

# Introduction to Cryptocurrencies

## Haskell and Cryptocurrencies

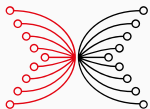
---

Dr. Lars Brünjes, IOG

Robertino Martinez, IOG

Karina Lopez, IOG

August, 2023



INPUT | OUTPUT

## Topics in this Lecture

---

- Blockchains & cryptocurrencies —
  - What are they?
  - How do they work?
- IOG in the cryptocurrency world
- Haskell at IOG

# What are blockchains?

---

## Short Definition of a Blockchain

A *blockchain* (...) is a distributed database that is used to maintain a continuously growing list of records, called *blocks*. Each block contains a timestamp and a link to a previous block.

---

*Wikipedia*

# Properties of Blockchains

Blockchains are...

# Properties of Blockchains

Blockchains are...

- “write-only” memory in the cloud
- decentralized
- a public ledger
  - for financial transactions
  - for university diplomas (GUNET,...)
  - for property rights (land, houses, cars,...)
  - ...

# What are Cryptocurrencies?

---



## Short Definition of a Cryptocurrency

A *cryptocurrency* (or *crypto currency*) is a digital asset designed to work as a medium of exchange using *cryptography* to secure the *transactions* and to control the *creation of additional units* of the currency.

---

*Andy Greenberg*

# Properties of Cryptocurrencies

- For us, a cryptocurrency is a blockchain specialized to financial transactions.
- So the ledger entries are *transactions*.
- Units of currency are associated with *public keys*.
- Everybody who knows the *private key* for a given key can control the money associated with that key.

# Beyond Generic Blockchains

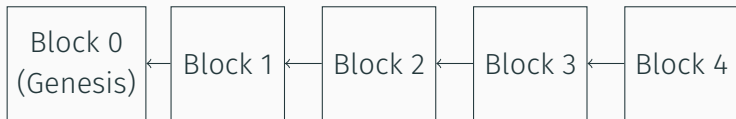
On top of the generic blockchain protocol, there are mechanisms for

- regulating creation of new coins,
- transaction fees and rewards for block creators,
- smart contracts,
- ...

# How Blockchains Work

---

# An Ideal Blockchain

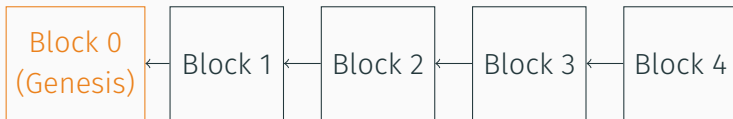


# An Ideal Blockchain



**Blocks** each contain a list of ledger entries (for example transactions) and reference the previous block, thus putting all entries in a linear order.

# An Ideal Blockchain



Blocks each contain a list of ledger entries (for example transactions) and reference the previous block, thus putting all entries in a linear order.

The first block is called the **Genesis Block**. It is publicly known and contains the initial state.

## Questions and Possible Attacks – Forgery and Manipulation

- How to prevent forgery of ledger entries?
- How to safeguard blocks against tempering?



# Cryptographic Hash Functions

A (cryptographically secure) *hash function* is a function  $H$  that takes arbitrary bitstrings to bitstrings of a fixed length with the following additional properties:

# Cryptographic Hash Functions

A (cryptographically secure) *hash function* is a function  $H$  that takes arbitrary bitstrings to bitstrings of a fixed length with the following additional properties:

- effectively computable
- collision free
- hiding
- (puzzle friendly)

# Digital Signatures

*A signature scheme* consists of the following three ingredients:

# Digital Signatures

A *signature scheme* consists of the following three ingredients:

- *Key Generation*: *genKeys*, given a desired key size, produces a pair of keys  $(pk, sk)$  of the desired lengths.
- *Signing*: given an arbitrary bitstring and a secret key  $sk$ , function *sign* returns a (fixed length) bitstring, the digital signature.
- *Verification*: Given an arbitrary bitstring, a public key  $pk$  and a signature, function *verify* checks whether the signature is valid.

# Digital Signatures

A *signature scheme* consists of the following three ingredients:

- *Key Generation*: *genKeys*, given a desired key size, produces a pair of keys  $(pk, sk)$  of the desired lengths.
- *Signing*: given an arbitrary bitstring and a secret key  $sk$ , function *sign* returns a (fixed length) bitstring, the digital signature.
- *Verification*: Given an arbitrary bitstring, a public key  $pk$  and a signature, function *verify* checks whether the signature is valid.

## Important

It should be infeasible to “guess” a valid signature for a given message and public key without knowing the private key!

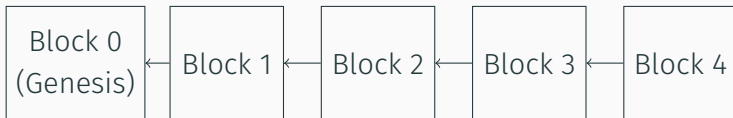
# Preventing Forgery with Dital Signatures

- Each ledger entry is digitally signed by somebody with the aproprate rights (for example the owner of the money being transferred).
- Each block is signed by the block creator.

# Preventing Blockchain Manipulation with Hashing

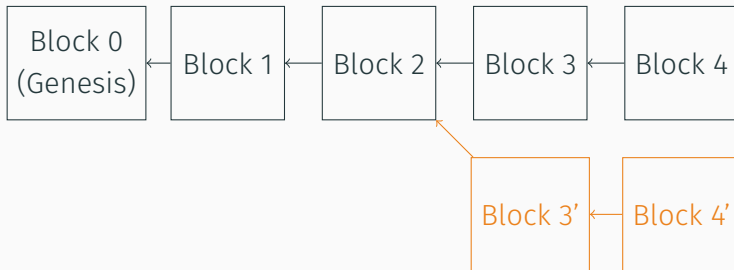
- Each block's link to the previous block contains/is the *hash* of the previous block.
- This means it is impossible to delete, insert or change blocks afterwards.

## Another Problem: Forking





## Another Problem: Forking



Nothing in the blockchain data structure enforces a *chain*. To prevent **forks**, something else is needed.

## Why Forks are a problem

- The *linear* ordering of all blockchain entries is essential.
- Forks destroy the linear order.
- In the context of cryptocurrencies, a fork means potential for “double spending”.

# Consensus Protocols

- Even in the absence of malicious players, forks cannot be prevented completely (network errors,...).
- A *Consensus Protocol* ensures that forks do not become too deep.
- *Common Prefix* should hold: After “throwing away” the last  $k$  blocks, each party has the same view of the blockchain (longest path in the tree).
- Idea: The right to create blocks is tied to some asset which “mostly” belongs to honest parties.
- For Bitcoin: *Proof-of-Work* (computing power).
- For Ada: *Proof-of-Stake*.

# How Cryptocurrencies Work

---

# Transaction Structure

*A transaction* contains the following data:

# Transaction Structure

A *transaction* contains the following data:

- A list of *inputs*, where each input references a previous transaction output.
- A list of outputs. Each output consists of
  - an amount of coins,
  - a receiver (often a public key).
- For each input, a *signature* of the transaction by somebody in control of the referenced output.

# Transaction Structure

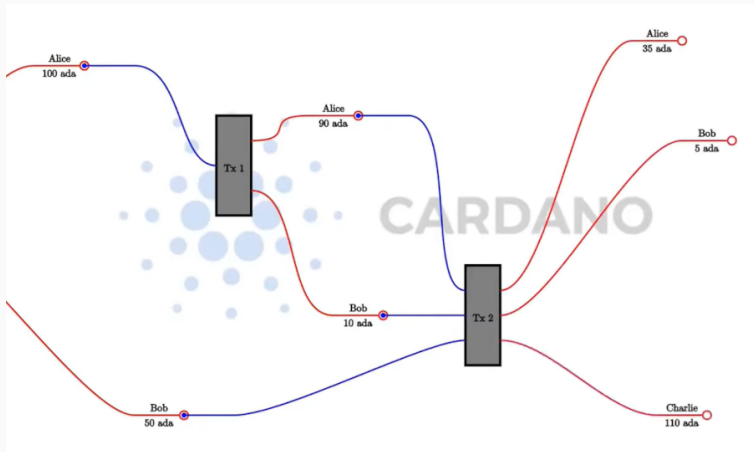
A *transaction* contains the following data:

- A list of *inputs*, where each input references a previous transaction output.
- A list of outputs. Each output consists of
  - an amount of coins,
  - a receiver (often a public key).
- For each input, a *signature* of the transaction by somebody in control of the referenced output.

## Note

Inputs are always used up completely.

# Transaction Structure (Continued)





# Valid Transactions

A transaction is *valid* if:

# Valid Transactions

A transaction is *valid* if:

- All input signatures can be verified.
- The sum of input amounts is at least as big as the sum of output amounts.

## Valid Block

A block, containing a list of transactions, is *valid* if:

# Valid Block

A block, containing a list of transactions, is *valid* if:

- The block creator had the right to create a block at that time (details depend on the consensus protocol).
- The signature of the block creator can be verified.
- All transactions in the block are valid.
- All transactions in the block are consistent: With respect to the linear ordering of transactions in the blockchain, all inputs have not been spent before.

# Valid Block

A block, containing a list of transactions, is *valid* if:

- The block creator had the right to create a block at that time (details depend on the consensus protocol).
- The signature of the block creator can be verified.
- All transactions in the block are valid.
- All transactions in the block are consistent: With respect to the linear ordering of transactions in the blockchain, all inputs have not been spent before.

## Note

The first two items above hold for all blockchains, the last two are specific to cryptocurrencies.

# Fees and Incentives

- If the sum of input values of a transaction exceeds the sum of output values, the difference is considered as *transaction fees*.
- Whether fees are obligatory depends on the individual cryptocurrency.
- In Bitcoin, the block creator gets the fees.
- In Bitcoin, the block creator also gets a reward for each block, which is implemented as a special “coin base” transaction.

# IOG and Cryptocurrencies

---

# IOG Role & Philosophy

- IOG is a “factory for cryptocurrencies”.
- Works on *Ethereum Classic* (in Scala) and *Ada* (in Haskell).
- Employs both academic researches and software developers.
- Is committed to *best practices*, both in academia (peer reviewed papers,...) and software development.
- Strives to be as rigorous as possible (mathematical proofs, formal verification,...).
- Develops a “toolbox” for cryptocurrencies, so that different consensus protocols, incentive schemes and other ingredients can be combined easily.
- Wants to get things right. — No cutting corners!



# Why Haskell?

---

# Why Haskell?

Haskell...

# Why Haskell?

Haskell...

- has an extremely high level of abstraction,
- is very expressive and terse,
- is perfect for writing DSL's (protocols,...),
- lends itself to formal verification, due to its declarative/functional nature.
- is statically typed with a sophisticated type system,

# Why Haskell?

Haskell...

- has an extremely high level of abstraction,
- is very expressive and terse,
- is perfect for writing DSL's (protocols,...),
- lends itself to formal verification, due to its declarative/functional nature.
- is statically typed with a sophisticated type system,
- is *fun*!