Notes on Bitcoin and Blockchain by a Mathematician: Explanation and Implementation

Daisuke Natthaworn Sakai

November 30, 2021

Contents

1 Encryption

A problem is said to be computationally easy iff it is computable in polynomial time. It is said to be hard or computationally infeasible iff it cannot. Computationally easy usually means that something can be computed in under 1 second. Computationally infeasible usually means it will take more than 100 years. Some computationally infeasible problems would take longer than the age of the universe to calculate using current computers.

Denote $\mathbb{S}$ as the set of all strings. Denote $\mathbb{N}$ as the set of all natural numbers.

Definition 1. A hash function takes in some string of arbitrary length and maps it to a string of fixed length, where:

1. Any minor difference in the input string drastically and unpredictably changes the output string.
2. The output is evenly distributed. That is, for any two output strings, the probablility that one of them will be outputted given an arbitrary input string is the same.
3. The output is easy to compute.

Definition 2. A hash function is further said to be cryptographic iff:

1. The fastest way to compute a string that produces a given output is by brute force; that is, one needs to try inputs one by one until one reaches the desired output.

A useful hash function is therefore irreversible: it is computationally infeasible to find an input for almost all possible hash outputs. Such hash functions exist; the SHA256 hash developed by the US National Security Agency outputs a string of length 64 of alphanumeric characters. The possible output combinations amount to $36^{64} \approx 4 \times 10^{99}$, which is around 50 magnitudes larger than the number of atoms which make up the earth.

Definition 3. A Public Key encryption protocol is an ordered triple (G, h, k) of functions such that:

1. G maps from $\mathbb{N}$ to $\mathbb{S} \times \mathbb{S}$. Given any output of G, the first string is called the "public key" and the second string is called the "private key".
2. The function h maps from $\mathbb{S} \times \mathbb{S}$ to $\mathbb{S}$. It takes in a private key and encrypts a message m and produces a garbled string.
3. The function k maps from $\mathbb{S} \times \mathbb{S}$ to $\mathbb{S}$. It takes in a public key and decrypts a message and produces the original message m.

The following conditions need to be satisfied:

1. It is easy to compute a public and private key given any natural number n. It will be usually easy to compute the public key from the private key.

2. It is computationally easy to encrypt a message using a public key.

3. It is computationally easy to decrypt a message using a private key.

4. It is computationally infeasible to decrypt any message given a public key.

5. It is computationally infeasible to deduce the private key from the corresponding public key and any encrypted message.

A public key protocol allows the public to send encrypted messages to an individual.

Suppose Bob wants to send Alice a secret message. First Alice will create a private and public key with a secret random initial seed. The public key is sent to Bob, and Alice keeps the private key secret. Bob uses the public key to encrypt the message. Then Bob sends the message to alice. Since only Alice has the private key, only she is able to decrypt the message.

However, this does not allow Alice to know whether the message she recieved was from Bob. The public key can be used by anyone who has it to write a message and send it to Alice. A digital signature by Bob addresses these issues.

Definition 4. A Digital Signature protocol is an ordered triple (G, S, V) of functions such that:

1. G maps from the natural numbers to all ordered pairs of two strings. Given any output of G, the first element of the ordered pair is called the "public key," and the second element the "private key".

2. Given the set of all possible private keys outputted by function G, which we denote $G[\mathbb{N}]^2$, the map S takes from $G[\mathbb{N}]^2 \times \mathbb{S}$ to the set $\mathbb{S}$. The output of S is called the "signature," or "tag". The input string is called the "message".

3. Given the set of all strings $\mathbb{S}$, the map V takes from $\mathbb{S} \times \mathbb{S} \times G[\mathbb{N}]^1$ to the set $\{0, 1\}$. If first input is the message and the second input is the signature, and the third input is the public key, then the function returns 1. Otherwise, with very high probability, it returns 0.

The conditions need to be as follows:

1. It is easy to compute the ouput of G given any $n \in \mathbb{N}$. It will be usually easy to compute the public key from the private key.

2. It is easy to compute the signature given any private key and message.

3. The function V is easy to compute. That is, it is easy to verify a message.

4. Given any message m and public key pk, and the functions of G, S, V, it is computationally infeasible to deduce a signature σ such that $V(m, \sigma, pk)$ gives a value 1.

5. Given signature, message, public key and the functions of G, S, V, it would be computationally impossible to deduce the private key.

Now suppose Alice wants to make sure that Bob's message is authentic. Bob creates a public and private key using a random large natural number for the digial signature protocol. Bob then shares the public key with Alice and keeps the private key secret. Bob then uses the private key and the message to generate a signature. Both the signature and the message is then sent to Alice, who has Bob's public key. Alice then inputs the message, signature, and public key into the function V. If the message or signature were tampered with, then the function with very high probability returns 0.

There are many of these functions and protocols which are publicly available for use and implementation.

2 Digital Coin Motivation: Debt

Let us say that Alice and Bob go out to eat, and Alice has forgotten her wallet. She asks Bob for 10 dollars to pay for mutton biryani, and writes an IOU. However, if Alice wrote "IOU 10 dollars - Alice", and she forgets about it, she wouldn't know if it had been forged in her handwriting. Furthermore, since Alice does this so many times with many people, Bob proposes that they agree to a protocol, as follows. Alice first creates a public key and private key for a digital signature protocol, and shares the public key with Bob, and signs the string with her private key:

```
IOU#1
Alice will pay 10 Dollars.
Bob owns this ticket.
```

and creates some signature *Alice_sig*. Alice and Bob then both agree that by signing this specific string, it means that alice needs to pay Bob 10 dollars, and that the bond number is 1 to distinguish between multiple bonds (For example, Alice might again ask for 10 dollars the next day, and we need to have a different string to say that alice owes Bob 10 dollars twice. It will clearer why we need to do this once we introduce how Bob confirms the IOU payment from Alice). The string doesn't have to be in the above format, of course, as long as Alice and Bob both agree on rules of how to interpret the string. For example, they can also instead to agree that

```
#1_Debtor:Alice_Creditor:Bob_AmountInDollars:10
```

would mean the same thing. For now let us stick with the original and easier to read format.

Then alice writes the following string for Bob to keep.

```
IOU#1
Alice will pay 10 Dollars.
Bob owns this ticket.
Alice_sig
```

Alice and Bob agree, that by signing the string, Alice is obliged to pay Bob back 10 dollars. Let us think about this for a second. Since Alice signed the string with her private key, the message and signature are sufficient for Bob to claim to Alice that she pays Bob, because only she can produce the signature corresponding to the string. Therefore, if Bob comes to Alice after one year, and asks her for the money, and because Alice owed 20 other people 10 dollars as well and couldn't keep track of who she owed money to, Bob can show Alice the string that she signed and claim that only she could have signed the string because only she had her private key. Both Alice and Bob can verify the signature with Alice's public key that indeed it was signed by Alice. Since they agreed beforehand that signing the string meant that Alice had to pay Bob back, this is what she must do. It is important that Alice should not leak her private key to Eve otherwise Eve can sign the following string:

```
IOU#2
Alice will pay 10 Dollars.
Eve owns this ticket.
```

and Alice would be obliged to pay Eve 10 dollars if she were to be shown the string with the signature (if they had made a similar agreement).

Now what if Alice already paid Bob back but he comes back with the same string a year later and asks Alice for money again? Alice and Bob agree to the following protocol to avoid this situation.

Bob also creates a public and private key, for a digital signature protocol. He keeps his private key secret, and shares his public key with Alice. They agree that, if Alice paid Bob back for a specific ticket, he would append the string "Alice has paid back the ticket to Bob." and obtain a new string

```
IOU#1
Alice will pay 10 Dollars.
Bob owns this ticket.
Alice_sig
Alice has paid back the ticket to Bob.
```

and sign the updated string with his signature. Then only Bob would be able to produce the signature such that the function V outputs 1 on the message, signature, and public key, so Alice and Bob both would be able to verify that it was indeed Bob who signed it.

Now if Alice had paid Bob back the next day, and Bob adds the string and signs it, then the contract is voided, and both Alice and Bob keep the final string. If Bob comes back a year later and shows Alice the string

```
IOU#1
Alice will pay 10 Dollars.
Bob owns this ticket.
Alice_sig
```

and requests money from Alice, then Alice can simply show the string

```
IOU#1
Alice will pay 10 Dollars.
Bob owns this ticket.
Alice_sig
Alice has paid back the ticket to Bob.
Bob_sig
```

to rebuff Bob's claim. Then Bob would have to accept that Alice did in fact pay back the money for the ticket that he presented.

Now what if Alice and Bob agreed instead that when Alice paid Bob back, that he would sign the following string?

```
IOU#1
Alice has paid back the ticket to Bob.
```

Then since the ticket number is shown so both Alice and Bob know to refer to which IOU ticket Alice paid back to Bob.

This is also a valid method, but it's best to keep things in a single file rather than multiple files. It's easier to lose things when you have too many of them.

Of course we would be able to specify more details of a the IOU, such as the interest rate, the expiry date, and so on, and include other information, such as date of crediting.

3 Digital Coin Contract

We have introduced a simple debt contract between two parties using the public key protocol, which allowed two parties to securely agree to some arrangement of money to be paid. What if we wanted to be able to trade this debt instrument like a bond in some sort of market? It then is a simple matter of transferring ownership from one party to another.

Suppose that Alice borrowed from Bob 9 dollars on Jan 1, 2021, and tells Bob that she will repay him 10 dollars next year. She writes Bob an IOU as the following string.

```
IOU#1 Jan 1, 2021
Alice will pay 10 Dollars on Jan 1, 2022.
Bob owns this ticket.
Alice_sig
```

Let's say that there is a market in which these sorts of contracts are made every day. Everyone in the market, including Alice and Bob agree to a single format of strings that they can interpret into a contract of obligations, and can all agree who needs to pay who if they look at the string. Everyone knows each other's public keys for their signatures, and can verify that Alice indeed signed the string declaring that she pay back the debt in one year.

Let us say further that we want to have a system of transferring ownership of debt. Bob, being aware that Alice has a habit of defaulting on her debts, decides to sell it to an unknowing Eve on April 1, 2021, who doesn't know

much about Alice. She gladly agrees to pay 9.5 dollars for the debt instrument. Everyone participating in the market of buying and selling debt needs to agree on a format of how this ownership transfer of a ticket works. Alice, Bob, and Eve agree that the owner of the ticket is the one who the debtor is obliged to pay back the money. They also agree that given a current owner x, and buyer y, by appending a string "y owns this ticket." at the end of the previous string and then appending x's signature on the whole string, the new string is interpreted to mean that y is the owner of the ticket. The process of selling would comprise of Bob recieving 9.5 dollars from Alice, and then appending the aforementioned strings to obtain the new string:

```
IOU#1 Jan 1, 2021
Alice will pay 10 Dollars on Jan 1, 2022.
Bob owns this ticket.
Alice_sig
Eve owns this ticket. April 1, 2021
Bob_sig
```

and now Alice, when shown the string by Eve, would be obliged to pay Eve 10 dollars on Jan 1, 2022, instead of Bob.

Now suppose that Alice won the Jackpot for a million dollars. She is now rich and can pay back any of her debts. Since Bob wants some of that money, on Jan 1, 2022, he shows Alice the string:

```
IOU#1 Jan 1, 2021
Alice will pay 10 Dollars on Jan 1, 2022.
Bob owns this ticket.
Alice_sig
```

and asks Alice for money. But she also has a claim from Eve. Eve can show the updated string to Bob, with his signature to show the transferral of ownership, making Bob's claim void. Therefore, Alice can look at all the claims for her money at expiry, and find the string with the longest chain of ownership, and the last person on the longest string is the current owner of the ticket. Therefore all participants can agree on who owns the ticket at expiry.

We now move on to define the abstract definition of a digital coin. For a market of participants, a digital coin is defined as follows.

Given participant n in our digital coin market, denote $Public\_n$ as the public key of the nth owner of a digital coin, and $Private\_n$ as the private key of the nth owner of a digital coin. Given two strings $str1$, $str2$, we denote $str1 + str2$ as the appendage of the two strings. Given a string str, denote $str + sig\_n$ to denote the signature of participant n signed with the message str.

A digital coin is a string of the following form:

```
Bitcoin_1:
Public_0
Public_1
sig_0
```

```
Public_2
sig_1
Public_3
sig_2
...
Public_n
sig_n-1
```

We will say that the chain of ownership is verified iff for each string starting from the top up to a signature, the ith signature belongs to the $i-1$th last public key in the string.

Now for reasons including data storage and efficiency in calculation (these reasons will be elaborated when we discuss blockchain and proof of work), we will incorporate the usage of a hash function in the digital coin.

When a bitcoin is created by owner 0, he will hash the following string:

```
Bitcoin_1:
Public_0
```

We will denote the hash as $hash\_0$, which we shall call the initial hash. This will then be appended to end of the string to obtain the following new string:

```
Bitcoin_1:
Public_0
Hash_0
```

When owner 0 decides to transfer the coin to owner 1, he $Public\_1$ and appends it to the end of the string. He then hashes $Hash\_0 + Public\_1$, the result which we denote as $Hash\_1$. This is then appended to the string. Finally, owner 0 digitally signs the $Hash\_1$, which we denote as sig_0 and appends it to the end of the string. So we get the following string:

```
Bitcoin_1:
Public_0
Hash_0
Public_1
Hash_1
sig_0
```

The string comprising of $Public\_i, Hash\_i, sig\_\{i-1\}$ is called the ith transaction. The procedure to obtain transaction i from the previous transactions is as follows:

Given a digital coin, to obtain transaction i, we take the previous hash and the public key of the ith owner and obtain $Hash\_\{i-1\} + Public\_i$ and hash it to obtain $Hash\_i$. Then $Hash\_i$ is signed by the owner $i-1$, obtaining $sig\_\{i-1\}$. Then the strings $Public\_i, Hash\_i, sig\_\{i-1\}$ are appended to the digital coin. This process is called a transaction, and we will say that the coin is transferred from owner $i-1$ to owner i.

There are several advantages to this approach. To verify signature $i-1$, it is enough to input $Hash\_\{i-1\} + Public\_i$ and the public key $Public\_\{i-1\}$

and $sig\_\{i-1\}$ into function V. For a million transactions, in the previous approach, the input message in V must be at least many million characters long, whereas since the hash has a limited size, we only have a short string each time we verify a signature.

Once owner i verifies the chain of ownership, he can then confirm that all previous transactions were valid, so he can accept the payment.

A digital coin contract is a set of rules describing the ownership of a digital coin that market participants agree to. In particular, these comprise of the following:

1. An agreement of the format of the digital coin, and the format of new transactions appended to the digital coin. In particular, this includes specification of the hash function used, the type of digital signauture protol, and how to name each coin, whether to include the time and date of transaction, and the format of time and date of transaction, if included, and so on.

2. The individual whose public key appears last in the digital coin is the owner of the coin.

3. The coin with the longest chain of ownership is the only contractually valid coin.

4. Participants may offer goods, services, or other assets in exchange for the transferral of a contractually valid digital coin to their name.

Now consider this in a practical real world setting. A group of friends can agree to this protocol, without relying on a central authority for legal enforcement of debt payments. If Alice creates an IOU certificate for Bob, and doesn't pay him back, then the rest of the market participants can beat her up. The framework of payments works out as long as the majority of the participants agree on the rules and beat up people who participate in the market, but breach the rules.

As it turns out, the digital coin is insufficient as a transaction system as it needs to address the double spend problem. This is discussed in the next section.

4 The Double Spend Problem

Let us return to the tradable bond created by Alice and Bob. Alice and Bob create the following string:

```
IOU#1 Jan 1, 2021
Alice will pay 10 Dollars on Jan 1, 2022.
Bob owns this ticket.
Alice_sig
```

which they interpret to mean that Alice is obliged to pay Bob 10 dollars on Jan 1, 2022.

Now suppose that Bob decides to sell to Carol the ticket. After Carol's payment on April 1, 2022 of 9.5 dollars, he signs the string, and both he and Carol have the following string:

```
IOU#1 Jan 1, 2021
Alice will pay 10 Dollars on Jan 1, 2022.
Bob owns this ticket.
Alice_sig
Carol owns this ticket.
Bob_sig
```

All is well, for now. But since this transaction occurred between only Bob and Carol, neither Alice nor Dave knew about the transfer of ownership. Bob realizes that he could make more money if he could sell Dave the bond if he showed Dave the original string before Bob sent the coin to Carol. Dave looks at the original string, and sees that Bob is the current owner, so gives Bob 9.5 dollars on April 2, 2022. Bob updates the string to the following:

```
IOU#1 Jan 1, 2021
Alice will pay 10 Dollars on Jan 1, 2022.
Bob owns this ticket.
Alice_sig
Dave owns this ticket.
Bob_sig
```

Now under our digital coin contract, both Carol and Dave have valid claims to Eve's money, because both versions of the coin the longest chain of ownership. Eve now has to pay 20 dollars in total rather than 10 at expiry, and Bob gets away with 19 dollars.

The above problem is called the "double-spending problem" or the "double spend problem", which is cleverly solved by the Blockchain protocol.

5 The Blockchain Protocol

A blockchain contract is a set of rules that participants of a digital coin market agree to in order to avoid the double spend problem.

We need some method so that if someone spends a coin, they cannot spend it again. So if a coin has been spent, we reject later attempts for the previous owner to spend it again.

In order to stop Bob from spending the same coin twice, everyone needs to know when Bob makes the transaction. If everyone knows that Bob spent the coin by sending it to Carol, then if he tries to sell the coin to Dave at a later date, Dave will reject it, as he knows that Bob doesn't posses the coin.

So we need the first rule:

- Once owner i of a coin completes a transaction and recieves goods, services or other assets from another market participant, other market participants

will not accept the coin as payment for goods, services, or other assets that they may possess.

This rule can only be enforced if all transactions and when they occurred have been made public. Along with a set of market participants, we also need to have a set of network participants which are witnesses of each transaction. Each network participant is called a node. Market participants can also be node participants and vice versa. One method is to publish these on some webpage or a newspaper, for everyone to see, but it would be highly inefficient to verify transactions this way. We need to have a network of computers who are sent the transactions, they are considered members of the public who can agree which transactions are valid. But this is not enough. What if a transaction made was only sent out to some of the nodes, and not all? Some users who were sent the transaction would reject all future double spending but those who did not would not reject it would not. In this way, the public would not be in agreement as to what happened. We need a method to create a single authoritative history of transactions which all nodes, whether they received a transaction or not, can agree to. Further, nodes can become offline at any given moment but return to the network and be able to obtain the history of what happened while he was offline. New nodes need to be able to join the network and be able to obtain the history of all transactions as well. The protocol propounded by Satoshi Nakamoto that solves all these problems is called the "Blockchain". We first describe the blockchain protocol for digital coins and then explain why works, and why it can be trusted.

Market participants and nodes all agree that when a transaction has been made, it must be broadcast to all available nodes. From this information, nodes will then add these transactions to what is called a blockchain, which is a series of blocks. Each node has their own version of the blockchain stored in their computer, but in the long run, the longest chain will be accepted by the most nodes and will be the chain that has only valid transactions. How the blockchain is created by the nodes is explained next.

To start a market, the initial block (known as the "genesis block") is created along with a hash of that block, which we call the block hash. The initial block must contain information such as the rules of spending and recieving and what kind of transactions are accepted for that blockchain, and the digital signature protocol. For example, if the particular blockchain is for bitcoin, then the initial block must specify that it records transactions for bitcoin and only bitcoin, not some other coin such as etherium.

When a market participant transfers a coin to some other market participant, he informs all nodes that the transaction took place. Each 10 minutes (approximately; how this is determined is explained afterwards), the nodes gather all the transactions which they recieved into a single string called a "block" of transactions. If a transaction is invalid, that is, a coin that has been spent is being spent again, then the nodes don't include that particular transaction in the block. Each node might recieve valid transactions at different times and might miss out on some transactions, but this is alright, for the moment. We then

obtain the hash of these transactions, which we will call the transaction root hash. The nodes then input the transaction root hash plus the previous block hash plus some number α (natural or alphanumeric) into a hash function. α is called a "nonce (number only used once)". This number α needs to be one such that the first n bits of the hash output is 0. This can only be done by brute force and expends computational power. n is calculated by looking at the rate at which previous blocks were added to the chain. Suppose M is the number of blocks that were added to the chain within the last 24 hours. Then n is then defined by the following equation:

$$n = n_{prev} + Activation(Round(144/M - 1))$$

where *Activation* is the activation function that outputs 1 if the input is positive, and outputs -1 if the input is negative, and *Round* is the rounding function which rounds to the nearest integer. Hence each block needs to be timestamped with a GMT time, which receiving nodes can verify at each time. If the number of blocks mined is on average once per 10 minutes in the past 24 hours, then we see that $n = n_{prev}$.

When a node finds α, we say that it "solved" the hash puzzle. Once such a number is found by some node N, it gathers the previous previous block hash, the root hash, and the nonce into one string called the "block header". This block header is then hashed to create the block hash for the current block. The block of transactions gathered by node N, the block header and the current block hash is then broadcast to all participant nodes. All participant nodes who receive this information can either accept or reject this block. They accept the block if and only if they are able to confirm:

1. All the coins are confirmed to not be already spent in the previous transaction blocks.

2. The purported transaction root hash is indeed obtained by hashing the transactions that node N purported to have received.

3. The purported block hash is indeed obtained by hashing the string which is previous block hash, the current root hash, and the nonce α.

4. The purported block hash is all zero in its first n bits.

If the nodes accept the block, then they will record the block and its header into their blockchain. Note that a set of transactions that one node received may not be the same as what another node received.

Market participants will need to agree on how to interpret this blockchain: The transaction at a given block came after transactions from any preceding blocks. They are able to query nodes, who agree to comply with these queries, about any past transaction in the blockchain of that particular node. If the majority of nodes agree on a certain transaction, the market participant accepts it as true.

Market participants can therefore give away their goods or services once they confirm that they have recieved the required amount on the blockchain.

We have therefore solved the double spend problem by agreeing on a single history of transactions which is continually broadcast to all participating nodes. Market participants who offer goods and services in exchange for ownership of digital coins are then able to query nodes to see if the single history agreed by nodes include the transferral of ownership of a coin to their name and then give their goods to the person transferring the coin to them.

The above is a summary of the blockchain. The reader surely has many questions about it, such as:

1. Why should the nodes need to solve a hash puzzle?
2. Why is n determined by the above equation?
3. How does each node determine how long they wait to collect transactions before trying to solve the hash puzzle?

These will be answered in the following sections.

6 Proof of Work and its Explanation

Define an honest node as a node in the network which sticks the agreed protocol exactly.

To solve the hash, one needs to expend electricity and cpu/gpu power. The underlying assumption of the nodes is that the majority of them are honest. If the majority of the CPU power of participating nodes are controlled by honest nodes, then the chain that grows the fastest is the correct record of transactions.

There are multiple incentives against nodes being dishonest.

7 Transaction Format, Ownership of Coins

Suppose we know that Bob bought a digital coin as recorded in the t_ith transaction in block i and bought another one as recorded in the t_jth transaction in block j. He knows t_i, i, because when the hash puzzle for block i was solved, the node that solved it broadcast t_i, i to Bob. The same applies for j.

Suppose Alice is selling one apple for 1 digital coin, and Bob wants to buy it. She creates a private and public key, and shares the public key with Bob. Bob sends Alice t_i, i, t_j, j. Alice then queries the nodes to confirm that Bob owns 2 digital coins by querying the nodes. Denote Alice's public key as *Alice_Public* and Bob's public key as *Bob_Public*. Bob then creates the following string:

```
Time_and_Date
In: 2
1: t_i,i
2: t_j,j
```

```
Bob_Public
Out:
Alice_Public: 1
Bob_Public: 1
```

and signs it to create

```
Time_and_Date
In: 2
1: t_i,i
2: t_j,j
Bob_Public
Out:
Alice_Public: 1
Bob_Public: 1
Bob_sig
```

In: 2 indicates that there are 2 previous transactions from which Bob obtained his wealth. Alice_Public: 1 means that Bob sent Alice 1 coin, and Bob_Public: 1 means that Bob sent himself back 1 coin. Bob then broadcasts this string to all available nodes. This string constitutes a transaction from Bob to Alice.

Now when a node receives this transaction, they verify that it is valid by:

1. Checking that the input transactions have not been spent already. This means that the string $t_i + ", " + i$ is not mentioned in any block after block i. The same is done for t_j and j. Strictly speaking, Bob does not have to include all the previous unspent coins, but it would be good practice to do so because it would be easier to keep track of his total unspent coins.

2. Checking that for each input transaction, the recipient is Bob.

3. Checking that the total that Bob recieved in each transaction totals the amount Bob sends to Alice and to himself. In general, Bob can send to as many people as he likes, he simply needs to include their public keys and amounts in the output.

4. Checking, using Bob's public key, that he indeed signed the message.

When all of these are satisfied, the nodes include it in their block to solve the hash puzzle. Nodes shall agree on the following rule:

- If block is verified to be valid, then the nodes will no longer work on mining their current block (unless they contain no overlapping valid transactions, which is highly unlikely), and immediately start gathering the next set of valid transactions. When they decide that enough transactions are collected, they start to solve its hash puzzle insead of the previous block. Further, when two verified blocks from different nodes are broadcasted at the same time, the nodes agree that they will use the block hash of the node that they received first to mine the next block. In this case we

say that there are two possible branches, or chains of the blocks that we will accept as the correct history of transactions. The tie is broken when one of the chains becomes longer than the other, and nodes continually update their version of the blockchain as the longest chain in the network of nodes.

In general, it is usually not necessary for the blocks to even wait after a block has been mined to collect transactions. When starts to solve the hash puzzle for a block, it can stop collecting including new transactions for the block each time it tries a nonce. The transactions that are broadcasted by market participants can then be stored by the side until a new block is mined. When the new block is mined the node looks at the transactions in it and removes the transactions that overlap with the already mined block to avoid double spending (This is because the other node could have received the transaction first and included it in the block it was mining). It then works on mining the rest of the transactions which it received while mining.

It is noted here that the chances of two chains branching indefinitely diminishes drastically over time. Furthermore, even if a certain proportion of nodes have a branched version of the blockchain, the further back in time we go, there will be less nodes disagreeing on the nth transaction block. So the more blocks are mined from a previous transaction, the more certain it becomes in accepted history.

8 Mining Digital Coins

Obviously without some incentive there would be no nodes willing to expend computation time and electricity for nothing in return. Therefore all network participants and network nodes agree to the following rule:

- When the network node who is also a market participant, with public key $Node\_Public\_key$ gathers transactions into a block, he adds the following string to the start of the block:

  ```
  Time_and_Date
  Node: Node_Public_key
  Amount: A
  ```

  Where A is a number that is given by

  $$\frac{\Psi}{2^{\lfloor t/T \rfloor}}$$

  where Ψ, T are natural numbers, and t is the current block number in the chain whose hash puzzle is to be solved. Then when the node solves this puzzle, all nodes and market participants agree that the node that solved it is currently in possession of the amount A of digital coin. Hence the process of obtaining a hash of a block is called "Mining". The above string

counts as a transaction that is inputted to obtain the root hash for the block. Our rule also stipulates that if the amount A does not follow the rules, and a greedy miner tries to make A larger than what was agreed upon, all nodes shall reject this block and continue mining their own block until they obtain the desired nonce.

In consequence, each miner needs to find their own nonce for their own unique block. The nonce that each miner seeks to find is therefore in general different. Further, we note that the sum of all mined coins cannot exceed a certain amount. This is what makes the digital currency "inflation free".

In the beginning, it is necessary to allow coins to be minted or mined in this way, but as time goes on, the incentive for nodes to solve the hash problem is gradually replaced by transaction fees. It will be described how this is implemented and why this is possible by purely market incentives.

9 Transaction Fees Bid and Ask and How They Determine Block Size

Aside from mining, each node can also ask for a transaction fee from each participant. Suppose Bob buys an apple from Alice for 1 coin. He then bids for his transaction to be included in the next block by offering 0.5 coin to the miner. The string that Bob signs is this:

```
Time_and_Date
In: 2
1: t_i,i
2: t_j,j
Bob_Public
Out:
Alice_Public: 1
Bob_Public: 0.5
```

He then broadcasts:

```
Time_and_Date
In: 2
1: t_i,i
2: t_j,j
Bob_Public
Out:
Alice_Public: 1
Bob_Public: 0.5
Bob_sig
```

to all the nodes.

Then each node can decide whether or not they include this in the current block that they are mining. If they do, then the difference between the output

and input is implied to be sent to the node which mined the block. We have the following rule:

- When a participant happens to be a node, they can point to the block that they mined and all participants and nodes shall agree that the participant can spend the coins which are the amount of the sum of the mined coins and the sum of all differences between inputs and outputs of each transaction in the block.

The ask price of a block can be estimated by looking at the average transaction fee in the previous block, and how fast they were accepted into a block. If a transaction is taking too long to be processed, it can be made again with a different transaction fee until it is included in the blockchain, it is only a matter of re-signing the desired string and broadcasting it to all available nodes.

Now if a transaction references many previous transactions, it will not only take longer to verify but also use up more disk space. Define transaction worth as the adjusted fee of a certain transaction. The function that determines transaction worth is up to each miner to decide, but it would list how worthy a transaction is to be included in their block. Transaction fees are determined by the current market conditions: how much a digital coin is worth, how many nodes are currently available, how many transactions there currently are, and so on.

Now given average transaction fee ϕ, the nodes earn the an amount of coins proportional to ϕ and the amount that is specified by mining protocol. Therefore there is incentive to collect a large number of transactions before starting to mine. Right after the previous block is mined and broadcasted to available nodes, the nodes check and (if everything is in order) accept the block. They then immediately start to gather unconfirmed transactions into a new block. When they decide that it is enough, they will put other transactions on hold for the next block and start mining the current block. If they wait too long, then other nodes will have already had a significant head start on mining, so it will be disadvantageous, as the likelihood of solving the hash puzzle first is diminished. If they start too fast however, the processing power and electricity expenditure would not justify the small number of transaction fees collected. Note that since we only input the transaction root hash and not all the transactions into the hash function, the amount of transactions does not change the processing power and electricity expenditure.

Here, it is also possible to introduce the following rule (although it is not strictly necessary), as Satoshi Nakamoto did:

- The block size cannot exceed 1 megabyte.

The reason for introducing this rule is because the broadcasting of the block cannot be too slow, otherwise other blocks might be mined in that time, and it could increase the probability that the single history of valid transactions could fork into multiple versions if this happens.

Now let us consider possible cases that might make blocks very long or very short. Suppose that the current market conditions cause transaction fees to be

very small compared to the amount minted by each block. For example, the rule that we agreed to might stipulate us to be able to obtain 300 coins by solving the hash puzzle, but each transaction is only worth 0.0001 coin. Then if we collected 10 transactions or 10000 transactions, we won't get any substantive difference in the amount of coins that we obtain by mining. So we might collect only 1 transaction and start mining the block. This poses a problem as for the system to work we need a substantive number of transactions to be included so that transactions can be completed in a relatively fast pace.

But actually, in fact if each transaction is worth a very small amount compared to the hash puzzle prize, it means that there are a very small number of transactions; that is, the demand is very low. If there is low demand, then that means only a small number of transactions need to be made at a given time, so including a small number of transactions will not reduce the speed of the transactions. Then if someone gets impatient and bids higher prices for their transactions, this would increase incentive for each miner to include more transactions in the block.

Conversely, if the transaction costs are currently very high compared to the hash puzzle prize, then miners would be willing to wait longer to collect transactions before starting to mine a block. In this case, as noted, they cannot wait too long otherwise the probability of that a competitor node mines it increases over time, and by then the demand for transactions to be included in the block chain may decrease, and transactions will be worth less.

So we can see that the market of transactions self adjusts. There is no need to dictate how many transactions need to be included in a certain block, or how long miners need to wait to collect transactions before starting to mine the block.

10 Self Perpetuating System

In a sense, the transaction fee one pays in the initial stages of the start of the blockchain is in the form of the amount that the coin decreases in value by the inflation of the value of the coins minted by the mining of new blocks. However, as the demand to use the digital currency increases, the market price of coins can also increase causing increased price of the coin (currency deflation) which offsets the initial inflation. When inflation is stronger, one wants to spend them. When deflation is strong, one wants to keep them. So initially, miners will ought to want to spend their newly minted coins sooner rather than later, lest the coins lose value over time. However, if a large number of market participants join as the currency gets more mainstream, more miners will join in mining the digital coin as the rewards for it are increased.

The incentive for each node to work starts with the ability to mine coins (their creation) and then as the mining reward diminishes, it can be replaced totally by transaction fees. This is only possible if the digital currency becomes widely used. Once certain merchants start to accept the digital currency, it is no longer necessary to mint new coins; it is enough to keep the currency going

with only transaction fees.

When the currency becomes widespread and accepted by various merchants, and the desirability to pay in the digital currency is high, then market participants would be willing to pay a transaction fee. However, when desirability to pay in the digital currency decreases, there will be fewer transactions per minute. When this happens, the market price of transactions decreases. When the market price of transactions decreases below the transaction fees of banks and other currencies, the demand rises again. All of this discussion of course needs to discount the price of volatility.

So the market is balanced: if the coin becomes too popular, the transaction fees increase and it becomes less popular. If it becomes less popular, transaction fees decrease, and it becomes more popular. Therefore there is no need for any external incentive or central authority to enforce transactions in a certain coin.

11 Possible Market Participant Attacks and Why They Don't Work

A market participant may try to double spend a coin by signing a transaction saying he transferred coins to Alice and then signing a transaction that he transferred it to Bob, and then sending some nodes the first transaction and other nodes the second transaction.

In this case, nodes which recieved the first transaction will say that Alice owns the coins and nodes that received the second transaction will say that Bob owns the coins. Say the block that a node mined contained the first transaction. Then the nodes that contained the second transaction will gather it to the next block, and see that it is no longer a valid transaction. It will therefore reject it and not include it in the block that they are currently mining.

Now suppose that by chance, two nodes mined the blocks at the same time, one containing the first transaction, and the other containing the second transaction. This is still not a problem, as once one chain becomes longer than the other, the nodes will all agree which one is valid. Market particiants need to simply wait until this happens to confirm the payment. To do this, they can query enough nodes to convince themselves that the majority of them accepted the payment as valid.

12 Possible Node Attacks and Why They Don't Work

We now consider cases where some network nodes are not honest.

12.1 Uncooperative/Stubborn Node

Suppose that all nodes currently are mining block n, and one node solves the hash puzzle and broadcasts the block. The honest nodes then stop mining and

start collecting a set of transactions for the next node to mine.

12.2 51% Attack

Suppose that a dishonest node has acqured the computation power that eclipses that of all the other nodes. If a node has such computation power it would mean that even with the rest of the computation power of the honest nodes combined, the longest chain will be that of the dishonest node.

The following enumerates what the 51% attack allows the dishonest node to do:

1. It allows the node to mine all future blocks on the blockchain from the point of time they gained the majority of computation power. This done as follows. When the node mines a block,

13 Other Remarks

13.1 Anonymity

13.2 Losing Coins

13.3 Maintenance