Fundação Getúlio Vargas

Modelagem Matemática

Programming a cryptocurrency in Agda

Student:
Guilherme Horta Alvares da Silva
Professor:
Doctor Flávio Codeço Coelho

Contents

1 Introduction

1.1 Cryptocurrencies

In 1983, David Chaum created ecash (Panurach, 1996) , an anonymous cryptographic eletronic money. This cryptocurrency use RSA blind signatures (Chaum, 1983) to spend transactions. Later, in 1989, David Chaum found an eletronic money corporation called DigiCash Inc. It was declared bankruptcy in 1998.

Adam Back developed a proof-of-work (PoW) scheme for spam control, Hashcash (Back et al., 2002). To send an email, the hash of the content of this email plus a nounce has to have a numerically value smaller than a defined target. So, to create a valide email, the sender (miner) has to spend a considerable CPU resource on it. Because, hash functions produces pratically random values, so the miner has to guess a lot of nounce values before find some nounce that make the hash of the email less than the target value. This idea is the same that is used in Bitcoin proof of work, because each block has a nounce guessed by the miner and the hash of the block has to be less than the target value.

Wei Dai propose b-money (Dai, 1998) for the first proposal for distributed digital scarcity. And Hal Finney created Bit Gold (Wallace, 2011), a reusable proof of work for hashcash for its algorithm of proof of work.

In 31 October 2008, Satoshi Nakamoto registered the website “bitcoin.org” and put a link for his paper (Nakamoto et al., 2008) in a cryptography mailing list. In January 2009, Nakamoto released the bitcoin software as open-source code. The identity of Satoshi Nakamoto is still unknown. Since that time, the total market of Bitcoin came to 330 billions dollars in 17 of December of 2018 and his value has a historic record of 20 thousands dollars.

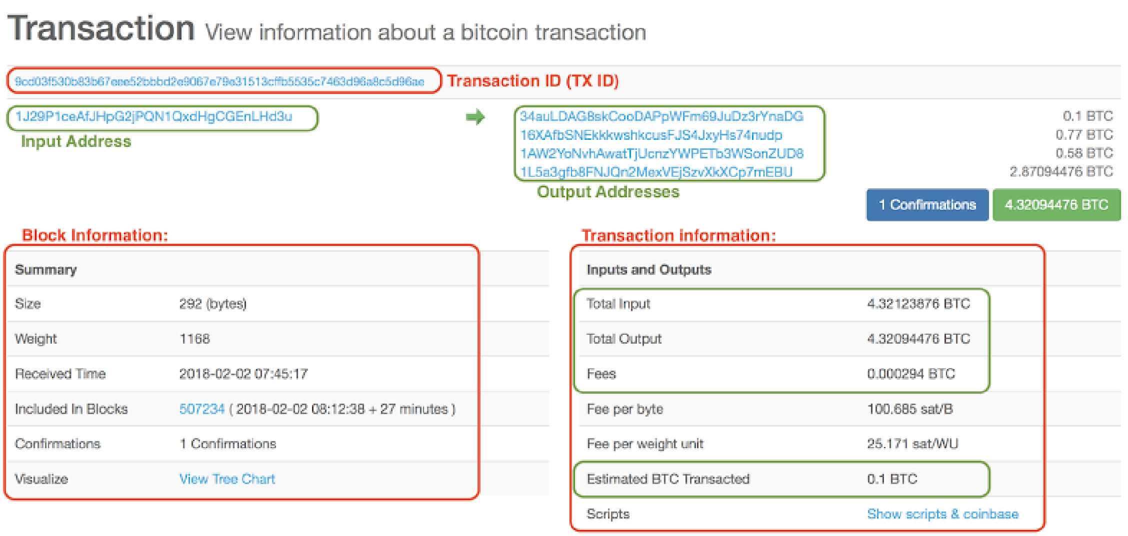
Other cryptocurrencies like Ethereum (Wood et al., 2014), Monero (Noether, 2015) and ZCash (Hopwood, Bowe, Hornby, & Wilcox, 2016) were created after Bitcoin, but Bitcoin is still the cryptocurrency with the biggest market value.

Ethereum is a cryptocurrency that uses account model instead of UTXO used in bitcoin for its transaction data struct. It uses Solidity as its programming language for smart contracts, it looks like Java Script, so it is easier to programming in it than in stack machine programming language of Bitcoin. Ethereum is now changing from proof of work (used in Bitcoin) to proof of stake.

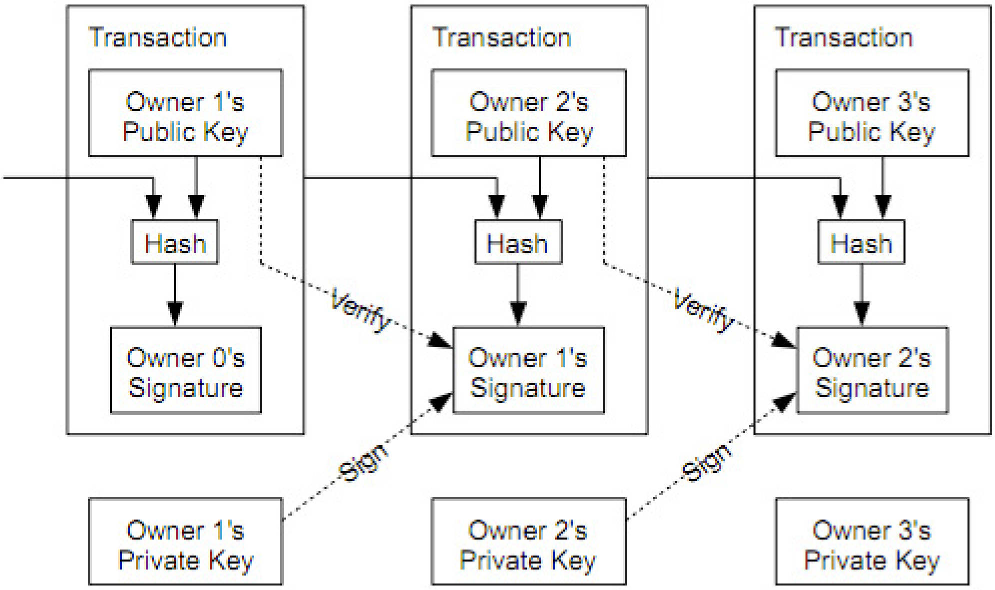
Monero and ZCash are both cryptocurrency that focus on fungibility, privacy and descentralization. Monero uses obfuscated public ledger, so anyone can send transcations, but nobody can tell the source, amount or destination. Zcash uses the concept of zero-knowledge proof called zk-SNARKs, which garantee privacy for its users.

1.2 Bitcoin

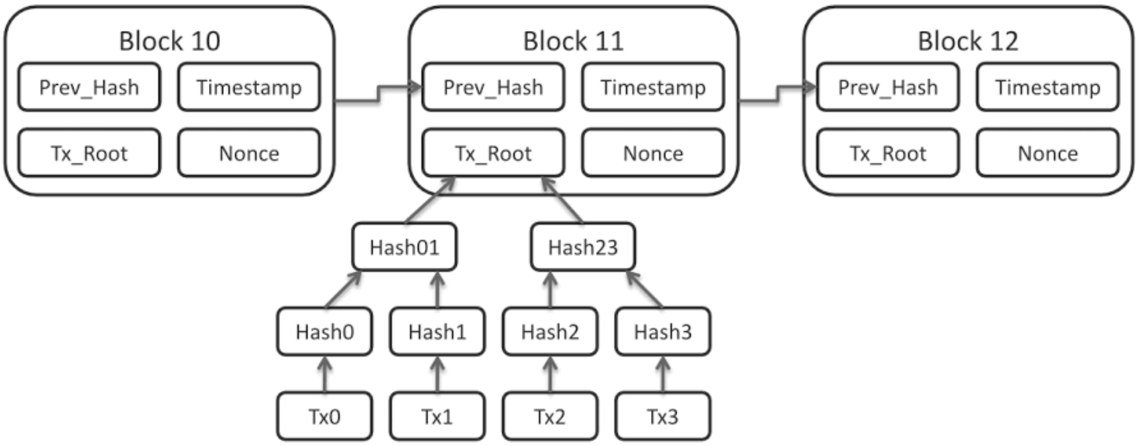
The bitcoin was made to be a peer to peer eletronic cash. It was made in one way that users can save and verify transactions without the need of a trust party. Because of that no authority or goverment can block the bitcoin.



Transactions in bitcoins are one array of input of previous transactions and one array of outputs. The mining transaction does not need to have an input. For each input of the transaction, it is necessary a signature signed with a private key to prove the ownerchip of the bitcoins.



Transactions are grouped in a block. Each block has in its header the timestamp that the block was created, the hash of the block, the previous hash and a nounce. A nounce is an arbitrary value that the miner has to choose to make the hash of the block respect some specific characteristics.



Each block has a limit size of 1 MB. Because of that, bitcoin has a blockchain (one array of blocks). Each block should be created in an average of 10 minutes. This time was choosen because in 10 minutes, it is enough to propagate the block in all the world. To make it possible, there is a concept called proof of work in bitcoin. So the miner has choose a random value as nounce that makes the hash of the block less than a certain value. This value is choosen in a way that each block should be generated in 10 minutes in average. If the value is too low, miners will take more time to find a nounce that make the hash block less than the it. If it is too high, it will be easier to find a nounce and they will find it faster.

When two blocks are mined in nearly the same time, there are two valid blockchains. It is because the last block in the both blockchains are valid but different. Because of this problem, in Bitcoin protocol, the largest chain is always the right chain. While two valid chains have the same size, it is not possible to know which chain is the right. This situation is called fork and when it happens, it is necessary to wait to see in which chain the new block will be.

In Bitcoin, there is a possibility of 51% attack. It happens when some miner, with more power than all network, mine secretly the blocks. So if the main network has 50 blocks, the miner could produce hidden blocks from 46 to 55 and he would have 10 hidden blocks from the network. When he shows their hidden blocks, his chain become the valid chain, because it is bigger. So all transactions from previous blockchain from 46 to 50 blocks become invalid. Because of that, when someone make a big transaction in the blockchain, it is a good idea to wait more time. So it is becoming harder and harder to make a 51% with more time. Bitcoin has the highest market value nowadays, so attacking the bitcoin network is very expensive. Nowadays, this kind of attack is more common in new altcoins.

1.3 Ethereum

Ethereum differs from bitcoin in having an Ethereum Virtual Machine (EVM) to run script code. EVM is a stack machine and turing complete while Bitcoin Script is not (it is impossible to do loops and recursion in Bitcoin).

Transactions in bitcoin are all stored in blockchain. In Ethereum, just the hash of it is stored in it. So it is saved in off chain database. Because of that, it is possible to save more information in Ethereum Blockchain.

In Bitcoin, the creator of the contract as to pay the amount proportional to its size. In Ethereum, it is different, there is a concept of gas. Each smart contract in Ethereum is made by a serie of instructions. Each instruction consume different computional effort. Because of that, in Ethereum, there is a concept of gas, that measure how much computional effort each instruction needs. So in each smart contract, it is well know how much computational effort will be necessary to run it and it is measured in gas. Because computional effort is a scarce resource, to execute the smart contract, it is necessary to pay an amount in Ether for each gas to the miner run it. Smart contracts that pay more ether per gas run first, because the miner will want to have the best profit and they will pick them. If the amount of ether per gas payed is not high enough, the contract will not be executed, because there are some other contracts that pay more that will be executed instead of this one.

Because Ethereum has its own EVM with more instructions than Bitcoin and it is Turing Complete, its considered less secure. Ethereum has its own high level programming language called Solidity that looks like JavaScript.

1.4 Introduction

Before my work, there were some research in this field. Antom Setzel (Setzer, 2018) already code the definitions of transactions and transactions tree of bitcoin. Orestis Melkonian start to formalize Bitcoin Script.

My work try to extend Antom Setzel model and make possible to use Bitcoin protocol from inputs and outputs from plain text. For example, the user send a transaction in plain text to the software and it validates if it is right. To use the Antom Setzel model, the user has to send the data and the proof that it is correct.

1.5 Agda Introduction

Agda is a dependently typed functional language developed by Norell at Chalmers University of Technology as his PhD Thesis. The current version of Agda is Agda 2.

1.6 Syntax

In Agda, Set is equal to type. In dependent type languages, it is possible to create a function that return a type.

```
bool→Set : (b : Bool) → Set
bool→Set b = if b then ℕ else Bool
```

Because of dependent types, it is possible to have a type that depend on the input

It is possible in Agda to do pattern match. So it breaks the input in their possible cases.

```
boolean→Set : (b : Boolean) → Set
boolean→Set true = ℕ
boolean→Set false = Bool
```

To create a new type with different pattern match, it is used data constructor.

```
data Boolean : Set where
  true : Boolean
  false : Boolean
```

Records are data types with just one case of pattern match

```
record Person : Set where
  constructor person
  field
    name : String
    age : ℕ

agePerson : (person : Person) → ℕ
agePerson (person name age) = age
```

Implicits terms are elements that the compiler is smart enough to deduce it. So it is not necessary to put it in argument of the function.

```
id : {A : Set} (x : A) → A
id x = x
```

Implicits arguments are inside . In case of the function id, the type of the input can be deduced by the compiler. For example, the only type that zero can be is Natural.

```
zeroℕ : ℕ
zeroℕ = id zero
```

1.7 Martin-Löf type theory

Agda is also a proof assistance based on intensional Martin-Löf type theory.

1.8 Types

In Martin-Löf type theory, there are 3 finites types and 5 types constructors. The 0 type contain 0 terms, it is called empty type and it is written bot.

The 1 type is the type with just 1 canonical term and it represents existence. It is called unit type and it is written top .

The 2 type contains 2 canonical terms. It represents a choice between two values.

The Boolean Type is defined using the Trivial type and the Either type

If statement is defined using booleans

1.9 Types Constructors

The sum-types contain an ordered pair. The second type can depend on the first type. It has the same meaning of exist.

```
data ∑ (A : Set) (B : A → Set) : Set where
  ⟨_,_⟩ : (x : A) → B x → ∑ A B

∑-elim : ∀ {A : Set} {B : A → Set} {C : Set}
  → (∀ x → B x → C)
  → ∑ A B
  ----------------
  → C
∑-elim f ⟨ x , y ⟩ = f x y
```

The pi-types contain functions. So given an input type, it will return an output type. It has the same meaning of a function

In Inductive types, it is a self-referential type. Naturals numbers are examples of that

```
data ℕ : Set where
  zero : ℕ
  suc : ℕ → ℕ
```

Other data structs like linked list of natural numbers, trees, graphs are too. Proofs in inductive types are made by induction.

```
ℕ-elim : (target : ℕ) (motive : (ℕ → Set)) (base : motive zero)
  (step : (n : ℕ) → motive n → motive (suc n) ) → motive target
```

```
ℕ-elim zero motive base step = base
ℕ-elim (suc target) motive base step = step target (ℕ-elim target motive base step)
```

Universe types are created to allow proofs written in all types. For example, the type of Nat is U0.

It looks like CoQ, but does not have tatics. Agda is a total language, so it is garanteed that the code always terminal and coverage all inputs. Agda needs it to be a consistent language.

Agda has inductive data types that are similar to algebric data types in non-depently typed programming language. The definition of Peano numbers in Agda:

```
data ℕ : Set where
  zero : ℕ
  suc : ℕ → ℕ
```

Definitions in Agda are done using induction. For example, the sum of two numbers in Agda:

```
_+'_ : ℕ → ℕ → ℕ
zero +' m = m
suc n +' m = suc (n + m)
```

In Agda, because of dependent types, it is possible to make some restrictions in types that is not possible in other language. For example, get the first element of a vector. For it, it is necessary to specify in the type that the vector should have at size greater or equal tha than one.

```
head : {A : Set} {n : ℕ} (vec : Vector A (suc n)) → A
head (x :: vec) = x
```

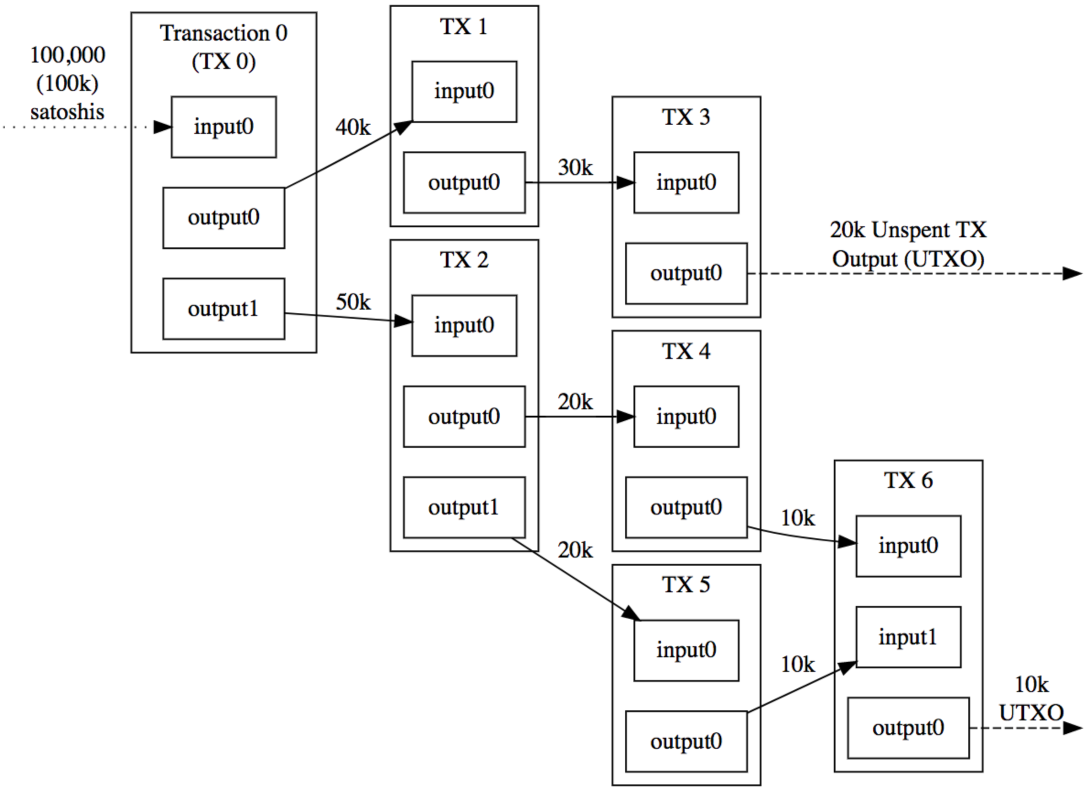
Another good example is that in sum of two matrices, they should have the same dimentions.

```
_+m_ : {m n : ℕ} (P Q : Matrix ℕ m n) → Matrix ℕ m n
[] +m [] = []
(vx :: P) +m (vy :: Q) = (vx +v vy) :: (P +m Q)
```

1.10 UTXO Bitcoin

There are two kinds of data structures to modeling accounts records and savings states. The UTXO model used in Bitcoin and the account model used in Ethereum.

In account model, it is saved the address and the balance of each address. For example, the data struct will look like this [(0xabc01, 1.01), (0xabc02, 2.02)]. So the address 0xabc01 has 1.01a of balance and the address 0xabc02 has 2.02 of balance. In this way, it is possible to easily know how much of balance each address has, but it is not possible to know how they got in this state.



Triple-Entry Bookkeeping (Transaction-To-Transaction Payments) As Used By Bitcoin

In UTXO model, each transaction is saved in the transaction tree. Every transaction is composed of multiples inputs and multiples outputs. But all inputs have to never been spent before.

Because of that, in UTXO model, it is easy to make a new transaction from previous one, but it is harder to know how much each one has. The wallet that calculate how much balance each address has.

In account model, there could be one kind of vulnerability that is less probabable to happen in UTXO model. Because there is an undesirable intermediary state that there is some address without balance while another has not already received his money.

For example:
bobBalance -= 1
Intermediary State
aliceBalance += 1

In account model, it is straight foward to know how much balance each address has.

In UTXO model, this calculation is made offchain. It can be a good thing, because each user has more privacy.

1.11 TXTree in Agda

2 Methods

2.1 Cripto Functions

The first thing that we define are the cripto functions that will be needed to make the cripto currency. Message can be defined in multiple ways, one array of bytes, one string or a natural number. Message in this context means some data.

Private key is a number, a secret that someone has. In Bitcoin, the private key is a 256-bit number. Private key is used to signed messages.

Public key is generated from private key. But getting private key from public key is impossible. To verify who signed a message with a private key, he has to show the public key.

Hash is an injection function (the probability of collision is very low). The function is used from a big domain to a small domain. For example, a hash of big file (some GBs) is an integer of just some bytes. It is very usefull to prove for example that 2 files are equal. If the hash of two files are equal, so the files are equal. It is used in torrents clients, so it is safe to download a program to untrusted peers, just have to verify if the hash of the file is equal to the hash of the file wanted.

These functions can be defined, but it is not the purpose of this theses. So they will be just postulates.

```
postulate _priv≡pub_ : PrivateKey → PublicKey → Set
postulate publicKey2Address : PublicKey → Address
postulate Signed : Msg → PublicKey → Signature → Set
postulate Signed? : (msg : Msg) (pk : PublicKey) (sig : Signature)
            → Dec $ Signed msg pk sig
postulate hashMsg : Msg → Hashed
postulate hash-inj : ∀ m n → hashMsg m ≡ hashMsg n → m ≡ n

record SignedWithSigPbk (msg : Msg)(address : Address) : Set where
  field
    publicKey  : PublicKey
```

```
pbkCorrect : publicKey2Address publicKey ≡ address
signature  : Signature
signed     : Signed msg publicKey signature
```

2.2 Transactions

In Bitcoin, there are some transactions. In each transactions, there are multiple inputs and outputs. Each input is named TXFieldWithId. The input of one transaction is the ouput of another transaction. Firsts outputs are generated from coinbase transaction (there is just one of this transaction at each block). Coinbase transactions are the miner reward.

```
data VectorOutput : (time : Time) (size : Nat) (amount : Amount) → Set where
  el : ∀ {time : Time}
    (tx : TXFieldWithId)
    (sameId : TXFieldWithId.time tx ≡ time)
    (elStart : TXFieldWithId.position tx ≡ zero)
    → VectorOutput time 1 (TXFieldWithId.amount tx)

  cons : ∀ {time : Time} {size : Nat} {amount : Amount}
    (listOutput : VectorOutput time size amount)
    (tx : TXFieldWithId)
    (sameId : TXFieldWithId.time tx ≡ time)
    (elStart : TXFieldWithId.position tx ≡ size)
    → VectorOutput time (suc size) (amount + TXFieldWithId.amount tx)
```

Vector output is the vector of outputs transactions. It is a non empty vector. In its representation, it is possible to know in what time it was created (time is the position of they in all transactions), what is his size (quantity of outputs fields) and the total amount spend in this transaction,

elStart is a proof that the position of TXFieldWithId is the last one. It is used after to specify wich input is in the transaction.

```
record TXSigned
  {time      : Time}
  {outSize   : Nat}
  {outAmount : Amount}
  (inputs    : List TXFieldWithId)
```

```
(outputs : VectorOutput time outSize outAmount) : Set where
field
  nonEmpty : NonNil inputs
  signed : All
    (λ input →
    SignedWithSigPbk (txEls→MsgVecOut input outputs)
                     (TXFieldWithId.address input))
                     inputs
  in≥out : txFieldList→TotalAmount inputs ≥n outAmount
```

A signed transaction is composed of a non empty list of inputs and outputs. For each input, there is a signature that confirms that he accepted every output in the list of outputs. And in the transaction, there is a proof that the total amount of money in all inputs are bigger than the total amount of outputs. The remainder will be used by the miner.

3 Conclusion

References

Back, A., et al. (2002). Hashcash-a denial of service counter-measure.

Chaum, D. (1983). Blind signatures for untraceable payments. In *Advances in cryptology* (pp. 199–203).

Dai, W. (1998). B-money. *Consulted*, *1*, 2012.

Hopwood, D., Bowe, S., Hornby, T., & Wilcox, N. (2016). Zcash protocol specification. *Tech. rep. 2016–1.10. Zerocoin Electric Coin Company, Tech. Rep.*.

Nakamoto, S., et al. (2008). Bitcoin: A peer-to-peer electronic cash system.

Noether, S. (2015). Ring signature confidential transactions for monero. *IACR Cryptology ePrint Archive*, *2015*, 1098.

Panurach, P. (1996). Money in electronic commerce: Digital cash, electronic fund transfer, and ecash. *Communications of the ACM*, *39*(6), 45–51.

Setzer, A. (2018). Modelling bitcoin in agda. *arXiv preprint arXiv:1804.06398*.

Wallace, B. (2011). The rise and fall of bitcoin. *Wired*, *19*(12).

Wood, G., et al. (2014). Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, *151*, 1–32.