Bitcoin: A Peer-to-Peer Electronic Cash System

Satoshi Nakamoto
satoshin@gmx.com
www.bitcoin.org

Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll generate the longest chain and outpace attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone.

1. Introduction

Commerce on the Internet has come to rely almost exclusively on financial institutions serving as trusted third parties to process electronic payments. While the system works well enough for most transactions, it still suffers from the inherent weaknesses of the trust based model. Completely non-reversible transactions are not really possible, since financial institutions cannot avoid mediating disputes. The cost of mediation increases transaction costs, limiting the minimum practical transaction size and cutting off the possibility for small casual transactions, and there is a broader cost in the loss of ability to make non-reversible payments for non-reversible services. With the possibility of reversal, the need for trust spreads. Merchants must be wary of their customers, hassling them for more information than they would otherwise need. A certain percentage of fraud is accepted as unavoidable. These costs and payment uncertainties can be avoided in person by using physical currency, but no mechanism exists to make payments over a communications channel without a trusted party.

What is needed is an electronic payment system based on cryptographic proof instead of trust, allowing any two willing parties to transact directly with each other without the need for a trusted third party. Transactions that are computationally impractical to reverse would protect sellers from fraud, and routine escrow mechanisms could easily be implemented to protect buyers. In this paper, we propose a solution to the double-spending problem using a peer-to-peer distributed timestamp server to generate computational proof of the chronological order of transactions. The system is secure as long as honest nodes collectively control more CPU power than any cooperating group of attacker nodes.

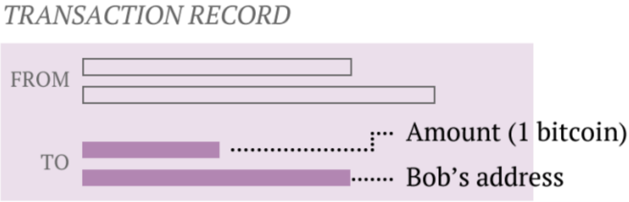
What is needed is an electronic payment system based on cryptographic proof instead of trust, [. . .] without the need for a trusted third party

We propose a solution [. . .] using a peer-to-peer distributed timestamp server to generate [. . .] proof of the chronological order of transactions

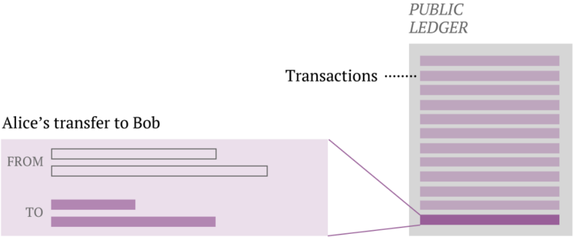
Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. Online, bitcoin.org/bitcoin.pdf. 2008

Transactions I

- Alice transfers bitcoins to Bob

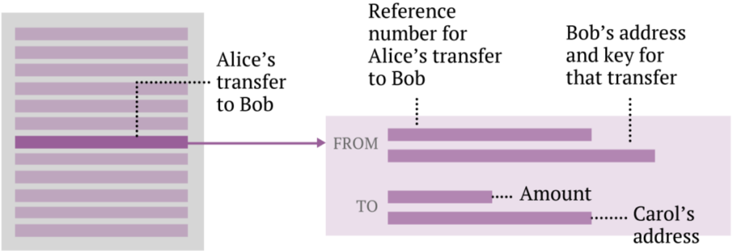


- this is written in a public ledger



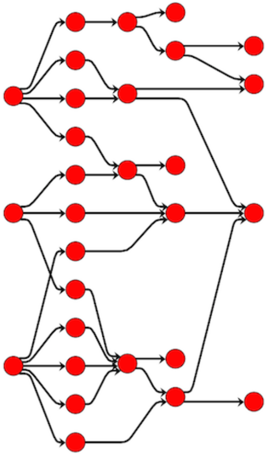
Transactions II

- Bob can then transfer to Carol



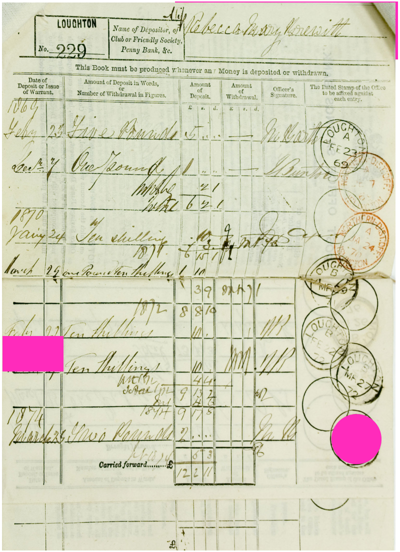
- Bob has to sign the new transaction, with asymmetric crypto
- simple
- combination of several inputs and outputs
- many, many outputs
- with coins to self

Graph of transactions

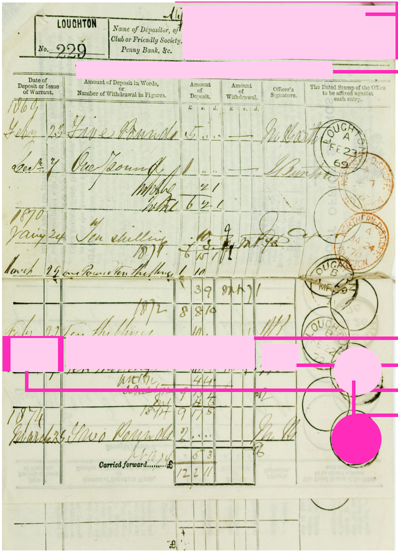


All of this needs to certified, agreed upon, etc

Blocks are chained



Blocks and the ledger



Rebbeca Mary Marewitt

"This book must be produced whenever any money is deposited or withdraw"

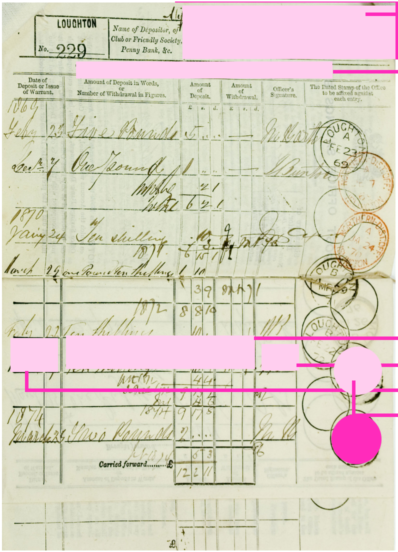
Transaction

Officer's signature

March 27, 1869

Date stamp of the office
to be affixed against each entry

Blocks and the ledger



Rebbeca Mary Marewitt Bitcoin address

"This book must be produced whenever any money is deposited or withdraw"

Public Ledger

Transaction

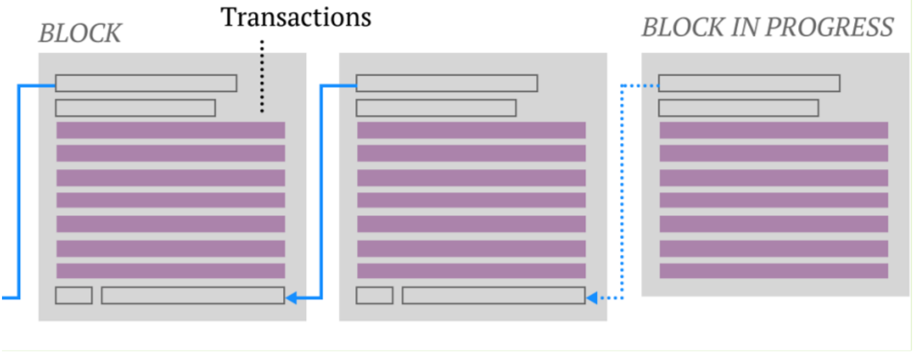
Officer's signature No officer, no signature

March 27, 1869

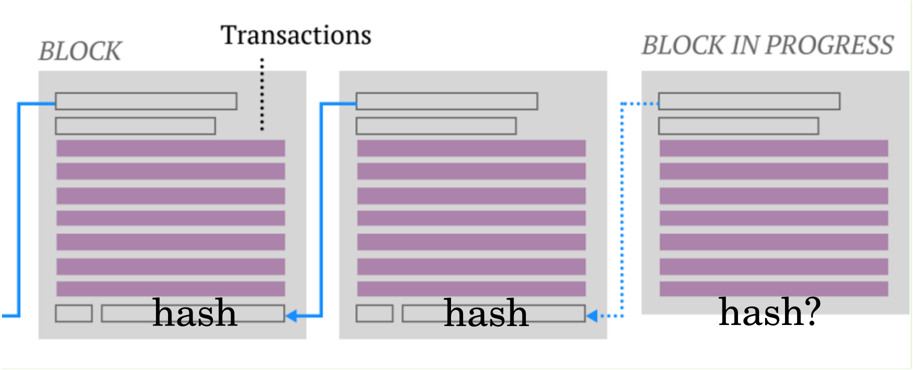
Date stamp of the office to be affixed against each entry

"mining"

Blocks are chained



Blocks are chained *and certified by a hash*



Cryptographic hash function

$$H : \begin{cases} \{0,1\}^* & \rightarrow & \{0,1\}^{256} \\ m & \mapsto & H(m) \end{cases}$$

- deterministic algorithm
- impossible to invert, predict, etc
- should look random (random oracle)
- no secrets, no keys (neither private, public, or secret)
  - it is **not** signature, **neither** encryption
  - it is **not** signature, **neither** encryption
  - it is **not** signature, **neither** encryption
  - ...

Cryptographic hash function

$$H : \begin{cases} \text{any bit string} & \rightarrow & \text{256 bits} \\ m & \mapsto & H(m) \end{cases}$$

- deterministic algorithm
- impossible to invert, predict, etc
- should look random (random oracle)
- no secrets, no keys (neither private, public, or secret)
  - it is **not** signature, **neither** encryption
  - it is **not** signature, **neither** encryption
  - it is **not** signature, **neither** encryption
  - ...

Cryptographic hash function

$$H : \begin{cases} \text{any byte string} & \rightarrow & \text{64 bytes} \\ m & \mapsto & H(m) \end{cases}$$

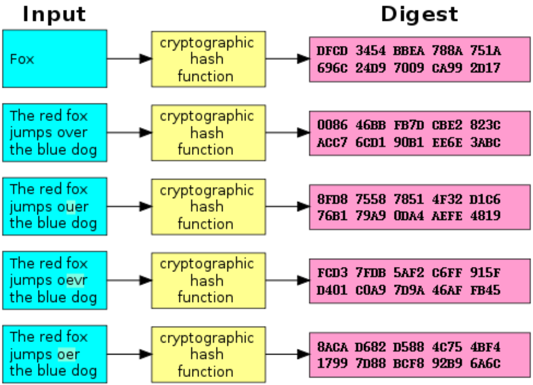
- deterministic algorithm
- impossible to invert, predict, etc
- should look random (random oracle)
- no secrets, no keys (neither private, public, or secret)
  - it is **not** signature, **neither** encryption
  - it is **not** signature, **neither** encryption
  - it is **not** signature, **neither** encryption
  - ...

Cryptographic hash function

$$H : \begin{cases} \text{any digitalized document} & \rightarrow & \text{64 bytes} \\ m & \mapsto & H(m) \end{cases}$$

- deterministic algorithm
- impossible to invert, predict, etc
- should look random (random oracle)
- no secrets, no keys (neither private, public, or secret)
  - it is **not** signature, **neither** encryption
  - it is **not** signature, **neither** encryption
  - it is **not** signature, **neither** encryption
  - ...

Cryptographic hash function



terminal demo

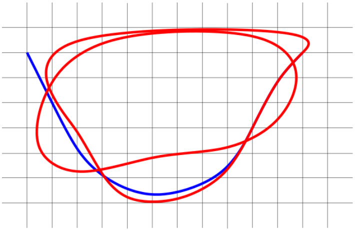
Difficult to create, only a few are in use: SHA1, SHA256, Keccak (SHA3)

Why such a thing would be useful

Most unsemantic function: given input x, the output $h = F(x)$ is random!

Usage

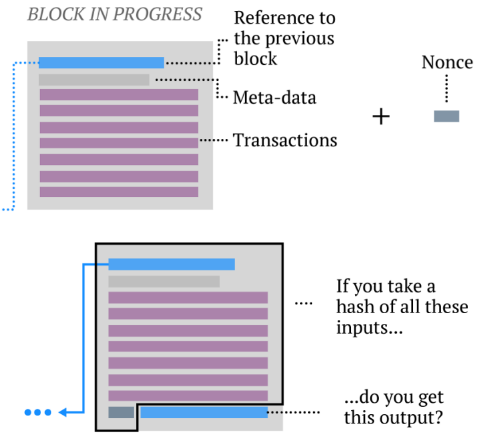
- Ensuring file integrity: $M \mapsto (M, h(M))$
  If $h(M)$ is secure, there can be non corruption on M
- Password storage
- Play heads or tails on the phone
- Blind registration of documents



d95b82d3187458f83ad36abd509c7688f60cbda4

Mining

Mining is finding a nonce wich contributes to a partially prescribed hash



nonce = an arbitrary number used only once

Proof-of-work I

bitcoin uses `SHA-256`^{2}, whose output is 256-bit

Proof of work (simplified)

- given an integer N
- to mine block-data:

    UNTIL hash starts with N zero bits

        nonce = next nonce

        hash = SHA-256(SHA-256(block-data || nonce))

Probability for success for one iteration

$$\mathbb{P} = \frac{1}{2^N}$$

No better strategy than iterating (the hash is random)

Proof-of-work II

Proof of work, with more granularity

- given a "target" $T \in [0, 2^{256})$
- to mine block-data:

UNTIL hash $<$ T

nonce $=$ random value

hash $=$ SHA-256(block-data || nonce)

Probability $\mathbb{P}$ for success for one interation: $\frac{T}{2^{256}}$

Feedback

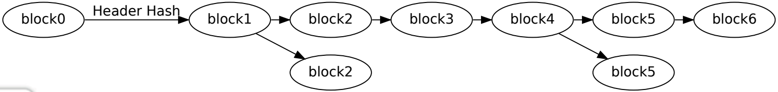
T is readjusted every 2048 blocks, to keep producing a block every 10 min

$$\mathbb{P} = \frac{1}{903,262,006,880,187,187,200} \approx 2^{-69.6} \approx 10^{-21}$$

Protocol summary

From Satochi's paper

1. *new transactions are broadcast to all*† *nodes*
2. *each*† *node collects new transactions into a block*
3. *each*† *node works on finding a difficult proof-of-work for its block*
4. *when a*† *node finds a proof-of-work, it broadcasts the block to all nodes*
5. *nodes*† *accept the block only if all transactions in it are valid and not already spent*
6. *nodes*† *express their acceptance of the block by working on creating the next block in the chain, using the hash of the accepted block as the previous hash*



Rules I

Coin creation, minting

By convention, the first transaction in a block [. . .] starts a new coin

This adds an incentive for nodes to support the network [. . .]

and provides a way to distribute coins, since there is no central authority

- the first years, the reward for successful mining was 50 bitcoins
- now 12.5
- this value halves every 210,000 blocks, no new bitcoins in 2140

Fees

If the output of a transaction is less than its value, the difference is a transaction fee

The incentive can transition to transaction fees [. . .]

Crypto and politics

Cryptography

- hash functions
- proof-of-work
- 51% rule: controling mining requires the majority of hash power
- electronic signature

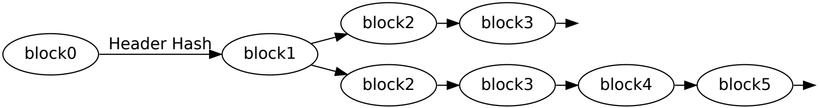
Non crypto, neither technical, choices

- deflationnist rule for bitcoin creation
- fees
- anonymity (sort of)
- protect bitcoin, not Alice or Bob

Governance I

Levels of control

1. reference implementation: github.com/bitcoin
2. the protocol: BIP (Bitcoin Improvement Proposals)
3. the miners: they choose (the protocol of) the blocks they accept



4. the economic power: those exchanging value for bitcoins

Forks did happen, mild for Bitcoin, severe for DarkCoin

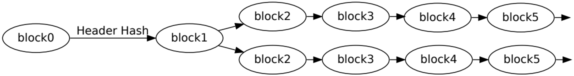
Governance II

The block size issue ("bitcoin crisis")

- max block size is 1 Mo (P2P performance), ≤ 7 transactions/second
- two clans: augment the block size or keep the block size fixed

The story of bitcoin-xt (Jul 2015 - Jan 2016)

1. devs could not agree
2. BIP101, still no agreement
3. accepted if 75% of last 1000 blocks are mined by `bitcoin-xt`
4. risk of fork, and two blockchains with incompatible bitcoin sets



5. bitcoin-xt failed

Cryptocurrencies

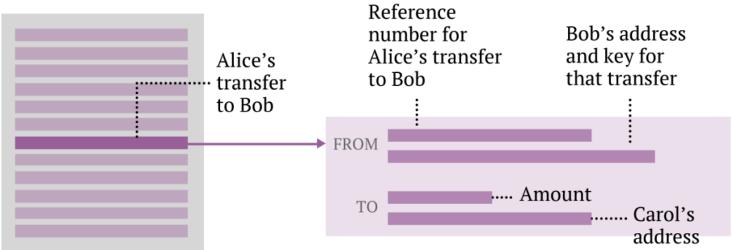
- Litecoin — another hash function
- PeerCoin — proof of stake
- DarkCoin/Dash — for the “DarkWeb”
- Monero — advanced crypto for privacy, anonymity
- ZeroCoin — zero-knowledge proofs

Goldfinger attack

- non profit, political attack, to get the majority of hash power
- Eligius mining pool destroyed CoiledCoin

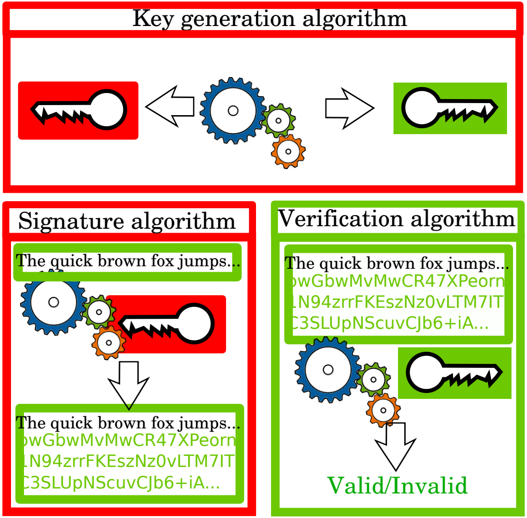
No Alice and Bob, adresses

- Bob has to *sign* its new transation

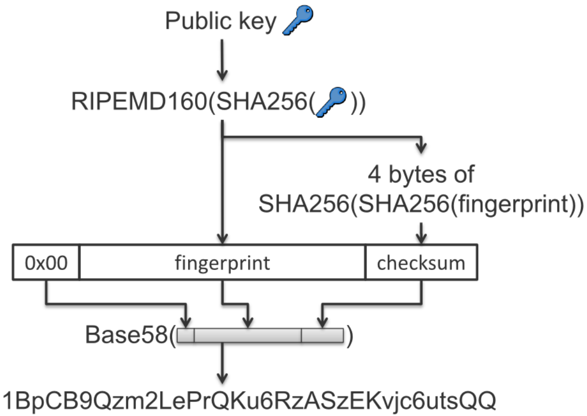


- a wallet controls a *private key*, enabling to *sign* transactions

Electronic signature

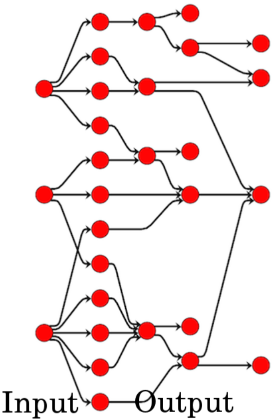


There are no Alice and Bob, but “adresses”



- bitcoin: no names, only *hash of* public keys
- users send money to *hashes of* public keys

Transaction Input and Output



A transaction is the sum of an input and of an output
A programming langage is used to describe inputs and outputs

Scripts

- the standard “output script” (FORTH-like langage)

  OP_DUP
  OP_HASH160
  404371705fa9bd789a2fcd52d2c580b65d35549d
  OP_EQUALVERIFY
  OP_CHECKSIG

- the standard “redeem”script

  304502206e21798a42fae0e854281abd38bacd1aeed3ee3738d9e1446618c4571d10
  (<Sig>)

  90db022100e2ac980643b0b82c0e88ffdfec6b64e3e6ba35e7ba5fdd7d5d6cc8d25c6b24150
  (<PubKey>)

Script execution

signature script

- <sig>
- <pubKey>

Output script of previous transaction

- OP_DUP
- OP_HASH160
- pubKeyHash
- OP_EQUALVERIFY
- OP_CHECKSIG

Script execution

<sig>

<pubKey>

OP_DUP

OP_HASH160

pubKeyHash

OP_EQUALVERIFY

OP_CHECKSIG

Script execution

	<pubKey>
	OP_DUP
	OP_HASH160
	pubKeyHash
	OP_EQUALVERIFY
<sig>	OP_CHECKSIG

Script execution

	OP_DUP
	OP_HASH160
	pubKeyHash
<pubKey>	OP_EQUALVERIFY
<sig>	OP_CHECKSIG

Script execution

	OP_HASH160
<pubKey>	pubKeyHash
<pubKey>	OP_EQUALVERIFY
<sig>	OP_CHECKSIG

Script execution

<pubKeyHash>	pubKeyHash
<pubKey>	OP_EQUALVERIFY
<sig>	OP_CHECKSIG

Script execution

pubKeyHash

<pubKeyHash>

<pubKey>

<sig>

OP_EQUALVERIFY

OP_CHECKSIG

Script execution

<pubKey>

<sig>

OP_CHECKSIG

Script execution

Script execution (by the miner)

Elaborate scripts

- Multisignature

```
OP_2
[A's pubkey]
[B's pubkey]
[C's pubkey]
OP_3
OP_CHECKMULTISIG
```

- "smart contracts": escrow, micropayment channel
- pay-to-script-hash (P2SH), example

```
out:  OP_HASH160 <scriptHash> OP_EQUAL
in:  ..signatures...  <serialized script>
```

Ethereum

- rapid diffusion: July 2015 (frontier), Feb. 2016 (homestead)
  - white paper, yellow paper
- currency is *ether*
- gaining traction: $\sim$ \$1 billion worth of ether (https://etherchain.org)
- press coverage NY Times, 2016/03/28

  "The system is complicated enough that even people who know it well have trouble describing it in plain English"

- so let us try plain computer science language

Abstract bitcoin

```
Transaction
404371705fa...
sends 3 to
0e3d7f56b4f...
<Signature>
```

```
State                              State
404371705fa... 10        ⟹        404371705fa...  7
0e3d7f56b4f...  8                  0e3d7f56b4f... 11
```

State machine

- imagine all executed bitcoin transactions have defined a *state* S
- a transaction defines a *state transition* T
- when a block pf transaction is mined, a new state S' is determined
- further confirmations show a consensus has emerged on S'

Slogan: Ethereum is bitcoin with a Turing complete language

Ethereum model

- adresses
  1. externally owned account: people, with a private key
  2. contracts: programs, with no private key
     - code, *storage*
- transactions (sent users) transfert ether and requests to programs
- messages sent by programs

State transitions APPLY(S,T)

- if destination is an external account: transfer value to the receiver
- if the receiving account is a contract: $\rightarrow$ run its code

Block validation

- from current state S, for each transaction T in the block, do $S \leftarrow \text{APPLY}(S, T)$
- check the proof-of-work on the hash of the new state S

“The world computer”

Computer Science view

- the blockchain of ethereum is irreversible history of a memory
- data and program are stored on the blockchain
- miners execute code, modify the memory

Seen on forum.ethereum.org

“havo” has a question

> *I am a student of economics and I already got some ideas for simple ethereum projects, however, I don't know how to code. I have only a little experience in python [. . .]*

A 101 Noob Intro to Programming Smart Contracts on Ethereum

```
contract FFAToken {
    // The keyword "public" makes those variables
    // readable from outside.
    address public minter;
    mapping (address => uint) public balances;

    // Events allow light clients to react on
    // changes efficiently.
    event Sent(address from, address to, uint amount);

    // This is the constructor whose code is
    // run only when the contract is created.
    function FFAToken() {
        minter = msg.sender;
    }
    function mint(address receiver, uint amount) {
        if (msg.sender != minter) return;
        balances[receiver] += amount;
    }
    function send(address receiver, uint amount) {
        if (balances[msg.sender] < amount) return;
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        Sent(msg.sender, receiver, amount);
    }
}
```

dapps

- etherpot.github.io — on line lottery
- github.com/maran/notareth — notary
- etherid.org — name registrar
- weifund.io — crowdfunding
- www.trustlessprivacy.com — interoperable electronic health records
- cetas.github.io — Decentralized KYC and Credit rating framework

a DAO is an [. . .] entity that exists as executable code on the block-chain

Economics

- the blockchain of ethereum is for every one, for every application
  - no need to build a blockchain with its reputation
  - more applications, more money spent, more miners
  - more money, the stronger the blockchain, stronger the appeal for devs
- it is a platform
  - two implementation (go, C++)
  - programming langages (solidity, serpent)
  - development framework


Download from
Dreamstime.com
ID 31069105
Anton Kovalenko | Dreamstime.com