http://oraclewizard.com/Oraclewizard/2018/10/blockchain-a-primer/

Oraclewizard

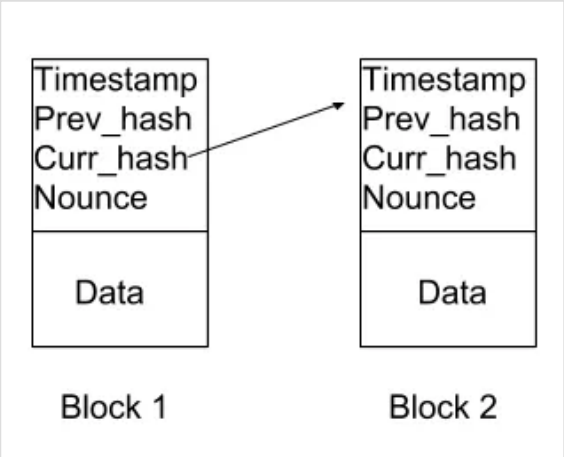
Veteran Owned HUBZONE Certified Small Business, Oracle ACE Director and Winner of the 2015 Oracle Developers Choice Award for Database Design. Oracle Database Security and Performance.

Blockchain A Primer

Posted on **October 22, 2018** by **rlockard**

Us technical nerds have a way of talking to each other, mostly we understand each other, sometimes we don't and frequently we throw out buzzwords, thinking everyone must know what we're talking about. This paper is going to address the subject of blockchain so anyone with a non-technical education can understand what it is and how it works. You may not be able to develop your own blockchain after reading this paper, but you'll have a good handle on what us nerds are talking about..

There seems to be some confusion about just what blockchain is. Many people I speak with automatically assume blockchain is Bitcoin. First off, Bitcoin (and other cryptocurrencies) are not blockchain and blockchain is not bitcoin. Bitcoin uses the secure data structure of blockchain to protect the data. Now because blockchain was described by Satoshi Nakamoto in 2008 to describe a Peer to Peer electronic cash system, and Bitcoin was the first application to utilize the security aspects of blockchain, this confusion is understandable. Now that we have established that bitcoin is not blockchain and blockchain is not bitcoin, lets address what blockchain is.
Blockchain is a way of linking transactions using a timestamp and cryptologic hash into a linked list to make the data immutable. Yea' that's a mouth full. But if you do much reading on blockchain, there have been some enhancements to what we can do besides making data immutable. The power of blockchain lies in, if you change anything in the blockchain, you break the cryptologic hash.



Basic Blockchain with two blocks.

Here we see two blocks that have been joined by block 1's current hash to block 2's previous hash. This is a simple linked list. Data is linked to other data by pointing to a unique value of its neighbor. Also a linked list can not have any branches, so in this instance we can not have two pieces of data pointing back to the same unique value of a neighbor.

Let's start taking this apart, to see what is actually happening and I'm going to start by defining what a hash is, what are the qualities of a good hash, and a tiny bit about how hashes work.

A hash is a one way cryptologic function that produces a value from some input. So if I pass to a hash function, "the rain in Spain falls mainly in the planes." the SHA256 hash function will produce: 4f5960e9f8aa23073bd14dfe85cce5020530e15f1f7dc4231d16cceb01d09e70. Now if I change one letter of the text to "The rain in Spain falls mainly in the planes." (just switching the t from lower case to upper case) The SHA256 hash function will produce: 5cadf57561ce0e294dd4c7b982be6ffad0c81281f0d6ab6fa64efccdfaf51061. As you can see two totally different results that don't even remotely resemble each other. I have read from many respected sources, the SHA256 hash is guaranteed to always return a unique value. This is not exactly a true statement. SHA256 produces an output of finite length, however there are an infinite number of combinations that can be passed to a hash function. Eventually, someone will find a hash collision (when two different inputs produce the same output) but for all practical purposes, it will produce a unique output for any given input.

The last thing you need to understand about hash functions, you should never be able to take the hash and find out what the original input was. That is if I was to give you this hash from SHA256 5cadf57561ce0e294dd4c7b982be6ffad0c81281f0d6ab6fa64efccdfaf51061 you should not be able to figure out what the input is. You may already see the problem with this statement. Because I used this hash in a previous example, you can go back and read what my input was. Because we know the input and we know the hash we can run it through the function to see if the message has changed. If they stay the same, we're golden, if not, the text was altered.

Difficulty: Tthe time to calculate a hash is amazingly fast, my last test was producing a SHA256 hash on a small input in 0.001001119613647461 seconds,. Yup, that's pretty quick. So, we actually need a way to slow down the ability to produce a hash so we introduce a difficulty factor to it. We do this by creating an artificial requirement that the hash must start with one or more zeros. In reality you can use anything you want; however the current standard people are using, is starting the hash with 0's. Each time we increase the difficulty, the time to calculate the hash increases exponentially. So with a difficulty of two zero, the average time to hash is 0.003002166748046875 seconds and a difficulty of three zeros the average time to hash is 0.2151656150817871. A difficulty factor of four and it took 200.69477248191833 seconds to calculate the hash. This is a pretty old computer I'm working on right now, so faster speeds are possible, but you get the point, as the difficulty increases, it becomes much harder to calculate a hash with the required difficulty.

The reason we are adding in a difficulty factor is to slow down the ability to rebuild the blockchain and defeat the security that is built, by linking the data (blocks) with a hash. So, if the speed to calculate the hash is slow, a potential attacker would need much more CPU power then the machines building the blockchain to change some data and then recalculate all the hashes in the blockchain, and then get ahead of the machines building the blockchain.

I promise if your not a techie, this is not that painful. What I want you to see is the technique that I used to calculate the hash with a given difficulty.

```
def f_calc_hash(self, data, nounce, difficulty):
s_curr_hash = sha256(data.encode()).hexdigest()
while not s_curr_hash.startswith('0'*difficulty):
```

```
self.nounce += 1
data = data + str(self.nounce)
s_curr_hash = sha256(data.encode()).hexdigest()
return s_curr_hash
```

This is actually pretty easy to read, we have a function to calculate the hash with the required difficulty. It takes the data we want to hash, a nounce that I'll get to in a minute and our difficulty factor. It starts by calculating the hash and checks to see if it starts with the required number of 0's. If it does not, then it adds 1 to the nounce, and puts that number at the beginning of the data we are hashing. Because we have now changed the data, we will get a different hash. We continue adding 1 to the nounce and testing the hash to see if it meets the difficulty requirements. If it does, great, return the hash, if not, keep adding 1 to the nounce and getting the hash again. There is no way you can figure out what nounce to use, therefore, the only way to get a hash with the required difficulty is to use brute force.

So a **Nounce** is just a variable we use to add to the data we are hashing to get a different hash. That's all. Where the term nounce came from, I don't know.

We also add a **Timestamp** to the transaction to prove when the transaction was created and also in some cases to make sure the transaction is unique and we can get a unique hash for each transaction.

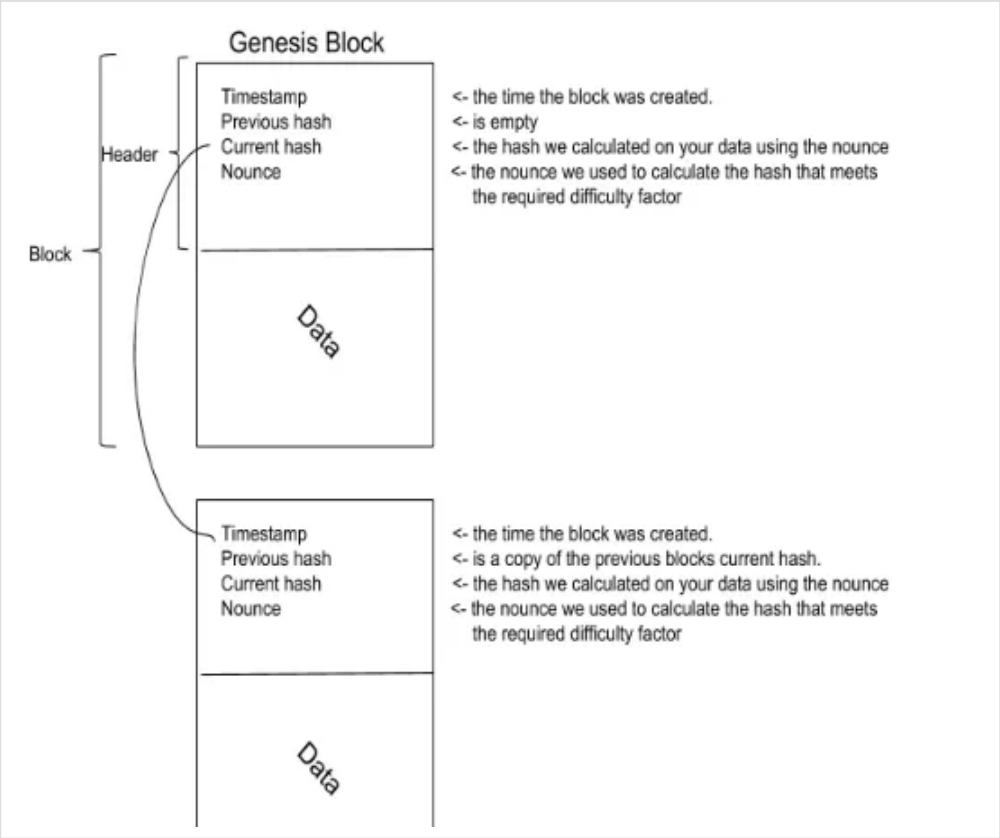
Transactions all go into a **Block**. You can have one or more transactions in a block, the number is up to you and the requirements for your system.

So let's use the simple example of a check register. These two transactions can go into a block to be added to the blockchain.

Check#	Date	For	Amount	Deposit
256	05 May 2018	Gas	50.00	
.	05 May 2018	Payroll Deposit	.	5,700

Before we go much further, we need to explain how to weld blocks together using a hash that meets the difficulty requirements.

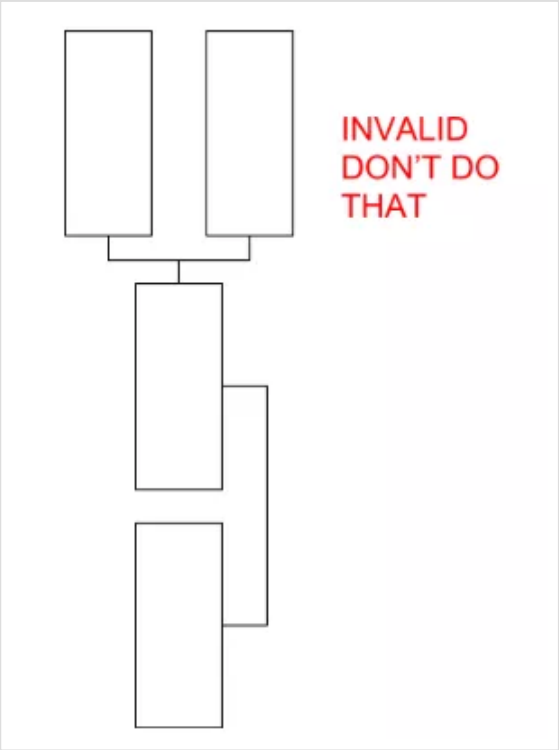
All blockchains start with a **Genesis Block**, this serves as the anchor to the blockchain by providing the first hash needed to weld the blocks together. There is one special thing about the genesis block, it does not have a prior hash in its header. There is one other thing about the genesis block you should understand, The data in it may not be of any value. It's just a seed for the rest of the blockchain. When we have built the genesis block, we get a hash that meets the difficulty requirements and put that in a field called current_hash.The **Current_hash** is the hash value we get when we calculate the hash with the required difficulty factor using the calculated nounce.



So when we add in block#2, it copies the current hash from the genesis block into the previous hash field. We then hash the block to get the current hash and nounce that meets the difficulty requirements. By joining block# 2 to the genesis block, we now have a blockchain with two blocks. Every block we add to the blockchain, we repeat this process.

Why is the data structure so powerful? Let's start with the assumption that we have a blockchain with 10,000 blocks. If someone were to change one piece of data, say in block 500, then the current hash in block 500 would not be right anymore. Let's say our hacker is clever and calculates a new nounce to get a hash that meets the difficulty requirements. Well then the previous hash for block 501 would not match the new hash that was calculated. So, the hacker now has to calculate a nounce for block 501 and to get a hash that meets the difficulty requirements. Then, 502, and 503, on and on. It would require a lot of CPU to recalculate the entire blockchain. If we did not have a difficulty factor built into the blockchain, the speed to calculate a hash is pretty darn quick. That difficulty factor puts a speed limit on just how fast you can write to a blockchain. Pretty darn clever if you ask me.

Anonymity vs Known Users or Public vs Private blockchains. We have the very basics of what a blockchain is, now we can look at a couple of different types of blockchains. First there is a public blockchain where anyone can add data to the blockchain. The typical public blockchain is cryptocurrencies such as Bitcoin. Note: Cryptocurrencies are also frequently called shadow currencies. In cryptocurrencies, anyone can add blocks to the blockchain and make money. Here everyone has a copy of the blockchain and when you add to the blockchain, you provide a proof of work and other nodes in the blockchain check your work. If you met the required difficulty and the nounce is correct, then your block is added, that is unless another node has added more blocks then you. Because we can not have branches on a blockchain, the network uses the node that is longest as the valid blockchain.



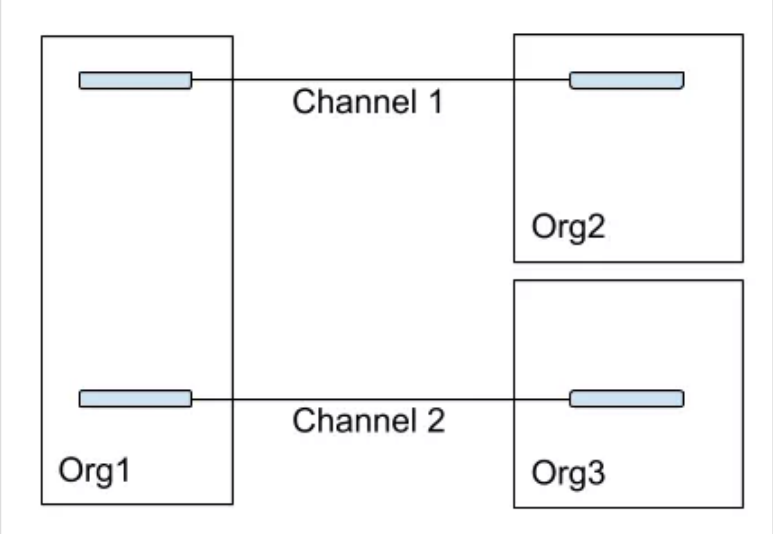
When we say branches, that means two or more blocks using the current hash of a previous blocks as their previous hash.

So now that we understand what a public blockchain is, let's explore what a private blockchain is.

In a private blockchain only computers (nodes) that are invited in can participate. Private blockchains can be used to store medical information that has regulatory protections, supply chain management, and trade between businesses.

In the public blockchain, we used proof of work to add data to the blockchain. In the private blockchain we use **Selective Endorsement** by using **Endorsing nodes.** Endorsing nodes are the machines that are authorized to add data to the blockchain. The endorsing nodes can be one node, many nodes, or all the nodes on the private blockchain.

When we have multiple entities on a private blockchain, not everyone is allowed to see all the data in the blockchain. We enforce this by using channels. If you are dealing with financial or healthcare data, you must control who has access to the data. When a government, business, or person joins a private blockchain, they subscribe to a channel. The simplest way to describe a channel is it is a standalone blockchain that is shared betw*een players on the system.



A member of a private blockchain can subscribe to one or more channels. In this example Org1 is subscribed to both channel 1 and channel 2. Org2 is subscribed to channel 1 and Org3 is subscribed to channel 2. Now, the beauty of this is, the data has been segmented. That is to say, nothing in channel 1 can appear or be written to channel 2 and nothing in channel 2 can appear or be written to channel 1. Each of these channels is in fact, a private blockchain.

What can we do with this? Glad you asked, because the information has not been changed, we can use the data in the blockchain to trigger **Smart Contracts.**

Smart contracts are a way to trigger an event after conditions are met. A simple example of a smart contract would be to send payment to a vendor. This would be accomplished with a simple if .. then .. else statement. In a purchasing blockchain a Smart Contract may exists that says, IF Shipment Received AND Shipment Verified AND Invoice Received AND Matching Purchase Order THEN Pay Invoice. Smart Contract are a good way to speed up the flow of business processes by embedding logic into the blockchain that all parties agree to.

RELATED

Upcoming Talks
August 9, 2018
In "infosec"

#POUG2018: That's a wrap; what a great trip.
September 21, 2018

2017 was a crazy year, 2018 is going to be challenging
It's been a crazy year. In 2017 I've done talks in Paris France, Helsinki and Rovaniemi Finland, Sofia Bulgaria, Moscow Russia, Denver Colorado, Las Vegas
November 22, 2017

In "infosec" In "Database Stuff"

This entry was posted in **infosec** and tagged **blockchain**, **blockchain channels**, **blockchain made simple**, **blockchain primer**, **blockchain smart contracts**, **genesis block**, **infosec**, **primer** by **rlockard**. Bookmark the **permalink [http://oraclewizard.com/Oraclewizard/2018/10/blockchain-a-primer/]** .

About rlockard

Robert Lockard is a professional Oracle Designer, Developer and DBA working in the world of financial intelligence. In 1987 his boss called him into his office and told him that he is now their Oracle Wizard then handed him a stack of Oracle tapes and told him to load it on the VAX. Sense then, Robert has worked exclusively as an Oracle database designer, developer and Database Administrator. Robert enjoys flying vintage aircraft, racing sailboats, photography, and technical diving. Robert owns and fly's the "Spirit of Baltimore Hon" a restored 1948 Ryan Navion and lives in Glen Burnie Maryland on Marley Creek

View all posts by rlockard →