INDIAN INSTITUTE OF SCIENCE EDUCATION AND RESEARCH, PUNE

Department of Mathematics

WINTER PROJECT 2019

A Vulnerability Review of Tangles & Blockchains

Author:
Ananyapam De, 18MS075
Indian Institute of Science Education and Research, Kolkata

Supervisors:
Dr. Anindya Goswami
Dr. Ayan Mohalnobis

December 2019

Contents

1 Introduction

Smart Contracts are one of the most promising features of blockchain technology and have created a lot of buzz over the years. They are essentially small computer programs that exist on the blockchain. This makes them immutable as long as the blockchain's integrity is not compromised. Therefore, the end-users can be sure that the program has not been changed or manipulated. This provides the people with trust in a distributed environment where they do not have to trust the other nodes or a centralised third party.

Blockchain has emerged as the most prevailing technology in recent decades. In the early stage, blockchain was simply applied to the direct payment system, represented by Bitcoin. Later, blockchain was employed as the distributed state machine to provide an executable environment for complicated tasks, represented by Ethereum and EoS. However, more and more specific situations, such as IoT, micropayments, and edge computing, require high properties on scalability, performance, and cost.

Ethereum is the second most valuable crypto-currency, just after Bitcoin. However, the most distinguishing feature of Ethereum was the introduction of smart contracts. This has fostered a new area of software development called decentralized applications (or dApps). These are applications that utilize the features of blockchains (immutability and lack of a central authority) in the backend and combine them with user friendly front-ends for non-technical users. The development of dApps for different application areas is an active area of research with the most popular ones being decentralised identity management, land record management, e-governance systems, etc.

Essentially, blockchains help in maintaining a tamper-proof log that is used to establish ownership of actions. Therefore, it has wide applications and can help establish the trust of the users in the system.

1.1 Blockchain Introduction

The first (and arguably the most popular) blockchain is Bitcoin - the brainchild of the pseudonym Satoshi Nakamoto. The main motive behind Bitcoin was to serve as a decentralised and distributed global currency - without any centralised banks. The ledger of transactions is stored as a chain (or a linked list) of blocks but the pointers connecting these blocks are hash pointers which are dependent on the block information. The block information does not store the transaction directly.

Each transaction becomes the leaf node of a Merkle tree, and the root of the Merkle tree is included in the block information. Therefore, any attempt to manipulate the data would not be possible without breaking (or forking) the chain. Also, as we are in distributed systems territory, we encounter one of the most famous problems of consensus, represented by the Byzantine General's Problem.

Essentially in a distributed system which is prone to Byzantine faults, how do we ensure that every node maintains the same ledger of transactions i.e. all the nodes are in consensus about the contents of the ledger? Bitcoin introduced an approach called proof of work (explained later in this section) to deal with it. However, other consensus algorithms like Proof of Stake and Proof of Burn, etc. are also used in other blockchains.

Finding the answer to the cryptographic hash puzzle (proof of work for Bitcoin) is called mining. This is because whoever finds the answer gets a block reward in bitcoin (essentially creating or mining new bitcoin).

1.2 Properties of Distributed Ledger Technologies(DLT's)

- *Decentralised Trust:* DLTs introduce a distributed consensus mechanism whereby users in a P2P network can interact and exchange data without the need for a trusted third party to guarantee the integrity of the ledger. In IoT this would allow devices with mutually exclusive trust domains to interact with one another without the need of a centralised server

- *Irreversibility, Immutability and Transparency:* A DLT grants an irreversible, immutable and transparent record of transactions. Each agent owns a copy of the ledger and can verify the correctness of any sequence of actions. In IoT this would allow multiple devices to orchestrate their behaviour in order to achieve a common goal;

- *Pseudo-Anonymity and Data Sovereignty:* Transactions in a DLT are pseudo-anonymous, due to the cryptographic nature of the addressing, and their contents can be encrypted. This allows each user to control access to and ownership of their own data;

- *Decentralised Trust:* Communication Security: The cryptographic nature of DLTs means that problems such as key management and distribution are handled intrinsically. In IoT each device would have its own unique ID and asymmetric key pair, and so light-weight security protocols (that would fit and stratify the requirements for the limited resources of IoT devices) become more feasible.

1.3 Consensus Algorithms

The Consensus Problem occurs when different nodes in a distributed system need to come to an agreement in the presence of malicious nodes or faulty communication channels. It is a very popular problem in distributed systems and Nakamoto gave an ingenious solution to it with proof of work. However, over the past decade different algorithms to solve this problem have come up with different blockchain platforms.

1.3.1 Proof of Work(PoW)

This approach was introduced in Bitcoin. Essentially, the miner (the person doing the work) has to find a value (also known as the *nonce*) which after being concatenated with the block contents and taking the hash gives a value which starts with a fixed number of zeroes (also known as the *difficulty*). The solution to finding this value is brute-force which is computationally very expensive.

Therefore if anyone has to change the blockchain contents, he/she has to calculate the nonce value for that block and all the blocks after it up to the current block. This is highly infeasible. Since the work requires high computational power, the nodes with higher computational power become the miners and control the blockchain. However if a miner (or a group of miners) have more than 51% of the hash power of the network, they essentially control what goes into the blockchain and the integrity is compromised. Also, the energy consumption of PoW blockchains is extremely high.

1.3.2 Proof of Stake(PoS)

The concept of proof of stake is very similar to share holder voting in any company. Who ever owns the most stake (or the most coins) gets the privilege to mine the next block. Some blockchains also define stake in terms of the amount a time a person holds a coin (also referred to as the coin-age). Ethereum's Casper Protocol is a Proof of Stake Protocol.

A 51% attack is generally considered more expensive in case of a proof of stake chain. However, many people are opposed to it ideologically as the it gives the 'rich' control over the blockchain. Also, there is a problem of 'nothing-at-stake' where a malicious miner looses nothing by betting on two different forks of the chain.

1.3.3 Remarks

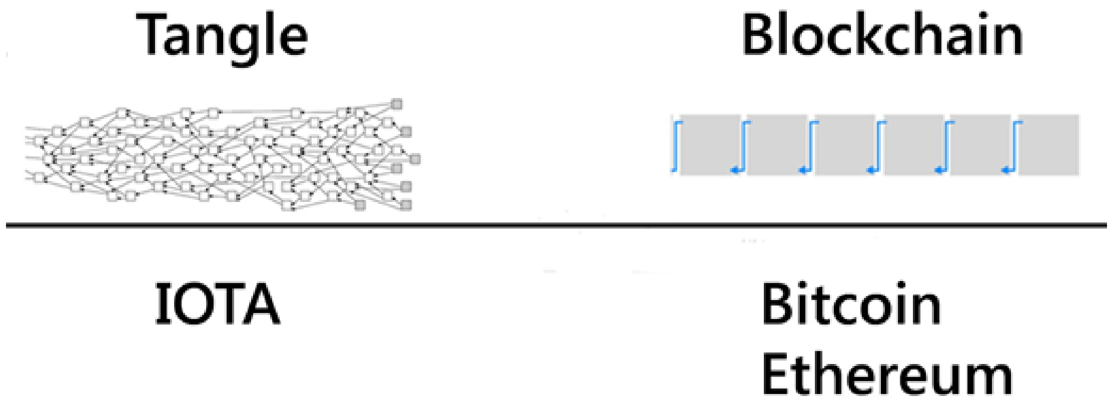
There are many varieties of DLT architecture, but not all are equally useful for IoT applications. In particular Blockchain, while responsible for the initial explosion of interest in DLT, has limited use in IoT applications: the large energy cost of mining, the volatility of bitcoin, long transaction approval times, transaction fees, and the inherent preference of miners to process large transactions rather than small ones, all create a bottleneck when thousands of devices communicate with each other many times per second.

However, the biggest advantage of smart contracts - their immutability also poses the biggest threat from a security standpoint. This is because any bug found in the smart contract after deployment cannot be patched. Recent attacks like the DAO attack and the Parity attack have caused massive monetary losses. More and more specific situations, such as IoT, micropayments, and edge computing, require high properties on scalability, performance, and cost.

2 The Tangle Structure

With these problems of scalability and low throughput, the structure of a Directed Acyclic Graph (DAG) is designed to improve the bottleneck of classic blockchain systems from the underlying structure. Different from the linear-chain in classic blockchains, DAG-based blockchain systems remove the limitation of blocks, expanding the network through a directed acyclic graph.

The following illustration explains the difference between the two structures elegantly-



Tangle structure is proposed by IOTA, one of the leading DAG-based systems. It aims to overcome the bottlenecks of current blockchain systems which include poor throughput on concurrency, low efficiency on performance, and high-cost on transaction fees.

The Tangle-based structure becomes one of the most promising solutions when designing DAG-based blockchain systems. The approach improves the scalability by directly confirming multiple transactions in parallel instead of single blocks in linear. However, the performance gain may bring potential security risks.

Newly generated transactions without packing into blocks establish the network by directly confirming their parent transactions. Through several iterative rounds, the main graph is formed with a low probability to be reversed.

Note the very major distinction in between the two structures: In proof-of-work blockchains like Bitcoin's and Ethereum's, there are two distinct types of participants in the system, those who issue transactions, and those who approve transactions, however in the tangle, every device works to maintain the ledger. Hence every node in the tangle is also a kind of miner, which is why we do not require additional miners, and hence no mining fees!

2.1 The Transaction Process

Every time a node wants to transfer some value, it must validate two previous transactions. This validation requires a small amount of Proof of Work(PoW) in order to secure the network, meaning that transactions are not strictly free. Since there is no distinct group of miners that must be compensated, though, there are no fees. This no-fee structure enables the kind of micro-transactions that would be impossible with bitcoin.

As a tangle transaction receives approvals, and the transactions approving it receive approvals in turn, the cumulative weight of that transaction builds up. Similar to confirmations for a bitcoin transaction, higher cumulative weights indicate more reliably immutable transactions. The gray boxes at the far right of the diagram, representing recent transactions that have received no validations, are called the *tips of the tangle.*

2.2 Consensus Without Blocks

Since transactions are not being shared all at once as blocks, divergences are more prone to happen on the tangle than the blockchain. It is important to note that the iota network is asynchronous. In general, nodes do not necessarily see the same set of transactions. It should also be noted that the tangle may contain conflicting transactions.

Eventually, some conflicting transactions are orphaned—not completed—while others stand. The tangle relies on incentives to reach consensus about these transactions' fate. If a node issues a new transaction that approves conflicting transactions, then it risks that other nodes will not approve its new transaction, which will fall into oblivion.

In order to find transactions to approve that are unlikely to lead its own transaction to be orphaned, a node runs a *tip selection algorithm*

2.3 Tip Selection Algorithm: Markov Chain Monte Carlo

The Markov Chain Monte Carlo (MCMC) algorithm would place at least two "random walkers" somewhere on the tangle. These random walkers should not be at the beginning as that would take too long, but not too recently as well, as then the quality of the selection would suffer. These move chronologically along the paths defined by validations, favoring paths linking transactions of similar cumulative weight. Say that transaction x (cumulative weight = 20) was approved by transaction y (= 19) and transaction z (= 3). The walker has a much higher probability of moving from x to y.

The rationale is that "lazy" nodes—ones that rarely issue transactions and therefore rarely validate others' transactions—will be at a disadvantage. Punishing lazy nodes is useful not just because it cuts down on free-riders, but because lazy nodes pose a risk for double-spend attacks: the white paper describes several such attacks and the ways an MCMC could be used to defend against them.

2.4 Advantages and Disadvantages

Tangle can be regarded as an expanding network formed by the continuously generated transactions. Transactions act as the smallest elements in the system to execute atomic operations, such as token transferring, witness validation, path extending, etc. The transaction-based Tangle possesses the properties of

1) **High throughput**: Transactions can be attached to the network from different directions and verified by previous transactions in parallel without serious congestion.

2) **High performance**: Newly arrived transactions are confirmed by the previous two transactions via a tiny Proof of Work (PoW) mechanism, where the computer consumption is negligible when compared to the traditional PoW.

3) **Low cost**: There is no transaction fees in Tangle-based systems, which perfectly fits for the high frequent situations including IoT, micropayment, and edge computing.

However, Tangle-based blockchain systems confront the potential threats caused by the forks of subgraphs which spread in different directions. Following the specified tip selection mechanism, the system forms a multi-directional expansive network to improve the scalability. More specifically, Tangle achieves the delayed confirmation based on previously verified transactions in each direction, instead of an instant confirmation such as Byzantine Fault Tolerance-style consensus.

The gap between delayed confirmation and instant confirmation leaves the blank of uncertainty and reversibility for attackers, same as other probabilistic consensus like PoW. Existing chains are also threatened by miners who own insurmountable computing power to send massive transactions. Newly issued transactions are unpredictably attached to different subgraphs without control. Forks will frequently happen and leave the system under the risk of parasite chain attack and double-spending attack. As a result, no leading subgraph can be formed to maintain the relatively stable states of the network.

2.5 Workarounds

In order to mitigate the issues, an additional centralized Coordinator is embedded in Tangle, for example the IOTA project, to resist the potential attacks. Transactions snapshot by the Coordinator is immediately considered to be confirmed with 100% confidence. In addition, milestones are periodically broadcast by the Coordinator to reach the consensus across multiple subgraphs for stability and record the history by removing useless branches. Nodes accordingly rebuild the Merkle tree which contains the Coordinator's address to verify the milestone.

3 Vulnerability Analysis

3.1 Blockchain 1.0 Vulnerabilities

3.1.1 51% Attack

In proof of work, the miners try finding the nonce value to solve the given cryptographic puzzle. However, if miner(s) get control of more than 51% of the compute power in the network then they essentially control what goes into the blockchain - compromising it's integrity.

3.1.2 Double Spending Attack

Double spending occurs when the attacker uses the same cryptocurrency more than once. This is done by leveraging race conditions, forks in the chain, or 51% attacks. A variant of this attack is the *Finney attack*

3.1.3 Selfish Mining

In selfish mining, a malicious miner does not publish the block immediately after solving the proof of work puzzle. Instead, reveals it only to its pool members which then work on the next block, while the other network continues working for essentially nothing. This was first demonstrated on Bitcoin, and recently on Ethereum as well.

3.1.4 Eclipse Attack

In eclipse attacks, the victim's incoming and outgoing connections are taken over by attacker (using a botnet or otherwise). This gives a filtered view of the blockchain to the victim. Several other attacks like selfish mining and double spending can be launched at the victim.

3.1.5 Private Key Security

Private keys are used to control the addresses in Ethereum. It is a security problem in itself to store these keys securely. As blockchain is a decentralised system, there is no way to report a stolen private key and prevent its misuse

3.1.6 BGP Hijacking Attack

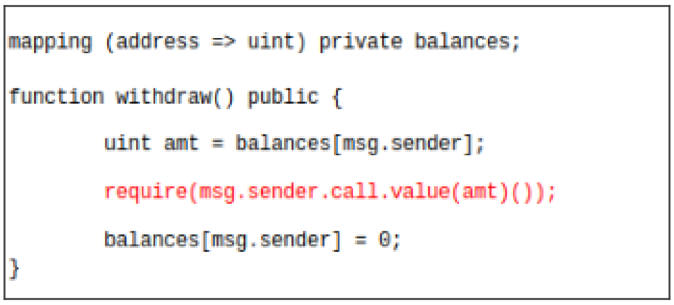
The border gateway protocol (BGP) is used for handling routing information over the internet. BGP hijacking is a common attack. However in the context of public blockchains, it can be used to create unwanted delays in the network and cause monetary losses to the miners.

3.2 Blockchain 2.0 Vulnerabilities

3.2.1 Re-entrancy

A re-entrancy condition is when a malicious party can call a vulnerable function of the contract again before the previous call is completed: once or multiple times. This type of function is especially problematic in case of payable functions, as a vulnerable contract might be emptied by calling the payable function repeatedly. The call() function is especially vulnerable as it triggers code execution without setting a gas limit.

To avoid re-entrancy bugs, it is recommended to use transfer() and send() as they limit the code execution to 2300 gas. Also, it is advised to always do the required work (i.e. change the balances, etc.) before the external call.



```
mapping (address => uint) private balances;

function withdraw() public {
        uint amt = balances[msg.sender];
        require(msg.sender.call.value(amt)());
        balances[msg.sender] = 0;
}
```

3.2.2 Transaction Order Dependence (Front Running)

The order in which the transactions are picked up by miners might not be the same as the order in which they arrive. This creates a problem for contracts that rely on the state of the storage variables. Gas sent is usually important as it plays an important role in determining which transactions are picked first.

A malicious transaction might be picked first, causing the original transaction to fail. This kind of race-condition vulnerability is referred to as transaction order dependence

3.2.3 Overflows and Underflows

Solidity can handle up to 256 bit numbers, and therefore increasing (or decreasing) a number over (or below) the maximum (or minimum) value can result in overflows (or underflows). It is recommended to use OpenZeppelin's SafeMath library to mitigate such attacks.

3.2.4 Timestamp Dependence

A lot of applications have a requirement to implement a notion of time in their applications. The most common method of implementing this is using the block.timestamp either directly or indirectly. However, a malicious miner with a significant computational power can manipulate the timestamp to get an output in his/her favour.

3.2.5 Forcing Ether to a Contract

Usually, when you send ether to a contract, it's fallback function is executed. However, if the transfer of ether happens as a result of a selfdestruct() call, then the fallback is not called. Therefore, a contract's balance should never be used in an if condition as it can be manipulated by a malicious user.

3.2.6 Bad Randomness

Online games and lotteries are common dApp use cases. For these applications, a common choice for the seed of the random number generator is the hash or timestamp of some block that appears in the future. This is considered secure as the future is unpredictable. However, a malicious attacker can bias the seed in his favour. Such attacks on Bitcoin have already been demonstrated.

3.2.7 Short Address Attack

It is an input validation bug that was discovered by the Golem Team. This allows the attacker to abuse the transfer function. The EVM automatically pads zeroes if the length of the address is less than the required length. This makes certain addresses with trailing zeroes vulnerabilities if proper sanity check is not done.

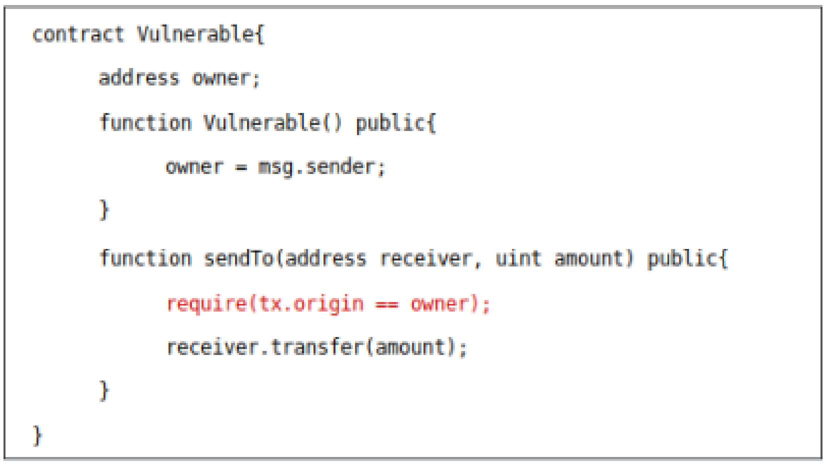
3.2.8 Unprotected Ether Withdrawal

Due to missing or inadequate access control mechanisms, it might be the case that anyone is able to withdraw Ether from the contract which is highly undesirable.

3.2.9 Authorization through tx.origin

This can be interpreted as a type of a phishing attack. In solidity, tx.origin and msg.sender are separate. The account calling a contract is defined by msg.sender. tx.origin is the original sender of the transaction, which might lead to a string of other calls. However, if tx.origin is used for authorization, and the actual owner is conned to call a malicious contract which in turn calls the victim contract, then the authorization fails.

In the example given below, the owner is set to the contract creator. However, if the contract creator is conned into calling a malicious contract which in turn calls the sendTo() function, the authorization passes and the attacker is able to siphon funds from the contract.



```
contract Vulnerable{
    address owner;
    function Vulnerable() public{
        owner = msg.sender;
    }
    function sendTo(address receiver, uint amount) public{
        require(tx.origin == owner);
        receiver.transfer(amount);
    }
}
```

3.2.10 Unprotected selfdestruct

selfdestruct kills a contract on the blockchain and send the contract balance to the specified address. The opcode SELFDESTRUCT is one of the few operations that costs negative gas as it frees up space on the blockchain. This construct is important because contracts may need to be killed if they are no longer required or if some bug is discovered. However, if this construct is put without proper protection mechanisms in place then anyone can kill the contract.

3.2.11 Function and Variable Visibility

Solidity has four visibility specifiers for functions and variables. However, being declared public is the most tricky from a security standpoint. If an important function like a payable function or a constructor with a wrong name is declared as public, then it can cause great monetary losses. Variable visibility does not have such drastic consequences as public variables get a public getter function.

3.2.12 Denial of Service

A denial of service attack from a smart contract's perspective happens when a smart contract becomes inaccessible to its users. Common reasons include failure of external calls or gas costly programming patterns.

3.2.13 Call to the Unknown

Ethereum Smart Contracts can make calls to other smart contracts. If the addresses of these smart contracts may be user provided then a malicious actor can utilize improper authentication to call a malicious contract. If the address is hard-coded, then it does not give the flexibility to update the contract to be called over time.

3.2.14 Exception Handling

Like in any object oriented programming language, exceptions may arise due to many reasons. These must be properly handled at the programmer level. Also, lower level calls do not throw an exception. They simple return a false value which needs to be checked and the exception should be handled manually.

3.2.15 Vulnerabilities introduced by Ethereum updates

Early in 2019, it was discovered by Chain Security that Ethereum's Constantinople update had an effect that might have made some smart contracts vulnerable to a re-entrancy bug. Such updates cannot be predicted by smart contract developers and the Ethereum Foundation ultimately delayed rolling out the update.

3.3 Tangle Vulnerabilities

3.3.1 Storage

Storage is an immediate concern for tiny, resource-constrained IoT devices. The white paper doesn't address this issue, but light-bulbs and toasters clearly aren't able to store the entire tangle, as full nodes do the entire 153 GB bitcoin blockchain or 338 GB Ethereum blockchain. Iota's development roadmap, published in March 2017, describes solutions including automated snapshotting—similar in principle to pruning—and a swarm client, which would allow devices to shard and collectively store the database.

3.3.2 Cryptography

In September, a team led by Neha Narula of the MIT Media Lab pointed out a flaw in the "curl" a cryptographic hash function developed in-house by the iota team. The impetus for designing the curl is the threat of quantum attacks, but designing a new hash function is a gamble, and it appears to have backfired.

3.3.3 Decentralization

If a party controls more than a third of the tangle's hashing power, the network is insecure. Bitcoin and Ethereum are generally considered secure as long as a single party does not control a majority of the network. In other words, while blockchains are vulnerable to 51% attacks, the tangle is vulnerable to a 34% attack.

Iota's implementation attempts to mitigate this vulnerability by amassing hash power itself using a "Coordinator" node which can be argued that this decision makes Iota centralized as the coordinator is now a single point of failure.

4 Conclusion

In this report, we looked at blockchains, specifically Ethereum and IoTa's Tangle from a security perspective. We began by discussing the basic concept of blockchains and few popular consensus algorithms and how tangle comes into the picture. We then discussed the structure of tangle and how the MCMC algorithm works in the context of tip selection. Finally we comprehensively reviewed all the attacks on the Ethereum blockchain and the IoTa's Tangle. Iota might be the our next natural step as the number of IoT devices further increases in number.

5 References

1. On the Resilience of DAG-based Distributed Ledgers in IoT Applications
2. Distributed ledger technology for iot: Parasite chain attacks
3. Iota-based directed acyclic graphs without orphans
4. On the security and performance of proof of work blockchains
5. The first glance at the simulation of the tangle: discrete model
6. Extracting tangle properties in continuous time via large-scale simulations
7. Analysis of Ethereum Smart Contracts - A Security Perspective
8. Security Analysis on Tangle-based Blockchain through Simulation