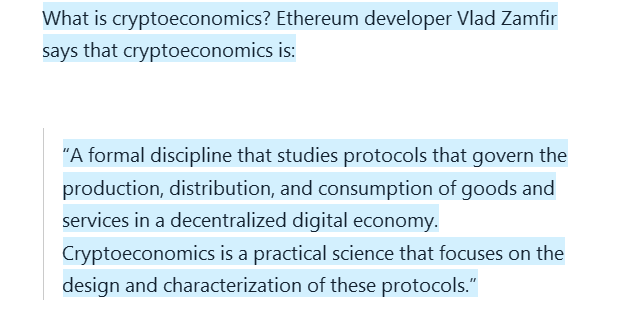
# Cryptoeconomics（加密经济）

## 是什么：

以太坊创始人的定义：

A formal 学科研究在去中心化数字经济中的商品的生产，分发，消耗的协议。

加密经济就是就是关于这些协议的设计与描述的实践的科学。



加密经济= 加密 + 经济。

## 基本特性（properties）

### Cryptography

动机（目的）

为什么需要应用加密技术？

### Economics

## 工作模型之区块链技术

区块链技术就是运行在加密经济的properties之上。

正是加密经济其中的“经济”使得区块链具有了独特的能力。

### 传统P2P网络：torrent技术的问题：

区块链并不是第一个去中心化的P2P系统。用于文件共享的Torrent技术就已经是并且应用了很多年多了。

Torrent技术的缺点：

Torrent系统内，任何人都可以通过去中心化的P2P网络共享文件，该技术的基本实现思路是，任何人通过从网络内下载文件时，同时也为网络内的其他人做种子，以便其他人也可以从本机上下载文件。

缺点：

Torront系统只有在具有分享精神有荣誉感的人之间才能更好的工作，因为实际情况是，很多人只是自私地从网络中下载文件，而拒绝为其他人做种子。

The problem was that this worked on an honor system. If you were downloading a file, then you were expected to seed as well. The problem is that humans are not really the most honorable of creatures and without any economic incentives it made no sense for people to keep seeding a file which took up unnecessary space in their computers.

# 区块链技术的创新：

1、激励机制的引入：在传统的去中心化网络中，加入了economic incentive，以激励人们“follow the rules”

2、共识机制的引入：共识机制是为了解决在去中心化的,相互独立的节点如何就某些事项达成一致的过程。简而言之，共识机制是在解决分布式系统中如何保持一致性的问题。

# Cryptoeconomic properties of BitCoin

* It is based on the blockchain technology where each block contains the hash of the previous block and forms a continuous chain.
* Each block will include transactions.
* The blocks will have a particular state which is subject to change according to transactions. Eg. if A has 50 bitcoins and wants to send 20 bitcoins to B. Then The new state should show that A has 30 bitcoins left and B has 20 new bitcoins.
* The blockchain must be immutable. It should be possible to add new blocks but the old blocks can’t be tampered with.
* Only valid transactions should be allowed.
* The blockchain should be downloadable and anyone anywhere can easily access and check a particular transaction.
* Transactions could be added quickly to the blockchain if a sufficiently high transaction fee is paid.

Cryptography：

# 概念术语

## 金融/会计基本概念

### financial statements:

（reference from wikipedia）

Financial statements (or financial report) is a formal record of the financial activities and position of a business, person, or other entity.

Relevant financial information is presented in a structured manner and in a form easy to understand. They typically include basic financial statements, accompanied by a management discussion and analysis:[2]

**A balance sheet or statement of financial position,** reports on a company's assets, liabilities, and owners equity at a given point in time.

**An income statement or statement of comprehensive income,** statement of revenue & expense, P&L or profit and loss report, reports on a company's income, expenses, and profits over a period of time. A profit and loss statement provides information on the operation of the enterprise. These include sales and the various expenses incurred during the stated period.

**A Statement of changes in equity or equity statement or statement of retained earnings**, reports on the changes in equity of the company during the stated period.

**A cash flow statement** reports on a company's cash flow activities, particularly its operating, investing and financing activities.

For large corporations, these statements may be complex and may include an extensive set of footnotes to the financial statements and management discussion and analysis. The notes typically describe each item on the balance sheet, income statement and cash flow statement in further detail. Notes to financial statements are considered an integral part of the financial statements.

### liability（负债）

wikipedia：

In financial accounting, a liability is defined as the future sacrifices of economic benefits that the entity is obliged to make to other entities as a result of past transactions or other past events,[1] the settlement of which may result in the transfer or use of assets, provision of services or other yielding of economic benefits in the future.

### assets（资产）

wikipedia：

In financial accounting, an asset is an economic resource. Anything tangible or intangible that can be owned or controlled to produce value and that is held by a company to produce positive economic value is an asset. Simply stated, assets represent value of ownership（所有权的价值体现） that can be converted into cash (although cash itself is also considered an asset).[1]

## 记账相关

### ledger（账本）

账本的作用：

传统账本技术：

每个市场参与者都维护（逻辑上的）一个账本，用来追踪自身的（某项）资产的增减变动（收入、支出）；用于展示核查资产状况；

wikipedia：

账本分三种类型：

debtors ledger：债务人账本（销售账本）记录了产生收益的交易，提供收入信息；

creitors ledger：债权人账本（支出账本），记录导致利益减少的交易，提供支出信息；

general leger：汇总账本，representing the five main[3][citation needed] account types: assets, liabilities, income, expenses, andCapital.

The ledger is a permanent summary of all amounts entered in supporting [journals](https://en.wikipedia.org/wiki/General_journal) which list individual transactions by date. Every transaction flows from a journal to one or more ledgers. A company's [financial statements](https://en.wikipedia.org/wiki/Financial_statement) are generated from summary totals in the ledgers.[[2]](https://en.wikipedia.org/wiki/Ledger#cite_note-2)

Ledgers include:

* [**Sales ledger**](https://en.wikipedia.org/wiki/Sales_ledger)**,** records [accounts receivable](https://en.wikipedia.org/wiki/Accounts_receivable). This ledger consists of the financial transactions made by customers to the company.
* [**Purchase ledger**](https://en.wikipedia.org/wiki/Purchase_ledger) records money spent for purchasing by the company.
* [**General ledger**](https://en.wikipedia.org/wiki/General_ledger) representing the five main[[3]](https://en.wikipedia.org/wiki/Ledger#cite_note-3)[[*citation needed*](https://en.wikipedia.org/wiki/Wikipedia:Citation_needed)] account types: [assets](https://en.wikipedia.org/wiki/Assets), [liabilities](https://en.wikipedia.org/wiki/Liability_(financial_accounting)), [income](https://en.wikipedia.org/wiki/Income), [expenses](https://en.wikipedia.org/wiki/Expenses), and[Capital](https://en.wikipedia.org/wiki/Financial_capital).

For every [debit](https://en.wikipedia.org/wiki/Debits_and_credits) recorded in a ledger, there must be a corresponding [credit](https://en.wikipedia.org/wiki/Debits_and_credits) so that the debits equal the credits in the grand totals.

The three types of ledgers are the general, debtors, and creditors.[[4]](https://en.wikipedia.org/wiki/Ledger#cite_note-4) The [**general ledger**](https://en.wikipedia.org/wiki/General_ledger)accumulates information from journals. Each month all journals are totaled and posted to the General Ledger. The purpose of the General Ledger is therefore to organize and summarize the individual transactions listed in all the journals. **The Debtor Ledger** accumulates information from the sales journal. The purpose of the Debtors Ledger is to provide knowledge about which customers owe money to the business, and how much. **The Creditors Ledger** accumulates information from the purchases journal. The purpose of the Creditors Ledger is to provide knowledge about which suppliers the business owes money to, and how much.

## 安全相关：

### Cryptographic hash function

参考：

<https://en.wikipedia.org/wiki/Cryptographic_hash_function>

A **cryptographic hash function** is a special class of [hash function](https://en.wikipedia.org/wiki/Hash_function) that has certain properties which make it suitable for use in [cryptography](https://en.wikipedia.org/wiki/Cryptography). It is a mathematical [algorithm](https://en.wikipedia.org/wiki/Algorithm) that [maps](https://en.wikipedia.org/wiki/Map_%28mathematics%29) data of arbitrary size to a [bit string](https://en.wikipedia.org/wiki/Bit_string) of a fixed size (a hash) and is designed to be a [one-way function](https://en.wikipedia.org/wiki/One-way_function), that is, a function which is [infeasible](https://en.wikipedia.org/wiki/Computational_complexity_theory#Intractability) to invert. The only way to recreate the input data from an ideal cryptographic hash function's output is to attempt a [brute-force search](https://en.wikipedia.org/wiki/Brute-force_search) of possible inputs to see if they produce a match, or use a [rainbow table](https://en.wikipedia.org/wiki/Rainbow_table) of matched hashes. [Bruce Schneier](https://en.wikipedia.org/wiki/Bruce_Schneier) has called one-way hash functions "the workhorses of modern cryptography".[[1]](https://en.wikipedia.org/wiki/Cryptographic_hash_function#cite_note-1) The input data is often called the *message*, and the output (the *hash value* or *hash*) is often called the *message digest* or simply the *digest*.

The ideal cryptographic hash function has five main properties:

* it is [deterministic](https://en.wikipedia.org/wiki/Deterministic_algorithm) so the same message always results in the same hash
* it is quick to compute the hash value for any given message
* it is [infeasible](https://en.wikipedia.org/wiki/Computational_complexity_theory#Intractability) to generate a message from its hash value except by trying all possible messages
* a small change to a message should change the hash value so extensively that the new hash value appears uncorrelated with the old hash value
* it is [infeasible](https://en.wikipedia.org/wiki/Computational_complexity_theory#Intractability) to find two different messages with the same hash value

#### 破解办法：

##### brute-force attack

In [computer science](https://en.wikipedia.org/wiki/Computer_science), **brute-force search** or **exhaustive search**, also known as **generate and test**, is a very general [problem-solving](https://en.wikipedia.org/wiki/Problem-solving) technique and [algorithmic paradigm](https://en.wikipedia.org/wiki/Algorithmic_paradigm) that consists of systematically enumerating all possible candidates for the solution and checking whether each candidate satisfies the problem's statement.

While a brute-force search is simple to [implement](https://en.wikipedia.org/wiki/Software_implementation), and will always find a solution if it exists, its cost is proportional to the number of candidate solutions – which in many practical problems tends to grow very quickly as the size of the problem increases ([combinatorial explosion](https://en.wikipedia.org/wiki/Brute-force_search#Combinatorial_explosion))[[1]](https://en.wikipedia.org/wiki/Brute-force_search#cite_note-1). Therefore, brute-force search is typically used when the problem size is limited, or when there are problem-specific [heuristics](https://en.wikipedia.org/wiki/Heuristic_%28computer_science%29) that can be used to reduce the set of candidate solutions to a manageable size. The method is also used when the simplicity of implementation is more important than speed。

##### Rainbow table

参考：<https://en.wikipedia.org/wiki/Rainbow_table>

A **rainbow table** is a [precomputed](https://en.wikipedia.org/wiki/Precomputed) [table](https://en.wikipedia.org/wiki/Lookup_table) for reversing [cryptographic hash functions](https://en.wikipedia.org/wiki/Cryptographic_hash_function), usually for cracking password hashes. Tables are usually used in recovering a [password](https://en.wikipedia.org/wiki/Password) (or credit card numbers, etc.) up to a certain length consisting of a limited set of characters. It is a practical example of a [space–time tradeoff](https://en.wikipedia.org/wiki/Space%E2%80%93time_tradeoff), using less computer processing time and more storage than a [brute-force attack](https://en.wikipedia.org/wiki/Brute-force_attack) which calculates a hash on every attempt, but more processing time and less storage than a simple [lookup table](https://en.wikipedia.org/wiki/Lookup_table) with one entry per hash. Use of a [key derivation function](https://en.wikipedia.org/wiki/Key_derivation_function) that employs a [salt](https://en.wikipedia.org/wiki/Salt_%28cryptography%29) makes this attack infeasible.

Rainbow tables were invented by Philippe Oechslin[[1]](https://en.wikipedia.org/wiki/Rainbow_table#cite_note-ophpaper-1) as an application of an earlier, simpler algorithm by [Martin Hellman](https://en.wikipedia.org/wiki/Martin_Hellman).[[2]](https://en.wikipedia.org/wiki/Rainbow_table#cite_note-:0-2)

##### a simple lookup table

### commitment scheme:

参考：

<https://en.wikipedia.org/wiki/Commitment_scheme>

A **commitment scheme** is a [cryptographic primitive](https://en.wikipedia.org/wiki/Cryptographic_primitive) that allows one to commit to a chosen value (or chosen statement) while keeping it hidden to others, with the ability to reveal the committed value later.[[1]](https://en.wikipedia.org/wiki/Commitment_scheme#cite_note-Goldreich-1) Commitment schemes are designed so that a party cannot change the value or statement after they have committed to it: that is, commitment schemes are *binding*. Commitment schemes have important applications in a number of [cryptographic protocols](https://en.wikipedia.org/wiki/Cryptographic_protocol) including secure [coin flipping](https://en.wikipedia.org/wiki/Coin_flipping#Telecommunications), [zero-knowledge proofs](https://en.wikipedia.org/wiki/Zero-knowledge_proof), and [secure computation](https://en.wikipedia.org/wiki/Secure_computation).

A way to visualize a commitment scheme is to think of a sender as putting a message in a locked box, and giving the box to a receiver. The message in the box is hidden from the receiver, who cannot open the lock themselves. Since the receiver has the box, the message inside cannot be changed—merely revealed if the sender chooses to give them the key at some later time.

Interactions in a commitment scheme take place in two phases:

1. the *commit phase* during which a value is chosen and specified
2. the *reveal phase* during which the value is revealed and checked

In simple protocols, the commit phase consists of a single message from the sender to the receiver. This message is called *the commitment*. It is essential that the specific value chosen cannot be known by the receiver at that time (this is called the *hiding* property). A simple reveal phase would consist of a single message, *the opening*, from the sender to the receiver, followed by a check performed by the receiver. The value chosen during the commit phase must be the only one that the sender can compute and that validates during the reveal phase (this is called the *binding* property).

## 区块链：

### Blockchain:

The blockchain is a chain of blocks where each block contains data of value **without any central supervision**. It is cryptographically secure and immutable.

区块链，每个区块中包含一些数据 of vlaue（具体应用场景不同，数据不一样）。整个区块链内的数据是密码学安全（只是密码学安全，不代表物理或其他安全）和不可修改的，另外整个区块链的运行维护的不依赖于一个中央控制节点。

### Decentralized

去中心化的。系统的运行维护不依赖于中央控制节点。

### Concensus Mechanism

共识机制。在去中心化的系统中，各个节点之间就某些事项达成一致的机制。

在中心化系统中，各个节点之间的交互往往依赖中心节点提供的某些服务基础上的。

### Miners

在区块链技术中，所有参与新的合法区块产生的节点（这些节点一般需要消耗某些资源才能产生新的区块）

### 创世区块（genesis block）

The genesis block is the first block of the blockchain, and the reason why it is special is that while every bock points to the block previous to it, the genesis block doesn’t point at anything.  So, the moment a new chain is created, the genesis block is invoked immediately.

# 背景动机目的（需求）

区块链技术

# 区块链技术的难点：（需求：系统需满足的要求）

## 1、保证安全性（Security）

区块链代码首先是开源的，所有人都可以看到，与一般开源项目不同，区块链项目代码中发现的漏桶很有可能会被黑客利用，而导致巨大的经济损失。正因为如此，所以区块链项目的开发一般进展比较慢。

## 2、资源管理（resource management）

it is important to keep pace with the network. You cannot fall too far behind and not keep up with all the network demands. You should be well equipped to handle remote and local queries

## 3、性能（performance）

区块链需要总是工作在最高的性能条件下，为此所选择的开发语言must be extremely versatile（万能的）。在区块链中，有些任务可以并行运行，而有些任务则必须顺序运行。

例如，其中一种可以并行运行的任务就是，数字签名的校验（校验的过程中所需要的输入只有三种：KEY,transaction,and signature）

而必须顺序执行的任务，则例如transaction execution （交易不能同时并行运行，一个时刻只能有一个交易运行，以避免类似double spending这种问题）。

有些语言擅长于并行运算，而有些语言擅长于非并行运算。

## 4、隔离（isolation）

问题需求：

在区块链开发中，所有的交易（transaction）都必须是确定的（deterministic）。也就是说，一个交易（transaction）不能behaves one way and then behaves another way the next day. 类似的在智能合约中，you cannot have smart contracts that work in two different ways on two different machines.

deterministic behavior的定义:

If A + B = C, then no matter what the circumstances, A+B will always be equal to C. That is called deterministic behavior.区块链技术中所使用的hash运算是确定性的功能（Hash functions are deterministic, meaning A’s hash will always be H(A).）。

解决方法：isolation

The only solution to this is isolation. Basically, you isolate your smart contracts and transactions from non-deterministic elements.（将区块链内要求严格确定性的功能与非确定性的功能分隔开）

开发语言选择：

There are some languages which fulfill most of these needs. If you are a blockchain developer, then you definitely need to have some basic knowledge of C++ and JavaScript.

While C++ may seem a little outdated, the truth is that it wonderfully satisfies all the functionalities that we have described above. In fact, Satoshi Nakamoto wrote the Bitcoin source code in C++.

Along with HTML and CSS it is one of the three core technologies in World Wide Web Content Production. Javascript is usually used to create highly interactive web pages.

# 区块链技术的组成：（满足了什么需求）

## 密码学技术：Cryptography

### Hashing：

hashing means taking an input string of any length and giving out an output of a fixed length. Bitcoin uses SHA-256 to take in an input string of any length and giving an out hash of 256 bits。

给定输入，总是产生特定的输出，而且从输出无法反推出输入；

问题：有完美哈希吗？

#### 在加密币中的应用：

1 Cryptographic hash functions.

2 Data structures.

3 Mining

##### Cryptographic hash functions

在区块链中应用的是一类称为Cryptographic hash的特殊hash函数。（详情参考上文概念术语部分）。

基本特性：

* **Deterministic（输出确定）:** An input A will always have the same output h(A) no matter how many times you parse it through the same hash function.
* **Quick Computation**: A function should return a hash of an input as quickly as possible.
* **Pre-Image resistance（单向）:** Given h(A) which is an output of a hash function, it should be infeasible to determine input A.
* **Collision resistance（无冲突）**: Given two inputs A and B and their hash outputs h(A) and h(B) it should be infeasible for h(A) = h(B).
* **Small changes（蝴蝶效应）**: in the input should drastically affect the output of the hash function.
* **Puzzle Friendly:** For every hash output Y and an input x. It is infeasible to find a value k, which will result in h(k|x) = Y

##### Data structures’不可篡改性，immutability

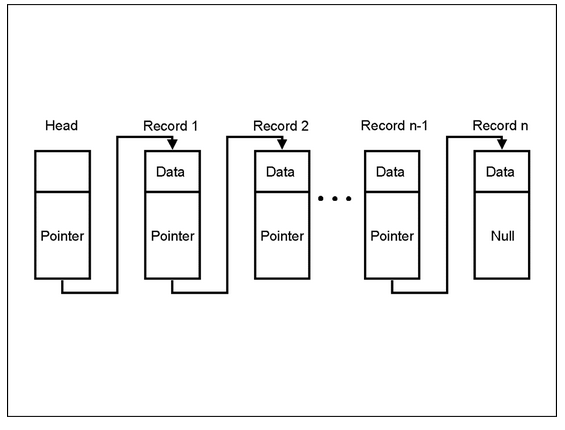
Hash的“Small changes”特性的为区块链的“immutability”特性提供了重要保障。

在区块链中，这一特性的应用就是：在区块链结构中的hash pointer的设计。

在区块链中有两个结构对于理解区块链比较重要：

Linked Lists and Hash Pointers。

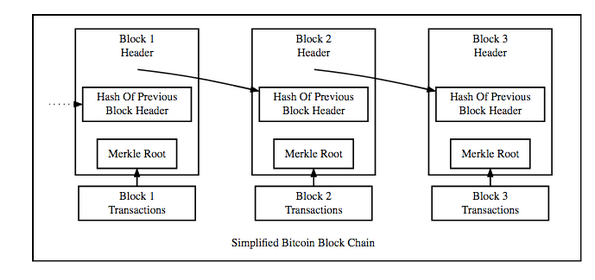
* **Linked Lists:** Linked lists are blocks of data which are connected to one after another. This is an example of a linked list:



Each block in the list is pointing to the other via a pointer.

* **Pointer:** Pointers are variables which include the addresses of the other variables. So they are variables which are literally pointing towards the other variables.
* **Hash Pointers:** Hash pointers are basically pointers which not only has the address of other variables but also the hash of the data in that variable. So how does that help in the context of a blockchain? （这是hash指针提供了区块链的不可篡改的特性，因为一旦某个区块被串改，会导致后一个区块保存的该区块的hash值与串改后的区块的hash值不一致，从而发现该区块被窜改过，被认定为不合法区块，所在的链也不合法。所以对于对于攻击者而言如果意图修改某一个区块链的内容，就需要保证将该区块之后的所有区块都修改（主要指每个区块中的hash值），而区块链的共识机制，hash等的设计导致这需要很大的资源消耗，所以攻击几乎不可能。

**This is what a blockchain looks like:**



The blockchain is basically a linked list where each new block contains a hash pointer which points to the previous block and the hash of all the data in it. Just this one property leads into one of Blockchain’s greatest qualities….its immutability

##### Mining

挖矿的过程就是各节点（或矿工节点）通过一种消耗资源（如时间，计算资源，电力等）方式计算出新的合法区块的过程。（至少在比特币中）这个过程是各个节点（或矿工节点）通过计算一个密码谜题来实现的，只有产生算法认可的计算结果时，该节点产生新的区块才被认为是合法的。**Hash在这个过程中的作用是**：挖矿过程中先设置一个难度系数（difficulty level），然后随机产生一个字符串称为“噪声”，“噪声”is appended to the hash of the new block and hashed again，然后检查最新计算出的hash值的特点(例如比特币中要求hash值的前n位必须是0)是否满足小于设置的难度系数，如果小于，则新区块被认为是合法的，可以加入到区块链中，然后系统会给矿工一定的奖励；如果不小于，则矿工继续产生新的“噪声”继续上面的计算过程。

### 数字签名

数字签名是密码学技术的一种常见的应用，在现有的互联网、金融等已经广泛引用。其实现是基于密码学中的非对称加密技术，

数字签名三种基本功能：

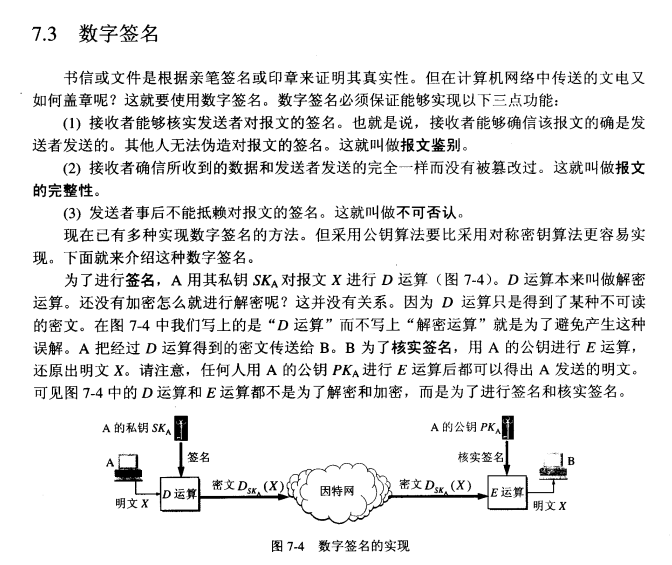
1、身份证明；

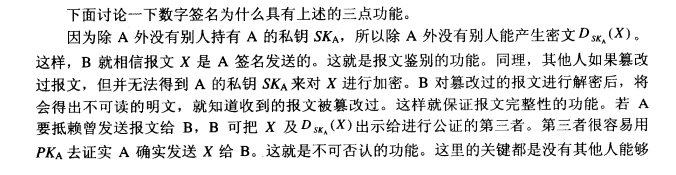
2、不可抵赖；

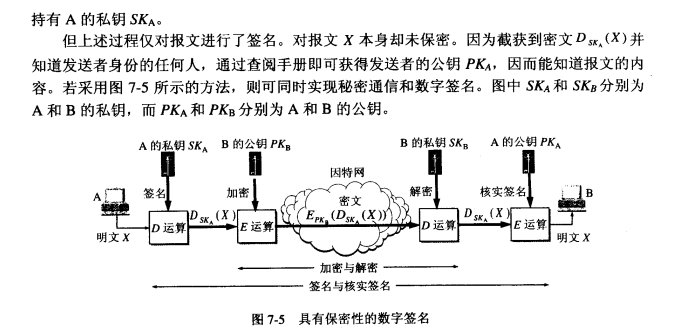
3、无法伪造；

在区块块中也是利用这三种功能，来实现区块链交易过程中节点的身份证明，交易的不可抵赖，和防止伪造的功能。

以下来自《计算机网络基础》对数字签名的原理介绍。







## 共识机制：

区块链技术中所采用的共识机制，主要是为了解决如何在各个诚实节点之间就新的区块的产生达成一致（对什么是合法的区块达成共识，对该区块是否可以入链达成共识。）。

对共识算法的要求是能不受拜占庭问题的困扰，依然正常运行（即就是在系统中可能存在叛徒节点的情况下，所有（或有权限的）诚实节点依然可以正常就一个区块是否合法是否可以入链达成共识）

### POW(proof of work)

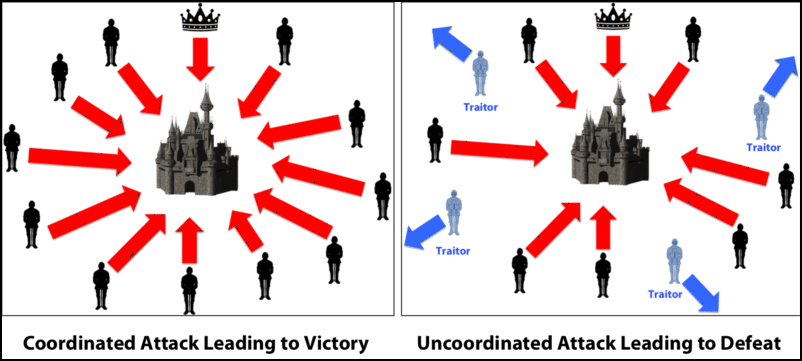
#### 原理介绍：

基于POW算法的关于什么是合法区块以及是否可以入链的共识是：只有满足一定哈希要求的区块才可以被认为是合法的可以入链的区块。

而算法对拜占庭问题的解决是： 只要诚实节点依据规定检测收到的区块是否合法即可，而对不诚实节点通过算法的设计，因为产生符合要求的哈希需要大量的计算（这个过程中耗费了电力等各种资源），而正是这样的设计增加了非法节点伪造篡改区块内容的代价（*注：并不能完全避免非法节点的篡改伪造，只是大幅提高到了代价*）。

When miners “mine” to form new blocks to add to the blockchain, the consensus system by which the blocks get approved and added is called “proof-of-work”. Miners use heavy duty computational power to solve cryptographical puzzles to satisfy a difficulty level. This is one of the most path-breaking mechanisms in blockchain technology. Earlier decentralized peer-to-peer digital currency systems used to fail because of something called the “Byzantine General’s Problem”. The proof-of-work consensus system finally provided a solution to this problem.

**What is the Byzantine General’s Problem?**



Ok so imagine that there is a group of Byzantine generals and they want to attack a city. They are facing two very distinct problems:

* The generals and their armies are very far apart so centralized authority is impossible, which makes coordinated attack very tough.
* The city has a huge army and the only way that they can win is if they all attack at once.

In order to make successful coordination, the armies on the left of the castle send a messenger to the armies on the right of the castle with a message that says “ATTACK WEDNESDAY.” However, suppose the armies on the right are not prepared for the attack and say, “NO. ATTACK FRIDAY” and send back the messenger through the city back to the armies on the left. This is where we face a problem. A number of things can happen to the poor messenger. He could get captured, compromised, killed and replace with another messenger by the city. This would lead to the armies getting tampered information which may result in an uncoordinated attack and defeat.

This has clear references to blockchain as well. The chain is a huge network; how can you possibly trust them? If you were sending someone 4 Ether from your wallet, how would you know for sure that someone in the network isn’t going to tamper with it and change 4 to 40 Ether?

Satoshi Nakamoto was able to bypass the Byzantine General’s problem by inventing the proof of work protocol. This is how it works. Suppose the army on the left want to send a message called “ATTACK MONDAY” to the army on the right, they are going to follow certain steps.

* Firstly, they will append a “nonce” to the original text. The nonce can be any random hexadecimal value.

* After that, they hash the text appended with a nonce and see the result. Suppose, hypothetically speaking, the armies have decided to only share messages which, on hashing, gives a result which starts with 5 zeroes.

* If the hash conditions are satisfied, they will send the messenger with the hash of the message. If not, then they will keep on changing the value of the nonce randomly until they get the desired result. This action is extremely tedious and time-consuming and takes a lot of computation power.

* If the messenger does get caught by the city and the message is tampered with, according to hash function properties, the hash itself will get drastically changed. If the generals on the right side, see that the hashed message is not starting with the required amount of 0s then they can simply call off the attack.

However, there is a possible loophole.

No hash function is 100% collision-free. So what if the city gets the message, tampers with it and then accordingly change the nonce until they get the desired result which has the required number of 0s? This will be extremely time-consuming but it is still possible. To counter this, the generals are going to use strength in numbers.

Suppose, instead of just one general on the left sending messages to one general on the right, there are 3 generals on the left who have to send a message to the ones on the right. In order to do that, they can make their own message and then hash the cumulative message and then append a nonce to the resulting hash and hash it again. This time, they want a message which starts with six 0s.

Obviously, this is going to be extremely time-consuming, but this time, if the messenger does get caught by the city, the amount of time that they will take to tamper the cumulative message and then find the corresponding nonce for the hash will be infinitely more. It may even take years. So, eg. if instead of one messenger, the generals send multiple messengers, by the time the city is even halfway through the computation process they will get attacked and destroyed.

The generals on the right have it pretty easy. All they have to do is to append the message with the correct nonce that will be given to them, hash them, and see whether the hash matches or not. Hashing a string is very easy to do. That, in essence, is the process behind proof-of-work.

* The process of finding the nonce for the appropriate hash target should be extremely difficult and time-consuming.
* However, the process of checking the result to see if no malpractice has been committed should be very simple.

### ZKP（zero knowledge proofs,零知识证明）

#### 原理：

在不向对方提供具体信息内容的情况下，向对方证明某一事项为真。后者称为证明者（prover），前者称为验证者（verifier）。在密码学中，这个技术为Prover提供了隐私保证。

有些零知识证明算法或协议的运行通过prover and verifier间的交互实现：

1、通过交互进行验证；

2、多次交互（验证），证明有效性（避免单次验证的偶然正确性，防止作弊）；

#### 举例：（不透漏额外信息而证明声明为真）

##### 1、Alibaba cave

An example of a ZKP is the Alibaba cave, let’s see how it works. In this example, **the prover (P) is saying to the verifier(V) that they know the password of the secret door at the back of the cave and they want to prove it to the verifier without actually telling them the password**. So this is what it looks like:

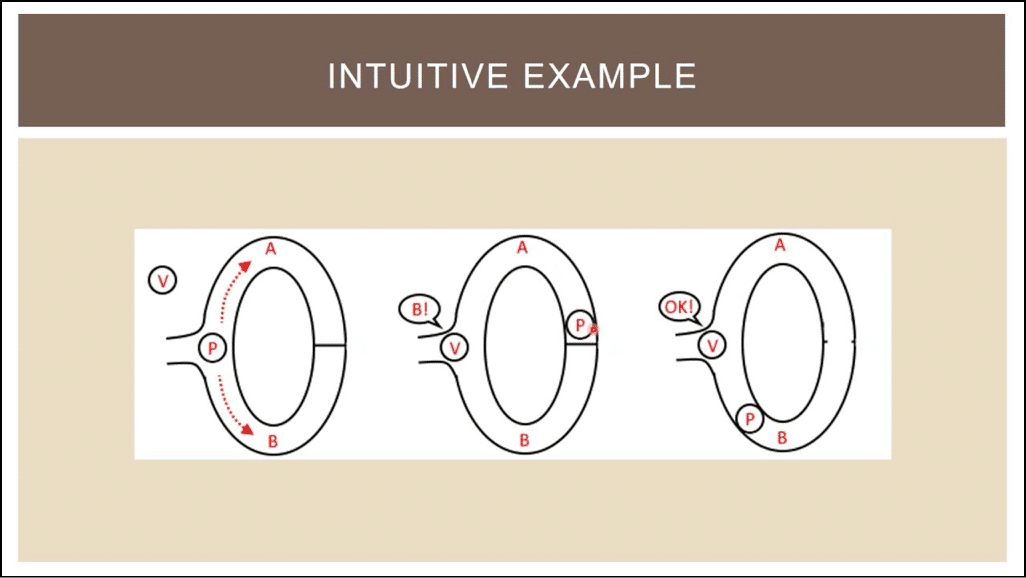


Image courtesy: Scott Twombly (YouTube channel)

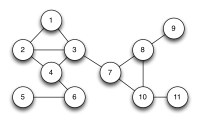
The Prover goes down any of the paths A and B, suppose they initially decide to go through path A and reach the secret door at the back. When they do so, the verifier V comes in at the entrance, with no knowledge of which path the prover actually took and declares that they want to see the prover appear from path B.

In the diagram, as you can see, the prover does indeed appear in path B. But what if this was dumb luck? What if the prover didn’t know the passcode, and took the path B, was stuck at the door and by sheer fortune, the verifier told him to appear from path B, the one they were originally on anyway?

So, **to test the validity**, **the experiment is done multiple times**. If the prover can appear at the correct path every single time, it proves to the verifier that the prover indeed knows the password even though the verifier doesn’t know what the password actually is.

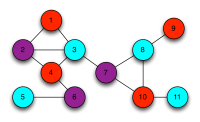
##### 2、三色问题：

For the purposes of this example, I’d like you to imagine that I’m a telecom magnate in the process of deploying a new cellular communications network. My network structure is represented by the graph below. Each vertex in this graph represents a cellular radio tower, and the connecting lines (edges) indicate locations where two cells *overlap*, meaning that their transmissions are likely to interfere with each other.

[](https://matthewdgreen.files.wordpress.com/2014/11/d5db3-uncoloredgraph.png)

This overlap is problematic, since it means that signals from adjacent towers are likely to scramble reception. Fortunately my network design allows me to configure each tower to one of three different frequency bands to avoid such interference.

Thus the challenge in deploying my network is to assign frequency bands to the towers such that no two overlapping cells share the same frequencies. If we use colors to represent the frequency bands, we can quickly work out one solution to the problem:

[](https://matthewdgreen.files.wordpress.com/2014/11/af15c-coloredgraph.png)

Of course, many of you will notice that what I’m describing here is simply an instance of the famous theory problem called the [graph three-coloring](http://en.wikipedia.org/wiki/Graph_coloring) problem. You might also know that what makes this problem interesting is that, for some graphs, it can be quite hard to find a solution, or even to determine *if* a solution exists.In fact, graph three-coloring — specifically, the decision problem of whether a given graph supports a solution with three colors — is known to be in the complexity class [NP-complete](http://en.wikipedia.org/wiki/NP-complete).

It goes without saying that the toy example above is easy to solve by hand. But what if it wasn’t? For example, imagine that my cellular network was very large and complex, so much so that the computing power at my disposal was not sufficient to find a solution. In this instance, it would be desirable to *outsource* the problem to someone else who has plenty of computing power. For example, I might hire my friends at Google to solve it for me on spec.

But this leads to a problem.

Suppose that Google devotes a large percentage of their computing infrastructure to searching for a valid coloring for my graph. I’m certainly not going to pay them until I know that they really have such a coloring. At the same time, Google isn’t going to give me a copy of their solution until I’ve paid up. We’ll wind up at an impasse.

In real life there’s probably a common-sense answer to this dilemma, one that involves lawyers and escrow accounts. But this is not a blog about real life, it’s a blog about cryptography. And if you’ve ever read a crypto paper, you’ll understand that the right way to solve this problem is *to dream up an absolutely crazy technical solution*.

###### A crazy technical solution (with hats!)

The engineers at Google consult with Silvio Micali at MIT, who in consultation with his colleagues [Oded Goldreich and Avi Wigderson](http://people.csail.mit.edu/silvio/Selected%20Scientific%20Papers/Zero%20Knowledge/Proofs_That_Yield_Nothing_But_Their_Validity_or_All_Languages_in_NP_Have_Zero-Knowledge_Proof_Systems.pdf), comes up with the following clever protocol — one so elegant that it doesn’t even require any computers. All it requires is a large warehouse, lots of crayons, and plenty of paper. Oh yes, and a whole bunch of hats.\*\*

Here’s how it works.

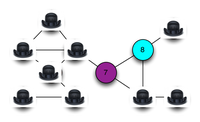
First I will enter the warehouse, cover the floor with paper, and draw a blank representation of my cell network graph. Then I’ll exit the warehouse. Google can now enter enter, shuffle a collection of three crayons to *pick a random assignment of the three agreed-upon crayon colors* (red/blue/purple, as in the example above), and color in the graph in with their solution. Note that it doesn’t matter which specific crayons they use, only that the coloring is valid.

Before leaving the warehouse, Google covers up each of the vertices with a hat. When I come back in, this is what I’ll see:

[](https://matthewdgreen.files.wordpress.com/2014/11/538c4-hats.png)

Obviously this approach protects Google’s secret coloring perfectly. But it doesn’t help me at all. For all I know, Google might have filled in the graph with a random, invalid solution. They might not even have colored the graph at all.

To address my valid concerns, Google now gives me an opportunity to ‘challenge’ their solution to the graph coloring*.* I’m allowed to pick — at random — a single ‘edge’ of this graph (that is, one line between two adjacent hats). Google will then remove the two corresponding hats, revealing a small portion of their solution:

[](https://matthewdgreen.files.wordpress.com/2014/11/12725-hatsminusone.png)

Notice that there are two outcomes to my experiment:

1. If the two revealed vertices are the same color (or aren’t colored in at all!) then I definitely know that Google is lying to me. Clearly I’m not going to pay Google a cent.
2. If the two revealed vertices are different colors, then Google *might* *not* be lying to me.

Hopefully the first proposition is obvious. The second one requires a bit more consideration. The problem is that *even after our experiment*, Google could still be lying to me — after all, I only looked under two of the hats. If there are *E* different edges in the graph, then Google could fill in an invalid solution and still get away with it most of the time. Specifically, after one test they could succeed in cheating me with probability up to (*E*-1)/*E* (which for a 1,000 edge graph works out to 99.9% of the time).

Fortunately Google has an answer to this. We’ll just run the protocol *again!*

We put down fresh paper with a new, blank copy of the graph. *Google now picks a new (random) shuffle of the three crayons*. Next they fill in the graph with a valid solution, but using the new random ordering of the three colors.

The hats go back on. I come back in and repeat the challenge process, picking a new random edge. Once again the logic above applies. Only this time if all goes well, I should now be slightly more confident that Google is telling me the truth. That’s because in order to cheat me, Google would have had to get lucky twice in a row. That can happen — but it happens with relatively lower probability. The chance that Google fools me twice in a row is now (*E*-1)/*E \**(*E*-1)/*E*(or about 99.8% probability for our 1,000 edge example above).

Fortunately we don’t have to stop at two challenges. In fact, we can keep trying this over and over again until I’m confident that Google is probably telling me the truth.

But don’t take my word for it. Thanks to some neat Javascript, you can [go try it yourself](http://web.mit.edu/~ezyang/Public/graph/svg.html).

Note that I’ll never be perfectly certain that Google is being honest — there’s always going to be a tiny probability that they’re cheating me. But after a large number of iterations (*E^2,* as it happens) I can eventually raise my confidence to the point where Google can only cheat me with [negligible](http://en.wikipedia.org/wiki/Negligible_function) probability — low enough that *for all practical purposes* it’s not worth worrying about. And then I’ll be able to safely hand Google my money.

What you need to believe is that Google is also protected. Even if I try to learn something about their solution by keeping notes between protocol runs, it shouldn’t matter. I’m foiled by Google’s decision to *randomize* their color choices between each iteration. The limited information I obtain does me no good, and there’s no way for me to *link* the data I learn between interactions.

#### 三个必要条件：

ZKP算法的正常工作必须要满足以下三个条件：

* **Completeness:** If the statement is true then an honest verifier can be convinced of it by an honest prover.只要声明事实上为真，则一个诚实的prover一定能说服另一个诚实的verifier。（也就是不管怎么交互，重复验证多少次，都没问题，或者说有问题的概率已经极小）
* **Soundness:** If the prover is is dishonest, they can’t convince the verifier of the soundness of the statement by lying.如果prover撒谎则无法说服verifier。（因为大量重复验证的过程中，总会有极大的概率露馅）
* **Zero-Knowledge:** If the statement is true, the verifier will have no idea what the statement actually is.除了声明为真这一信息之外，verifier无法获取其他任何关于声明具体内容的额外信息；（解释如下）

##### **示例：**

###### **Completeness**

（在三色问题中，The protocol will eventually convince me (with a negligible error probability), provided we run it enough times.）

###### **Soundness**

（在三色问题举例中，soundness is also pretty easy to show here. If Google ever tries to cheat me, I will detect their treachery with overwhelming probability）

###### zero knowledge:

为了说明“zero knowledge”究竟是什么意思，我们继续用三色问题构造一个思想实验进行说明。

假设google工程师并没有想出解决方案，为了能欺骗对方他们已经有解决方案，他们决定采用研发的时光机。在验证时，如果发现随机抽选的两个帽子下面的颜色一样，则工程师可以利用时光机回到几分钟之前，重新更换上色方案，从而每次验证都通过。实际上，利用时光机，google工程师可以在任何时候更改他们的假方案，使其可以继续看来仍然是正常的，最终说服验证者，他们的方案是真的（这个实际上就已经违背了零知识证明协议三要素中的“soundess”，即他们通过撒谎作弊等手段，使a honest verifier相信了他们的声明为真）。

对google工程师来说，这一方法虽然导致他们的方案实际运行时耗费更多的时间，但是对验证者来说，实际是完全认识不到的。虽然，在作弊方案中，证明者即google工程是也完全不知道真正的方案是什么样的，但对验证者来说，真正的方案和作弊的方案至少从结果的统计意义来说，完全没有区别。两个方案都传达了传达了同样多的信息。

believe it or not，this proves something very important.

特别是，假设verifier有一种特殊的策略可以通过观察the honest protocol的运行，从而从中提取中其他的有用信息。那么，真正的方案协议和作弊方案协议因为对verifier来说，在运行时are statistically identical,the verifier can not tell the difference,那么在作弊方案下这种策略就也应该也能提取出同样的有用信息。然而，事实是， google工程师在作弊方案中所输入的信息量实际为0。因此反过来说，这也暗示者，即使是在真实世界，真正方案的证明过程中也应该肯定不会泄露任何有用的信息。当然，事实上我们并不会运行 a protocol with hats。And nobody has a literal time machine.

零知识协议之commitment scheme：

我们将讨论回归到计算机世界，这要求我们所设计的零知识证明协议应该包含一个类似于“hat”的机制，一方面它可以在证明的过程中帮助prover隐藏背后的真实数字信息，另一方面，simutaneously “binding”(or “committing”)the maker to it, so she can’t change her mind after the fact.（同时又能限制prover一旦向verifier发出信息（声明）后或者说做出承诺后，就无法再作弊或改变，不可抵赖）。

在密码学技术中，已经有现成技术可以满足这样的需求：commitment scheme（详细解释参考上文“概念术语”部分）。

A commitment scheme allows one party to ‘commit’ to a given message while keeping it secret, and then later ‘open’ the resulting commitment to reveal what’s inside. Commitment schemes are designed so that a party cannot change the value or statement after they have committed to it: that is, commitment schemes are *binding*.

零知识协议之----除了commitment scheme：

除了commitment scheme之外，怎么才能证明一个协议是零知识的呢？

What we can now prove is the following theorem: if you could ever come up with a computer program (for the Verifier) that extracts useful information after participating in a run of the protocol, then it would be possible to use a ‘time machine’ on that program in order to make it extract the same amount of useful information from a ‘fake’ run of the protocol where the Prover doesn’t put in any information to begin with.

And since we’re now talking about *computer programs*, it should be obvious that rewinding time isn’t such an extraordinary feat at all. In fact, we rewind computer programs all the time. For example, consider using virtual machine software with a snapshot capability.

|  |
| --- |
| [https://i2.wp.com/download.parallels.com/desktop/v4/docs/en/Parallels_Desktop_Users_Guide/snapshotmanager.gif](http://download.parallels.com/desktop/v4/docs/en/Parallels_Desktop_Users_Guide/snapshotmanager.gif) |
| Example of rewinding through VM snapshots. An initial VM is played forward, rewound to an initial snapshot, then execution is forked to a new path. |

Even if you don’t have fancy virtual machine software, any computer program can be ‘rewound’ to an earlier state, simply by starting the program over again from the beginning and feeding it exactly the same inputs. Provided that the inputs — including all random numbers — are fixed, the program will always follow the same execution path. Thus you can rewind a program just by running it from the start and ‘forking’ its execution when it reaches some desired point.

Ultimately what we get is the following theorem. If there exists any Verifier computer program that successfully extracts information by interactively running this protocol with some Prover, then we can simply use the rewinding trick on that program to commit to a random solution, then ‘trick’ the Verifier by rewinding its execution whenever we can’t answer its challenge correctly. The same logic holds as we gave above: if such a Verifier succeeds in extracting information after running the real protocol, then it should be able to extract the *same amount of information* from the simulated, rewinding-based protocol. But since there’s no information going into the simulated protocol, there’s no information to extract. Thus the information the Verifier can extract must always be zero

#### 起源：

参考：

<https://blog.cryptographyengineering.com/2014/11/27/zero-knowledge-proofs-illustrated-primer/>

The notion of ‘zero knowledge’ was first proposed [in the 1980](http://groups.csail.mit.edu/cis/pubs/shafi/1985-stoc.pdf)s by MIT researchers Shafi Goldwasser, Silvio Micali and Charles Rackoff. These researchers were working on problems related to [interactive proof systems](http://en.wikipedia.org/wiki/Interactive_proof_system), theoretical systems where a first party (called a ‘Prover’) exchanges messages with a second party (‘Verifier’) to convince the Verifier that some mathematical statement is true.\*

Prior to Goldwasser *et al.*, most work in this area focused the [soundness](http://en.wikipedia.org/wiki/Soundness) of the proof system. That is, it considered the case where a malicious Prover attempts to ‘trick’ a Verifier into believing a false statement. What Goldwasser, Micali and Rackoff did was to turn this problem on its head. Instead of worrying only about the Prover, they asked: what happens if you don’t trust the *Verifier?*

The specific concern they raised was *information leakage.* Concretely, they asked, how much extra information is the Verifier going to learn during the course of this proof, beyond the mere fact that the statement is true?

（之前证明系统的研究主要集中在如何防止恶意的Prover时verifier相信了家的声明，*而MIT的*Goldwasser则反过来，研究的是如果verifier不诚实怎么办？具体来说，在证明的过程中，除了声明为真这一事实本身之外，verifer需要了解多少额外信息，并防止verifier将这些额外信息泄露出去。这一问题的现实需求举例如下：）

It’s important to note that this is not simply of theoretical interest. There are real, practical applications where this kind of thing matters.

Here’s one: imagine that a real-world client wishes to log into a web server using a password. The standard ‘real world’ approach to this problem involves storing a [hashed version of the password](http://en.wikipedia.org/wiki/Cryptographic_hash_function#Password_verification) on the server. The login can thus be viewed as a sort of ‘proof’ that a given password hash is the output of a hash function on some password — and more to the point, that the client actually *knows* the password.

Most real systems implement this ‘proof’ in the absolute worst possible way. The client simply transmits the original password to the server, which re-computes the password hash and compares it to the stored value. The problem here is obvious: at the conclusion of the protocol, *the server has learned my cleartext password.* Modern password hygiene therefore involves a good deal of praying that servers aren’t compromised.

（如登陆系统设计中，客户端通过密码登陆服务器这一操作，标准的实现通常是在服务器端存储密码的一个哈希值，然后客户端登陆的过程，就可以看成是向服务器证明某个密码哈希值就是某个密码哈希后的结果，更直白说，就是证明客户端确实知道密码。而现实世界的实际实现通常是最坏的一种实现，即客户端只直接将明文密码发送给服务端，由服务端重新计算密码的哈希值并与存储的值比较。而这种实现很明显的问题就在于，服务器知道了明文密码）

What Goldwasser, Micali and Rackoff proposed was a new hope for conducting such proofs. If fully realized, zero knowledge proofs would allow us to prove statements like the one above, while provably revealing *no information* beyond the single bit of information corresponding to ‘this statement is true’

#### 在区块链技术中的应用：

Many blockchain based technologies are using Zk-Snarks, in fact, even [Ethereum](https://blockgeeks.com/guides/what-is-ethereum/) in its Metropolis phase is planning to bring in [Zk-Snarks](https://blockgeeks.com/guides/what-is-zksnarks/) and add it to its arsenal. Zk-Snarks stands for “Zero-Knowledge Succinct Non-Interactive Argument of Knowledge” and it proves a computational fact about the data without revealing the data itself.

They can be used to generate a proof of statement to verify each and every transaction by just taking a simple snapshot of each transaction which is enough to prove to the receiving side that a transaction was done without revealing the transaction itself.

**This achieves two things:**

* The integrity and privacy of the transaction is maintained.
* By not revealing the inner workings of the entire transaction the system maintain abstraction which makes it infinitely easier to use.

So these are some of the important cryptographical functions which are being used by the blockchain. Now let us look at the second pillar, Economics.

# 区块链实现示例：

## 区块

### 结构：

### 操作：

创建：

就是区块结构中的各个成员的初始化；

## 区块链：

创世区块（genesis block）

### 结构：

### 操作：

#### 创建：

#### 添加新的block(入链)：

流程：

1、获取前一个block的hash，存到新block的previous hash；

2、计算新block的hash，存到新block的hash；

3、入链；

#### 区块链vlaid检查：

遍历所有的block，需要满足：重新计算的每个block hash值等于其中存储的值；每个block中存储的前一个block的hash值确实等于其前一个block的hash值；

# 智能合约（smart contract）

# 问题：

## 加密币的价值来源

区块链中的加密币的价值来源，应该是有多种。

来源0（价值起点）：

其中其最基本的或者说最小价值等于为了产生加密币所消耗的时间，电力，计算设备等资源的价值。

来源1：使用

示例：