Blockchain and Distributed Ledger

Introduction

- Black box cryptography
- Currencies and cryptocurrencies
- Decentralization vs Distributed systems
- Web3
- Do you need a blockchain?

Black box cryptography

Definition

- Cryptography
  - from Ancient Greek **κρυπτός** (kryptós,"hidden, secret") and **γράφειν** (graphein, "to write")
  - techniques for **secure communication** in the presence of **adversaries**
- It is about protocols that **prevent third parties** from **reading private messages**
  - aspects such as **confidentiality**, **integrity**, **authentication** and **non-repudiation** are central to modern cryptography

Secure communication

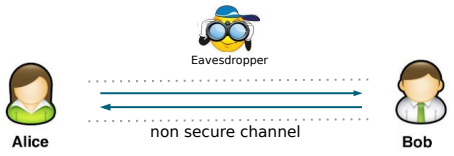
Secure communication

Alice and Bod could use a **secret algorithm** to **hide their communication**...

Secure communication

...but they use a **known algorithm**, and **hide the parameters**, e.g. the **keys**

Secure communication



The simplest algorithm is a **substitution cipher**, in which the letters or words in the message are substituted for others, based on a code shared in advance between the parties

Example: Caesar Cipher, or ROT 13, where the **key is n=13**

Two main methods

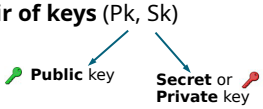
- 1. Symmetric cryptography
- 2. Asymmetric cryptography

Symmetric cryptography

- The **same key** is used to **encrypt** and **decrypt**
- It is **shared** by Alice and Bob
- They must **exchange** the key **in a secure way**
  - It might be difficult and costly
- Analogy: Alice and Bob have a **lock** and one copy each of **a key that opens/closes it**
  - Alice just puts her message in a box, passes it to Bob, only he can open the box

Asymmetric cryptography

- One key for encryption
- Another key for decryption
- The two keys must match, e.g., they are mathematically related
- Each user has a **pair of keys** (Pk, Sk)



Asymmetric cryptography

- **Both keys are required to perform an operation**
  - Data encrypted with the private key is deciphered with the public key
  - Data encrypted with the public key is deciphered with the private key
- Asymmetric cryptography is **often used to exchange the secret key** to prepare for using **symmetric cryptography** to encrypt data
- Encrypting data with the private key creates a **digital signature**

Analogy

- A **special lock** that is closed by one key and opened by another one
- **Encryption:** Alice makes the "closing key" public (everybody has a copy) and keeps the "opening key"
  - Bob puts the message in a box, closes it with the public key
  - Only Alice can open the box
- **Signature:** Alice makes the "opening key" public
  - She puts the message she sent in a box
  - If Bob can open the lock with Alice's "opening" key, the message must come from her

Asymmetric cryptography

- Alice has a pair of keys (Pk_A , Sk_A)

Asymmetric cryptography

- Alice has a pair of keys (Pk_A , Sk_A)
  - Bob sends a message M to Alice, and uses her public key

M ----------Pk_A----------> C (**encrypt**)

Asymmetric cryptography

- Alice has a pair of keys (Pk_A , Sk_A)
  - Bob sends a message M to Alice, and uses her public key

M ----------Pk_A----------> C (**encrypt**)

  - Alice receives the ciphertext C, and uses her secret key

C ----------Sk_A----------> M (**decrypt**)

Asymmetric cryptography

- Alice has a pair of keys (Pk_A , Sk_A)
  - Bob sends a message M to Alice, and uses her public key
    M ----------Pk_A----------→ C (**encrypt**)
  - Alice receives the ciphertext C, and uses her secret key
    C ----------Sk_A----------→ M (**decrypt**)
- This guarantees **confidentiality**

Asymmetric cryptography

- **Everyone knows the public key and the algorithm**, but without the private key it is not possible to decrypt
- The algorithm uses a **one-way** function which is easy to compute, but difficult to invert
  - Examples
    - List of phone numbers
      - Easy to find a phone number for a given name (if names are sorted)
      - Given a phone number, difficult to find the corresponding name
    - Given two prime numbers, p and q
      - **Easy** to compute m = p*q, even for large values of p and q (**multiplication**)
      - Given m, **difficult** to find p and q (**factorization**)

Prime numbers

- Multiply is easy 19 * 23 ?
- Factorize is difficult 323 ? (semiprime)
- With large numbers is very difficult also for computers
- The algorithms to compute keys use large prime numbers, modular arithmetic, exponential and logarithmic operations…

Hash Functions

- Hash come from *hasher*, cut in small pieces, e.g. chop
  - Good representation of data
  - One-way
- Example in real life


  - From vegetables you can prepare a soup (**easy**)
  - From a soup is **difficult** to obtain the vegetables back
- Hash functions provide a way to map **data of arbitrary size** to **fixed-size values**

Hash Functions


Message M of **arbitrary** length

h()

00100111010101 Digest h of **fixed** length

Hash Functions


mobydick.txt

sha256sum

4dffc0bf542de6751f3918787fdd5afadce51c3fdff062ab70349d0834a56395

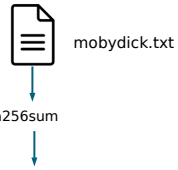
Properties of h()

- 1. **Deterministic**
- 2. **Quick** to compute the hash
- 3. **One-way** (also called pre-image resistant)
  - Given y it is computationally infeasible to find m' such that h(m') = y
- 4. **Collision resistance**
  - It is computationally infeasible to find two messages m, m' such that h(m) = h(m')

Properties of h()

- 5. With **minor modifications** in the input, **h() changes totally**, e.g., you cannot "adjust" hashes
- Notice that
  - A big book with hundreds of pages can be hashed. By changing only one letter in one page, the new hash is totally different
  - It is possible to hash any kind of data (image, text, video); from the hash value it is not possible to recover the type of the original document
  - Hashes are similar to **fingerprints** in real life: from a fingerprint you **cannot recover information** of the person, you need to **compare** the fingerprint with others

Hash Functions



4dffc0bf542de6751f3918787fdd5afadce51c3fdff062ab70349d0834a56395

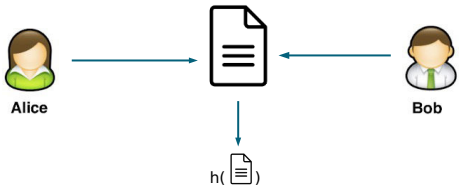
Adding one "." in the file

1eb64b7c370af468e85d320cd915a6aa71aa5d3eed6df32fd697c8111a3994e5

Example

Alice and Bod want to **share a contract** and they want to be sure that **none of them changes the contract**

Example



They can compute the **hash of the contract** and store it in a safe place. If there are changes in the contract, by applying again the hash function a different result is obtained (so it is possible to check)

Uses of h()

- **Integrity checks**
  - modifications on documents/messages can be malicious or can occur because of errors (data loss over the network)
- **Digital signatures**
  - combined use of asymmetric cryptography and hash functions

Digital signature

- A digital signature is a **sequence of bits** associated to a message to **authenticate the person who signed it**
- Bob receives a message from Alice and he wants to be sure that the message comes from her
- **Alice** is the **signer** and she has a pair of keys (Pk_A, Sk_A)
  - Alice will use the **secret key Sk_A** to **sign**
  - Bob can **verify the signature** with **Pk_A**

Digital signature

- **Alice** has a **message M** and computes **h(M)**
- To **sign** she applies **Sk_A** to h(M) and produces **sign(M)**
- She sends the signed message, e.g. pair **(M,sign(M))**

Digital signature

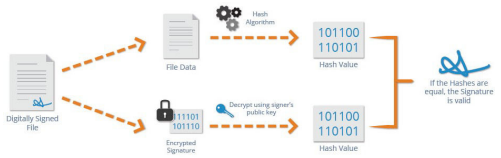
- **Alice** has a **message M** and computes **h(M)**
- To **sign** she applies **Sk_A** to h(M) and produces **sign(M)**
- She sends the signed message, e.g. pair **(M,sign(M))**

Digital signature

- **Bob** receives the signed message **(M,sign(M))**
- Splits the message **M** from the signature **sign(M)**
- **Extracts h(M)** from the signature using **PK_A**
- **Computes his version of h(M)** from the message M just received and checks if they are equal

Digital signature

- **Bob** receives the signed message **(M,sign(M))**
- Splits the message **M** from the signature **sign(M)**
- **Extracts h(M)** from the signature using **PK_A**



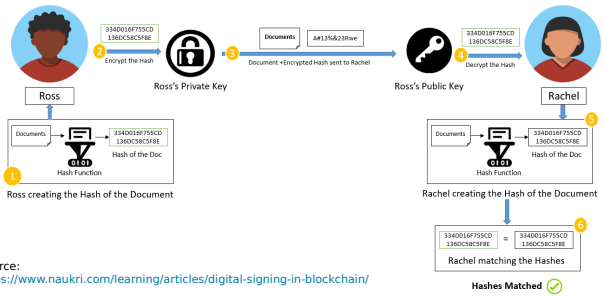
Digital signature

- If the two hashes are different
  - there is a problem with the identity of the sender (Alice)
  - or the message has been altered
- and the message is discarded

Digital signature

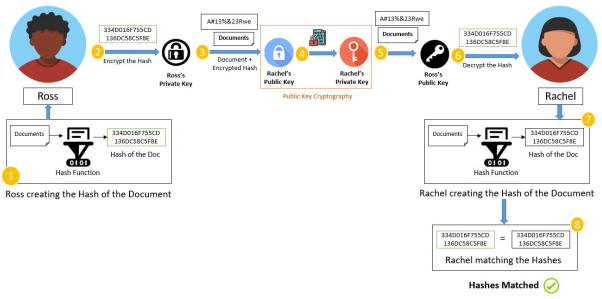
- 1. **Authentication**
  - it is possible to verify the identity of the sender using their public key
- 2. **Non repudiation**
  - the sender can not say they did not send a message signed with their private key
- 3. **Integrity**
  - modifications in the message invalidate the hash

Problem: anyone can read

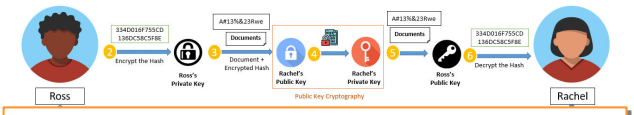


Source:
https://www.naukri.com/learning/articles/digital-signing-in-blockchain/

Solution: use $PK_{receiver}$



Solution: use $PK_{receiver}$



As we will see, **digital signatures** in blockchains (and elsewhere) are used to **authenticate transactions**

The user has to prove to the network that they are authorized to issue transactions and spend their balance

Hashes Matched

Currencies and cryptocurrencies

Currencies

- **Gift economy**
  - Valuables were not sold, but rather given without an explicit agreement for immediate or future rewards
- **Barter**
  - System of exchange in which participants in a transaction directly exchange goods or services for other goods or services
  - Reciprocal exchange, not delayed in time

Currencies

- **Commodity money**
  - Derived from the commodity out of which the money is made

  - It has an **intrinsic value** and it was a convenient form of trade in comparison with the barter system

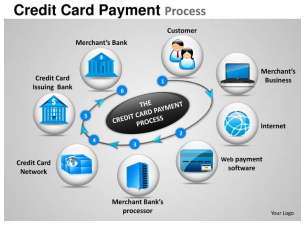
Currencies

- **Fiat money**
  - Type of money that is **issued and regulated by the government**. Its most important feature is that it **has no intrinsic value of its own**, it holds value only because the government issues, maintains, and regulates it

FIAT MONEY VS COMMODITY MONEY

Currencies

- Having fiat-based currency requires a lot of third-party consensus to try to avoid frauds in the system
- For example



Ledger

- A **ledger** is a **record for economic transactions** that includes cash, accounts receivable, inventory, accounts payable, debt, costs, salaries, expenses, …
- It is the **primary record used by banks and other financial institutions to reconcile book balances**
- The financial statements of banks, financial institutions, and enterprises are compiled using ledger accounts

Ledger

- For financial transactions with fiat currency, **third-party trust systems** (VISA, MasterCard, banks) **maintain information** about every transaction on their ledgers
- Blockchain has changed the landscape making everyone part of the ledger…

Cryptocurrencies

- A cryptocurrency is a **digital currency**
- Digital means that there is nothing physical
  - It is possible to **pay** for objects and **exchange** with other crypto or fiat currencies
- Cryptocurrecies are **decentralized**, e.g. there are no central authorities
- **Maths replaces banks**
  - Instead of trusting some "entity", you **trust cryptography**

Cryptocurrencies

- Other cryptocurrencies (who did not survive) were studied before Bitcoin
- Also PayPal was initially introduced as a cryptocurrency, but then it became the payment system we know today
- Suppose we want to create a cryptocurrency

Create a new coin

- Steps
  - Create new coins
  - Transfer coins

Create a new coin

- Steps
  - Create new coins
  - Transfer coins

Create a coin, e.g., a file

Create a new coin

- Steps
  - Create new coins
  - Transfer coins

Send coin to a friend Tom

Create a new coin

- Steps
  - Create new coins
  - Transfer coins

Send coin to a friend Tom

Is this a good model?

Create a new coin

- Steps
  - Create new coins
  - Transfer coins

Send coin to a friend Tom

Am I really the sender?

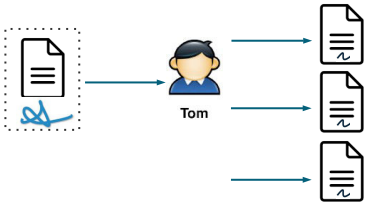
Create a new coin

- Steps
  - Create new coins
  - Transfer **signed** coins

Send **signed** coin to a friend Tom

Create a new coin

- Now suppose Tom cheats



Tom **spends** the **same coin multiple times**

How can we avoid this?

Create a new coin

- To avoid double spending **all actors** in the system must **communicate**
- "I am **Alice** and I gave **1 coin to Tom today**"

sender

amount

receiver

when

Create a new coin

- To avoid double spending **all actors** in the system must **communicate**
- "I am **Alice** and I gave **1 coin to Tom today**"

sender

amount

receiver

when

Problem: being in a network, messages can arrive late

Create a new coin

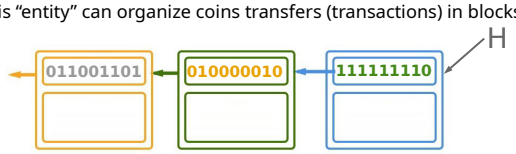
- We need to add some **other piece** of **cryptography**
  - All actors in the network can **hash** the **history** of the transactions
  - If **two actors know a different history**, theirs **hashes are different**, and they can check what happened
- Unfortunately, even with this solution, point-to-point communication is not realistic…

Create a new coin

- We can replace real identities with **digital identities**
  - **Public key** can be used, but it is too long
  - **Hash** the public key and get the **Address**
- We also need an **initial issuer** of the coins
- And we need some "entity" keeping track of the history

Create a new coin

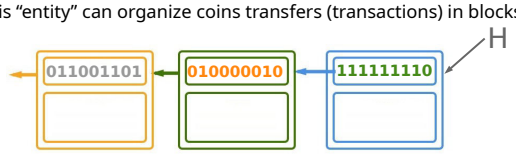
- This "entity" can organize coins transfers (transactions) in blocks



- Blocks are connected via **hash pointers**
  - Data structure that contains the **address** of a given information (a block in this case) and the **hash** of the information so that it is possible **to check the integrity of the information**

Create a new coin

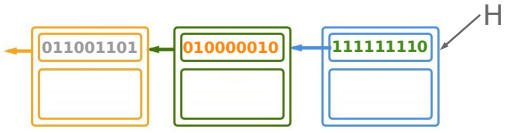
- This "entity" can organize coins transfers (transactions) in blocks



- Blocks contains issues of new coins and valid payments
- This is an **abstract model of a blockchain!**

Create a new coin

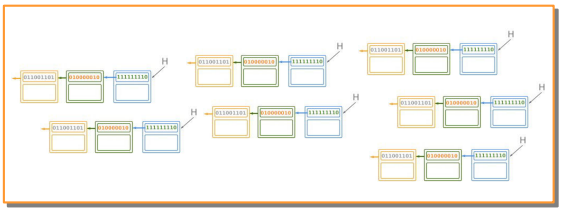
- This "entity" can organize coins transfers (transactions) in blocks



- A **problem** of this model is that the "entity" is too powerful and it is also a single point of failure
- Also, we need to trust the "entity"
- We will see how this was solved in the Bitcoin blockchain

Create a new coin

- **Multiple "entities"** share the same state of the system



Decentralized vs Distributed Systems

Distributed systems

- A **distributed system** is one in which the application and its architecture are distributed over a large number of machines, and possibly physical locations
- This means that multiple computers must **coordinate** to achieve the goals of the overall application

Distributed systems

- The **advantages** of distributed systems are many
  - **Resiliency**, e.g., the ability to adapt and keep working in presence of changes
  - **Redundancy**, e.g., each part of the system can be built to have backups so that if it fails, another copy can be used
  - **Parallelism**, e.g., work can be divided efficiently so that many less expensive computers can be used instead of a single (very expensive) fast computer

Decentralized systems

- All **decentralized systems must be distributed**, but distributed systems are not necessarily decentralized
- The difference has to do with location and redundancy versus **control**
- Metaverse, X, YouTube
  - Highly distributed services, with servers and data centers worldwide. They are distributed, with fault tolerance, extensive coordination, redundancy, and so on

Decentralized systems

- But
  - These services are still "centralized" because, with no input from other stakeholders, **they can change the rules**
  - Platforms others depend on, but with no reciprocity
- The last decade has seen the growth of many highly distributed, yet highly centralized companies, such as AirBnB, Uber, BlaBlaCar, and others

Decentralized systems

- In a decentralized system, there is no super powerful stakeholder with the ability to make and enforce rules without the permission of other network users
- To judge how decentralized a system is, the are several factors to consider

Decentralized systems

- **Open access**
  - At the beginning Internet was seen as revolutionary in part because of its open access: anyone can join
- **No hierarchy**
  - If each member in the system has identical power, control can be exercised through influence or reputation

Decentralized systems

- **Diversity**
  - A diverse system stands in opposition to monoculture, as was for instance Windows for a long time and today is probably Google for web searches
- **Transparency of operation**
  - If some actors have information dominance, e.g., access to greater information, they can centralize the system

Decentralized systems

- Decentralized system have their downsides
- **Speed**
  - Depending on the type of event, decentralized systems can be slower
  - The Bitcoin blockchain can handle approximately 7 transactions a second; VISA and MasterCard are distributed (but not decentralized) transaction-handling systems that can handle more than 50000 transactions a second

Decentralized systems

- Decentralized system have their downsides
- **Censorship resistance**
  - Decentralized systems tend to be much harder to censor because of a lack of central authority
  - This is not necessary a disadvantage but some information (child pornography, hate speech, bomb making instructions) is considered dangerous and immoral and should be censored
  - This is not always possible (for example it is not possible with blockchain)

Decentralized systems

- Decentralized system have their downsides
- **Chaos**
  - Decentralized systems tend to be more chaotic than centralized ones by their nature
  - Each actor works according to their own desires and not in response to an authority
- For all these reasons, decentralized systems are difficult to predict

Decentralized systems

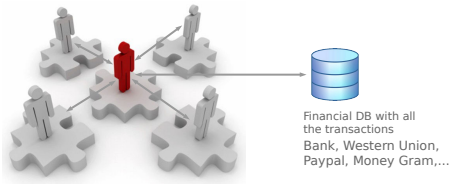
- Blockchain is the base "ingredient" to foster decentralized (**Web3**) applications
- Enables the **Internet of Value**, e.g., an online space where people can instantly transfer value between each other, eliminating the need for the middlemen and most costs
- https://gatehub.net/blog/what-is-the-internet-of-value/

Do you need a blockchain?

Paper available at https://ieeexplore.ieee.org/document/8525392

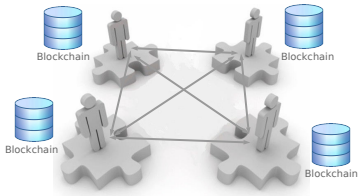
Do you need a blockchain?

- Payments **before** Bitcoin and its blockchain



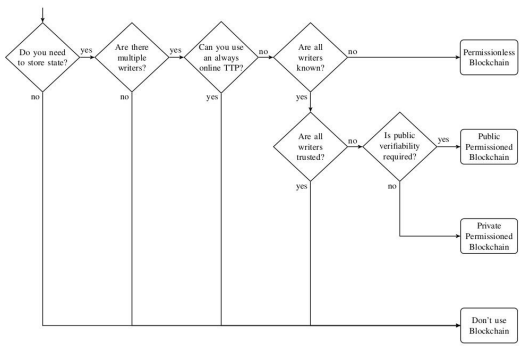
Do you need a blockchain?

- Payments **after** Bitcoin and its blockchain



Do you need a blockchain?

- **Payments** without a central authority are the **first killer application** of the blockchain
- But after that?
  - New proposals appeared and many start-ups were created around the blockchain ecosystem
  - But in many cases a (distributed) database could be enough



Suggested reading: https://ieeexplore.ieee.org/document/8525392

Decentralized Systems

Bitcoin

Huge expectations…

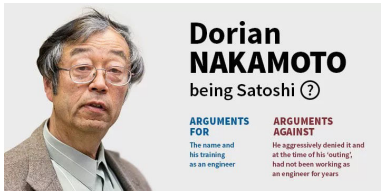
As the Internet transformed and commoditized how society communicates, blockchain will transform and commoditize how society agrees, trusts, and exchange values. How did this begin?

2008 White Paper

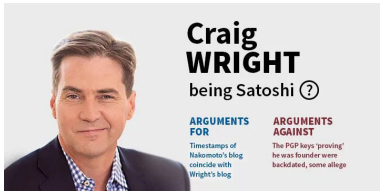
Bitcoin: A Peer-to-Peer Electronic Cash System

By: Satoshi Nakamoto

Who is Satoshi?



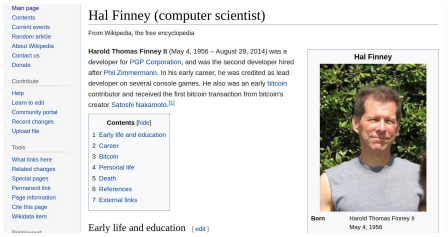
Who is Satoshi?



Who is Satoshi?

https://cointelegraph.com/news/from-dorian-nakamoto-to-elon-musk-the-incomplete-list-of-people-speculated-to-be-satoshi-nakamoto

Who is Satoshi?



What is Blockchain?

- Introduced in 2008/2009 by **Satoshi Nakamoto**
- **Replicated linked list of blocks** that are **secured through cryptography**
- This list is **maintained by a large number of nodes** to tolerate malicious behaviors of a small group of nodes
- It is also called **Distributed Ledger**

Different types of blockchains

Different types of blockchains

Open and decentralized
Any peer (node) can join and leave the network as writer or reader at any time

No central “entity”

The content (e.g., the transactions) is readable to any peer

Different types of blockchains

Only authorize a limited set of readers and writers

A central entity decides and attributes rights to peers

Can be public or private

What is Bitcoin?

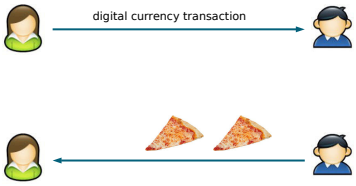
- Bitcoin (BTC) is the most famous and used cryptocurrency
- **First** open-source cryptocurrency
- More **gold** than coins

What is Bitcoin?

- Bitcoin is a worldwide payment system that **does not require intermediaries or banks**

Pizza for Bitcoin

- (May 2010, 2 pizzas paid 10.000 BTC)



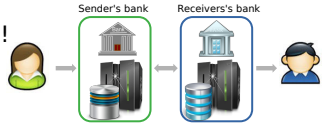
https://www.youtube.com/watch?v=A_3kHupuuxQ

Pizza for Bitcoin, risks

- Payment requires the use of hardware/software devices
  - bartender and customer are **not able to visually check the monetary transaction**
- In addition, currency virtualization introduces the problem of **double spending**
  - the owner of a digital money could make a copy of it and try to spend it two (or more) times

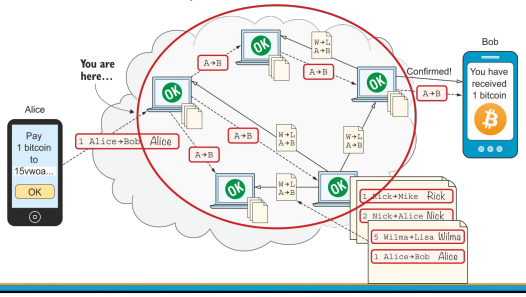
In case of bank transfer

- The transaction is managed by **a few trusted intermediaries**
  - Banks
  - Financial agencies
  - Lenders
- We need to trust banks!



Bitcoin ecosystem

- No intermediaries, but **nodes** in a **P2P network**



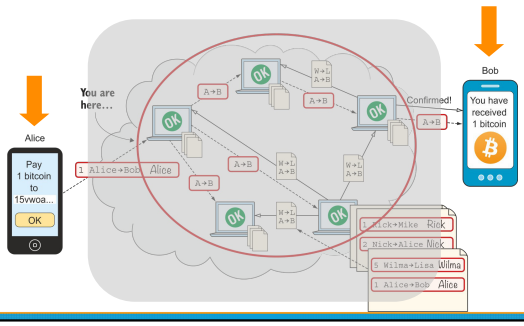
Nodes (or peers)

- Peers are **heterogeneous**, presenting difference in hardware and software
- Many peers **store data about the blockchain**
  - **Full blockchain**
  - **Pruned blockchain** (up-to-date version of the blockchain up to a few days, plus metadata about all known blocks and unspent transactions)
  - **Simplified payments verification** (SPV) clients (up-to-date version of blockchain headers, usually deployed in mobile devices)

Nodes (or peers)

- Peers can be classified for their **functionalities**
  - Wallet
  - Miner
  - Validation and relaying
  - Others: DNS, block explorer, exchange service,..
- A peer may perform more functionalities at the same time

Wallet



Wallet

- To become a node of type wallet we need
  - an Internet connection
  - a Bitcoin wallet (software program or hardware)
- A Bitcoin wallet can be compared to the **user's bank account** and it is thus necessary for sending and receiving Bitcoins
- It is identified by **two mathematically-connected keys**
  - a private key
  - a public key

Private key

- The **secret key (Sk)** is a **256-bit** randomly generated number which must meet a specific format

Private key

- The **secret key (Sk)** is a **256-bit** randomly generated number which must meet a specific format

64 characters in the range 0-9 or A-F

Private key

- The **secret key (Sk)** is just a **big random number** in **hexadecimal format**

ef235aacf90d9f4aadd8c92e4b2562e1
d9eb97f0df9ba3b508258739cb013db2

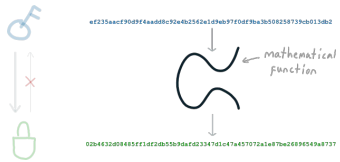
- It is used to **authorize Bitcoin transfers** and it can be thought of as a **password** or the **secret PIN** of a card associated with the bank account

Public key

- The **public key (Pk)**
  - is **calculated from the private key**, by means of an algorithm established by the Bitcoin protocol
  - is a **public data** and it can be thought of as the public information of a bank account
- The private key should not be easily recoverable from the public key
- Pk is used for **transaction verification** and **validity check**

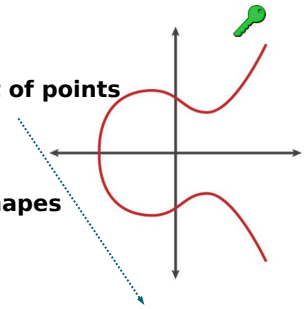
Public key

- The Bitcoin protocol uses **elliptic curve cryptography** (ECC) to compute keys for digital signatures (ECDSA)



ECC

- An elliptic curve is the **set of points** described by the equation
- Depending on the values of a and b, elliptic curves may assume **different shapes**



$4a^3 + 27b^2 \neq 0$

Condition on a and b

$y^2 = x^3 + ax + b$

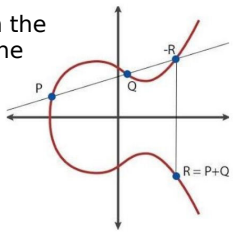
Equation of a generic elliptic curve

ECC

- Elliptic curves are **symmetric on the x-axis**
- Form a **group**, e.g., the elements of the group are all the points on the curve
- The **inverse** of a point is the one symmetric on the x-axis

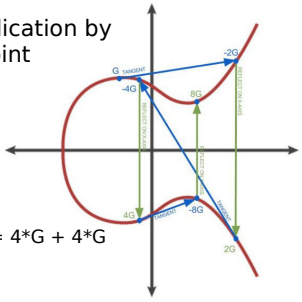
ECC: sum

- The sum of two points on the curve is a third point of the curve
- P + Q = ?
  - Find -R
  - Reflect on the x axis



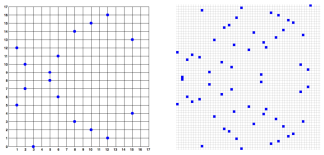
ECC: scalar multiplication

- Given a point G, its multiplication by a scalar value returns a point on the curve
- 2*G = G + G
  - Take the **tangent** in G
  - Reflect on x axis
- Same for 4*G= 2*G + 2*G, 8*G= 4*G + 4*G



ECC: scalar multiplication

- Given a known **base point G**, it can be multiplied by a private key **Sk** to find the corresponding public key **Pk**
- **Modular arithmetic**
  - **Pk** = (**Sk** * **G**) mod **p** (with p prime number)



https://www.youtube.com/watch?v=qCafMW4OG7s

Bitcoin Secp256k1

- Specific elliptic curve: $y^2 = (x^3 + 7) \bmod p$
  - **a** = 0, **b** = 7
  - **p** = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFC2F

    $= 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$
  - **G** = (0x79be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798, 0x483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08f5b166e6c)
  - **Sk** is 256 bit
- How secure is 256 bit security? https://www.youtube.com/watch?v=S9JGmA5_unY

Bitcoin address

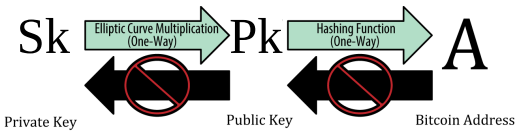
- The **Bitcoin address A** is a sequence of digits and letters computed from the public key* with 2 hash functions

  A = Base58Check(**RIPEMD160**(**SHA256**(Pk)))
  - it is encoded in **Base58**, to avoid I,l,O,0 https://tools.ietf.org/id/draft-msporny-base58-01.html
  - it starts with **1** (legacy), **3** (multi-signature transactions) or **bc1** (SegWit transactions, using Bench32 encoding)
- It serves as **identifier of the wallet**
  - it can be seen as the **IBAN**, needs to **be shared** for receiving payments

* things are more complex

Wallet

- Any Bitcoin wallet is associated and identified with (at least) **three numbers**



Source: https://www.oreilly.com/library/view/mastering-bitcoin-2nd/9781491954379/ch04.html

Wallet

- If the **private key Sk** of a Bitcoin wallet **is lost**, it cannot be recovered and the crypto associated with that private key are lost as well!
- And this happens...

https://news.bitcoin.com/analyst-1500-bitcoins-lost-every-day-less-than-14-million-coins-will-ever-circulate/

Wallet

- Stores keys, not Bitcoin!
  - Bitcoins are transactions on the blockchain
- It can be
  - **Custodial**, e.g., provided by third parties (exchanges) which have control over private keys (and therefore, over crypto)
  - **Non custodial**, e.g., the owner is the only one with access to their private keys, and therefore, has complete control over their assets

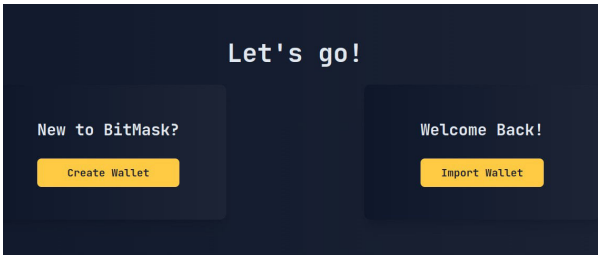
Pros	Cons
Less responsibility held by users	Private keys are controlled by third party
Simple and easy to use for beginners	Custodial wallets are vulnerable to hackers
Can reset password to regain access to digital assets	KYC and AML verification for account creation
	Less advanced features available for experienced crypto users

Custodial

Pros	Cons
You control your keys	Impossible to recover digital assets if users lose private keys and/or recovery phrases
Fast and easy to create new wallets	More technical knowhow needed to use advanced features
Funds won't be impacted in cases of exchange hacks	
No KYC or AML process necessary for creating/storing	
More advanced functions and features available than custodial services	

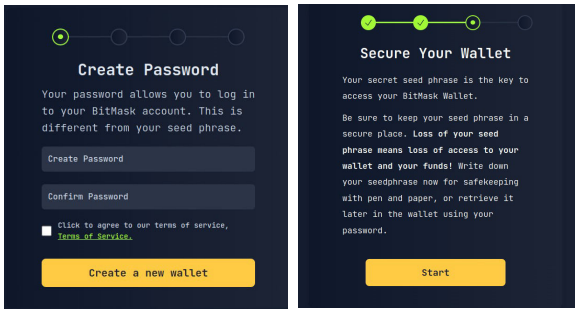
Non custodial

BitMask

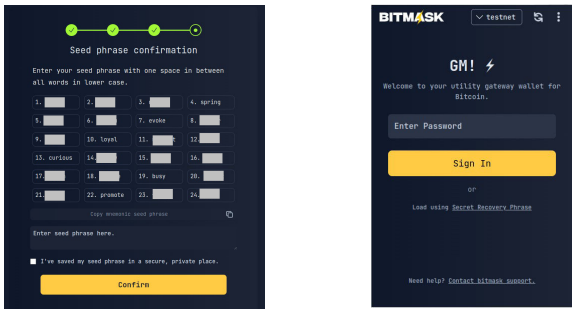


Note: We will use **MetaMask** (Ethereum wallet)

BitMask

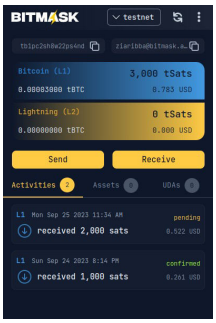


BitMask



Faucet

- Website or application to get fake cryptocurrency for testnet https://bitcoinfaucet.uo1.net/
- Sometimes they award registered users with small amounts of cryptocurrency in exchange for completing a simple task such as viewing an ad or participating in a short survey



Seed phrase

- Created using a specific algorithm (BIP 39) which converts a **randomly generated number** (**entropy**) into a **sequence of words** from a predetermined word list
  - **128 bits** of entropy → **12 words**
  - **256 bits** of entropy → **24 words**
- Seed phrases enable users to regenerate all private keys associated with their wallets

COMPLETE LIST OF BIP-39 SEED RECOVERY PHRASE WORDS FOR WALLETS

abandon ability able about above absent absorb abstract absurd abuse access accident account accuse achieve acid acoustic acquire across act action actor actress actual adapt add addict address adjust admit adult advance advice aerobic affair afford afraid again age agent agree ahead aim air airport aisle alarm album alcohol alert alien all alley allow almost alone alpha already also alter always amateur amazing among amount amused analyst anchor ancient anger angle angry animal ankle announce annual another answer antenna antique anxiety any apart apology appear apple approve april arch arctic area arena argue arm armed armor army around arrange arrest arrive arrow art artefact artist artwork ask aspect assault asset assist assume asthma athlete atom attack attend attitude attract auction audit august aunt author auto autumn average avocado avoid awake aware away awesome awful awkward axis baby bachelor bacon badge bag balance balcony ball bamboo banana banner bar barely bargain barrel base basic basket battle beach bean beauty because become beef before begin behave behind believe below belt bench benefit best betray better between beyond bicycle bid bike bind biology bird birth bitter black blade blame blanket blast bleak bless blind blood blossom blouse blue blur blush board boat body boil bomb bone bonus book boost border boring borrow boss bottom bounce box boy bracket brain brand brass brave bread breeze brick bridge brief bright bring brisk broccoli broken bronze broom brother brown brush bubble buddy budget buffalo build bulb bulk bullet bundle bunker burden burger burst bus business busy butter buyer buzz cabbage cabin cable cactus cage cake call calm camera camp can canal cancel candy cannon canoe canvas canyon capable capital captain car carbon card cargo carpet carry cart case cash casino castle casual cat catalog catch category cattle caught cause caution cave ceiling celery cement census century cereal certain chair chalk champion change chaos chapter charge chase chat cheap check cheese chef cherry chest chicken chief child chimney choice choose chronic chuckle chunk churn cigar cinnamon circle citizen city civil claim clap clarify claw clay clean clerk clever click client cliff climb clinic clip clock close cloth cloud clown club clump cluster clutch coach coast coconut code coffee coil coin collect color column combine come comfort comic common company concert conduct confirm congress connect consider control convince cook cool copper copy coral core corn correct cost cotton couch country couple course cousin cover coyote crack cradle craft cram crane crash crater crawl crazy cream credit creek crew cricket crime crisp critic crop cross crouch crowd crucial cruel cruise crumble crunch crush cry crystal cube culture cup cupboard curious current curtain curve cushion custom cute cycle dad damage damp dance danger daring dash daughter dawn day deal debate debris decade december decide decline decorate decrease deer defense define defy degree delay deliver demand demise denial dentist deny depart depend deposit depth deputy derive describe desert design desk despair destroy detail detect develop device devote diagram dial diamond diary dice diesel diet differ digital dignity dilemma dinner dinosaur direct dirt disagree discover disease dish dismiss disorder display distance divert divide divorce dizzy doctor document dog doll dolphin domain donate donkey donor door dose double dove draft dragon drama drastic draw dream dress drift drill drink drip drive drop drum dry duck dumb dune during dust dutch duty dwarf dynamic eager eagle early earn earth easily east easy echo ecology economy edge edit educate effort egg eight either elbow elder electric elegant element elephant elevator elite else embark embody embrace emerge emotion employ empower empty enable enact end endless endorse enemy energy enforce engage engine enhance

Abandon is 0001.

Ability is 0002.

Able is 0003 etc. etc.

Seed phrase

- **Start** from the **entropy**, e.g., a sequence of random bits (128 or 256)
- **Compute hash** = SHA256(entropy)
- **Take** the first n = len(entropy) / 32 bits from **hash** (128/32=4, 256/32=8)
- **Append** these bits at the end of the entropy (checksum) (128+4=132, 256+8=264)
- **Split** bits into groups of 11 and convert them into integers (132/11=12 words, 264/11=24 words)
- **Find** the corresponding words in the word list

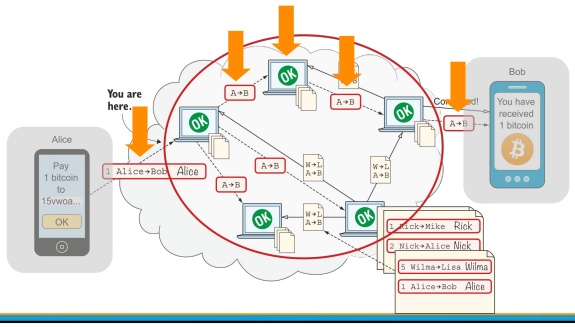
Seed phrase

- The process is **reversible**, if you loose your password and cannot access your private key(s) you can restore the wallet starting from the seed phrase
- The seed phrase is used to recompute the starting random number (the **master private key**) which generates the rest of the other private keys

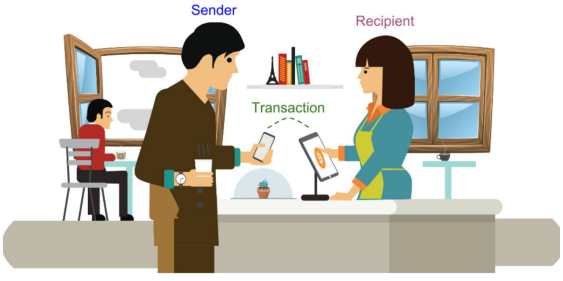
DIY (Do It Yourself)

- Install METAMASK

Broadcast transactions



Bitcoin transaction



Bitcoin transaction

- **Payments** in the Bitcoin system are **transactions** that represent **Bitcoins movements** from source addresses (**input**) to destination addresses (**output**)
- The protocol forces input addresses to spend the exact amount of a previously received transaction
- At any given moment, an **output** may be
  - **Already spent**
  - **Unspent** transaction output (**UTXO**)

Bitcoin transaction: metadata

- Information about the transaction itself
  - creation date
  - dimension
  - transaction identification number calculated using a hash function
  - segwit flag (for segregated witness, added later)
  - See https://www.blockchain.com/explorer

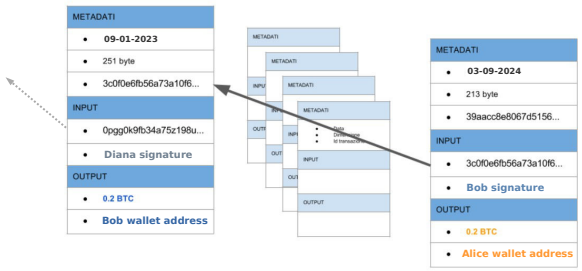
Bitcoin transaction: input

- Consists of
  - a transaction **sender**, for example Bob
  - the transaction **identification number** of a **previous transaction** with **Bob as receiver** (UTXO)
  - Bob's **digital signature** (by means of his private key)
  - A **script** to prove Bob owns the money

Bitcoin transaction: output

- Consists of
  - the **recipient**'s Bitcoin address, for example Alice's one
  - the **number of Bitcoin/Satoshi** to be transferred to Alice
  - a **lock script**, which means that the coins can only be used as inputs in future transactions by people who can unlock them

From Bob to Alice

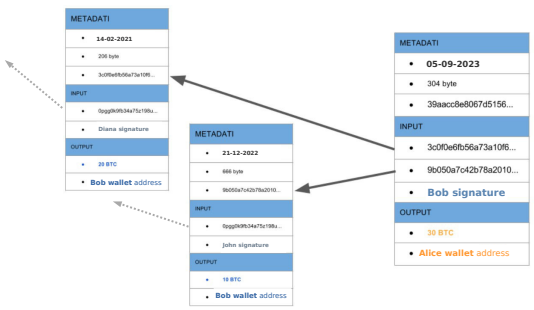


Bob can spend UTXO coming from his past history

Multiple input

- Suppose **Bob wants to pay** the amount of **30 BTC** to Alice, but he is not the recipient of any single transaction of 30 BTC
- Bob can insert into the input of the new transaction **one or more transaction IDs** he has previously received
  - one of **10 BTC**
  - one of **20 BTC**
- In the **input of a transaction**, the sender can insert a **list of transactions previously received**

Multiple input

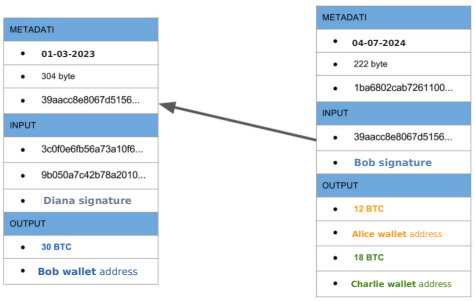


Multiple output

- Suppose **Bob wants to pay**
  - **12 BTC** to Alice
  - **18 BTC** to Charlie

  and he wants to pay using a transaction of **30 BTC** he has previously received
- Bob can insert into the output of the new transaction both Alice's address, to which he sends 12 BTC, and Charlie's, to which he sends 18 BTC
- In the **output of a transaction**, the sender can insert a **list of addresses** of the payment recipients

Multiple output



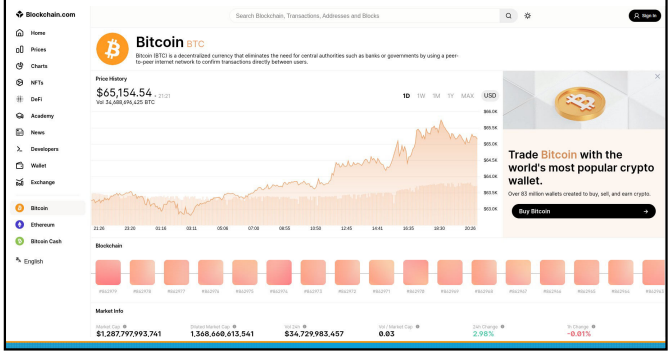
Change

- **Charlie can be Bob** himself and in this case we have a **change** of 18 BTC going back to Bob
- Bob can use the same address or a different one (as often suggested)
- Each user can have **multiple Bitcoin addresses**

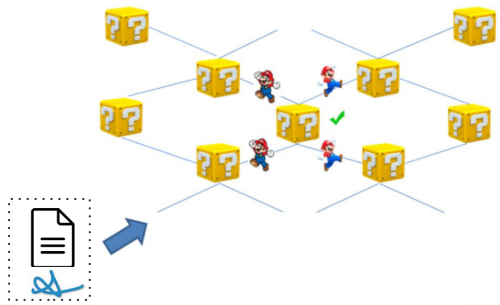
Transaction fee

- The **transaction fee** does not depend on the amount spent but on the **weight in terms of KB**
- The **number of UTXOs** used to compose a transaction is relevant to determine the fee of a transaction
- Therefore, to pay a lower fee it is better to **use a UTXO close to the amount to spend** rather than using several

Live explorer



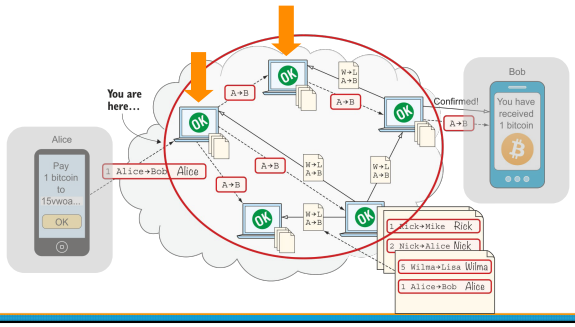
Transaction transmission



Transaction transmission

- All **transactions** broadcast in the Bitcoin network **wait to be confirmed** in the **mempool**
- A high mempool size indicates more network traffic which will result in longer average confirmation time and higher priority fees

Storing transactions



Mining

- Mining is the process of **adding new blocks** of valid transactions to the blockchain
- Miners are also responsible for the **creation of all new Bitcoins** and are a fundamental part of the Bitcoin ecosystem
- Most Bitcoin users do not mine

Nice explanation here: https://learnmeabitcoin.com/beginners/mining

Solo miner

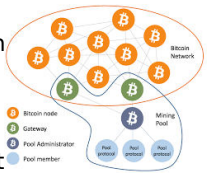
- Past: individual CPU, GPU, home-made hardware configurations
- They had a full copy of the blockchain to validate transactions
- They also needed a wallet to manage their mining reward

Solo miner

- Today: mining is split in two tasks
  - Block creation is done by peers that store the full blockchain
  - Hashing is done with dedicated hardware (ASIC), to speed up the computation

Mining pools

- Today: mining is not individual anymore and we have **mining pools**
- Organizations which compute hash to get a **reward**
  (see https://btc.com/stats/pool)



- Few mining pools concentrate a lot **power**
  → still real decentralization?

Six steps to be a miner

- **Listen for transactions** and **check them** by
  - checking that Bitcoins have not been spent before (no double-spending)
  - checking the signatures
- Maintain the blockchain and **listen for new blocks**
- **Prepare** a new block
  - group transactions into a new block (e.g., a file) that extends the latest known block

Six steps to be a miner

- **Reach the mining target** (see Proof-of-Work)
- **Hope** your block is accepted
  - e.g., other miners start mining on top of it
- If yes, **profit!**

Prof-of-Work

- Mining requires a task that is very **difficult to perform**, but **easy to verify**
- Satoshi proposed a sort of lottery/race/brute-force activity called **Proof-of-Work**
- Uses cryptography, with a **hash function** called **double SHA-256**

Prof-of-Work

- The mathematical challenge faced by miners consists in finding the **NONCE**
- SHA256(SHA256(**NONCE** + **hash previous block** + **transactions**)) <= **00000000000007C96A72BBC4445A5634BD**
- Trial and error procedure
- Try a new NONCE until the TARGET is reached

Prof-of-Work

- The mathematical challenge faced by miners consists in finding the **NONCE**
- SHA256(SHA256(**NONCE** + **hash previous block** + **transactions**)) <= **00000000000007C96A72BBC4445A5634BD**
- Trial and error procedure
- Try a new NONCE until the TARGET is reached

Prof-of-Work

- The average number of “trial and errors” steps needed to meet the target is the **Difficulty**
- **Every two weeks it is adjusted** as follows:
  - **1 block** on average every **10 minutes**
  - **20160 minutes** in **2 weeks** (around 2016 blocks)
  - **T = time required to mine the last 2016 blocks**
  - NewDifficulty = Difficulty * (**20160/T**)
  - If T < 20160, the NewDifficulty increases (and the target decreases)

Prof-of-Work

- The Difficulty is established by the Bitcoin network to moderate the mining speed: **every 10 minutes** a new block should be added to the blockchain

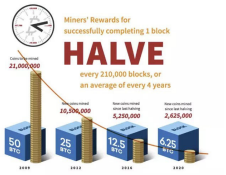
Why 10 minutes?

Reward for miners

- To encourage their work, miners receive incentives (rewards)
  - **fees** of the transactions contained in the block
    - priority to the transactions with highest fees
  - **newly created Bitcoins**
    - Coinbase transaction

Reward for miners

- The Coinbase transaction
  - is the **only way to create new Bitcoins**
  - reward per block was 50 BTC in 2008
  - this value is halved every 4 years
- **April 25, 2024**
  Last Bitcoin halving: the reward dropped to 3.125 Bitcoin per block
  See bock 840000:
  https://www.blockchain.com/explorer/blocks/btc/840000



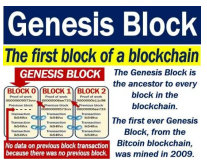
Issuing new Bitcoins

- Bitcoins are finite!
  - the maximum and total amount of Bitcoins that can ever exist is **21 million**
  - there are **less than ???? million** Bitcoins left to be mined
  - Satoshi estimated that the last Bitcoin should be mined sometime in the year 2140

https://www.buybitcoinworldwide.com/how-many-bitcoins-are-there/

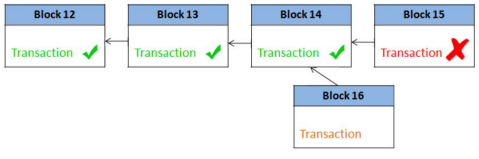
Genesis block

- It was mined over the course of six days by Nakamoto to start the blockchain
- https://www.blockchain.com/btc/block/000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f
- https://ma.ttias.be/retrieving-the-genesis-block-in-bitcoin-with-bitcoin-cli/



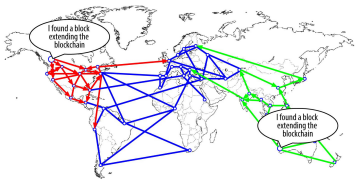
Forks

- The miner that has the right to insert a new block into the blockchain, adds it after the last known block
- But we are in a **decentralized** network and...

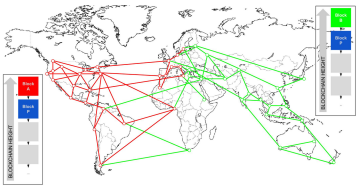


Forks

Forks



Forks



Forks

Forks

The longest chain wins!

- A block is valid only after other blocks are added on top of it
  - in the Bitcoin network usually wait for 6 blocks (around 1 hour) to consider a block really valid
- Blocks in the shorter chain are “canceled” and the unspent transactions return into the mempool

How many confirmations?

- **0** - can still be reversed! Wait for at least one
- 1 - enough for small payments, less than $1,000
- 3 - enough for payments $1,000 - $10,000. Most exchanges require 3 confirmations for deposits
- **6** - for large payments between $10,000 - $1,000,000 Standard for most transactions to be considered secure
- 60 - suggested for large payments greater than $1,000,000. Less is likely fine, but this is to be safe!

Decentralized Systems

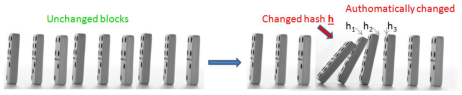
Bitcoin (cnt)

Hash pointers and Merkle trees

Hash pointer

- A **hash pointer** is a pointer to where some information is stored together with a cryptographic hash of the information

- This data structure makes the blockchain immutable



Merkle tree

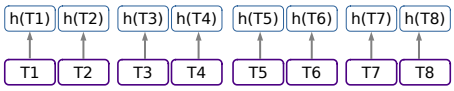
- A Merkle tree is a **binary tree with hash pointers**
- Consider the **Bitcoin transactions** of a block
  - they can be stored using a Merkle tree
- Transactions are grouped into pairs, if the number of transactions is odd, the last transaction is duplicated

Merkle tree

- A Merkle tree is a **binary tree with hash pointers**
- Consider the **Bitcoin transactions** of a block
  - they can be stored using a Merkle tree
- Transactions are grouped into pairs, if the number of transactions is odd, the last transaction is duplicated

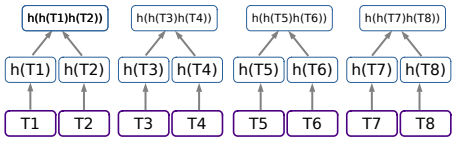
Merkle tree

- Compute the hash of each transaction

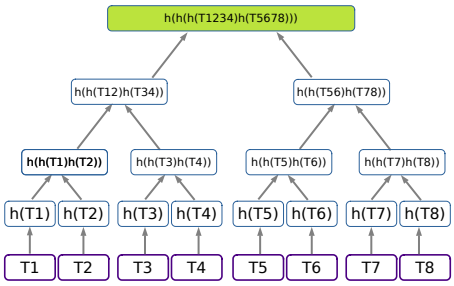


Merkle tree

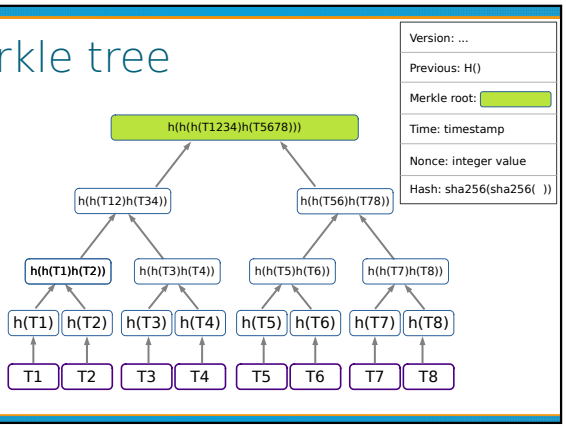
- Compute the hash of each transaction
- Concatenate the digests and compute the hash
- Continue...



Merkle tree



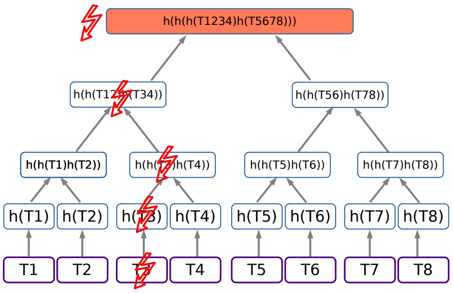
Merkle tree



Merkle tree

- The **root of the tree** is enough to check whether any transaction of the block has been modified
- If an attacker **tampers a transaction** or an intermediate leaf and the hash is cryptographic, then **the root changes**

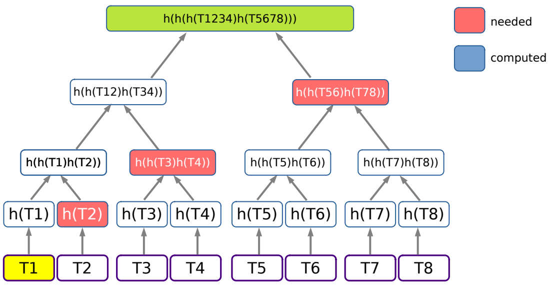
Tampering in a Merkle tree



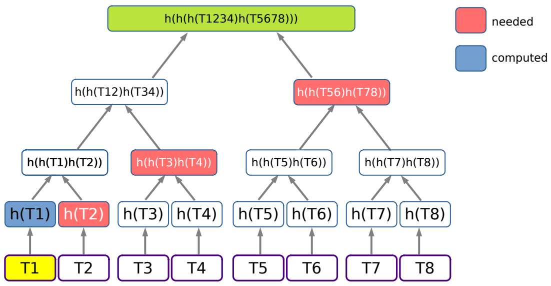
Proof of validity

- Problem
  - Bob wants to check if a certain Bitcoin transaction T is contained in a block and it has not been tampered
- Solution
  - He only needs to ask for the hashes of each sibling, he can recompute the root, and check if the two hashes match
  - The number of operations is $\log_2(n)$ where n is the number of transactions in the block

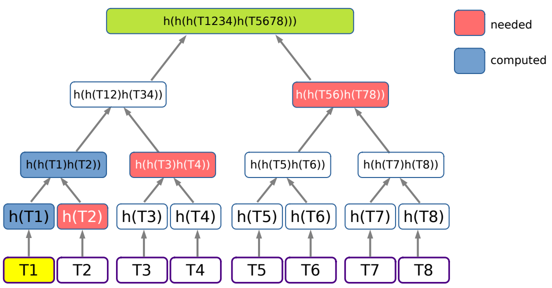
Proof of validity for T1



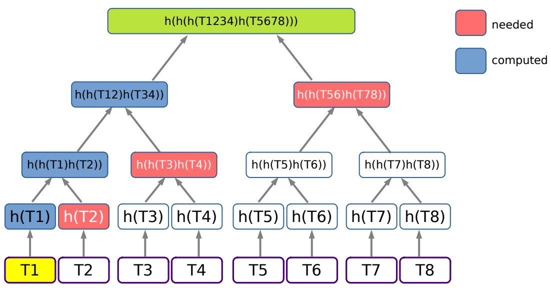
Proof of validity for T1



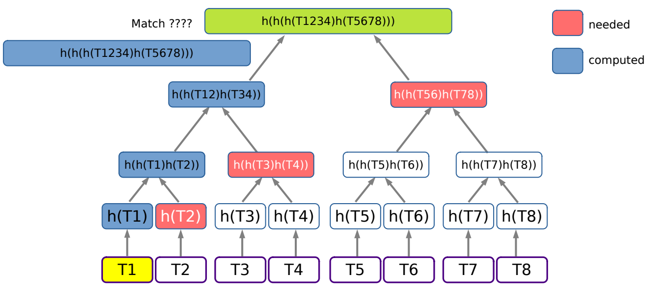
Proof of validity for T1



Proof of validity for T1



Proof of validity for T1



Coinbase extra-nonce

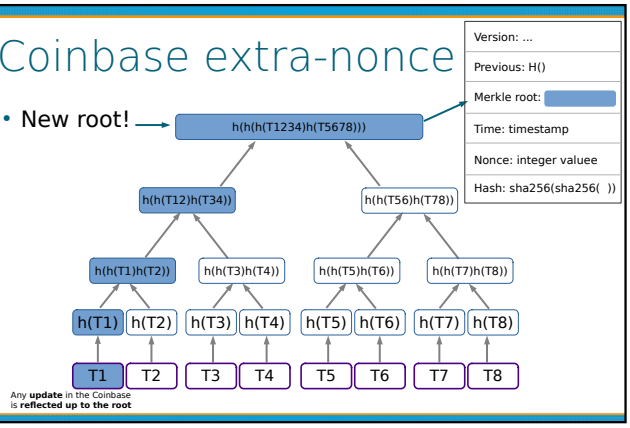
- The Coinbase transaction looks like a normal transaction but
  - generates brand-new coins, it has only one (invalid) input, a **null hash pointer**, and an **extra-nonce field**
  - the **reward** is sent to **one or more outputs** (miners can distribute the block reward to other addresses)
- For each block we are guaranteed to produce different hashes

Coinbase extra-nonce

- Miners change the nonce field in the block's header
- There are **2^{32} possible values for the nonce**
- Once tried all possible values, miners can **change the extra-nonce** of the Coinbase
- Computationally expensive, **the Merkle tree needs to reflect this change**

Coinbase extra-nonce

- New root!



Bitcoin network

Bitcoin Network

- The ecoystem of Bitcoin relies on a **non-structured P2P overlay network** with some similarities with Gnutella
- **Flooding** or **gossip protocols** are used for the propagation of the required information
- In this context, the information which is "stored" by the P2P network are **transactions** and **blocks**

Bitcoin Network

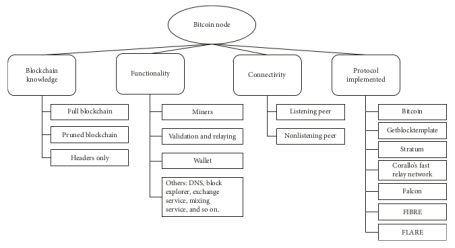


Figure 3: Bitcoin node classification.

Source: Cryptocurrency Networks: a New P2P Paradigm https://www.hindawi.com/journals/misy/2018/2159082/

Bitcoin Network

- **Users** are "identified" in the blockchain by their pairs of **public key/address**, **peers** are identified in the Bitcoin network by their **IP** or **Onion addresses**
- With respect to **connectivity**, they can be classified as
  - **listening** peers, e.g., nodes accepting incoming connections (Bitcoin daemons by default listen for incoming connections on TCP ports 8333 and 18333)
  - **nonlistening** peers, e.g., nodes not doing so

Bitcoin Network

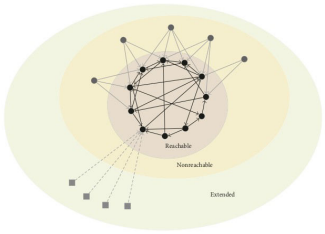
- The P2P network is composed of
  - The **reachable** network, formed by **all listening nodes** that talk the Bitcoin protocol (a sort of "servers")
  - The **nonreachable** network, formed by nodes that talk the Bitcoin protocol but do not listen for incoming connections (a sort of "clients", e.g., peers behind NAT or firewall)
  - The **extended** network formed by all nodes in the ecosystem (DNS, exchanges, explorers)

Bitcoin Network

- By default all peers maintain **up to 125 connections** with other peers
  - Each node, when joins the network **tries to connect to 8 other** peers (**outgoing** links)
  - Each node **can accept up to 117** connections from potential peers (**incoming** links)

Bitcoin Network

- See https://bitnodes.io/



Source: Cryptocurrency Networks: a New P2P Paradigm https://www.hindawi.com/journals/misy/2018/2159082/

Bitcoin Network

- No central authority
- Designed having in mind
  - **Reliability** (by design, every peer stores all relevant information, and this is highly inefficient from the storage point of view)
  - **Security** (properties of transactions and blocks can prevent flooding attacks, tampering of the data, and others)

Bitcoin Network

- **Flooding/DoS attacks**
  - Transaction flooding is prevented by **not relaying invalid transactions** and by requiring fees for valid transactions
  - Block flooding is prevented by **only relaying valid blocks** which must contain a **valid nonce** for the PoW
  - **Reputation-based** mechanism in which each node keeps a penalty score for every connection. The penalty increases with the number of malformed messages sent on the connection. Misbehaving peers whose penalty scores reach a given value are banned for 24 hours

Bitcoin Network

- **Join**
  - Many Bitcoin nodes usually operate behind a NAT or firewall
  - When a node **joins the network** it must **discover its own public IP address**
  - It **can send a GET request** to two hard-coded websites which reply with the address
  - It is also possible to **manually configure** the node with the public IP address and the port in the node's settings (e.g., via bitcoin.conf)

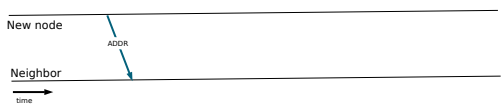
Bitcoin Network

- **Join**
  - After joining the network, the node may **discover other IP addresse**s from the peers it connects with
  - Each peer **keeps a list of IP addresses** associated with its connections into a local database
    - **tried** table: **IP + timestamp** of other peers known from past connections
    - **new** table: new **IP + timestamp**, populated by addresses learned by **DNS seeders** or by querying **neighbors**

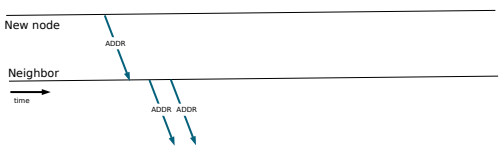
Bitcoin Network

- **Address propagation**
  - Peers can request addresses sending **GETADDR** messages to their neighbors
  - They also receive unsolicited **ADDR** messages which contain IP addresses
  - Each node can decide to forward them or not

Bitcoin Network



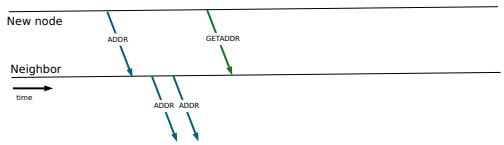
Bitcoin Network



The neighbor receiving the **ADDR message** from the new node will **store the received IP** in the local database and, in turn, will **forward it to its neighbors**

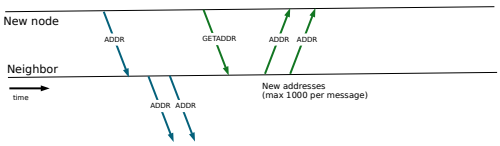
This ensures that a newly connected node is well known by the network

Bitcoin Network



Additionally, a **GETADDR** message can be sent by the new node asking its neighbors to return a list of IP addresses of other peers

Bitcoin Network



Bitcoin Network

- Peers try to always maintaining their 8 outgoing connections, selecting new nodes when their neighbors leave the network
- Neighbors are selected from the local database following a **pseudo random procedure** that gives the network **high dynamism** and keeps its **overall structure unknown**

Bitcoin Network

- Peers exchange also data structures
  - **Transactions** are the data structures **most usually seen** (see https://www.blockchain.com/charts)
    - Every single node can take part in a transaction by simply using a wallet, no matter of its type
  - **Blocks are less frequent**, on average 6 blocks per hour

Transaction propagation

- Bitcoin's method for transactions is **flooding-based**
- When a node **creates** a transaction
  - **serializes** it in hexadecimal format
  - **announces** it using an **INVENTORY** message

```
Message header:
    Command: "inv"
    Payload length: N

Variable-length payload:
    Count (4-byte unsigned integer): The number of inventory entries in the message.
    Inventory entries (variable length):
        Type (1-byte unsigned integer): The type of object that the inventory entry identifies.
        Hash (32-byte hash): The hash of the object.
```

0 Block
1 Transaction
2 Filtered block
3 Witness data
4 Compact block

Transaction propagation

- Bitcoin's method for transactions is **flooding-based**
- When a node **receives** an **INVENTORY** message
  - requests the transaction using a **GETDATA** message
  - the sender node sends the full transaction data to the requesting nodes using the **TX** message
  - the receiving nodes validate the transaction and **add it to their mempool** (temporary store for unconfirmed transactions)

Transaction propagation

- Nodes keeps **valid transactions** in the **mempool** and answer requests for them
- Some Bitcoin clients uses **Bloom filter** encoding the transaction IDs in their mempool, thus only missing transactions can be exchanged
- Nodes periodically **exchange Bloom filters** to avoid forwarding transactions to those neighbors who already have them

Block propagation

- When a **miner finds a nonce**, it creates a block with
  - the block header (previous block hash, timestamp, nonce, etc.)
  - the list of transactions
  - the Merkle root (the cryptographic hash of all the transactions)
- It **advertizes** the new block to its neighbors with an **INVENTORY** message containing the hash of the block

Block propagation

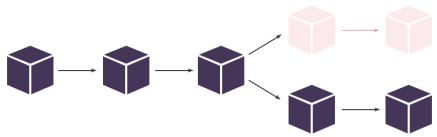
- When neighbors **receive** the INVENTORY message, they check if they already have the block
- If they do not have it, they respond with a **GETDATA** request, asking for the full block
- The **miner** (peers) **sends** the block with a **NEWBLOCK** message
- The neighbors **verify the new block** and, if valid, they **forward** its to their neighbors

Block propagation

- Since blocks are larger and more critical for maintaining consensus, the Bitcoin network uses several optimizations to improve the speed and efficiency of block propagation
- The protocol specification is rather complex, if interested see https://en.bitcoin.it/wiki/Network (optional)

Block propagation

- Of course, delayed blocks may lead to **forks**
- Forks should be avoided as they are symptomatic for inconsistencies among the replicas in the network



Bitcoin network summary

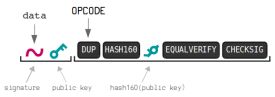
- By design
  - **high level of reliability** thanks to its **redundancy**: the availability of a single node in the network contains the information to keep the system alive
  - **high inefficiency** in terms of **storage space**
  - **slow**
  - **computationally intensive (energy-intensive)**

Decentralized Systems

Bitcoin (cnt)

Bitcoin Script

(a language with no name)



Valid payments

- To ensure the **correctness of a transaction**, each node verifies
  - its input Bitcoins have not been spent yet (UTXO)
  - the sender's signature
- To **automatically control** the transaction correctness, each node carries out the **transaction script**

Bitcoin Script

- Specialized **programming language** used in Bitcoin to define the rules for spending and validating transaction
- **Automatically processed** by peers (wallets, miners, and other nodes behind the scene)
- **Stack-based**: every instruction is executed exactly once

Bitcoin Script

- Used to **define the conditions that must be met** in order to spend a particular UTXO
  - requires the **recipient** to provide a **valid signature** with the private key corresponding to the public key associated with the output
  - implements a variety of features, such as **multi-signature** wallets, **escrow** payments, and **time-locked** transactions

https://en.bitcoin.it/wiki/Script

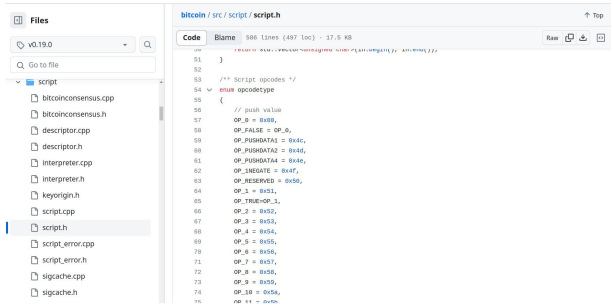
Bitcoin Script

- No Turing complete (no infinite loops or complex logic)

Bitcoin Script

- Design goals
  - simple, compact, support for cryptography
- Composed of
  - **Data**: for example public keys and signatures
  - **Opcodes**: instructions (there is room only for 256 opcodes)
    - basic arithmetic (+, -, *, ...)
    - basic logic, returning earlier, etc...
    - special-purpose instructions to compute and verify signatures

Bitcoin Script

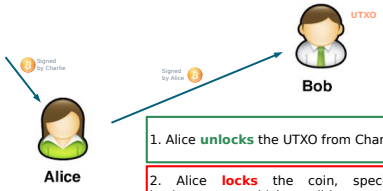


https://github.com/bitcoin/bitcoin/blob/v0.19.0/src/script/script.h
https://en.bitcoin.it/wiki/Script

Bitcoin Script

Bitcoin Script

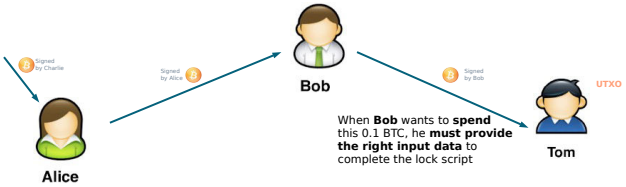
Bitcoin Script



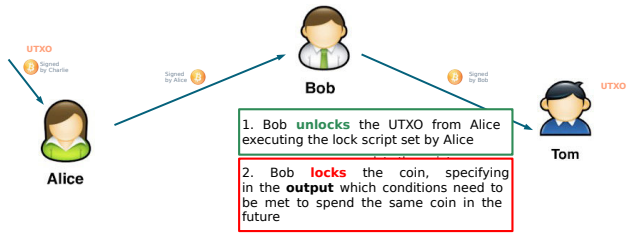
1. Alice **unlocks** the UTXO from Charlie

2. Alice **locks** the coin, specifying in the **output** which conditions need to be met to spend the same coin in the future

Bitcoin Script



Bitcoin Script



Bitcoin Script

- Every **output** is given a **locking script**
- **Data** must be provided in the **input** for executing the **unlocking script** to reedem and spend coins
- The two scripts **are concatenated** and, if the **resulting script runs with success**, the **transaction is valid**

Bitcoin Script

- Most transactions use the same **small set of simple instructions**
- Bitcoin nodes have a **list of standard scripts** and they refuse to accept scripts that are not in the list
- This makes the Bitcoin script language inherently **more secure** due to the limited number of operations that it can perform

Pay-to-PubKey-Hash (P2PKH)

- P2PKH allows a Sender (Alice) to send coins to a Bitcoin address (one of the most used script before SegWit)
- To spend the coins (redeem) the Receiver (Bob) must show the ownership

Unlock script (ScriptSig)

3044022041c09a16ee9db2315d09df490d0b4ba3b34f40b148fe39bd756b93dffe27b31802204c1230b9415657c176369a7556283398b7552039f2a5cd3bfb17a22be0a77b0f01

02f5e548d2ab03cb235fc979c8cce56620fd09114362f926f0cc31c7e317f36e91

Lock script (ScriptPubKey)

OP_DUP
OP_HASH160
7b134de21ecf69a197ae05305e5264a691b58753
OP_EQUALVERIFY
OP_CHECKSIG

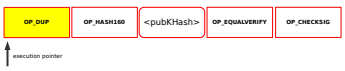
Pay-to-PubKey-Hash (P2PKH)

Pay-to-PubKey-Hash (P2PKH)

3044022041c09a16ee9db2315d09df490d0b4ba3b34f40b148fe39bd756b93dffe27b31802204c1230b9415657c176369a7556283398b7552039f2a5cd3bfb17a22be0a77b0f01

Provided by Bob

Pay-to-PubKey-Hash (P2PKH)



02f5e548d2ab03cb235fc979c8cce56620fd09114362f926f0cc31c7e317f36e91

3044022041c09a16ee9db2315d09df490d0b4ba3b34f40b148fe39bd756b93dffe27b31802204c1230b9415657c176369a7556283398b7552039f2a5cd3bfb17a22be0a77b0f01

Provided by Bob

Pay-to-PubKey-Hash (P2PKH)

02f5e548d2ab03cb235fc979c8cce56620fd09114362f926f0cc31c7e317f36e91

Result of OP_DUP

02f5e548d2ab03cb235fc979c8cce56620fd09114362f926f0cc31c7e317f36e91

3044022041c09a16ee9db2315d09df490d0b4ba3b34f40b148fe39bd756b93dffe27b31802204c1230b9415657c176369a7556283398b7552039f2a5cd3bfb17a22be0a77b0f01

Pay-to-PubKey-Hash (P2PKH)

The input is hashed twice: first with SHA-256 and then with RIPEMD-160, e.g., Bitcoin address

7b134de21ecf69a197ae05305e5264a691b58753

Result of OP_HASH160

02f5e548d2ab03cb235fc979c8cce56620fd09114362f926f0cc31c7e317f36e91

3044022041c09a16ee9db2315d09df490d0b4ba3b34f40b148fe39bd756b93dffe27b31
802204c1230b9415657c176369a7556283398b7552039f2a5cd3bfb17a22be0a77b0f01

Pay-to-PubKey-Hash (P2PKH)

7b134de21ecf69a197ae05305e5264a691b58753

Provided by Alice in the lock script when sending coins to Bob's address

7b134de21ecf69a197ae05305e5264a691b58753

02f5e548d2ab03cb235fc979c8cce56620fd09114362f926f0cc31c7e317f36e91

3044022041c09a16ee9db2315d09df490d0b4ba3b34f40b148fe39bd756b93dffe27b31
802204c1230b9415657c176369a7556283398b7552039f2a5cd3bfb17a22be0a77b0f01

Pay-to-PubKey-Hash (P2PKH)

Result of OP_EQUALVERIFY

02f5e548d2ab03cb235fc979c8cce56620fd09114362f926f0cc31c7e317f36e91

3044022041c09a16ee9db2315d09df490d0b4ba3b34f40b148fe39bd756b93dffe27b31
802204c1230b9415657c176369a7556283398b7552039f2a5cd3bfb17a22be0a77b0f01

Pay-to-PubKey-Hash (P2PKH)

A script is valid if the top and only element left on the stack is a 1

1

Result of OP_CHECKSIG

Pay-to-PubKey-Hash (P2PKH)

A script is invalid if

- The final stack is empty
- The top element on the stack is 0
- There is more than one element left on the stack at the end of execution
- The script exits prematurely (e.g. OP_RETURN)

0

Result of OP_CHECKSIG

Pay-to-Pubkey (P2PK)

- P2PK (Pay To Pubkey) is a script pattern that locks an output to a **public key** (which is longer than a Bitcoin address)
- P2PK can be found in Coinbase transactions in the earlier blocks in the blockchain
- Other scripts can be found in the Coinbase, if OP_1 appears in the output script, it is likely associated with SegWit outputs...

Pay-to-Multisig (P2MS)

- Third party (**escrow**/arbiter) may optionally be involved in transactions
- To unlock a P2MS script, a **required number of signatures** (2) need to be provided among a set of specified public keys (3)
- Few locking scripts are P2MS, so they can be hard to find

Unlock script **Lock script**

* OP_0 is required because of a bug in the original implementation

Pay-to-Multisig (P2MS)

- Third party (**escrow**/arbiter) may optionally be involved in transactions

OP_CHECKMULTISIG	174	0xae	x sig1 sig2 ... <number of signatures> pub1 pub2 <number of public keys>	True / False	Compares the first signature against each public key until it finds an ECDSA match. Starting with the subsequent public key, it compares the second signature against each remaining public key until it finds an ECDSA match. The process is repeated until all signatures have been checked or not enough public keys remain to produce a successful result. All signatures need to match a public key. Because public keys are not checked again if they fail any signature comparison, signatures must be placed in the scriptSig using the same order as their corresponding public keys were placed in the scriptPubKey or redeemScript. If all signatures are valid, 1 is returned, 0 otherwise. Due to a bug, one extra unused value is removed from the stack.

Unlock script **Lock script**

* OP_0 is required because of a bug in the original implementation

NULL DATA (OP_RETURN)

- NULL DATA is a standard **locking script** that can be used to **store data** on the blockchain
- The instruction OP_RETURN
  - returns immediately an error
  - instructions after OP_RETURN are not processed
- The data after OP_RETURN could be at maximum of 80 bytes

NULL DATA (OP_RETURN)

Unlock script Lock script

- Any output with a OP_RETURN on it is **unspendable**
- 80 bytes are sufficient for a hash function output (32 bytes for SHA-256)
- The NULL DATA script is used to **timestamp public and immutable data** in the blockchain

Proof-of-burns

- All Bitcoin transactions with the OP_RETURN instruction in their outputs can not be redeemed
- That is equivalent to **burning** coins!
- But it is possible to take advantage of the ash and store data permanently on the blockchain

https://en.bitcoin.it/wiki/OP_RETURN

Script types today

Output Type Volume (90 Days)	?
Pubkey Hash	33.11%
Script Hash	9.69%
SegWit v0 Pubkey Hash	43.64%
SegWit v0 Script Hash	12.68%
Taproot	0.94%
Output Type Counts (90 Days)	?
Bare Multisig	341,730
Non-standard	5
Nulldata (OP_RETURN)	33,218,115
Pubkey	27
Pubkey Hash	9,466,300
Script Hash	9,280,866
SegWit v0 Pubkey Hash	58,596,281
SegWit v0 Script Hash	1,894,231
SegWit Unknown Version	7
Taproot	23,141,129

https://bitcoin.clarkmoody.com/dashboard/

Do you remember RFC?

Bitcoin Improvement Proposals

- In Bitcoin we have BIP (https://en.bitcoin.it/wiki/BIP_0001)
- A BIP is a **design document** providing **information** to the Bitcoin community, or describing a **new feature** for Bitcoin or its **processes or environment**. The BIP should provide a concise technical specification of the feature and a rationale for the feature
  - A **Standards Track BIP** describes any change that affects most or all Bitcoin implementations [...]

Hard fork

- Changes in the Bitcoin protocol can lead to
  - **Hard fork**
  - **Soft fork**
- A **hard fork** is a **radical change** to the protocol that makes previously valid blocks/transactions invalid (or vice-versa)
- A hard fork requires all nodes or users to upgrade to the latest version of the protocol software

Hard fork

- Nodes of the **newest version of a blockchain no longer accept the older version(s)** of the blockchain and this creates a **permanent divergence** from the previous version of the blockchain
- Generally, after a short time, those on the old chain will realize that their version of the blockchain is outdated and quickly upgrade to the latest version

Soft fork

- A **soft fork** is a change to the software protocol which is **backward compatible**
- Only a majority of the nodes upgrade to enforce the new rules, as opposed to a hard fork that requires all nodes to upgrade and agree on the new version
- The **blockchain accepts the new rules** and, therefore, accepts both the updated blocks and the old blocks of transactions at the same time

New scripts to solve some issues

The issue of scalability

- Visa more than 1700 transactions per sec
- Bitcoin 4-7 transactions per sec (average 4.6)

https://towardsdatascience.com/the-blockchain-scalability-problem-the-race-for-visa-like-transaction-speed-5cce48f9d44

Segregated witness (SegWit)

- **Protocol upgrade** (proposed in Dec 2015, BIP 141, adopted in Aug 2017) intended to **increase block capacity**
- It addresses also **malleability** of transactions

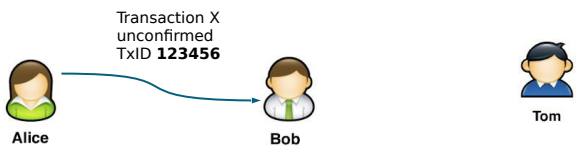
Malleability

- Bitcoin code allows **digital signatures to be altered** while a transaction is **still waiting to be confirmed**
- The **signature alteration** can be done in such a way that, by running a mathematical check, the **signature is still valid** for the network, but...
- ... its **hash is different**!
- And hash values identify transactions in the network

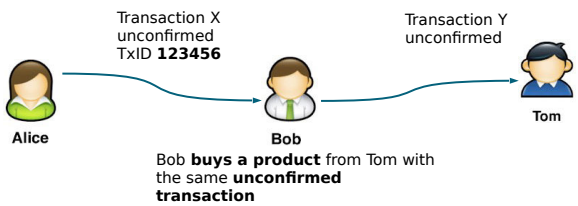
Malleability

	Original Tx	Altered Tx
Signature	3	7-4
Mathematical value	3	3
Hash value (new TxID!)	1121cfccd5913f0a63fec40a6ffd44ea64f9dc135c66634ba001d10bcf4302a2	48fd8983f9799e20d3ff8813dfe48e38ed1bd20dc0514894357cc2bdf141a0da

Malleability

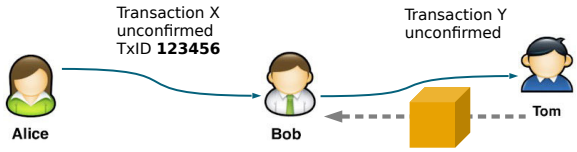


Malleability

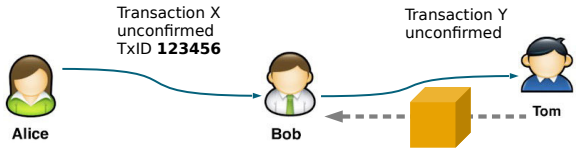


Bob **buys a product** from Tom with the same **unconfirmed transaction**

Malleability

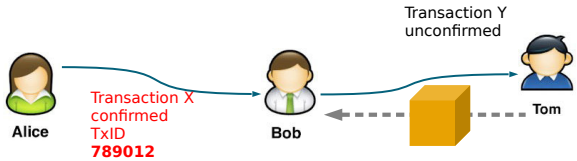


Malleability

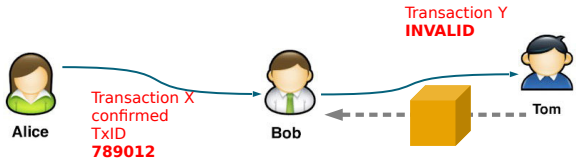


After receiving the product, Bob maliciously changes (**malleates**) **Alice's payment** so that her transaction gets confirmed but with a **different TxID**

Malleability



Malleability

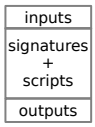


Transaction Y is now **invalid**, since it relies on TxID 123456 which no longer exists and **Tom will not get paid**

https://en.wikipedia.org/wiki/Transaction_malleability_problem

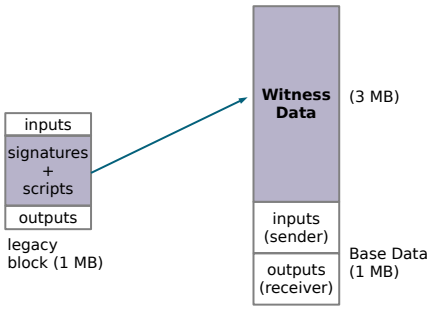
Segregated Witness (SegWit)

- Legacy blocks 1 MB, but
  - 90% of the space is used by transactions
  - 65% of the space of the transactions is used by signatures

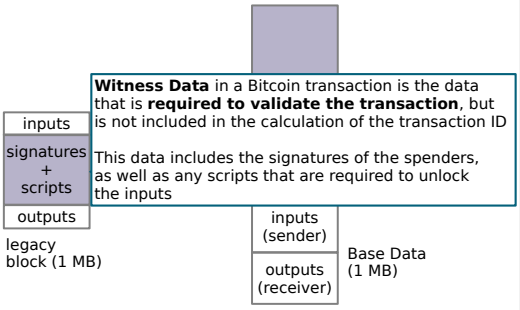


- What about separating (**segregating**) signatures (**witness data**) from the rest of the transaction data?

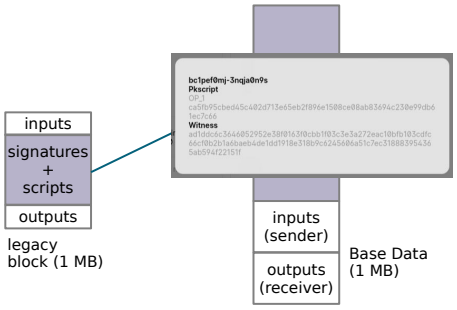
Legacy vs SegWit



Legacy vs SegWit



Legacy vs SegWit



SegWit

- Increases block capacity and it is **compatible** with the original Bitcoin protocol, e.g., is a **soft fork**
- The network accepts both SegWit and Non SegWit blocks
  https://transactionfee.info/charts/transactions-spending-segwit/
- Pay-to-Witness-Pubkey-Hash (P2WPKH) and Pay-to-Witness-Script-Hash (P2WSH)

SegWit

- **Pros**
  - more transactions can fit in a 1 MB block, **more scalable**
  - addresses also **malleability**, by moving the signature outside the base transaction block, changes in the transaction signature will not affect TxID
  - SegWit enables **second layer scaling solutions**
- **Cons**
  - Legacy wallets do not support SegWit (addresses start with bc1, legacy addresses start with 1)

Measuring blocks

Legacy blocks	SegWit blocks
Measured in **size**	Measured in **weight**

Measuring blocks

Legacy blocks	Instead of simply increasing the block size to accommodate more transactions, Bitcoin developers introduced the concept of **block weight** to ensure backward compatibility with the existing 1 MB block size limit while allowing for more efficient data usage
Measured in **size**	

Measuring blocks

Block Weight = (3 * Base Size) + Total Size

Legacy transaction	SegWit transaction
No ability to strip witness data the weight is always: **4 * Total Size**	Can strip witness data, the weight is always: **less than 4 * Total Size**
	Miners favor SegWit transactions

Measuring blocks

Block Weight = (3 * Base Size) + Total Size

Legacy transactio	With the introduction of block weight, the block size limit (1 MB) was replaced by a **block weight limit of 4,000,000** weight units (WU)
No ability to strip witness the weight is always: **4 * Total Size**	A block is still restricted to around 1 MB of non-witness data (Base Size), to preserve compatibility with older, non-SegWit nodes
	Miners favor SegWit transactions

Taproot

- **Protocol upgrade** activated in Nov 2021 (BIPs 340, 341, 342)
- Improves privacy, scalability and security of the network thanks to
  - **Schnorr signature**, that is smaller, faster, and more efficient than previous ECDSA signature used in Bitcoin
  - **MAST** (Merkelized Abstract Syntax Trees), that allows multiple signatures to be combined into a single signature, without revealing whether the signature is a single signature or a multisignature

Taproot

  - Collection of **new opcodes** which extends Bitcoin Script
  - Taproot, like SegWit, is implemented as a **soft fork**
- **Pay-to-Taproot** (P2TR) is the name of the scripts which are compatible with this upgrade
  - they are smaller and therefore cheaper in term of space occupied on the blockchain

Reduce consumption?

CLIMATE, ENERGY

Cryptocurrency's Dirty Secret: Energy Consumption

BY JEREMY HINSDALE | MAY 4, 2022 Comments

Though skeptics may characterize cryptocurrency as "fake money," "worse than tulip bulbs," or a "greater fool" scheme, it is a very real business. The market capitalization of the almost 19,000 cryptocurrencies in circulation is currently around $1.75 trillion — about the same as the gross domestic product of Italy, the world's eighth largest economy. Even though you might not be able to buy a loaf of bread with Bitcoin at the corner store, many investors are putting a lot of legal tender money into cryptocurrencies.

But crypto has a dirty little secret that is very relevant to the real world: it uses *a lot* of energy. How much energy? Bitcoin, the world's largest cryptocurrency, currently consumes an estimated 150 terawatt-hours of electricity annually — more than the entire country of Argentina, population 45 million. Producing that energy emits some 65 megatons of carbon dioxide into the atmosphere annually — comparable to the emissions of Greece — making crypto a significant contributor to global air pollution and climate change.

And crypto's thirst for energy is growing as mining companies race to build larger

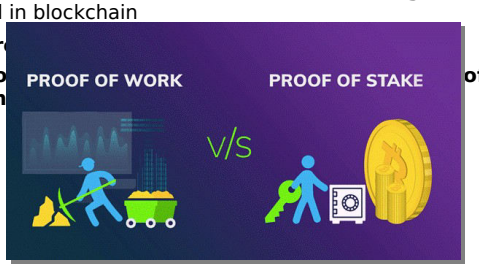
https://news.climate.columbia.edu/2022/05/04/cryptocurrency-energy/

Changing consensus

- Bitcoin Prof-of-Work is the **first consensus algorithm** used in blockchain
  - **Pros**: **anyone can join** the network
  - **Cons**: generally **slow**, requires a **huge amount of energy**

Changing consensus

- Bitcoin Prof-of-Work is the **first consensus algorithm** used in blockchain
  - **Pr**
  - **Co** **of** **en**



Proof-of-Stake

- Alternative consensus mechanism proposed for the firt time in 2012 with Peercoin (https://www.peercoin.net/)
- Blocks are **forged** or **minted** and not mined
- **Validators** not miners
- Two types of PoS
  - Chain-based
  - Consortium consensus

Chain-based PoS

- The leader (**validator)** adds the next block to the blockchain and the other peers agree on it
- **Leader election** is done by considering the **amount of investment** (**stake**) peers have committed to be part of the process **plus some randomness**
- Nodes have an **economic incentive to behave honestly** so as not to devalue the network and their stake in it
- We will see Ethereum

Decentralized Systems

Bitcoin Security

Bitcoin Security

- Bitcoin has the ambition of being **secure**, **decentralized** and **trustless**
  - i.e., the system is secure even if everybody is selfish and ill-intentioned
- To what extent it really is?

Mining Pools: Just a Few More Details

Mining Pools Are Centralized

- A **coordinator** prepares **block templates**
  - They choose the included transactions
  - The template pays block rewards to the coordinator
- Pools have an **internal difficulty** that is much lower than the Bitcoin one
  - When you solve a block with the internal difficulty, you get a "**share**" that allows you to share profits
  - When you solve a **real block**, the profits will be distributed among all those that have a share (and a fee for the coordinator)
- The coordinator needs to be **trusted**

51% Attack

51% Attack

- Bitcoin is considered secure **only if more than half of the hashing power is composed by "honest" nodes**
  - "Honest": nodes behaving according to the protocol
- Idea here: if there is a malicious fork, the "honest" one will finally win because they will build more blocks
- What if, however, attackers get to **more than that**?
- Source: How A Mining Monopoly Can Attack Bitcoin, blogpost by Eyal and Gün Sirer

Double Spending

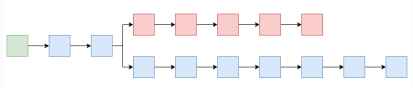


image from chowles.com

- A 51% attacker can always create a fork and eventually get a longer chain than the previous one
- They can spend all their bitcoins in the upper chain and then work on a fork where they're not spent

Transaction Censorship

- A 51% attacker can also "censor" transactions they don't like—simply not work on top of blocks that contain them
  - Those "orphan" blocks will eventually be in a shorter fork, and hence be censored
- This can take many forms
  - Total denial of service
  - Extortion ("pay me a huge fee otherwise you'll never get to use your bitcoins")
  - Targeting selected miners, transactions and addresses

Loss of Decentralized Narrative

- A 51% miner may not launch any attack
  - This will likely make BTC value fall, hence they might lose
  - However, many people choose cryptocurrencies **because they don't want to trust anybody**
  - Does it work when **you have to trust somebody**?

Are 51% Attacks Feasible? (1)

Source: CoinTelegraph (Jan 2020)

Source: Ars Technica

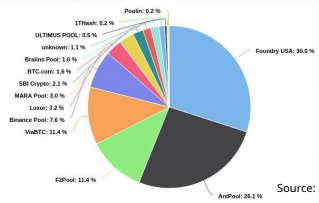
Are 51% Attacks Feasible? (2)

- What about today's Bitcoin?
  - You need to buy and maintain specialized ASIC (Application-Specific Integrated Circuit) equipment
  - The largest part of the bill is electricity
- Difficult to estimate easily
- Let's try another approach…

Are 51% Attacks Feasible? (3)

- Back-of-the-envelope calculation, October 2024:
  - Peak hash rate (source: blockchain.com) has been $6.39*10^{20}$ hash/second (639 exahash!)
  - Such a rate results in 38.4 million US$ per day earning for the miner (source: coinwarz.com)
  - Over a year, that's 14 billion US$ (9.3B$ last year)
- A miner has to operate at a profit, so costs must be lower
- Feasible long term if a powerful player is motivated
  - 0.62% of Italy's GDP, 0.08% of EU GDP, 0.05% of USA GDP
  - ~14% of Apple's yearly profits

Mining Pool Situation, October 2023



Source: btc.com (not existent anymore)

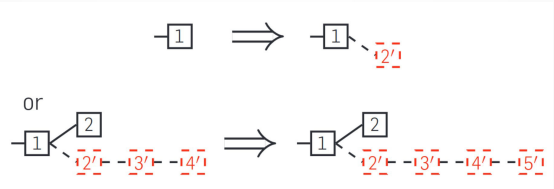
- Current data is quite similar
- Unlike Ghash (the 2014 55% pool), Foundry USA appears to be not anonymous
- Collusion between the top two pools would get a 51% attack

Selfish Mining

A Counterintuitive Strategy

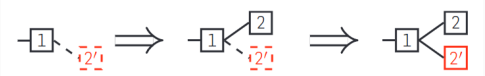
- Majority Is Not Enough: Bitcoin Mining Is Vulnerable, (Eyal and Gün Sirer, 2018)
  - (Login with your university password to access the PDF)
- A miner (more realistically: a pool) can gain more by **not respecting the protocol**, **withholding** (i.e., not revealing immediately) blocks they mined
- Idea: the selfish pool has a **secret fork** they are working on that other miners have no access to

Selfish Mining Strategy (1)



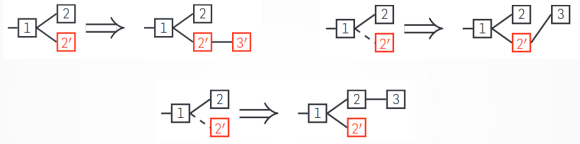
- When the pool finds a block, in most cases, they keep working on their secret (red dashed boxes) fork
  - Exception: two branches of length 1 (we'll see later)

Selfish Mining Strategy (2)



- If the pool was ahead by 1 and others find a block, it **immediately publishes** (solid red) the withheld block
  - The pool keeps working on their fork
  - Other peers will be split between the two (protocol: work on the first you've heard about)
  - In which fractions will the miners split between the two forks? **Key part we'll see later**

Selfish Mining Strategy (3)

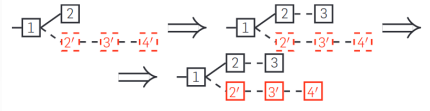


- Depending on who finds the next block we see if the pool "wins" on the bet they made
  - They lose if **others** find the next block **and** they add it to the "**honest" fork**

Selfish Mining Strategy (4)

- If the pool was ahead by more than 2 and the others find a block, no problem at the moment
  - The pool can however reveal a block now to get their rewards right away

Selfish Mining Strategy (5)



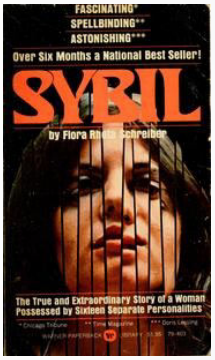
- If the lead drops to one, the pool **immediately publishes** their fork

Analysis

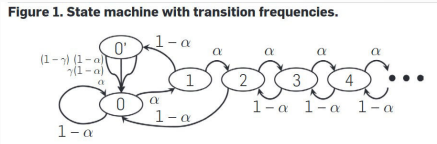
- Idea: compute the **fraction of rewards** that goes to the selfish pool
  - In the long run, **after difficulty adjustments**, the total rewards for miners will be **the same**
- Key parameter γ: the fraction of nodes that will work on the selfish fork

Sybil Attack

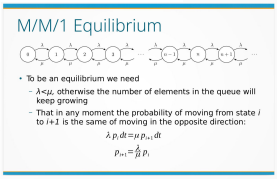
- Name from a book about a woman with 16 personalities
- In P2P systems: an attacker creates a very large number of nodes to subvert the system
- In the tie situation, Sybils join the Bitcoin P2P network and **only propagate the attackers' block**, thereby **raising γ**
- Proposed countermeasure: in case of competing tied forks, honest nodes propagate all of them



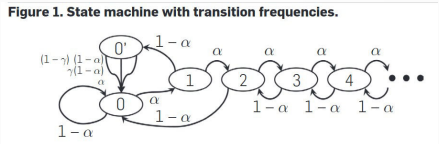
Analysis (1)



- Does this remind you of anything?

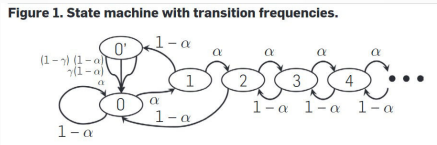


Analysis (2)



- **Markov Chain** analysis of how the system behaves
- Numbers are the **selfish fork's lead**
- The **0' state** is the **tie situation** in which a fraction γ of "honest" nodes works on the attacker's fork

Analysis (3)



- Find the **equilibrium distribution** of **probabilities to be in a given state**
- Associate a **reward** to every transition
- Sum up all **rewards weighted by their frequency** and compare with **those of the pool**

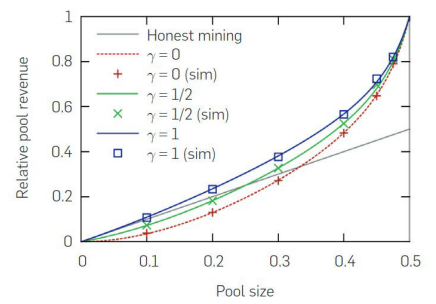
Analysis (4)

$$r_{\text{others}} = \overbrace{p_{0'} \cdot \gamma(1-\alpha)}^{\text{Case (c)}} \cdot 1 + \overbrace{p_{0'} \cdot (1-\gamma)(1-\alpha)}^{\text{Case (d)}} \cdot 2 + \overbrace{p_0 \cdot (1-\alpha)}^{\text{Case (e)}} \cdot 1$$

$$r_{\text{pool}} = \overbrace{p_{0'} \cdot \alpha}^{\text{Case (b)}} \cdot 2 + \overbrace{p_{0'} \cdot \gamma(1-\alpha)}^{\text{Case (c)}} \cdot 1 + \overbrace{p_2 \cdot (1-\alpha)}^{\text{Case (g)}} \cdot 2 + \overbrace{P[i>2](1-\alpha)}^{\text{Case (h)}} \cdot 1$$

$$R_{\text{pool}} = \frac{r_{\text{pool}}}{r_{\text{pool}} + r_{\text{others}}} = \frac{\alpha(1-\alpha)^2(4\alpha+\gamma(1-2\alpha)) - \alpha^3}{1-\alpha(1+(2-\alpha)\alpha)}$$

Results, Finally!



Announcement

Bitcoin Is Broken

bitcoin security broken selfish-mining

Ittay Eyal and Emin Gün Sirer — November 04, 2013 at 10:30 AM

← Older — Newer →

Bitcoin is broken. And not just superficially so, but fundamentally, at the core protocol level. We're not talking about a simple buffer overflow here, or even a badly designed API that can be easily patched; instead, the problem is intrinsic to the entire way Bitcoin works. All other cryptocurrencies and schemes based on the same Bitcoin idea, including Litecoin, Namecoin, and any of the other few dozen Bitcoin-inspired currencies, are broken as well.

(link)

Public Discussion

- Incredulity and skepticism in the Bitcoin community
- Vitalik Buterin (at the time editor of Bitcoin Magazine): *"No honest (or semi-honest) miner would want to join a selfish pool," he suggested. "Even if they do have a small incentive to [join], they have an even greater incentive to not break the Bitcoin network to preserve the value of their own Bitcoins and mining hardware."*
- Authors were skeptical about this and other critiques
- Follow-up paper (slides) dealing with the interplay with the difficulty adjustment algorithm

"Uncle" Blocks

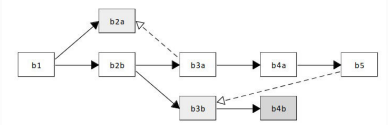


Image from Ritz and Zugenmaier

- Some cryptocurrencies (e.g., Ethereum, which had proof-of-work) reward linking "orphan" blocks in the blockchain
- Ritz and Zugenmaier (slides) show that selfish mining is still profitable: the selfish miner risks less in ties

Did It Really Happen, Though?

- Nobody has observed real-world selfish mining yet
- As described, it is detectable:
  - "Orphan" blocks (outside of the main chain) do happen due to latency
  - There are pattern: longer orphan chains, timing
- Bahrani and Weinberg proposed an "untraceable" variant (video) wrt orphan chains
  - Requires 38.2% of mining power
  - Impossible to differentiate between honest network and slightly increased latency
  - Can it be done to make timing still look natural?

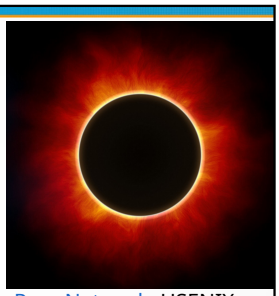
Reasons Why It Didn't Happen

Bahrani and Weinberg (Appendix A):

- No entity has enough hashpower/connectivity to be profitable
  - Makes sense only on large cryptocurrencies like Bitcoin
  - In public pools, you'd need public instructions to selfishly mine
- Engineering costs (e.g., dealing with ASICs) might outweigh benefits
  - Even assuming this is true now, it might not be on the long run
- If it's traceable, there may be targeted repercussion
  - E.g., leaving the pool or not accepting the selfish miners' blocks or transactions
- If it's statistically detectable, the value of the cryptocurrency may suffer as a result

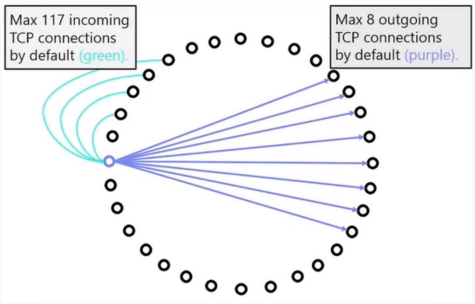
Eclipse Attack

Source

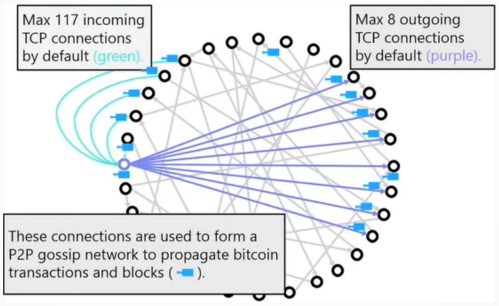


- Heilman et al., Eclipse attacks on Bitcoin's Peer-to-Peer Network, USENIX Security Symposium 2015
  - The link contains both the article and a video of the presentation

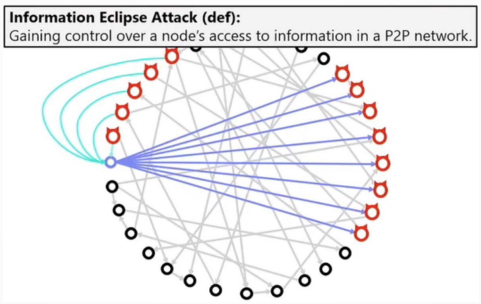
The Bitcoin P2P Network (1)



The Bitcoin P2P Network (2)



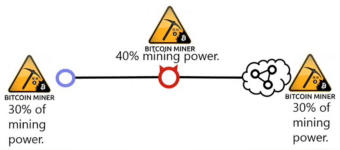
Eclipse Attack (1)



Eclipse Attack (2)

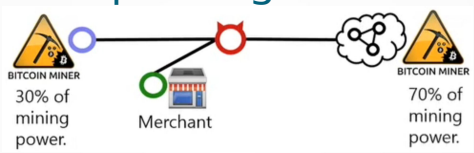
- The attacker **eclipses** the attacked node from the rest of the network
- It can control **everything** the attacked node sees
- Note: the attacker is not **physically** between the attacked node and everybody else, it is only in the **P2P overlay**

51% Attack with Less Mining Power



- The attacker **doesn't relay blocks** between the two sides and **outcompetes both**
- An eclipse attack can be used to **improve selfish mining** by **eclipsing others' blocks**

Double Spending



- Spend some BTCs at the merchant's and **use the eclipsed** ("honest") **miner's blocks** to confirm it
- Spend the same BTCs **somewhere else** in the network
- The first transaction will be in a **shorter blockchain**, hence **cancelled**

"Tried" and "New" Tables

- "Tried": **64 buckets** (size 64 each) containing nodes with which connections were **already established**
- "New": **256 buckets** for not yet tested nodes, gossiped by other nodes ("hey, try these new nodes")
- Each address is split between "first 16 bits" and "rest"; the first 16 (e.g., "130.251" in "130.251.222.83") is the **group** which is hashed to determine 4 buckets, then the full address determines the final one
- A countermeasure against this kind of attack

How the Attack Works

- The attacker sets up a collection of **Sybil nodes**
- They **contact the attacker node** and gossip information about **other attacking or non-existing nodes**
  - Filling up both "tried" and "new"
- Wait for a node **restart**
- If there are enough "groups" and with some luck, the attacked node will **connect only to the attacker**

Restarts

- A restart (disconnection/reconnection) is needed for the Eclipse attack to work
- In the Bitcoin network, machines restart quite frequently
  - Biryukov and Khovratovich show that a machine with a public IP has a 25% chance of going offline in 10h
- Security updates: either restart or stay vulnerable
- Some attacks can force restarts

Is the Attack Realistic?

- The per-IP address group hashing strategy is the strongest countermeasure faced by the attack
- A "normal" attacker will have IP addresses only from a **few groups**, hence filling only a few buckets
- Unfortunately, **botnets** (i.e., groups of vulnerable devices violated and controlled by a single actor) have large IP diversity
- Authors show the attack is **realistic** even for pretty small botnets (by analysis and attacking their own nodes)

Vulnerabilities and Countermeasures

- Authors proposed six countermeasures to patch Bitcoin's client
- They were eventually all included in the Bitcoin Core client
- Interestingly, they are based on ideas inspired by **botnets defenses**
  - Good guys often try to **eclipse botnets** to make them unreachable by the attacker

Countermeasure 1: Eviction

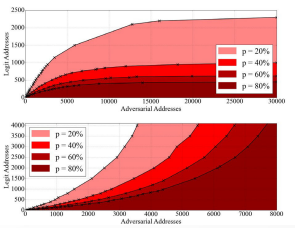
- The "Tried" and "New" tables had this eviction policy, when a bucket got full:
  - Take 4 random IPs in the bucket
  - Evict (remove) the one with the **oldest timestamp** (last time seen/heard of)
  - Put the new node in its place
- The attacker could avoid eviction by making sure its **timestamps were fresher**
- Countermeasure: just hash the IP to get a single slot in the bucket

Countermeasure 2: Peer Selection

- A node would connect to peers randomly, but with a strong bias towards **"fresher" peers** (once again, those with newer timestamps)
- Once again, the attacker can ensure its nodes have fresh timestamps
- Countermeasure: select nodes uniformly at random (i.e., each node in Tried and in New has the same probability)

Countermeasure 3: Test Before Evict

- Rather than evicting nodes in the Tried table, try to connect to them and evict them only if they don't respond
- Plots show the probability of success for the attacker
- If enough honest nodes are in the table, at some point the probability of success "plateaus" for the attacker



Countermeasure 4: Feeler Connections

- Run test connections towards random nodes in New:
  - If they succeed, move the address from New to Tried
  - If they fail, remove them from New
- This countermeasure cleans trash from New and increase the number of nodes in Tried that are likely available when a node restarts

Countermeasure 5: Anchor Connections

- A new table: Anchor nodes
- Inspired by a mechanism implemented in Tor, nodes connect to the two oldest Anchor nodes that accept incoming connections
- The attacker needs to disrupt communications to Anchor nodes to make the Eclipse attack feasible

Countermeasure 6: More Buckets

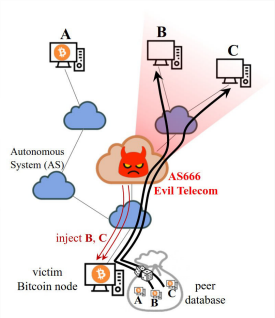
- Quite obviously, more buckets require more Sybil IPs to be filled
- New goes from 256 to 1024 buckets, Tried goes from 64 to 256

Erebus: A More Powerful Attacker

- Sources:
  - Tran et al., IEEE S&P 2020 (attack)
  - Tran et al., USENIX Security 2021 (countermeasure)

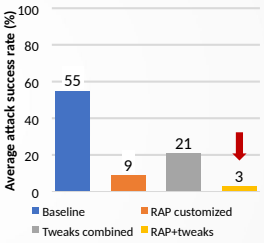
The Attack

- Here we have a **network adversary**: an entity, such as a telecom, that forwards traffic for **many IPs**
- The telecom uses **shadow IPs**, i.e., runs malicious replicas of legitimate IP addresses
- Very difficult to detect



The Defense

- Routing Aware Peering: diversifying based on routing
  - Unfortunately, there are some **false negatives**
  - The attacker can **exploit them**
- Tweaks (changing parameters such as table sizes) to improve resilience
  - Alone, limited effectiveness
- Strategies are effective when combined



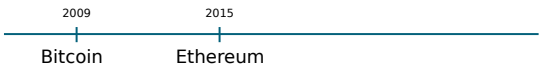
Decentralized Systems

Ethereum

Blockchains

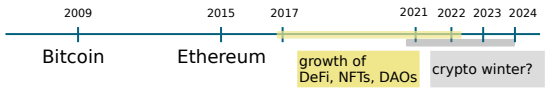
- Public **append-only** data structure, secured by replication and incentives
- Provides **coordination between many parties**, when there is **no single trusted party**
- **Fixed supply asset** (21 million BTC) used for digital payments, and more

Blockchains



- Blockchain computer: a **fully programmable environment**
  ⟹ public programs that manage digital and financial assets
- Composability: applications running on chain can call each other
- *The "slowest computer" in the world*

Blockchains



- **Crypto winter**: prolonged period of pricing weakness in the cryptocurrency market
  - Current crypto winter began in **late 2021**, and it is difficult to say when it will end
  - Need for new killer applications?
  - The market will eventually recover and cryptocurrencies will continue to play a role in the global financial system?

Ethereum

Ethereum is an open source, public, blockchain-based distributed computing platform and operating system featuring smart contract functionality

Vitalik Buterin (https://vitalik.ca/)

Launched on 30 July 2015 with 11.9 million coins pre-mined thanks to a crowd sale

Ethereum

- Designed to create, develop and spread **Smart Contracts**
- Uses its own ledger, different from the Bitcoin blockchain
- Implements a built-in **powerful scripting language** (Turing-complete)

Ethereum cryptocurrency

- The **native currency** of the Ethereum blockchain is called **Ether**
  - listed under the diminutive **ETH** and traded on cryptocurrency exchanges
  - also used to **pay for transaction fees and computational services** on the Ethereum network
- **Oct. 15, 2023:** 1 ETH is $1,554.19 USD (CoinMarketCap)
- **Oct. 15, 2024:** 1 ETH is $2,583.30 USD

Ethereum cryptocurrency

- **Ethereum Classic** (**ETC**) is another blockchain/cryptocurrency **created in 2016** as a result of a **hack of The DAO** (more later on this)
- Theft of over $60 million ETH, and the Ethereum community was divided on how to respond:
  - **reverse the hack** and return the stolen funds (Ethereum)
  - **code is law**, it is not possible to violate the principle of immutability (Ethereum Classic)
- Ethereum Classic is the original Ethereum blockchain, with a smaller community and market capitalization
  - **Oct. 15, 2023:** 1 ETC is $14.99 USD (CoinMarketCap)
  - **Oct. 15, 2024:** 1 ETC is $19.32 USD

Ethereum cryptocurrency

- Many other cryptocurrencies are based on Ethereum
- These are known as **Ethereum tokens**, and they are created and deployed on the Ethereum blockchain. Ethereum tokens can be used for a variety of purposes, such as representing assets, paying for goods and services, or granting voting rights (more later on this)

Ethereum denominations

Unit	Denominations	
Wei	1	1
Kwei	1,000	10^3
Mwei	1,000,000	10^6
Gwei	1,000,000,000	10^9
Szabo	1,000,000,000,000	10^12
Finney	1,000,000,000,000,000	10^15
Ether	1,000,000,000,000,000,000	10^18
KEther	1,000,000,000,000,000,000,000	10^24
MEther	1,000,000,000,000,000,000,000,000	10^24
GEther	1,000,000,000,000,000,000,000,000,000	10^27
TEther	1,000,000,000,000,000,000,000,000,000,000	10^30

https://www.geeksforgeeks.org/what-are-the-different-units-used-in-ethereum/

Ethereum Virtual Machine

- The entire Ethereum **P2P network** is a mass of nodes (computers) connected to one another, all understanding the same protocols
- Can be visualized as a **single** (giant) **entity** called the Ethereum Virtual Machine or **EVM**
- The EVM provides the environment in which users can exchange cryptocurrencies and smart contracts can be deployed and run

Ethereum Virtual Machine

- **Smart contracts** are written in a variety of programming languages, and **compiled into EVM bytecode**, which is the machine-readable code executed by the EVM
- The bytecode has its own **instruction set**, which is used to perform operations such as arithmetic, logical operations, and memory access, including the **JUMP instruction** which is used to implement a variety of control flow statements such as loops, conditional statements, and functions

Ethereum Virtual Machine

- The **state of Ethereum** is a **snapshot of the blockchain** at a given point in time. It includes all account balances, smart contract storage, and other data necessary to execute transactions and run smart contracts
- A **change of state** occurs when a **transaction is successfully processed and added to the blockchain**. This can change the accounts' balances, the storage of smart contracts, or other data in the state

Ethereum Virtual Machine

- The vision of the Ethereum project is to have **one computer distributed across the entire Internet**
- **Each full node joining the Ethereum network** must **keep in its memory** the **whole state** of this Ethereum computer. This state will include
  - all smart contracts bytecode
  - all input and output to the smart contracts (past and present)
  - all communications among smart contracts

Ethereum accounts

- Two types of accounts
  - User account (Externally Owned Account, **EOA**)
  - Contract account (**CA**)
- **EOAs** are similar to Bitcoin accounts
  - 20-byte **address** (hash(S_k))
  - controlled by **key pairs** (ECDSA)
  - **balance**
  - can send messages by creating and signing **transactions**
  - **nonce** (number of transactions coming from that address)

Ethereum accounts

- **CAs** (Contract Accounts)
  - 20-byte **address** (hash(CreatorAddr, CreatorNonce))
  - controlled by their **contract code**
  - can **read** and **write** to internal storage, **send messages** to other contracts or **create new contracts**
  - **nonce** (number of creations of contract with that address)

UTXO vs Account model

- Bitcoin uses the **UTXO model** which is based entirely on **individual transactions**, grouped in blocks
- The **global state** is represented with the **entire graph** of transactions
- Account balances are calculated on the client-side (wallet) by adding the UTXOs
- The verification process checks if transaction output is unspent

https://www.horizen.io/academy/utxo-vs-account-model/

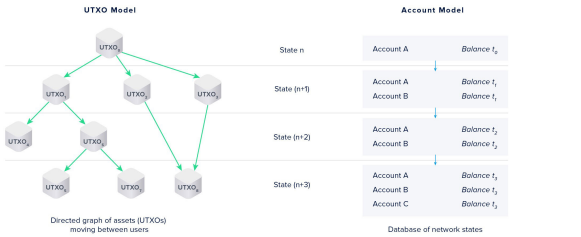
UTXO vs Account model

- Ethereum uses the **account model**, based on balances within accounts, similar to bank accounts
- A transaction in this model triggers nodes to **decrement the balance of the sender**'s account and **increment the balance of the receiver**'s account
- To prevent **replay attacks**, e.g., a malicious node retransmits a valid transaction that has already been executed, when a **wallet** sends a transaction, it **increments the nonce** for the sender's account
- This ensures that each transaction has a unique identifier

UTXO vs Account model

The first significant difference between the two balance models is how the state of the system is recorded

HORIZEN ACADEMY



Transactions

- Transactions are used to send to the EVM
  - **payments**, **smart contracts**, **calls to contracts methods**
- Many fields
  - timestamp
  - sender address and signature
  - receiver address
  - amount of Ether to transfer
  - gas detail
  - optional data
  - nonce

Smart contract: deployment

- To deploy a smart contract, a **transaction** that contains EVM bytecode in its data field is **sent to address 0**
- The contract will be accessible under an **address** that is **derived** from the deploy **transaction sender's address** and their **nonce** (the count of how many transactions they have sent)

Smart contract: validity

- Smart contracts do what they are programmed to do
  - the code **automatically executes exactly as programmed** without possibility of downtime, censorship or third-party interference
- If a "buggy" smart contract has a bad behavior or a contract is deemed invalid in a court, how is this solved?
  - Still an open problem

Ethereum gas

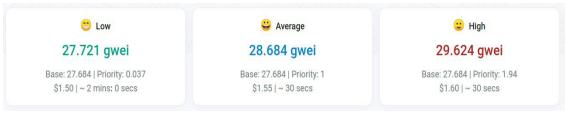
- To send a **transaction** or interact with **on chain applications** requires network's computation and users need to **pay a fee** (for miners/validators)
- Fees are commonly referred to as **gas**
  - gas is essentially a measurement of the computational effort needed to execute an operation on Ethereum

Ethereum gas

- EVM is a **Turing-complete** machine **limited by the amount of gas** required to run any instruction
- The **fee for the execution** depends on the operation performed, for example
  - a single step calculation like if (2 > 1) will cost 1 gas unit
  - a single hash operation will cost 20 gas unit
  - each byte in the data field will cost 5 gas unit

Ethereum gas price

- The **gas price** determines the amount the user pays **per unit of gas** used and does not change the amount of gas needed to execute the transaction



- A sender can specify any gas price they want, according to the current estimation

https://etherscan.io/gastracker

Ethereum gas cost

- Once fixed the gas price, the **gas cost** depends on the number of **gas unit** needed to perform operations on chain
- Different transactions require different amount of computation
  - token transfers require relatively small amount of gas: 21,000 unit
  - more complex transactions need a larger amount of gas unit

Ethereum gas limit

- The **gas limit** is the maximum amount of gas that a user is willing to use for a single transaction
- Ethereum users can specify their desired gas limit when sending a transaction
  - changing the gas limit does not change the actual amount of gas that is needed to execute an operation
  - the gas limit is a **safeguard** that protects users from malicious or buggy applications on chain that may try to use too much gas

Ethereum gas limit

- When submitting a transaction users (their wallets) need to specify the gas limit
  - if the transaction requires more gas than the gas limit, then the transaction will fail
  - any unused gas below the gas limit is returned to the sender's wallet
- **October 15, 2023:** the maximum gas limit for Ethereum blocks is **30 million gas** (after London fork)

Ethereum gas auction

- Why would a user bid to pay a high gas price when they can choose to pay the minimum?

Ethereum gas auction

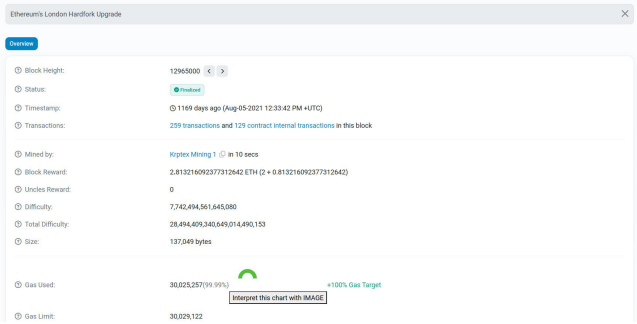
- Why would a user bid to pay a high gas price when they can choose to pay the minimum?

Higher gas leads to faster transaction confirmation, since miners would prioritize more convenient transactions

Ethereum gas

- **Before** London upgrade (hard fork)
  - Users could decide how much they wanted to pay for their transactions and this led to **high fees**, especially during times of high **network congestion**
  - Example
    The gas limit for transactions is 21,000 gas unit
    The chosen gas fee is 20 Gwei per unit
    Total tx fee (gas limit x gas fee) = 21,000 * 20 = 420,000 Gwei (0.00042 ETH)

London upgrade: EIP-1559, August 2021, block 12965000



Ethereum gas

- **After** London upgrade (EIP-1559)
  - Major change to the way fees are computed
  - **Base fee**, e.g., the minimun amount of gas per unit to be paid per each transaction
  - **Adjusted by the protocol**, e.g., when the previous block is filled more than 50%, the base fee of the next block will increase, and vice versa
  - The base fee is **burned** to help ETH to steadily **increase in value**, as supply decreases (deflationary coin)

🔥 0.110079055157238802 ETH

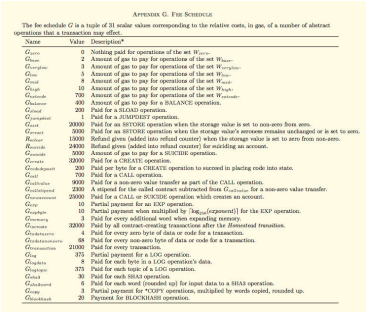
Ethereum gas

- **After** London upgrade (EIP-1559)
  - Users can add a **priority fee**, or **tip**, to **speed up their transactions**
  - The gas limit per block is doubled, but the target gas usage is set to 50% of the new limit
- Many miners were not happy as this upgrade reduced their rewards for completing a new block

Ethereum gas summary

- Price to be paid for the computation performed by the network
- Incentive ensuring that developers write quality applications
  - Infinite loops cannot run forever: **Out-of-Gas exception** once the gas is exhausted
  - **Wasteful code costs more**

Ethereum Yellow Paper



APPENDIX G. FEE SCHEDULE

The fee schedule G is a tuple of 31 scalar values corresponding to the relative costs, in gas, of a number of abstract operations that a transaction may effect.

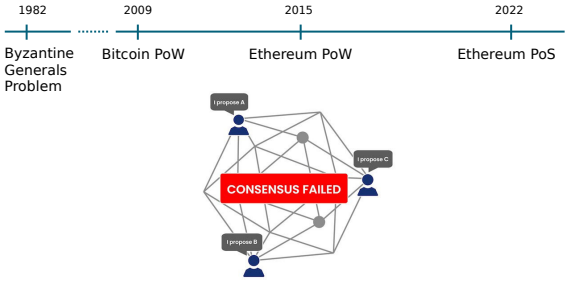
Name	Value	Description*
G_{zero}	0	Nothing paid for operations of the set W_{zero}.
G_{base}	2	Amount of gas to pay for operations of the set W_{base}.
$G_{verylow}$	3	Amount of gas to pay for operations of the set $W_{verylow}$.
G_{low}	5	Amount of gas to pay for operations of the set W_{low}.
G_{mid}	8	Amount of gas to pay for operations of the set W_{mid}.
G_{high}	10	Amount of gas to pay for operations of the set W_{high}.
$G_{extcode}$	700	Amount of gas to pay for operations of the set $W_{extcode}$.
$G_{balance}$	400	Amount of gas to pay for a BALANCE operation.
G_{sload}	200	Paid for a SLOAD operation.
$G_{jumpdest}$	1	Paid for a JUMPDEST operation.
G_{sset}	20000	Paid for an SSTORE operation when the storage value is set to non-zero from zero.
G_{sreset}	5000	Paid for an SSTORE operation when the storage value's zeroness remains unchanged or is set to zero.
R_{sclear}	15000	Refund given (added into refund counter) when the storage value is set to zero from non-zero.
$R_{selfdestruct}$	24000	Refund given (added into refund counter) for suiciding an account.
$G_{selfdestruct}$	5000	Amount of gas to pay for a SUICIDE operation.
G_{create}	32000	Paid for a CREATE operation.
$G_{codedeposit}$	200	Paid per byte for a CREATE operation to succeed in placing code into state.
G_{call}	700	Paid for a CALL operation.
$G_{callvalue}$	9000	Paid for a non-zero value transfer as part of the CALL operation.
$G_{callstipend}$	2300	A stipend for the called contract subtracted from $G_{callvalue}$ for a non-zero value transfer.
$G_{newaccount}$	25000	Paid for a CALL or SUICIDE operation which creates an account.
G_{exp}	10	Partial payment for an EXP operation.
$G_{expbyte}$	10	Partial payment when multiplied by $\lceil \log_{256}(exponent) \rceil$ for the EXP operation.
G_{memory}	3	Paid for every additional word when expanding memory.
$G_{txcreate}$	32000	Paid by all contract-creating transactions after the *Homestead transition*.
$G_{txdatazero}$	4	Paid for every zero byte of data or code for a transaction.
$G_{txdatanonzero}$	68	Paid for every non-zero byte of data or code for a transaction.
$G_{transaction}$	21000	Paid for every transaction.
G_{log}	375	Partial payment for a LOG operation.
$G_{logdata}$	8	Paid for each byte in a LOG operation's data.
$G_{logtopic}$	375	Paid for each topic of a LOG operation.
G_{sha3}	30	Paid for each SHA3 operation.
$G_{sha3word}$	6	Paid for each word (rounded up) for input data to a SHA3 operation.
G_{copy}	3	Partial payment for *COPY operations, multiplied by words copied, rounded up.
$G_{blockhash}$	20	Payment for BLOCKHASH operation.

Decentralized Systems

Ethereum (cnt)

Consensus in Ethereum

Consensus algorithms



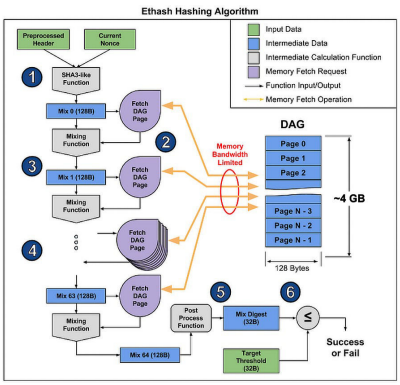
Past: Proof-of-Work

- **Ethash** is the PoW algorithm **previously used** in Ethereum
- **Memory-hard** algorithm, it requires a large amount of memory to mine efficiently (ASIC resistant)
  - miners must download and store a **randomly generated dataset** known as **DAG** (Directed Acyclic Graph)
  - a **new DAG** file of increasing file is generated every **mining epoch** (**30000 blocks**) , and miners must always use the most recent version
  - see https://minerstat.com/dag-size-calculator

Past: Proof-of-Work

- The difficulty of the PoW dynamically adjusts such that, on average, **one block** is produced by the network **every 12-15 seconds**
  - Orphan blocks, mined but not included on the main chain
  - Transactions included in an orphan block will not be finalized until they are included in a block on the main chain
- Miners compute the **mix hash** (Keccak-256) to reach the target and win the block reward
- Also in this case, difficult to compute, easy to verify

Past

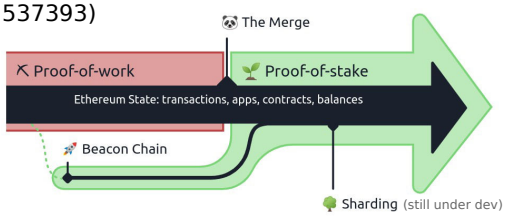


Past: Proof-of-Work

- Steps for the miners
  - select a random starting point (page) in the DAG
  - follow a path through the DAG, visiting each node (page) only once
  - compute the Keccak-256 hash of the node's data and the mix hash from the previous step
- Compare the final mix hash with the target

Now: Proof-of-Stake

- Sept 15, 2022: **The Merge** (block number 15537393)



https://ethereum.org/en/upgrades/merge/
https://www.youtube.com/watch?v=EEuPmA8w0Kc

Now: Proof-of-Stake

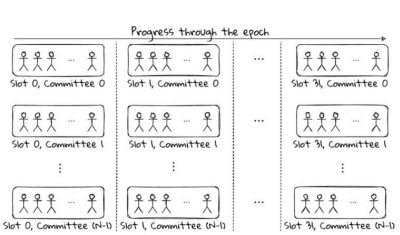
- To participate in PoS validation, users must **stake at least 32 ETH**, **individually** or through a **staking pool**
  - staked ETH is locked up and validators must be online to participate in the consensus process
- **Validators** are **randomly selected** to propose and attest new blocks
- If a validator proposes an **invalid block** or attempts to **double-spend** their ETH, they are **penalized**: some locked coins are burned! This is called **slashing**

Now: Proof-of-Stake

- The algorithm works in **epochs of 32 slots** (12 sec per slot)
- For each epoch, **32 committee** are formed, their **128 members** being randomly selected from the pool of all active validators using a pseudo-random function
  - this ensures that the committees changes dynamically from epoch to epoch, preventing centralization of power within the network
- Committees
  - **elect their proposers** responsible of the **next block** in their slot
  - **vote to validate their block** (attestation)

Now: Proof-of-Stake

- Proposers and validators are rewarded for their job



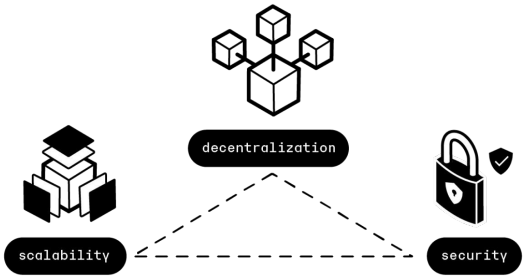
PoW vs PoS

- **Gas fees will remain the same** because the Merge does not affect the execution layer where the fees are determined
- **Transaction speed might slightly improve** but will not dramatically increase (a new block every 12 sec)
- **Malicious validators are slashed**, e.g., a significant part of their stake can be burned, up to the whole stake of 32 ETH in the worst case
- **Note: staked ETH can now be withdrawn** (after Shanghai upgrade, April 12, 2023, block 16112764)

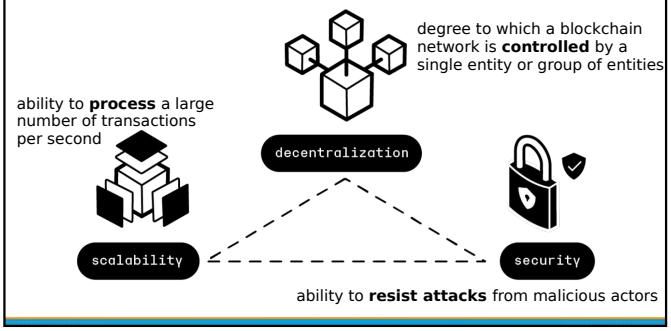
Blockchain Trilemma

- Ethereum improved by 99% (not sure!) in terms of energy consumption after The Merge, and set the stage for future upgrades that will make it faster and more scalable (sharding)
- Attempt to tackle the Blockchain (or Scalability) Trilemma

Blockchain Trilemma



Blockchain Trilemma



Tokens

Tokens

- **Tokens can represent anything** physical or digital goods, e.g., a ticket for a concert, a collectible, points at the supermarket, casino coins, etc.
- In Ethereum they
  - are managed by the underlying blockchain
  - can be issued with a **few lines of code**
  - are accessible with a (often dedicated) wallet
  - https://etherscan.io/tokens

Tokens

- Can be implemented on different layers of the technology
  - **Protocol tokens**
  - intrinsic, native, or built-in tokens, have the role to **keep the network safe** from attacks (mining rewards) and for **preventing transactions spam** (transaction fees)
  - BTC, ETH

Tokens

- Can be implemented on different layers of the technology
  - **Second-layer tokens**
  - these are application tokens which **can be created via a smart contract** (Ethereum) or issued in a Layer 2 network

Tokens

- Think of Ethereum like the internet and all the dApps as websites that run in it
- dApps are decentralized, not owned by a single entity, they are owned by people
- This usually happens by a crowd-sale called the **ICO** (Initial Coin Offer) or the **ITO** (Initial Token Offer)

Tokens

- ICOs have changed the way projects are funded
  - Assemble your team and create a product
  - Create the Tokens to be sold during the ICO
  - Write the White Paper
  - Run a well designed website and social channels
  - ...

Ethereum tokens

- Various standards
  - **ERC-20** for fungible tokens (ERC-223, ERC-777)
  - **ERC-721** for non fungible tokens (NFTs)
  - **ERC-1551** supports non-fungible and fungible tokens

ERC = Ethereum Request for Comments

ERC-20

- Defined in https://eips.ethereum.org/EIPS/eip-20
  - They have a property that makes **each token be exactly the same** (in type and value) **of another token**
  - They have functionalities to transfer tokens from one account to another, to get the current token balance of an account, to delegate other accounts to spend tokens
  - See https://www.openzeppelin.com/

ERC-721, known as NFT

- Defined in https://eips.ethereum.org/EIPS/eip-721
  - A non-fungible token (NFT) is a **special type** of cryptographic token which **represents something unique**
  - NFT have varying properties, and **represent scarce assets** like art, collectibles or real estate
  - Can also represent **identities**, **certificates** (licenses, degrees), voting rights, medical data, etc.

ERC-721, known as NFT

- See for example
  - https://www.cryptokitties.co/
  - https://www.larvalabs.com/cryptopunks
  - https://opensea.io/
  - https://www.coingecko.com/research/publications/10-most-expensive-nfts-ever-sold

Data Structures

State machine

- Ethereum uses the idea of the **world state** which is stored as a mapping between account addresses and account states (balance, contract code, contract storage)

Source: https://ethereum.org/en/whitepaper/

State machine

- Users send **transactions** Tx[i] which force nodes in the network to **change state** S[i+1]
  - Owned → Owned: transfer ETH between users
  - Owned → Address 0: deploy contract
  - Owned → Contract: call contract with ETH & data
  - Contract → Owned: contract sends funds to user
  - Contract → Contract: one program calls another (can send funds)

State machine

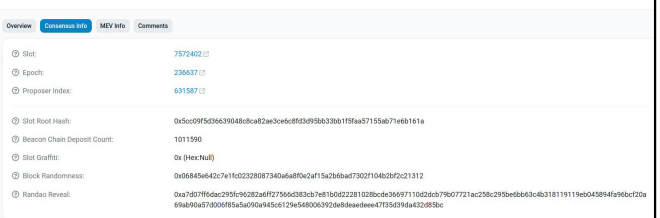
- Validators collect transactions Tx[0]...Tx[n-1] from users and the Proposer creates a new block
- To produce a block the Proposer
  - for i=0,...,n-1: execute (APPLY) state change of Tx[i] sequentially (can change state of > n accounts)
  - record updated world state in the block
- Other validators re-execute all Txs to verify the block and, if valid, sign it; enough signatures → epoch is finalized

World state data structure

- The world state is stored in a optimized tree structure known as **Merkle Patricia Trie**
  - **P**ractical **A**lgorithm **T**o **R**etrieve **I**nformation **C**oded **I**n **A**lphanumeric
  - Tree-like data structure in which each node represents a prefix of a string. The root node represents the empty string, and the child nodes represent the strings that start with the prefix of the parent node
  - Plus hashes... (Merkle tree)

Block header data (simplified)

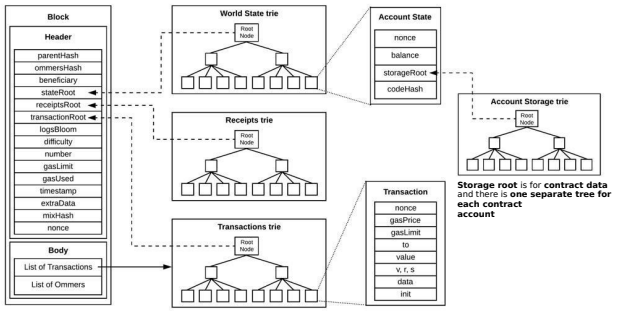
(1) Consensus Info: proposer ID, parent hash, etc.



Block header data (simplified)

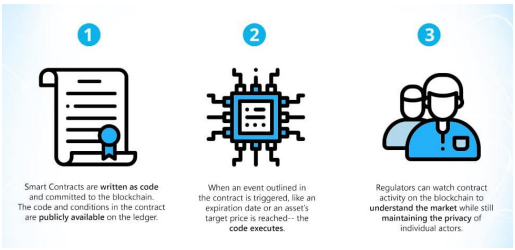
(2) Address of gas beneficiary: where fees will go
(3) Gas used: used to adjust gas price

(4) Root hashes



Smart Contract

Smart Contract



Smart Contract

- 1993: Nick Szabo defined **smart contracts** as a set of **promises**, specified in digital form, including **protocols** that facilitate, verify, or enforce how parties perform these promises
  - A set of promises (IF…THEN…ELSE)
  - Digital form
  - Protocols for communication and performance
  - Self executing program

Smart Contract

- Everyday example
  - the vending machine

- A modern example
  - installment payment of a car: a smart contract is able to check if the user has regularly payed the installment
  - if the check is negative then the smart contract renders the car inoperable

Smart Contract

Turing complete

- Any instruction can run
- Maximum freedom
- Easiest to make mistakes

Restricted Instructions

- Limited instructions
- Purpose-specific actions
- Hard to make mistake
- Harder to hack

Off-Chain execution

- Contract code is not on blockchain
- Code distribution has been coordinated
- Hard to make mistakes

On-Chain execution

- Contract code is stored on blockchain
- Code changes are part of blockchain consensus
- Very transparent

Smart Contract

- Ethereum is the **largest ecosystem of decentralized applications** or dApps built on a **decentralized network**
- Smart contracts on Ethereum are **public** and **transparent** – like open APIs – so dApps can also reuse code written by others
- Smart contracts are as good as the developers building them: “buggy” smart contracts stay in the blockchain forever

Web2 vs Web3 (bard)

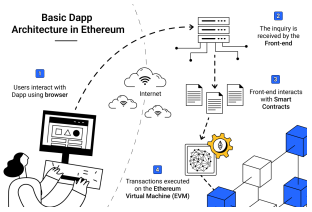
Feature	Web 2 app	Web 3 dApp
Ownership	Centralized	Decentralized
Data storage	Centralized	Decentralized
Control	Controlled by a single entity	Controlled by a network of computers
Security	Vulnerable to hacks and data breaches	More secure and resistant to censorship
Transparency	Not transparent	Transparent and verifiable
Accessibility	More accessible	Less accessible, requires some technical knowledge

Popular dApps (bard)

- **DeFi (Decentralized Finance)**
  - Uniswap
  - Aave
  - MakerDAO
  - Compound
  - SushiSwap
  - Curve Finance
  - PancakeSwap
- **Gaming**
  - Axie Infinity
  - Decentraland
  - Gods Unchained
  - Alien Worlds
  - Dark Country
  - My Crypto Heroes
  - Chainmonsters
- **NFTs (Non-Fungible Tokens)**
  - OpenSea
  - Rarible
  - LooksRare
  - SuperRare
  - Axie Marketplace
  - Magic Eden
  - SolanaArt
- **Other**
  - Brave Browser
  - Filecoin
  - Storj
  - Golem
  - Audius
  - Livepeer
  - Ocean Protocol

Activity

- Please select one dApp each, google 5 minutes and tell the class what it is about



https://info.etherscan.com/decentralized-applications-an-overview/

Decentralized Systems

Solidity & Remix demo

Write Smart Contract

Solidity

- Introduced by the Ethereum foundation to write smart contracts
- To work with Solidity we need
  - A framework to **write** and **compile** a contract
  - An Ethereum node to **deploy** the contract
  - A library to build the **front-end** and to **interact** with the contract through JSON-RPC calls

Solidity

- Extension **.sol**
- Needs
  - **License** (// SPDX-License-Identifier: MIT)
  - **Version** (**pragma** solidity *version_number)*
- **Contracts** are similar to **classes** and contain declaration of
  - Typed variables
  - Constructors
  - Functions and function modifiers
  - Events

Solidity: data location

- **Data location**
  - **Storage**: for permanent data (stored on blockchain, expensive)
  - **Memory**: used to save temporary variables during function execution (often required in return parameters)
  - **Calldata**: non-modifiable and non-persistent data location, default location for function arguments

The amount of gas used during a transaction depends on the data location used in the smart contract. The best practice is to write an optimized code that uses a minimum amount of storage

Solidity: variables

- **Variables**
  - **Private**: accessible only by functions within the contract itself
  - **Public**: part of the **contract interface**, can be accessed both inside and outside of the contract
  - **Internal**: (default) can only be accessed by functions within the contract itself, and by functions in derived contracts

It is important to choose the correct visibility for Solidity variables, as this can have a significant impact on the security of the contract

Solidity: functions

- **Functions**
  - **Private**: can only be called inside the current contract
  - **Public**: part of the contract interface, they can be called via transactions and from other contracts
  - **View**: do not modify the state
  - **Pure**: do not modify the state and do not read from the state
  - **Payable**: can receive Ether

Use the payable modifier only when necessary and be careful with reentrancy attacks (more later)

Solidity: contract scheleton

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.8.2 <0.9.0;
contract DummyContract {
    state variables
    events
    modifiers
    constructor
    functions
}
```

Solidity: contract scheleton

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.8.2 <0.9.0;
contract DummyContract {
    state variables
    events
    modifiers
    constructor
    functions
}
```

Solidity: data types

- **Boolean** (bool; default false)
- **Integer** (int8,..., int256, uint8, uint256; default 0)
  - The use of uint prevents programming errors for values which cannot be negative
  - In case of int, uint the compiler allocates 256 bits

Overflows of integer variables were exploited by malicious actors to attack smart contracts (today not possible anymore)

- **Strings** (text data; default “”)
- **Bytes** (raw binary data; default 0x00)

Solidity: data types

- **Array** (values of the same type)
  - uint256[ ] lottery;
  - string[12] months;
- **Struct** (group of related variables)
  - struct Player {
    string playerNickname;
    uint8 intelligence;
    uint8 strenght;
    }
    Player player1;
    Player player2;

Solidity: data types

- **Address** (store Ethereum addresses, 20 bytes)
  - Built in function **balance** to read the balance of a given account
- **Mapping** (key/value pairs)
  - Example: we want to keep track of the players in a game

    1) Player[ ] players
    (integer indexes in the array)
    2) mapping (address => Player) players
    (we can keep track of players’ addresses)

Solidity: constant

- **Constant**
  - *Variables whose values cannot be changed after initialization*
  - **Constant** variables are initialized at **compile time**

    Examples:
    uint constant MAX_SUPPLY = 100000;
    address constant OWNER_ADDRESS = 0x1234567890....;
  - Stored in the contract's **bytecode**, they can be accessed without accessing the contract's storage
  - **Do not consume any gas when accessed**, since they do not require a storage read

Solidity: immutable

- **Immutable**
  - *Variables whose values cannot be changed after initialization*
  - **Immutable** variables are **initialized during contract deployment**, in the contract constructor (see later)

    ```
    Examples:
    uint immutable InitialBalance;
    address immutable OwnerAddress;
    ```
  - Saved in the **bytecode**, they are useful for storing values such as addresses, hashes, and other **data that do not need to change over time** but are not known at compile time

Solidity: variable scope

- **Global** scope
  - **State variables**, stored in contract **storage**, are accessible from all functions in the contract
  - Special **built-in variables** are global (accessible from all contracts)
    - msg.sender
    - address.balance
    - block.timestamp
- **Local** scope
  - Variable defined into functions, stored in **memory**

Solidity: constructor

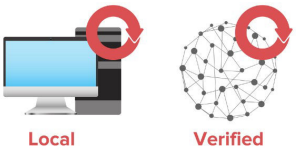
- **Constructor**
  - optional function that is executed upon contract creation

```
contract DummyContract {
    address public immutable OwnerAddress;
    constructor() {
        OwnerAddress = msg.sender;
    }
    …
}
```

Known only at deploy time

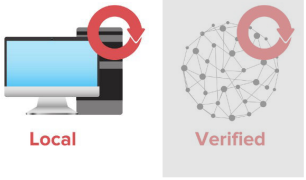
Solidity: execution

Smart contract functions have two modes of execution



Solidity: call

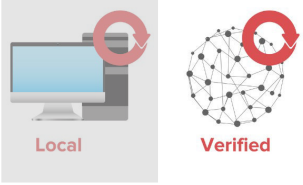
Smart contract functions have two modes of execution



- A **call** is a local invocation of a function that **does not broadcast or publish anything on the blockchain**
- **Read-only** operation, does **not consume any gas**
- It is **synchronous** and the return value is sent back immediately
- Used for view and pure functions
- Its underlying JSON-RPC is eth_call

Solidity: transaction

Smart contract functions have two modes of execution



- A **transaction** is **broadcast to the network**, processed by validators
- If **valid,** it is **added to a block**
- **Write-operation** that updates the state of the blockchain and **consumes gas**
- It is **asynchronous**
- The **immediate return value** is always the **tx's hash**
- Its underlying JSON-RPC is eth_sendTransaction

Solidity: compiler

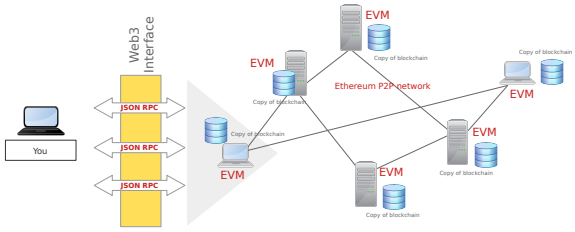
- Produces
  - the **bytecode**, e.g., the executable code on the EVM
  - the Application Binary Interface (**ABI**), e.g., an interface to interact with the EVM bytecode (JSON-like syntax)

Deploy Smart Contract

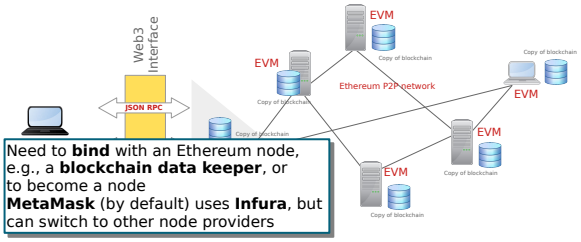
Deploy smart contract

- You need
  - bytecode
  - ABI
  - wallet with a balance
  - infrastructure

Infrastructure



Infrastructure



Need to **bind** with an Ethereum node, e.g., a **blockchain data keeper**, or to become a node
MetaMask (by default) uses **Infura**, but can switch to other node providers

Software

- Huge ecosystem…
- We will start with
  - Remix IDE (to write, compile, deploy demo smart contracts)
  - Metamask (to transact on the testnet)
  - Sepolia testnet (to see what happens)

Remix IDE

- You can access the Remix IDE in two different ways:
  - via a **web browser** (https://remix.ethereum.org/)
  - from a locally installed copy (npm install remix-ide -g)
- It provides various tools to write, compile, and deploy smart contracts

Remix IDE

- Remix Execution Environments
  - **JavaScript VM**: a sandbox blockchain implemented with JavaScript in the browser to emulate a real blockchain
  - **Injected Web3**: a provider that injects web3 such as Metamask, connecting you to your local blockchain or to a test net
  - **Web3 Provider**: a remote node with geth, parity or any Ethereum client. Can be used to connect to the real network, or to your private blockchain directly without MetaMask in the middle

Useful links

- https://solidity-by-example.org/
- https://cryptozombies.io/
- https://ethernaut.openzeppelin.com/

Decentralized Systems

Solidity (cnt'd)

Solidity: contract scheleton

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.8.2 <0.9.0;
contract DummyContract {
    state variables
    events
    modifiers
    constructor
    functions
}
```

Solidity: modifiers

- Modifiers can **change the behavior of functions**
- Called before function execution, for example to restrict access, validate inputs, guard against reentrancy hack
  1. **Declaration** with the keyword **modifier**
  2. **Usage** with the **name of the modifier** in the declaration of the function; the modifier's code is executed before the code in the function itself
- **Can be reused** in the smart contract

Solidity: modifiers

```
// Modifier to restrict the access only to the owner
modifier onlyOwner() {
  // Check if the caller is the owner
  require(msg.sender == owner, "Not authorized");
  _;      // Continue execution of the function
}
```

See examples at https://solidity-by-example.org/function-modifier/

Solidity: events

- Used to **log** and **broadcast information** from a smart contract to the outside world, they notify external applications of important actions within the contract
  1. **Declaration** with the keyword **event**
  2. **Usage** with the keyword **emit** inside the body of a function
- **Each emitted event**
  - generates a **log entry** on the Ethereum blockchain
  - has a **cost** in **gas**

Solidity: events

- Smart contract WishOfDay: we can add one event emitted each time a new wish is written in the blockchain

Solidity: events

- Smart contract WishOfDay: we can add one event emitted each time a new wish is written in the blockchain

1. **Declaration**

```
event WishAdded(uint256 _data, string _message, string indexed _author);
```

The keyword indexed is used to make authors filterable when querying the event logs

Solidity: events

- Smart contract WishOfDay: we can add one event emitted each time a new wish is written in the blockchain

1. **Declaration**

```
event WishAdded(uint256 _data, string _message, string indexed _author);
```

The keyword indexed is used to make authors filterable when querying the event logs

2. Usage

```
function setOneWish(string memory _message, string memory _author)
public {
 ...
 ...
  emit WishAdded(block.timestamp, _message, _author);
}
```

Solidity: events

- Events contribute to the **transparency** and **accountability** of smart contracts since they provide a **historical record of actions** and state changes within a contract, making it easier to verify the correctness of a contract's behavior
- Powerful tool that helps make smart contracts more interactive and responsive, bridging the gap between on-chain and off-chain systems

Solidity: transaction logs

- **Transaction logs**
  - are **stored in a special area** of the blockchain called the transaction log, which is **accessible externally** but **not within smart contracts** themselves
  - are **cheaper to store** compared to storing data in the contract storage
  - consist of **log entries**, each of which can have multiple **topics** and **data fields**

Solidity: transaction logs

- A **log entry** has the following fields
  - **From:** the address of the smart contract that generated the log entry
  - **Topic:** event signature (hash of the event's name and its parameter types)
  - **Event:** the event name
  - **Args:** the parameters associated with the event. At most three parameters can be **indexed**, and they can be used for filtering *(indexed parameters are hashed and placed in separate topics to make searching for these values efficient)*

Solidity: payable

- Functions and addresses declared **payable** can **receive and handle Ether**
- Ether sent to a payable function is read in **msg.value**
- The balance of a contract is read in **address(this).balance**
- **tranfer()** is the preferred way to send Ether, automatically reverts transactions in case of errors
- **send()** is less common, returns a boolean value to indicate success or failure, and it requires manual checking of the return value to handle potential errors

Solidity: payable

- A contract receiving Ether can also use the functions
  - **receive()** external payable
  - **fallback()** external payable

- receive() is automatically called (if present) whenever a contract receives a message with empty calldata, e.g., when Ether is sent to the contract without a specific function call
- fallback() is automatically called (if present) whenever a contract receives a message that is not handled by any of the contract's other functions; it does not take any arguments and it is no longer considered a best practice to handle Ether with fallback()

Solidity: errors

- Compile time errors (easier to correct)
- Run time errors
  - Out of gas
  - ~~Overflow and Underflow errors~~
  - Revert errors

Solidity: errors

- **revert()**, **require()**, and **assert()** are three different functions used to handle different types of conditions and errors in Solidity

Solidity: errors

- **revert()**, **require()**, and **assert()** are three different functions used to handle different types of conditions and errors in Solidity
- **revert(condition,message)**
  - if condition is **false**, cancel and **revert** the current transaction, **without returning any remaining gas** to the sender; send a message to explain the reason of the revert
  - if condition is **true**, **continue**
- On Etherscan it is possible to find out why a transaction was reverted

Solidity: errors

- **revert()**, **require()**, and **assert()** are three different functions used to handle different types of conditions and errors in Solidity
- **require(condition,message)**
  - used for validating user inputs and contract state to ensure that certain conditions are met
  - if condition is **false**, the transaction is **reverted** and the **remaining gas is sent back to the sender**; send a message to explain the reason of the revert
  - if condition is **true**, **continue**

Solidity: errors

- **revert()**, **require()**, and **assert()** are three different functions used to handle different types of conditions and errors in Solidity
- **assert(condition)**
  - used to check for **conditions that should never evaluate to false**;
    if the condition is false, the transaction will be reverted, and all remaining gas consumed; no message is sent back to the sender
  - used for debugging and ensuring the contract's internal consistency, catches **situations that should never occur** in a correctly functioning contract

Solidity: assert() example

```
function sum (uint256 a, uint256 b) public pure returns (unit256) {
    uint256 result = a + b;
    assert (result >= a);   // if false, possible overflow :(
    return result;
}
```

- If assert() sees something “wrong”, halts the execution and burns all remaining gas

Solidity: SafeMath library

- Before Solidity 0.8, this library was used for **preventing integer overflow and underflow** in arithmetic operations
  https://docs.openzeppelin.com/contracts/2.x/api/math#SafeMath
- Unexpected behavior can lead to security vulnerabilities, especially when dealing with tokens or financial transactions

Solidity 0.8 added **built-in overflow and underflow checks** to the language level.
This means that the compiler will automatically check for these errors and revert the transaction if they are found.

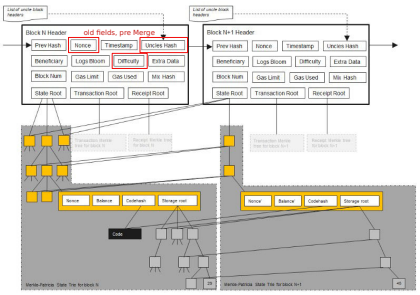
Solidity: libraries

- Solidity libraries can be included in smart contract using the **import** statement (with local and external files)

When choosing a (Solidity) library consider its

- **security**: the library should be audited and have a good reputation
- **gas efficiency**: the library should be gas-efficient to minimize the cost of deploying and using the library
- **community**: the library should have a large and active community that can provide support

Abstract view of the storage



Decentralized System

Solidity (cnt’d)

Verify Smart Contract

Verification what and why?

- Smart contract verification is the process of **matching** a smart contract’s **source code** to its **on-chain bytecode**
- Important step to ensure its **transparency** and **security**
  - The source code becomes visible in the blockchain
  - Full advantage of **tools that rely on source code**
  - Contribute to the **trustlessness** of the entire ecosystem

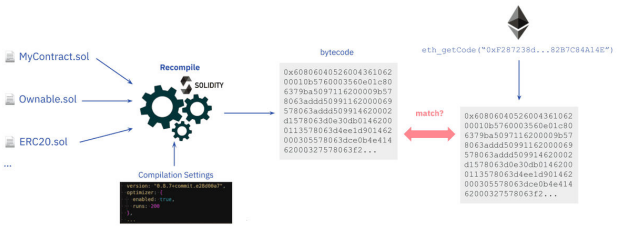
Verification how to?

- **Steps**
  - Compile the Solidity code
  - Obtain the bytecode and the ABI
  - Deploy the bytecode to the Ethereum blockchain (testnet)
  - Visit the Explorer (Etherscan) and find the page of the contract using its address

Verification how to?

- **Steps**
  - Open the contract details page and click on the “Verify and Publish” link
  - Input verification details: solidity compiler version, source code, appropriate license, any optimization settings
  - Enter constructor arguments, if any, in the correct order in the verification form
  - Click the “Verify and Publish” or equivalent button to submit the contract for verification

Verification how to?



You can also use Remix for Smart Contract verification, by installing a proper extension

Decentralized Systems

Smart Contract (Standards for tokens)

Blockchain tokens

- **Digital representations of assets** that exist on a blockchain
  - can represent different assets, from cryptocurrencies to digital representations of physical assets, ownership rights, or even access to specific services
- **Secured through cryptographic techniques**, they are tamper-resistant and guarantee the integrity of the transactions and ownership
- Built thanks to standard **smart contracts**

ERC-20

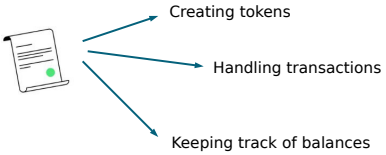
ERC-20 Tokens (2015)

- ERC-20 tokens
  - blockchain-based **assets** that have value and can be sent and received
  - known as **fungible tokens**, ERC-20 tokens are **identical** from one another
  - **represent virtually anything**, e.g., lottery tickets, points on an online platform, skills for a character in a game, fiat currency, etc.

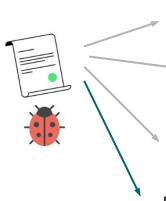
ERC-20 Tokens

- They are indistinguishable, interchangeable, divisible
- Can be **transferred** from accounts
- It is possible to
  - get the **current token balance** of an account
  - find out **how much** of a token’s entire supply **is available**
  - know if **certain quantity** of tokens from one account **can be spent by a third-party account**

ERC-20 Tokens



ERC-20 Tokens



Creating tokens

Handling transactions

Keeping track of balances

Be careful that buggy smart contracts, and therefore buggy tokens, once published will remain on the blockchain forever

ERC-20 Tokens

Ethereum Improvement Proposals

All Core Networking Interface ERC Meta Informational

Standards Track: ERC

ERC-20: Token Standard

Authors Fabian Vogelsteller <fabian@ethereum.org>, Vitalik Buterin <vitalik.buterin@ethereum.org>

Created 2015-11-19

Table of Contents

Simple Summary

A standard interface for tokens.

https://eips.ethereum.org/EIPS/eip-20

ERC-20 Tokens

- The ERC-20 standard implements an **API for tokens**

Optional	**Required**	**Not standard**
• name	• totalSupply	• mint
• symbol	• balanceOf	• burn
• decimals	• transfer	*Only the owner can create (mint) or destroy (burn) tokens*
	• transferFrom	
	• approve	
	• allowance	

ERC-20 Tokens

- Go to Etherscan https://etherscan.io/tokens
- Check some smart contract
- Even without any real ETH, you can make calls (Read Contract)

Build your token!

- RibbaToken: 
- Following the standard, a token has
  - A totalSupply (mandatory)
  - A name (optional), e.g. RibbaToken
  - A symbol (optional), e.g. RIB
  - A decimal value (optional)

Build your token!

- RibbaToken:  Transfer
  - function **balanceOf**(address _owner) constant returns (uint256 balance);
  - function **transfer**(address _to, uint256 _value) returns (bool success);
  - event **Transfer**(address indexed _from, address indexed _to, uint256 _value);
  - balanceOf uses a state variable (mapping)

address1	address2	address3	.....
Amount of RibbaToken	Amount of RibbaToken	Amount of RibbaToken	.....

Build your token!

- RibbaToken:  Delegated transfer
  - function **approve**(address _spender, uint256 _value) returns (bool success);
  - function **transferFrom**(address _from, address _to, uint256 _value) returns (bool success);
  - function **allowance**(address _owner, address _spender) constant returns (uint256 remaining);
  - event **Approval**(address indexed _owner, address indexed _spender, uint256 _value);
  - allowance uses a state variable (mapping of mapping)

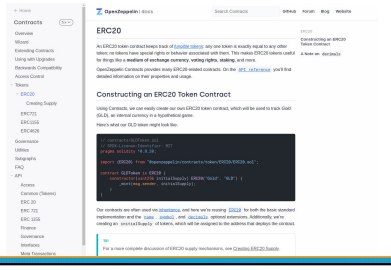
address1		address2		address3		.....
address2	Delegated RibbaToken	address1	Delegated RibbaToken	address2	Delegated RibbaToken	.....
address3	Delegated RibbaToken			address4	Delegated RibbaToken	
address4	Delegated RibbaToken					
address5	Delegated RibbaToken					

Build your token!

- RibbaToken:  Events
  - **event** Transfer(address indexed _from, address indexed _to, uint256 _value);
  - **emit** Transfer(msg.sender, _to, _value);
- Signal that “something” (a transaction) took place
- Stored in the blockchain, they are a good way of broadcasting relevant information
- Cannot be accessed by other smart contracts, but only from external applications

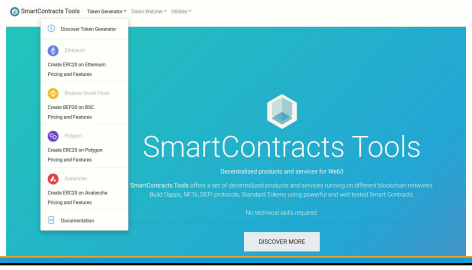
Build your token!

- You can use libraries, see for example: https://docs.openzeppelin.com/contracts/2.x/erc20



Build your token!

- Or online tokens generators, see for example: https://www.smartcontracts.tools/



ERC-721 Tokens (2018)

- Standard that describes how to build **non-fungible** or unique tokens (**NFT**), they are the **digital collectibles of the 21st century**
- NFT ownership is recorded on public blockchains, allowing **anybody to easily verify** their legitimacy and ownership at any moment
  - Defined in EIP-721: https://eips.ethereum.org/EIPS/eip-721

ERC-721 Tokens

- An NFT token represents **ownership of "something"**, for example an image
  - function ownerOf(uint256 _tokenId) external view returns (address);
- NFT files are too large to be stored directly on the blockchain
- They are stored "somewhere" else and the files' locations (among other details) are stored in the NFT metadata

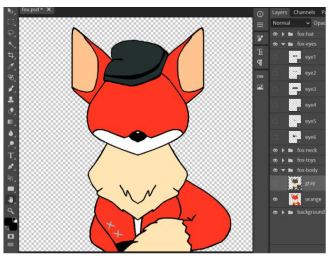
ERC-721 Tokens

- **Metadata** are **JSON documents** that contain various **information**
  - NFT's name
  - description
  - link to the file
- **Traits**, like
  - background color, clothing, facial expressions, accessories
  - rarity levels like common, rare, epic, legendary
  - strength, speed, other powers

ERC-721 Tokens

- In generative NFT collections, traits are often combined algorithmically
  - A base set of traits is predefined and a computer program mixes and matches them to create thousands of unique NFTs, each with its own trait combination
- NFT metadata are the input of the smart contract
- Despite NFTs and metadata are **stored off-chain**, they are **minted** on-chain

Build your collection



```
const layerConfigurations = [
  {
    growEditionSizeTo: 3000,
    layersOrder: [
      { name: "Background" },
      { name: "fox-body" },
      { name: "fox-eyes" },
      { name: "fox-hat" },
      { name: "fox-toys" },
      { name: "fox-neck" },
    ],
  },
];
```

Build your collection


```
"attributes": [
  {
    "trait_type": "Background",
    "value": "bgcolor3"
  },
  {
    "trait_type": "fox-body",
    "value": "white"
  },
  {
    "trait_type": "fox-eyes",
    "value": "eye5"
  },
  {
    "trait_type": "fox-hat",
    "value": "hat6"
  },
  {
    "trait_type": "fox-neck",
    "value": "neck3"
  },
  {
    "trait_type": "fox-toys",
    "value": "toy1"
  }
],
```


ERC-1155 Tokens

- Token standards like ERC-20 and ERC-721 require a separate contract for each token type or collection
- With ERC-1155 it is possible to have **multiple tokens in the same contract** and also to add new ones later on
- **Less contracts** means **less space** in the blockchain (less gas)
- More complex to implement, are becoming more and more popular

ERC-1155 Tokens

- ERC-1155
  - **enables batch transfers**, which allow multiple tokens to be sent in a single transaction
  - **supports atomic swaps**, allowing users to trade one type of token for another in a single, trustless transaction
- Especially useful for gaming, where users might want to exchange one type of item for another, without relying on an intermediary
- See https://www.youtube.com/watch?v=FkUn86bH34M

Decentralized System

Solidity (cnt'd)

Payable smart contract

- RibbaShop is an off-line shop that sells candies and stickers in change of Sepolia wei
- It has an initial stock for candies and stickers, and some rules
  - For each transactions it is possible to buy
    - Only one sticker
    - At most five candies
  - Each address can own at most three stickers
  - There is no limit on the number of candies (if they are available in stock)

Payable smart contract

- Only the owner of the smart contract can refill the stocks (both candies and stickers)
- Only the owner of the smart contract can transfer the balance of the smart contract to their own address

Tasks

- **Audit the code** of the smart contract to check for any vulnerabilities or missing functions
- **Add the following functionalities**
  - Stickers can be transferred to other addresses (easy)
    (of course not only on the blockchain but also off-chain)
  - Stickers can be sold to other addresses, possibly with a different price (more difficult)

Audit of smart contracts

- **Manual Code Review**
  - Check for naming conventions and comments; code should be understandable to other developers
  - Review each function to confirm it meets the intended logic
- **Automated Analysis**
  - Use automated tools to scan the contract and catch common vulnerabilities, gas inefficiencies, and areas prone to reentrancy attacks
  - Review automated findings

Decentralized Systems

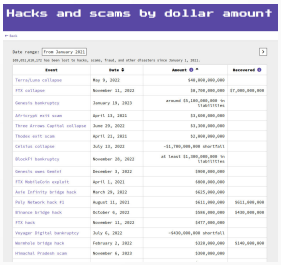
Smart Contract Security

Smart Contract Security

- Smart contracts exist in an **extremely adversarial** world
  - If there's a bug, it's likely that **real-world savings** are lost
- "Traditional" bugs are difficult enough to handle
  - Smart contracts are immutable, even if there are ways around this
- Here, we'll discuss a selection of security stories

Plenty More Other Stories

- We're discussing some stories that may be interesting and instructive
- A lot more have happened, and many are bigger!



https://web3isgoinggreat.com/charts/top

References

- Zubin Pratap. Reentrancy Attacks and The Dao Hack. Chainlink.
- Wayne Jones. The DAO Attack: Understanding What Happened. Crypto.news.
- Hess et al. Ethereum's DAO Wars Soft Fork is a Potential DoS Vector. Hacking, Distributed.

The DAO Hack and Reentrancy Attacks

Decentralized Autonomous Organizations

- Organizations governed through smart contracts
- The organization has an account
  - With cryptocurrency, any kind of tokens, ...
- You get in the organization by obtaining/buying **its tokens**
  - They give you voting power to decide what to do with the funds
- Idea for organizing both commercial and no-profit enterprises

The DAO

- Confusing name of **one** large DAO launched in 2016
- Worked as a decentralized venture capital fund
- Attracted several people, due to the claimed benefits of transparency, shareholder control, flexibility and autonomous governance without middlemen
- In May 2016, it was worth more than **150 million US$** and contained around **14% of all ETH** existing then

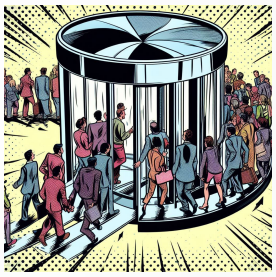
Code Is Law

- From The DAO's Explanation of Terms and Disclaimer:

  *The terms of The DAO Creation are set forth in the **smart contract code** existing on the Ethereum blockchain at 0xbb9bc244d798123fde783fcc1c72d3bb8c189413. **Nothing** in this explanation of terms or in any other document or communication **may modify or add any additional obligations** or guarantees beyond those set forth in The DAO's code. Any and all explanatory terms or descriptions are merely offered for educational purposes and **do not supercede or modify** the express terms of **The DAO's code** set forth **on the blockchain**; to the extent you believe there to be any conflict or discrepancy between the descriptions offered here and the functionality of The DAO's code at 0xbb9bc244d798123fde783fcc1c72d3bb8c189413, **The DAO's code** controls and sets forth **all terms of The DAO Creation**.*

Reentrancy

- From “re-entry”: a program (function, subroutine) is **reentrant** when you can run it execute it when another instance/call of it is concurrently running
- In our case, we have code recursively called by itself



Reentrancy Attacks

- Here, the attacker writes a **malicious smart contract** that will result in calling multiple times, recursively, a **vulnerable victim smart contract**
- Vulnerable code will mess up something dealing with **global state**
- The attacker will exploit this to **gain something**

Vulnerable Code

```
contract Dao {
    mapping(address => uint256) public balances;

    function deposit() public payable {
        require(msg.value >= 1 ether,
            "Deposits must be no less than 1 Ether");
        balances[msg.sender] += msg.value;
    }

    function withdraw() public {
        // Check user's balance
        require(
            balances[msg.sender] >= 1 ether,
            "Insufficient funds.  Cannot withdraw"
        );
        uint256 bal = balances[msg.sender];

        // Withdraw user's balance
        (bool sent, ) = msg.sender.call{value: bal}("");
        require(sent, "Failed to withdraw sender's balance");

        // Update user's balance.
        balances[msg.sender] = 0;
    }

    function daoBalance() public view returns (uint256) {
        return address(this).balance;
    }
}
```

- The sender may execute code **before** its account is set to zero
- What if withdraw() is called again?

Attack

```
interface IDao {
    function withdraw() external ;
    function deposit()external  payable;
}

contract Hacker{
    IDao dao;

    constructor(address _dao){
        dao = IDao(_dao);
    }

    function attack() public payable {
        // Seed the Dao with at least 1 Ether.
        require(msg.value >= 1 ether,
        "Need at least 1 ether to commence attack.");
        dao.deposit{value: msg.value}();

        // Withdraw from Dao.
        dao.withdraw();
    }

    receive() external payable{
        if(address(dao).balance >= 1 ether){
            dao.withdraw();
        }
    }

    function getBalance()public view returns (uint){
        return address(this).balance;
    }
}
```

- When the receive() method is called by withdraw(), it calls withdraw() again…
- withdraw() will gladly continue because the sender’s balance is still not set to zero!

Fixing the Vulnerability (1)

- Make sure the user’s balance is set to zero **before** sending them currency
- In this way, the second call to `withdraw` will fail

```
function withdraw() public {
    // Check user's balance
    require(
        balances[msg.sender] >= 1 ether,
        "Insufficient funds.  Cannot withdraw"
    );
    uint256 bal = balances[msg.sender];

    // Update user's balance.
    balances[msg.sender] = 0;

    // Withdraw user's balance
    (bool sent, ) = msg.sender.call{value: bal}("");
    require(sent, "Failed to withdraw sender's balance");

    // Update user's balance.
    balances[msg.sender] = 0;
}
```

Fixing the Vulnerability (2)

- Write a **modifier** to **forbid reentrancy** in critical code
- This way we’ll be sure the function will **never call itself recursively**

```
Contract Dao {
    bool internal locked;

    modifier noReentrancy() {
        require(!locked, "No reentrancy");
        locked = true;
        _;
        locked = false;
    }

//……
    function withdraw() public noReentrancy {

    // withdraw logic goes here…

    }
}
```

The DAO Hack

- The code had a `splitDAO` function that was vulnerable to a reentrancy problem that was similar to what we have seen, where the funds would end up in a “child DAO”
- On June 17, 2016, an active reentrancy attack was found
  - It was, somehow slowly, draining the DAO’s founds to a child “Dark DAO” controlled by the attacker
  - The DAO’s code locked these funds for 28 days

Whitehat DAO

- A group of “white hat hackers” started replicating the attack, trying to empty the funds of The DAO **faster than the attacker**
- They eventually managed to put US$100M worth of ETH in a “Whitehat DAO” child contract as opposed to the US$50M of the Dark DAO
- The Whitehat DAO would return funds to the original DAO investors

Soft Fork Proposal

- Vitalik Buterin, lead Ethereum developer, proposed a soft fork that would make all transactions taking currency from the Dark DAO invalid
- Hess et al. found it was vulnerable to a denial-of-service attack that would waste miners' computation without spending gas and make Ethereum (mostly) unusable:

```
for(uint32 i=0; i < 1000000; i++) {
   sha3('some data'); // costly computation
}
DarkDAO.splitDAO(...); // render the transaction invalid
```

The Attacker (?) Responds

- An open letter (with an incorrect crypto signature):

*[...] I have carefully examined the code of The DAO and decided to participate after **finding the feature** where **splitting is rewarded with additional ether**. **I have made use of this feature** and have **rightfully claimed 3,641,694 ether**, and would like to thank the DAO for this reward. [...] A soft or hard fork would amount to **seizure of my legitimate and rightful ether**, claimed legally through the terms of a smart contract. [...]*

The Hard Fork

- While ETH price went down from US$20 to US$13, discussion raged over a **hard fork** that would return all currency to the owners before the attack
- Main point of the debate: didn't we say that **code is law**? Is this still a decentralized system?
- The fork was accepted with an 85% majority of voters
- Block 1,920,000 transferred everything from the Whitehat and Dark DAO to a contract that allowed DAO investors to recover their ETH

Aftermath

- Somebody didn't accept the fork and established the Ethereum Classic (ETC) blockchain
- The attacker got away with around ETC US$8.5 million at that time's evaluation
- 12 November '23: ETC is worth ~€22, while ETH is worth ~€3100
- Do we still believe in decentralization and that code is law?
- Many other reentrancy attacks have been run

Flash Loans

Traditional Loans

- In traditional economy, a loan carries the risk of not being paid back
  - **Collateral** exists: resources that the lender can take if the borrower doesn't pay
  - We need **enforcement mechanisms**: police, tribunals, etc., often paid via taxes
  - And **interest rate** needs to compensate for the risk of not being paid back

Flash Loans: Certainty of Being Repaid

- Flash Loans are loans that are taken and repaid **within a single transaction**
- Smart contracts that are reverted (gas is lost) if the loan fails
- No risk (unless bugs exist) for the loaner

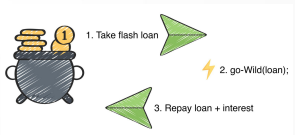


Image by Qin et al.

Flash Loan Usage

- Arbitrage
  - Say contract A sells RIB at 1ETH and contract B buys RIB at 1.1 ETH
  - I can flash-loan 1000 ETH, buy 1000 RIB, sell them for 1100 ETH, gain 100 ETH minus fees
- Wash trading
  - I create a lot of fake "trading" of RIB ("see? People spent millions on RibbaTokens! I'll buy some!")
- Exploits: often, one needs capital to exploit bugs

A Simple Exploit

- The Beanstalk project had a governance mechanism allowing to vote for changes to its code, with one vote per token
- In April 2022, an attacker used a flash loan to obtain enough tokens to get 67% of the votes
- That percentage allowed the attacker to vote for a code change that sent themselves the $182 million in the system's reserve
- After repaying the flash loan, they profited $80 million

Frontrunning

Transaction Pools Are Public

- Before transactions are finalized and put in a block, they are gossiped through the peer-to-peer network
- They are a "preview" of what will appear in the network
- If an attacker can make sure their transactions are processed **first**, they can exploit this information to profit

Examples

- A billionnaire got interested in RIB and sends a transaction that invests one billion dollars in it
  - I want to buy lots of RIB (maybe using a flash loan) just before the price goes up
- Somebody found a transaction that will make them richer (arbitrage, an attack, solving a crypto puzzle)
  - I want to do that in place of them
- There's an auction for some resource
  - I want to know other's offers to win the bid at the lowest price

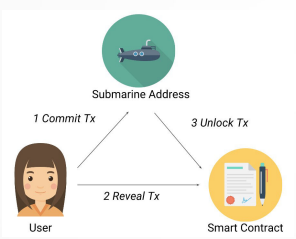
How To Front Run a Transaction

- **Just pay more gas!**
- Miners/validators will execute the transactions that pay more fees first
- There are **bots** that regularly front-run profitable transactions
- Example: in 2021, somebody found a vulnerability in the DODO DEX, and two bots stole the exploit netting $3.1 million

Countermeasures

- Setting a limit to the amount of gas that can be paid
- Commitment schemes:
  - Cryptographic mechanisms allowing to commit to a choice without revealing the choice
  - Require ad-hoc modifications to work in a blockchain

Submarine Transactions

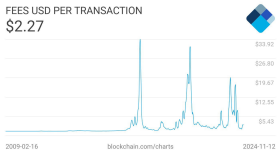


- **Commit**: send money
  - Not distinguishable from sending to any address
  - Nobody has the private key for the address
- **Reveal**: show that the tx was indeed a commitment
- **Unlock**: the smart contract (and only it!) can reclaim the money in the committed tx

Decentralized Systems

Payment Channels

Transaction Cost



- When the number of transactions grows, transaction fees go up
- Scalability limit: fixed number of slots per second, demand may go up

First Ever Answer On Bitcoin

```
Satoshi Nakamoto wrote:
> I've been working on a new electronic cash system that's fully
> peer-to-peer, with no trusted third party.
>
> The paper is available at:
> http://www.bitcoin.org/bitcoin.pdf

We very, very much need such a system, but the way I understand your
proposal, it does not seem to scale to the required size.

For transferable proof of work tokens to have value, they must have
monetary value. To have monetary value, they must be transferred within
a very large network - for example a file trading network akin to
bittorrent.

To detect and reject a double spending event in a timely manner, one
must have most past transactions of the coins in the transaction, which,
naively implemented, requires each peer to have most past
transactions, or most past transactions that occurred recently. If
hundreds of millions of people are doing transactions, that is a lot of
bandwidth - each must know all, or a substantial part thereof.
```

Source: James A. Donald, archived by the Satoshi Nakamoto Institute

Off-Chain Transactions

- There is an inherent scalability limit on a single blockchain
  - With n users, $O(n^2)$ total storage
- Writing on a blockchain is **expensive** and it has **limited throughput**
  - We are writing **too many copies of them**
- We will look at alternatives that allow transacting **without touching the blockchain**

References

- J. Poon and T. Dryja. The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments.
- N. Narula and T. Drya, Cryptocurrency Engineering and Design. MIT OpenCourseWare, lectures 13 and 14.

The Bitcoin Lightning Network

One-Way Channels: Idea

- Incremental payment channels for **recurring payments from A to B**
- A sets aside a given amount of money in a transaction
- When they are both done, the channel is closed and each gets their remaining balance
- Key: doing it **without trust involved**

One-Way Channels: Funding

- A "channel" is just a **multi-signature** output
  - an output for Alice and Bob can only be spent by transactions signed by **both Alice and Bob**
- Alice **funds** the channel to **spend at Bob's**

Funding transaction – on blockchain	
Input	Output
From: Alice's output Alice's signature	To: Alice's and Bob's multi-sig 10 coins

One-Way Channels: Refund

- Transactions have a **lock time:**
  - output can be spent only after the lock time (block height)
- The refund transaction is signed **before** the funding one
  - Needs segwit to work

Refund transaction – **LOCKED UNTIL December 15, held by Alice**	
Input	Output
From: funding transaction's id (txid) Bob's signature (Alice's signature not there yet)	To: Alice's address 10 coins

One-Way Channels: Full Setup

Funding transaction – on blockchain	
Input	Output
From: Alice's output Alice's signature	To: Alice's and Bob's multi-sig 10 coins

Refund transaction – **LOCKED UNTIL December 15, held by Alice**	
Input	Output
From: funding txid Bob's signature (Alice's signature not there yet)	To: Alice's address 10 coins

- Now, in the worst case, Alice can simply wait until she can get the refund
  - Alice won't lose money

One-Way Channels: Spending (1)

Funding transaction – on blockchain	
Input	Output
From: Alice's output Alice's signature	To: Alice's and Bob's multi-sig 10 coins

Spending transaction 1 **Held by Bob**	
Input	Output
From: funding txid Alice's signature (Bob's signature not there yet)	To: Alice's address 9 coins
	To: Bob's address 1 coin

- Bob can post the transaction and **close the channel**
  - He should do before the refund transaction becomes valid

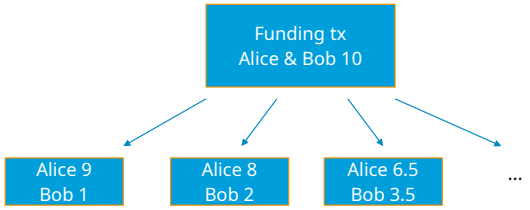
One-Way Channels: Spending (2)

Spending transaction 1 **Held by Bob**	
Input	Output
From: funding txid Alice's signature (Bob's signature not there yet)	To: Alice's address 9 coins
	To: Bob's address 1 coin

Spending transaction 2 **Held by Bob**	
Input	Output
From: funding txid Alice's signature (Bob's signature not there yet)	To: Alice's address 8 coins
	To: Bob's address 2 coins

- Now Bob can forget about the first transaction: the **second is better**
- Spending transactions **aren't posted to the blockchain**
  - Until A and B want to close the channel

One-Way Channels: Spending (3)



One-Way Channels: Outcome

- Bob keeps getting half-signed transactions with more money going to him
  - **Fast** (off-chain payments) and with **no fees**
  - He **must** sign and broadcast one before the refund time!
- Currency flows **only one way**
  - From Alice to Bob, **never vice versa**
- Channels are only useful **before they expire**
  - Expiration can't be too far away: if Bob disappears Alice's funds are locked
- Common case: how many transactions on the blockchain?
  - Two: funding and closing (last expense by Alice)

Bidirectional Payment Channels

- A construct that, like what we have seen until now
  - Avoids paying on-chain most transactions
  - Doesn't require trust between Alice and Bob
- But
  - Is bidirectional (money can flow in either direction)
  - Doesn't expire
- We'll follow the Bitcoin Lightning protocol
  - Raiden is the equivalent solution for Ethereum

Bitcoin Timing Opcodes

`OP_CHECKSEQUENCEVERIFY`

- Argument: sequence
- Requires that the input has at least n confirmations
  - i.e., it is ≥n blocks old
- Otherwise, the transaction fails

`OP_CHECKLOCKTIMEVERIFY`

- Argument: locktime
- Requires that the transactions' output has at least n confirmations to be spent
- Otherwise, the transaction fails

Using the New Opcodes

- "This transaction's output is spendable by *[the private key corresponding to public]* key X after 100 blocks"
- "or by this other *[private key corresponding to public]* key Y"
- "or by *[…]* keys W and Z in conjunction"

Revokable Transactions

Held by **Alice**	
Input	Output
From: funding txid Bob's signature (Alice's signature not there yet)	To: Alice's key after 100 blocks OR To Bob and AliceR keys 2 coins
	To: Bob's address 8 coins

Held by **Bob**	
Input	Output
From: funding txid Alice's signature (Bob's signature not there yet)	To: Alice's address 2 coins
	To: Bob's key after 100 blocks OR To Alice and BobR keys 8 coins

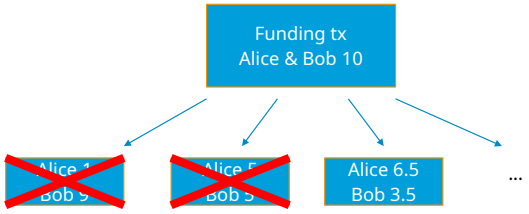
- Both Alice and Bob build new AliceR/BobR key pairs

Reveal to Revoke

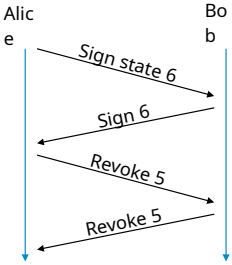
- Bob can **revoke** this transaction by sending Alice the BobR private key
- If he ever tries to broadcast this transaction, Alice can use BobR to recover **all** the funds in the channel
- Key assumption: **Alice has to remain online!**

Held by **Alice**	
Input	Output
From: funding txid Bob's signature (Alice's signature not there yet)	To: Alice's key after 100 blocks OR To Bob and AliceR keys 2 coins
	To: Bob's address 8 coins

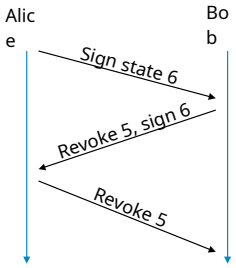
Updating States



Message Order



Optimized Message Order

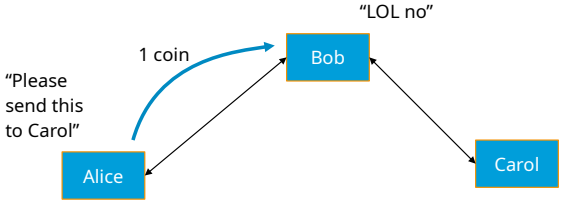


Storing Revocation Secrets

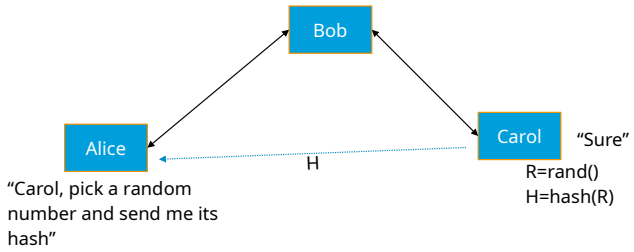
- How many revocation secrets should one store?
  - (Almost) all those in the history of the channel!
  - 32 bytes per state
- A clever optimization: Elkrem tree
- With n transactions
  - You just need to store O(log n) secrets
  - You need to perform O(log n) hashes to get to the secret

Multi-Hop Transfers

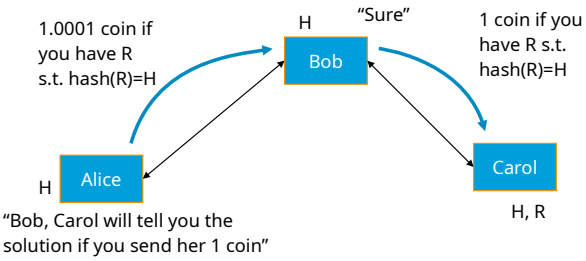
Trust Issues



The Method (1)



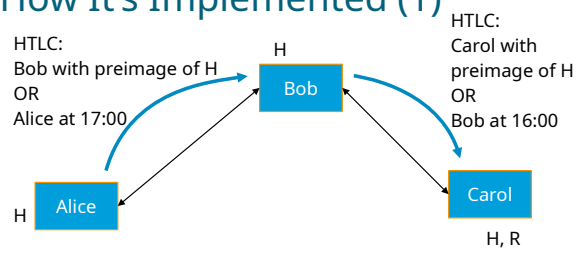
The Method (2)



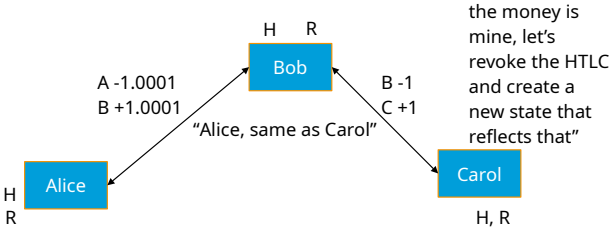
Hash/Time Locked Contract (HTLC)

Held by **Bob**	
Input	Output
From: funding txid Alice's signature (Bob's signature not there yet)	To: Alice's address 0.9999 coin
	To Bob's key after 100 blocks OR To Alice and BobR keys 8 coins
	To Bob if he knows the preimage to H **OR** **To Alice at 17:00 (uses block height)** **1.0001 coin**

How It's Implemented (1)



How It's Implemented (2)



"Bob, here's R, the money is mine, let's revoke the HTLC and create a new state that reflects that"

Implementation

Revocation Keys as Hash Preimages

Held by **Alice**	
Input	Output
From: funding txid Bob's signature (Alice's signature not there yet)	To: Alice's key after 100 blocks OR To Bob and AliceR keys 2 coins
	To: Bob's address 8 coins

Held by **Alice**	
Input	Output
From: funding txid Bob's signature (Alice's signature not there yet)	To: Alice's key after 100 blocks OR To Bob **if he knows P s.t. H(P)=AliceR** 2 coins
	To: Bob's address 8 coins

- Less storage on blockchain (~20B vs ~80B), same concept

An Elliptic Curve-Based Trick

- Remember the slides on elliptic curves? Say we have two private/public key pairs (b, B) and (c, C)
- (b+c, B+C) is a working key pair because
  - bG=B, cG=C → (b+c)G = B+C

ECC: scalar multiplication 🔑

- Given a known **base point G**, it can be multiplied by a private key **Sk** to find the corresponding public key **Pk**

Even Better: Just Sum Keys

Held by **Alice**	
Input	Output
From: funding txid Bob's signature (Alice's signature not there yet)	To: Alice's key after 100 blocks OR To Bob and AliceR keys 2 coins
	To: Bob's address 8 coins

Held by **Alice**	
Input	Output
From: funding txid Bob's signature (Alice's signature not there yet)	To Alice's key after 100 blocks OR To **KeyR=Bob+AliceR** 2 coins
	To: Bob's address 8 coins

- Even shorter script
- Compactness matters this much on the blockchain!

The Reduced Script

```
OP_IF KeyR
OP_ELSE
  <delay>
  OP_CHECKSEQUENCEVERIFY
  OP_DROP
  KeyA
OP_ENDIF
OP_CHECKSIG
```

- Stack-based language
- Working inputs on stack:
  - 1 SigR
  - 0 SigA (after the delay is passed)
- OP_CSV doesn't consume the stack for the soft fork backwards compatibility

The Lightning P2P Network

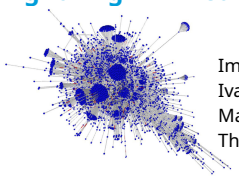
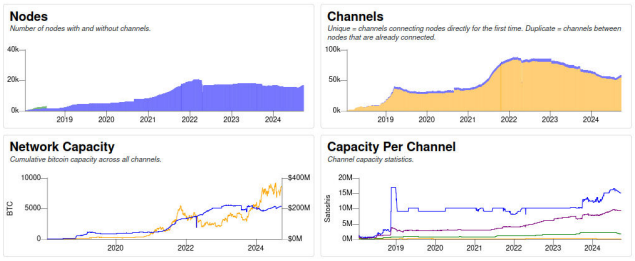


Image from Ivan Gallo's Master Thesis

The Network

- Nodes use **gossiping** to exchange information about the available channels
- Finding a way to send money from A to B is a graph problem: find paths from A to B
  - Easy if you have all the network
  - Some research on doing it with privacy about existing channels
- Not very big, but appears to have grown fast
  - +1212% in 2021-2023 (source: river.com)

Some Stats



Source: Bitcoin Visuals

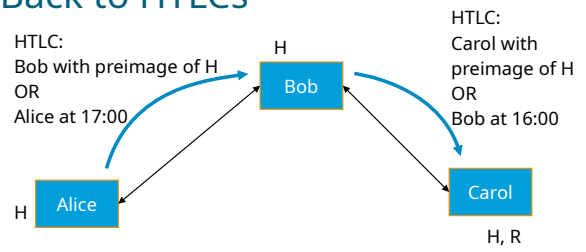
Cross-Chain Swaps

The Tragedy of Centralized Exchanges

Trustless Cross-Chain Trading

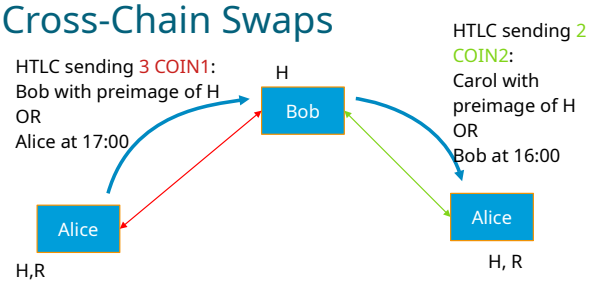
- Requirement:
  - A sends X COIN1 to B
  - B sends Y COIN2 to A
- A and B agree on the exchange rate
  - But don't trust each other!
- Idea: use payment channels for this!

Back to HTLCs



This works even if the two channels are on **different blockchains**!

Cross-Chain Swaps



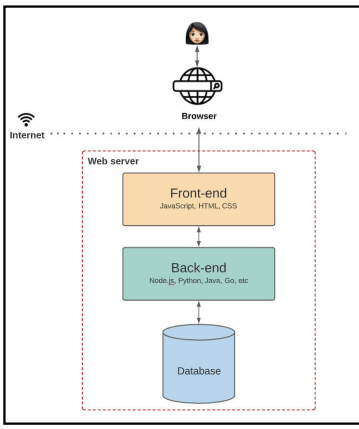
Here, Alice is exchanging 3 COIN1 for 2 COIN2 with Bob

Decentralized Systems

Web3

The Web3

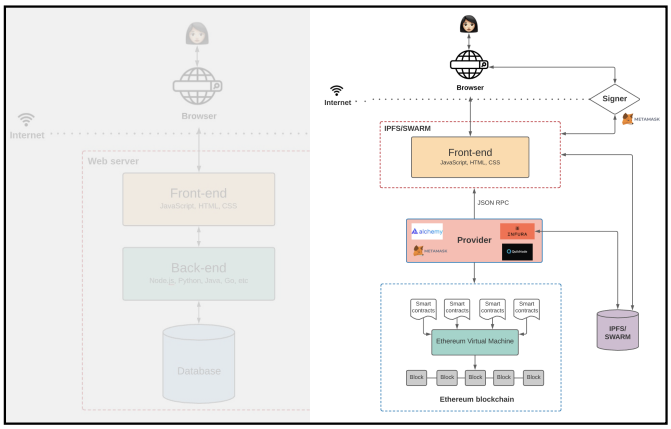
- The Internet we have today is “broken”
  - We do not control our data
  - Every time we interact, online copies of our data are sent to the servers of some (few) “trusted” tech companies
- While the **Web2** was a **front-end revolution**, the **Web3** is a **back-end revolution** proposing a **decentralized state layer** built **on top of blockchain** technologies



Suggested reading:
https://www.preethikasireddy.com/post/the-architecture-of-a-web-3-0-application

Web server for the back-end logic

“Centralized” database storing the application state



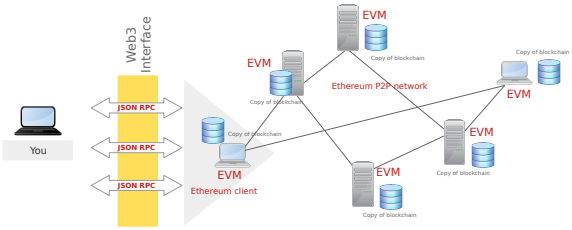
The Web3

- **Decentralization** is at the core of Web3, which is
  - Verifiable
  - Self-governing
  - Permissionless
  - Provides native built-in payments
- Producers and consumers of **Web3 services** (computing, storage, bandwidth, identity, hosting) **pay**, similarly to what happens today for cloud services. But in Web3 **rewards go directly to the network participants** who keep the overall ecosystem up, running, and secure

Ethereum stack

- LEVEL 1: EVM ✔
- LEVEL 2: Smart Contracts ✔
- LEVEL 3: Ethereum nodes
  Computers running software (e.g., an Ethereum client implementing the JSON-RPC standard) they **collectively store the state of the Ethereum blockchain** and **reach consensus** on transactions to mutate the blockchain state
- LEVEL 4: Ethereum Client APIs ← **Today**
  See: https://ethereum.org/en/developers/docs/apis/javascript/
- LEVEL 5: End-User Applications ✔
  Standard web and mobile apps

Ethereum ecosystem



web3.js

- **web3.js** the the first **JavaScript library** developed to **interact** with the **Ethereum** blockchain
- It supports different APIs
  - **web3-eth** for the Ethereum blockchain and smart contracts
  - **web3-shh** for the whisper protocol, p2p communication and broadcast
  - **web3-bzz** for the swarm protocol, the decentralized file storage
  - **web3-utils** contains useful helper functions for dApp developers
  - See https://docs.web3js.org/

ethers.js

- JavaScript library designed to interact with the Ethereum blockchain. It has many classes:
  - **1. Provider**: class for a **connection** to the Ethereum blockchain
  - **Read-only access** to the blockchain
    - provider.**getBalance**(address)
    - provider.**getBlockNumber**()
    - provider.**getTransaction**(txhash)

ethers.js

- JavaScript library designed to interact with the Ethereum blockchain. It has many classes:
  - **2. Contract**: class to **interact with a specific deployed contract**, accessible like a JavaScript object
    - ethers.**Contract**(address, abi, provider)
  - Given the contract instance, it is possible to call its **methods** as if they were local **JavaScript functions**

ethers.js

- JavaScript library designed to interact with the Ethereum blockchain. It has many classes:
  - **3. Signer**: can **sign messages and transactions** to perform **write operations** that have a cost
    - Usually connected to a Provider
    - ethers.**Wallet**, for non custodial accounts
    - ethers.**JsonRpcSigner**, for custodial accounts with private keys managed by other services

ethers.js

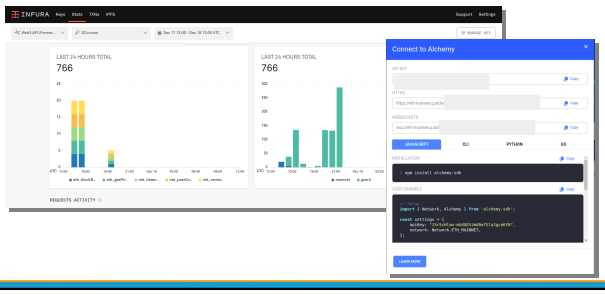
- JavaScript library that also includes numerous **utility functions** for common operations:
  - ethers.utils.formatEther / ethers.utils.parseEther
  - ethers.utils.keccak256 / ethers.utils.sha256
- **BigNumber** class (to handle **large numbers**, especially for Ethereum balances in wei)
  - ethers.BigNumber.from (creates a BigNumber from a number, string, or hex value)

How to

- To use web3.js or ether.js you need
  - **node.js**
  - the **JS library of your choice** (npm install ...)
  - an **Ethereum node** which provides access to the **Ethereum JSON-RPC API method library** that interacts with the Ethereum blockchain
    see
    https://www.alchemy.com/overviews/blockchain-node-providers

Connect to a node

- You need to create an account and get an **API Key**



Connect to a node

- With the **API Key** you can connect to Ethereum
- I will use **INFURA API Key** (but others seems more popular today)
- Hint: for private and API keys consider the **dotenv module** to store your private information outside the code and load them from a **.env** file

  See: https://www.npmjs.com/package/dotenv

Connect to a node

- According to ChatGPT ;)

How to Choose the Best Provider?

- **For Developers**: Use Alchemy or QuickNode for robust tools and enhanced APIs.
- **For Decentralization**: Consider Pocket Network or Ankr.
- **For Enterprises**: Chainstack or Blockdaemon are excellent options.
- **For Cost-Conscious Projects**: Start with Infura (free tier) or Cloudflare.

Each provider has strengths tailored to different needs, so the "best" choice depends on your project's requirements (e.g., speed, decentralization, cost, or features).

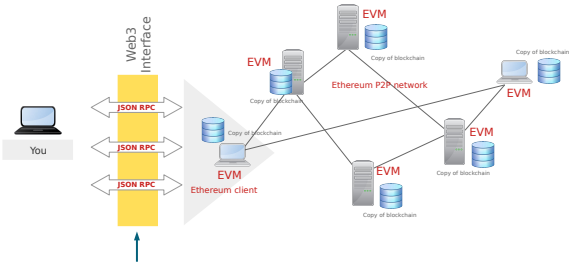
And interact :)

- There are **two ways** to interact with accounts and smart contracts: **reading** state or **writing** state
  - **Reading** from the blockchain **does not cost**: the function call is carried out by the connected node and it is free
  - **Writing** into the blockchain **has a cost**, paid in gas, which determines **the fee** required to successfully conduct a transaction or execute a contract to **update the state** of Ethereum

Decentralized Systems

Web3 (cnt)

Ethereum ecosystem



https://ethereum.org/en/developers/docs/apis/json-rpc/

Examples of today

- Ethers.js and Node.js
  - Sending ETH to other accounts
  - Writing on the blockchain
  - Reading events from the blockchain
- Ethers.js and JavaScript
  - Interacting via browser with WishOfDay smart contract

Solidity: events (recall)

- Smart contract WishOfDay: we can add one event emitted each time a new wish is written in the blockchain

1. **Declaration**

```
event WishAdded(uint256 _data, string _message, string indexed _author, address indexed _from);
```

The keyword indexed is used to make authors and addresses filterable when querying the event logs

2. Usage

```
function setOneWish(string memory _message, string memory _author)
public {
 ...
 ...
 emit WishAdded(block.timestamp, _message, _author, msg.sender);
}
```

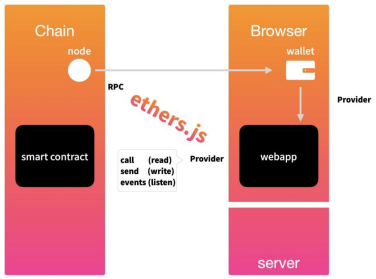
Solidity: events

- Emitted events are used for **logging information** that external web3-based applications can listen to
- Used to **store data** on the blockchain, outside the contract storage, that **can be queried later**
- When an event is emitted, the **topics** are the **indexed parameters of the event** (max 3) and they are the key to enabling efficient searching and filtering of blockchain logs
- The **first topic** always represents the hash of the **event signature** (event name and parameter types), allowing for a quick identification of the event

Solidity: events

- EventLog typically includes
  - **address** of the contract that emitted the event
  - **topics**
  - **data**, e.g., the actual payload, an hex value which can be decoded using the ABI, and contains the values that were emitted with the event

Connect MM with a web3-app



https://dev.to/yakult/a-tutorial-build-dapp-with-hardhat-react-and-ethersjs-1gmi

Connect MM with a web3-app

- In the user interface of a web3-app, wallets can connect the application with the blockchain, using ethers.js or other libraries
- The wallet becomes the **Provider**, thanks to the **window.ethereum** object injected into the browser when the wallet is installed
- With ethers.js it is possible to read and write on smart contracts, thanks to this provider

Connect MM with a web3-app

- MetaMask can be used by
  - Users ✔

Connect MM with a web3-app

- MetaMask can be used by
  - Users ✔
  - Developers ✔

The **window.ethereum object** API provides a standardized way for web3 applications to interact with Ethereum-compatible wallets

Bridge between the browser environment and the Ethereum blockchain

Connect MM with a web3-app

- MetaMask can be used by
  - Users ✔
  - Developers ✔

Injected into the browser when the wallet is installed

Interaction through **asynchronous methods** (e.g., Promises) for blockchain queries and wallet actions

https://docs.metamask.io/wallet/reference/provider-api/#ethereum-provider-api

Decentralized Systems

Web3 (cnt)

Smart contracts

- Are smart contracts really immutable

Smart contracts

- Are smart contracts really immutable

Yes and No

The **logic of smart contracts is immutable** after deployment, but they **change the state** (balances, storage variables)

It is also possible to **migrate** a smart contract to a different address to change its logic, but we need to convince users to move and be sure we do not lose any state data

Smart contracts

- Are smart contracts really immutable

Yes and No

Upgradable Smart Contracts can be modified after they have been deployed to the blockchain

Upgradable smart contracts

- **Advantages**
  - **Fixing bugs**: if a bug/vulnerability is discovered in a deployed smart contract, it can be fixed by upgrading the contract; of course, this **improves the security** of the smart contract
  - **Adding new features**: new features can be added to a smart contract without having to redeploy the contract from scratch

Upgradable smart contracts

- **Proxy pattern**
  - Uses a **proxy contract** to **delegate calls** to the **implementation contract**. The proxy contract can be upgraded to point to a new implementation contract
  - **Separates the logic** (implementation) **from its address** and **storage** and allows for upgrades without changing the contract's address or losing state data

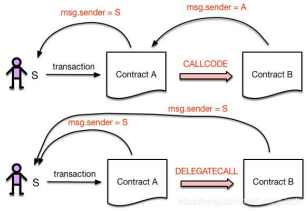
Upgradable smart contracts

- **Implementation** contract
  - Contains all the logic of the contract (v1)
  - To upgrade we deploy a new implementation of the contract (v2)
- **Proxy** contract
  - Points to the "current" implementation (v1)
  - Routes all function calls to that implementation
- **Users** make calls through the proxy
- **Admin** is the only actor who can upgrade to a new implementation contract

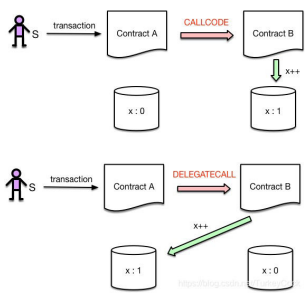
Delegate call

`delegatecall` is a low level function similar to `call`.

When contract A executes `delegatecall` to contract B, B's code is executed with contract A's storage, `msg.sender` and `msg.value`.



Delegate call



https://solidity-by-example.org/delegatecall/

Proxy contract structure

- **State variables**: those of the implementation plus at least an **address** to store the address of the currently active implementation contract
- **Constructor**: takes the address of the initial implementation contract as an argument and sets the address accordingly
- **Upgrade function**

```
modifier onlyAdmin() {
        require(msg.sender == admin, "Only admin can call this function");
        _;
    }
function upgradeContract(address _newAddress) external onlyAdmin {
      implementation = _newAddress;
}
```

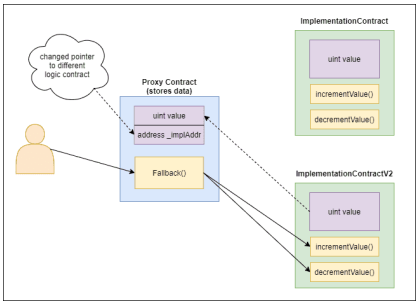
Proxy contract structure

- **Fallback** functions handles calls to functions that do not exist in the contract, can receive Ethers (if defined as payable)
- In a proxy contract act as the **default entry point** for all external calls
- Intercept incoming transactions, extract the function selector and calldata, and delegate the call to the implementation contract using the delegatecall opcode

Proxy contract structure

- The **delegatecall** function is available on the **address type**
  - One parameter, e.g., a bytes array containing the encoded function call data
  - This data is typically generated using the abi.encodeWithSelector or abi.encodeWithSignature functions, used to encode function calls into the low-level bytes format required for interacting with other contracts
- The **data** include
  - The function selector (the first 4 bytes of the Keccak256 hash of the function signature)
  - The arguments encoded in ABI format

Proxy contract structure



Reading data from the outside world: Oracles

Thanks to Ali Haider and Giacomo Pedemonte

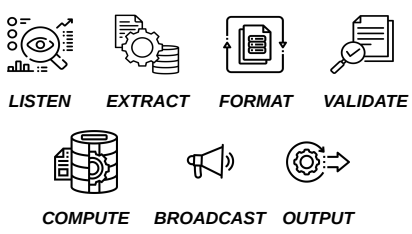
The Oracle problem

- By being purposely isolated, blockchains obtain
  - **strong consensus** on the validity of user transactions
  - **prevention of double-spending** attacks
  - **mitigation** of network **downtime**

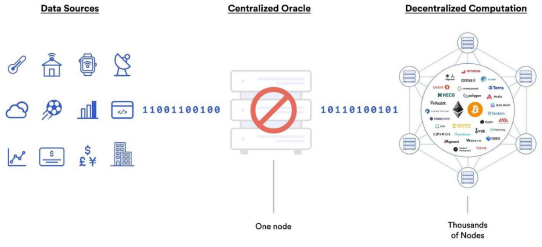
Blockchain Oracle

- Securely interoperating with **off-chain systems** from a blockchain requires an additional piece of infrastructure known as an “oracle” to bridge the two environments
- The vast majority of smart contract use cases like DeFi (Decentralized Finance) require **knowledge of real-world data and events** happening off-chain

Blockchain Oracle

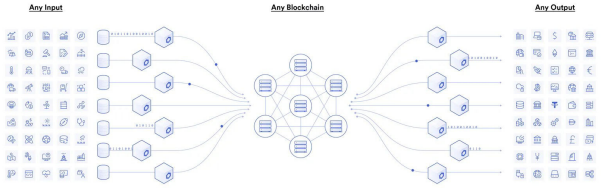


Centralized Oracle



Centralized oracles are a **single point of failure**

Decentralized Oracle



Combines **multiple independent oracle** node operators and multiple reliable data sources to establish end-to-end **decentralization**

Take away points on Smart Contracts and Web3

Take away points

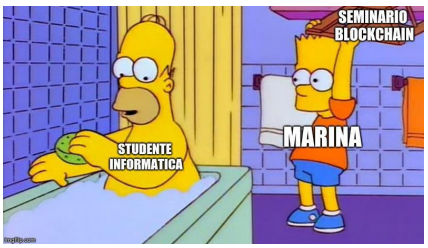
- Smart contracts are **as good as people writing them**
  - a smart contract that receives bad or incorrect information from the network will still execute, possibly propagating errors
- **Testing is fundamental** before deployment since it is critical to check for every possible way things could go wrong (and it happened)
- In case of misbehavior, **who is responsible**? The legal validity of smart contracts is still an open problem

Take away points

- Smart contracts allow for the development of dApps and the **overall ecosystem is complex**
- Also, remember that there is no way to guarantee that the current value of cryptocurrencies and other tokens will be maintained in the future
- There is much more over there, **we scraped only the surface**...

Take away points

Now it is your turn, enjoy!



Decentralized Systems

IPFS & ENS

InterPlanetary File System

(thanks to Marina Ribaudo)

A Centralized Internet

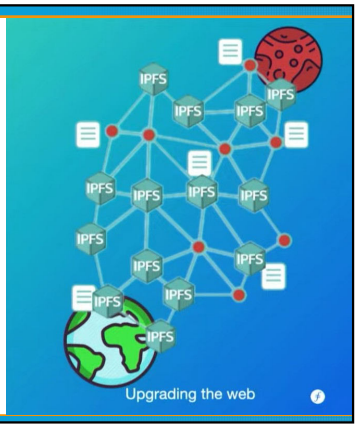
- We do **not own our data**
- Storing all information in central servers can cause problems, like having **single points of failure** and **censorship**
- Nowadays, the Internet is way less centralized than one may imagine, and that even causes catastrophic failures…

A Centralized Internet

- The current way in which information is provided to users is **location-based**
  - Users refer to resources by URL
  - http://gallery.com/meme1.png

InterPlanetary File System

BitTorrent + git → IPFS



A Centralized Internet

- The current way in which information is provided to users is **location-based**
  - Users refer to resources by URL
  - http://gallery.com/meme1.png
- Problem: images can change behind the same URL

IPFS

- The InterPlanetary File System (**IPFS**) is a protocol and P2P network for **storing** and **sharing data** in a distributed file system
  - **Ethereum** does **computation**
  - **IPFS** does **storage**
- IPFS allows users to not only receive but host content, in a similar manner to BitTorrent

IPFS

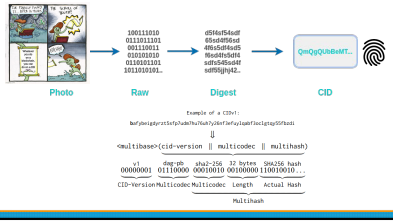
- IPFS lets users **back up files and websites** by hosting them across **numerous nodes**
- Content is **resistant to censorship** and **centralized points of failure**, such as server issues or **coordinated attacks**
- Rather than using file locations, **IPFS points towards contents**, which could be stored on any number of computers around the world

Content-Based Addressing

- Addresses content by **what** it is, rather then **where** it is stored
  - Computer → file://path-to-file/index.html
  - Internet → https://domain.com/path-to-file/index.html
  - IPFS → ipfs://[CID]/path-to-file/index.html
- Example CID: `QmcniBv7UQ4gGPQQW2BwbD4ZZHzN3o3tPuNLZCbBchd1zh`
  - Long binary string encoded to be **printable**

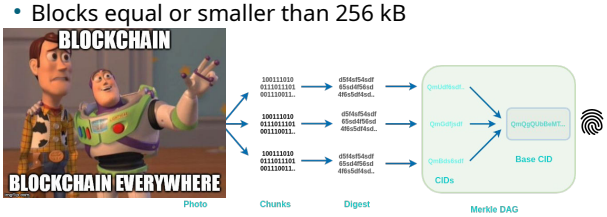
Content-Based Addressing

- For each file, and for each folder, a Content Identifier, or **CID,** is computed using a hash function
- This hash is used to store and retrieve all files in IPFS

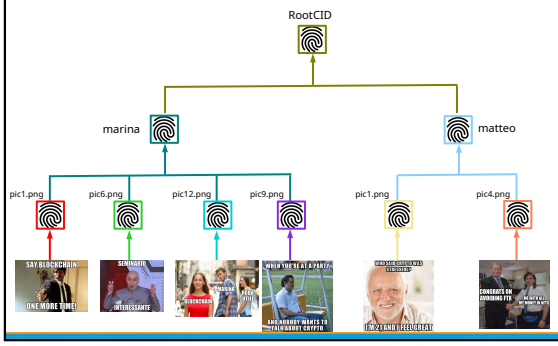


Content-Based Addressing

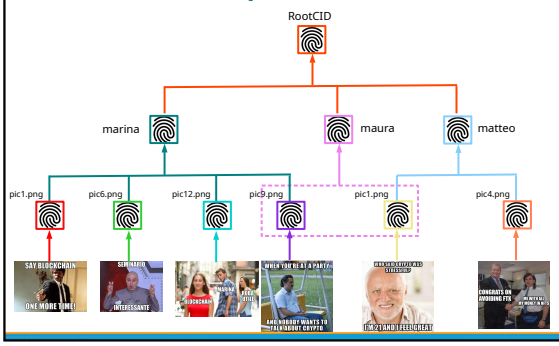
- Large files are **chunked**, **hashed**, and **organized** into **Merkle** DAG objects https://docs.ipfs.tech/concepts/merkle-dag/
- Blocks equal or smaller than 256 kB



IPFS: files and folders

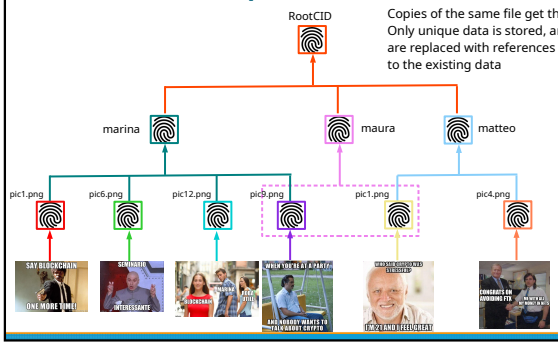


IPFS: deduplication



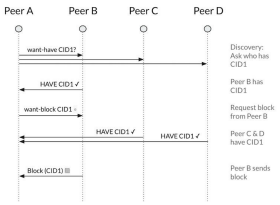
IPFS: deduplication

Copies of the same file get the same CID
Only unique data is stored, and duplicates are replaced with references or pointers to the existing data



Upload/Download

- Uses the BitSwap protocol, inspired by BitTorrent
- Remember? A **swarm** of peers, exchanging a file made of **small blocks**
  - Tit-for-tat incentives
- Here, a single swarm for **all of IPFS**



https://docs.ipfs.tech/concepts/bitswap/

Content Storage

- After you download a file/block, you **cache** it and **serve it to others**
  - Similarly to **seeding** in BitTorrent
- When disk space is over, you will erase **old pieces of data** you didn't access anymore
- You can decide you'll keep storing a piece of data by **pinning** it
  - It won't be evicted from the cache
- A file remains available as long as **somebody is still keeping it**

Versioning: IPNS

- IPFS supports **versioning** through IPNS (InterPlanetary Naming System)
  - Instead of a CID, you use the **hash of a public key**
  - As content, you have a file with
    - The full public key
    - The **version number**
    - The CID of the **current version**
    - Signature
- Storage nodes will store the **last correctly signed version**

DNSLink

- You can have a **human-readable** IPFS address by putting **an IPFS link in your DNS record**
- It's going to be automatically substituted to your DNS name

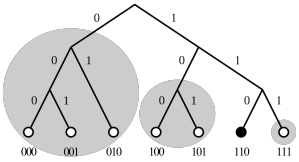
```
$ dig +noall +answer TXT _dnslink.docs.ipfs.tech
_dnslink.docs.ipfs.tech.  34  IN  TXT "dnslink=/ipfs/QmVMxjouRQCA2Qy
```

https://docs.ipfs.tech/concepts/dnslink/

- Here, `/ipns/docs.ipfs.tech/introduction/` will be converted to `/ipns/QmVMxjouR.../introduction/`

Lookup: DHT

- Current connections may be missing some CIDs
- IPFS uses the **Kademlia DHT** (remember?) to support routing and discovery of content and peers on the network
- Peer Ids are directly mapped to CIDs



Content Lookup

- IPFS implements content lookup through a distributed hash table (**DHT**) and **CIDs**
  - Each **node** in the IPFS network is assigned a **unique identifier**, typically a **256-bit value**
  - When a node wants to find a particular piece of content, it queries the DHT with the **content's CID** (**256-bit value**)
- Lookup is performed on Kademlia; at last one will get the **addresses of nodes having that CID** (if any)
  - They're added to the list of local neighbors, and download proceeds from there

Notable Properties

- **Nodes can cache content** they have recently accessed, making it available to other nodes in the network, this encourages content replication across multiple nodes, enhancing fault tolerance and availability
- IPFS uses **deduplication** to help **reducing redundancy at the storage level** by ensuring that identical content is only stored once
- IPFS uses **caching** to help **improving access times** by keeping frequently accessed content closer to the requester. The decision to use caching and the specific caching policies can vary among different IPFS implementations

IPFS Nodes (1)

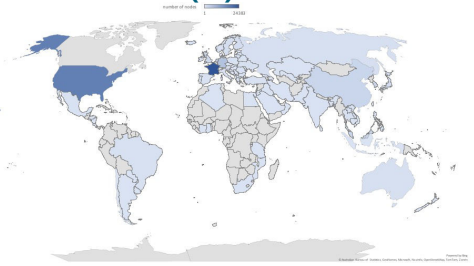
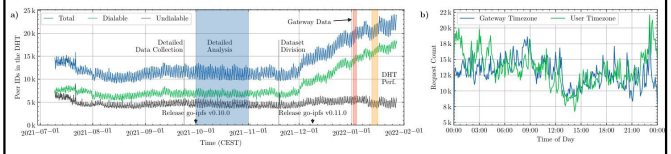


Fig. 5: Number of IPFS nodes per country.

Source: Confais et al., 2023

IPFS Nodes (2)



- Source: Trautwein et al., ACM SIGCOMM '22
- Something I don't understand in the plots, if anybody wants to download the data and try to reproduce them as their seminar...

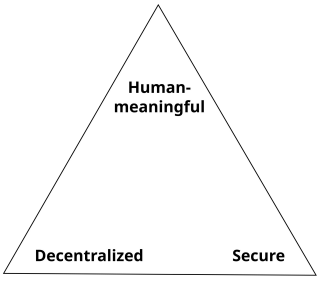
Filecoin

- Ad-hoc blockchain complementing IPFS by providing **incentives to store data**
- To store data persistently, you can pay some filecoin to ensure "miners" store your data for some time
- Based on crypto-magic (ZK-SNARKs, seminar opportunity for adventurous students):
  - Proof of space-time
  - Proof of replication

Ethereum Name Service
(thanks to Federico Fontana)

Zooko's Triangle

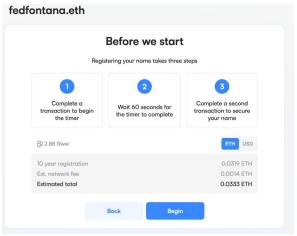
- A 2001 conjecture about the impossibility of having names that are at once **meaningful**, **decentralized** and **secure**
- Q: What about CIDs? Ethereum/Bitcoin addresses? DNSLink?
- It has been **proven false**
  - ENS is one of the solutions



Ethereum Name Service

- A way to associate **human-readable names** to **Ethereum addresses**
- Solves Zooko's triangle: decentralized, secure, meaningful
- Based on Ethereum smart contracts
- Contracts are updateable thanks to the proxy pattern (you'll see)

As a User: Registration Fee

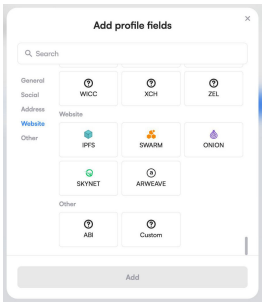


Source: Federico Fontana

- Prices vary (5-640US$ per year) depending on the **length of the name one wants to register**
- Expressed in dollars in the contracts thanks to **an oracle** (you'll see it in the next lesson!)
- Can be renewed until expiration (+ a grace period)

As a User: Fields

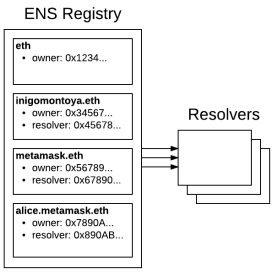
- As with DNS, you can insert **multiple fields** in your record
- Some are standardized, others just use free-form fields
  - Like DNSLink for IPFS+DNS



Source: Federico Fontana

Registry and Resolvers

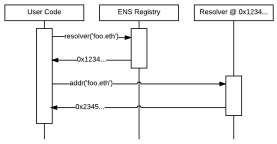
- A **single registry** which maps to **resolvers** for each subdomain
- Resolvers are **smart contracts** anyone can implement
  - They must implement at least the `addr` method



https://ens.readthedocs.io/en/stable/introduction.html

ENS Queries

- The client calls the **registry** to get the resolver's address
- The resolver answers the queries



https://ens.readthedocs.io/en/stable/implementers.html

Registry

- Stores mapping between names and resolver, owner and caching time-to-live (TTL)
- Emits events such as `NewOwner`, `Transfer`, `NewResolver`
- Interacts with another contract, the *Controller*, responsible for registration, fees, renewal
  - Previously (up to 2020): auction model, no renewal
  - Now: commit-reveal pattern to avoid frontrunning

Other Facilities

- NameWrapper: allows delegating a subdomain, but limiting functionalities (e.g., further transfers, change records, create subdomains)
- Reverse registrar: address 0x<addr> can claim the `<addr>.addr.reverse` name (and optionally transfer it)
- DNS in ENS: thanks to another oracle, one can use their DNSSEC (DNS+crypto extensions) name online

Decentralized Systems

Optimistic Rollups

Smart Contract Scalability

- Ethereum is “the slowest computer” (and most expensive) in the world
- **Every verifier executes everything**
  - Worst redundancy possible
- We can extend payment channels to **state channels**
  - Limited to two-player interactions (e.g., chess)
- Can we do better?

Layer-2 Blockchains

- **Blockchains on top of blockchains**
- Goal:
  - Layer-2 (L2) transactions are run on **a few machines**
  - The underlying L1 blockchain **guarantees correctness**
- We’ll see a technique called **optimistic rollup**
  - Reference implementation: Arbitrum

References

- Kalodner et al. Arbitrum: Scalable, private smart contracts. Usenix Security 2018.
- Finematics. ROLLUPS - The Ultimate Ethereum Scaling Strategy? Arbitrum & Optimism Explained – YouTube video, 2021.
- Arbitrum. Inside Arbitrum Nitro.

Optimistic Rollup

Rollup Transaction

- A transaction that “rolls up” several blocks of the L2 blockchain and bundles it all in one transaction
- Posted by a **Manager** that created the blocks
- Blockchain state as a Merkle tree, input messages as attachments to the L1 block
  - We’ll see more on this later
- Meaning: “The state with hash A and this input leads to state with hash B”

Optimistic Rollup Idea

- There is a **happy path** followed **most of the time**
  1) A **manager** publishes its assertion “from state A and this input we get to state B” (and send these messages/currency from this chain to these L1 wallets)
  2) Some other entities (confusingly also called managers) verify that the computation is correct
  3) If after some time (e.g., blocks) nobody disputes that, the new state is **confirmed**

Optimistic Rollup: Challenge

- Managers need to **stake** some currency
- If an assertion is **challenged**:
  - The L2 chain’s progress is paused
  - The conflicting managers play a **game** on the L1 chain
  - The L1 smart contract of the L2 chain will make sure it will be won by whoever is correct
  - The loser pays part of its stake to the winner, the rest goes somewhere else (e.g., the verifier)
- It *should* be irrational to willingly post **wrong states**

Challenge: Bisection Game

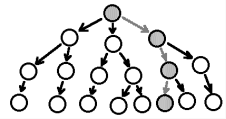
- A: “State A leads to state B in N steps”
- B: “I challenge it!”
- A: “State A leads to state C in N/2 steps, and C to B in N/2 steps”
- B: “Prove you get to from A to C (or C to B) in N/2 steps”
- A: “You get from A to D in N/4 steps, and from D to C in N/4 steps”
- …

Bisection Game: Result

- A and B get in log(n) steps to **a single step**
  - n is the number of execution steps in the whole "rollup block"
  - If either doesn't answer within a deadline, **they lose**
- That step gets **replayed on the main chain** to decide the winner
- How to do this?

Running a Single Step

- Remember that a state is a **Merkle tree**
- A reveals **only the parts of the state needed to run the instruction**
  - **Merkle proof:** you expand the nodes containing the parts of the state touched by the instruction
  - The L1 chain **verifies the hashes**
  - It **runs the instruction**
  - It **verifies the result**



Sequencing Input

- A **sequencer** takes all the messages, compresses them, and posts them to the blockchain as a binary "blob"
  - Such blobs are **erased after some time** (~18 days in Ethereum)
  - Enough time to **challenge** the state, then they're lost
  - Blocks have been introduced as **Proto-Danksharding** in March '24 (input was stored in CALLDATA before)
- Blobs are way cheaper than storing blockchain data
  - The security comes from the fact they're signed, as usual

Avoiding Censorship

- To make sure the sequencer can't censor messages, you can also **post** the ignored ones **on the blockchain**
- They will **have to be included** in the next rollup block
- No punishment for the sequencer: they may not have seen it in good faith

Putting It Together

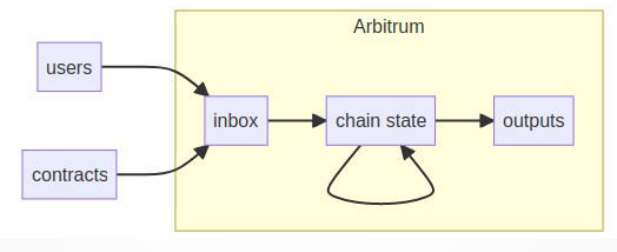
- An L2 chain that is secure if the L1 chain is secure
- …and if there's at least a honest manager verifying the rollup transactions
- The L2 chain can be faster and cheaper, because it's verified by much fewer nodes
  - Also, gas limits can be way larger
- Users have ways to bring currency from the L2 to the L1 chain

Arbitrum

About Arbitrum

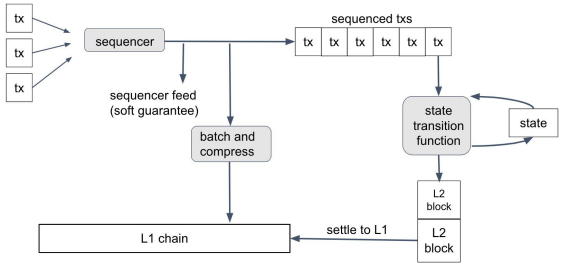
- The most popular L2 network at the moment
- Based on the protocol described in a 2018 USENIX Security paper
- Currently used by many users, while still under development

Architecture (1)



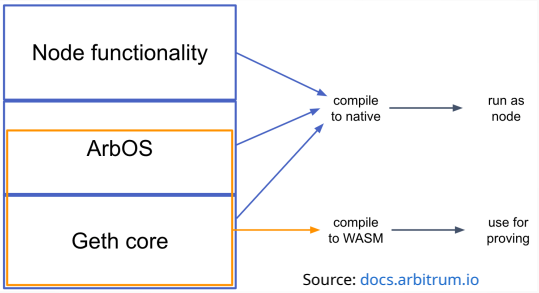
Source: docs.arbitrum.io

Architecture (2)



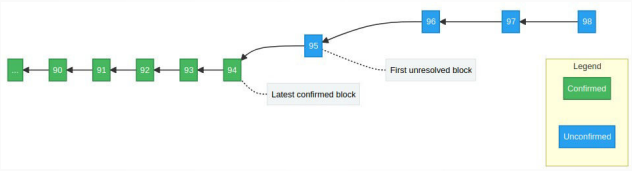
Source: docs.arbitrum.io

Implementation



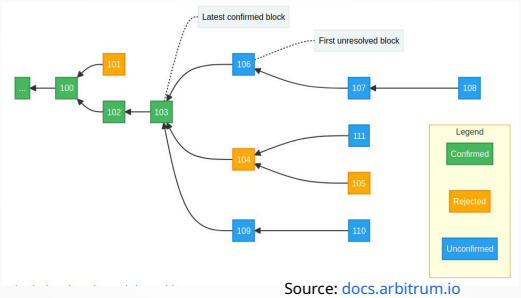
Source: docs.arbitrum.io

Happy Path



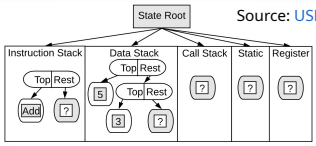
Source: docs.arbitrum.io

(Very) Unhappy Path



Source: docs.arbitrum.io

Virtual Machine for Proofs



Source: USENIX Security '18 paper

- The state as a Merkle tree is visible to the VM
- Micro-instructions that only touch elements close to the top of the stack
- EVM instructions are library calls
- O(1) (rather than O(log n)) to run on the L1 blockchain

K-Way Dissection

- Alice: "State X leads to Y in N steps"
- Bob: "No. State X leads to:
  - $X_{1,B}$ in N/K steps
  - $X_{2,B}$ in N/K steps from $X_{1,B}$
  - ...
  - $X_{K,B}$ in N/K steps from $X_{K-1,B}$"
- A: "No. State $X_{2,B}$ leads to
  - $X_{(2,1),A}$ in N/K^2 steps..."
- Less rounds and messages involved in the dissections

Status

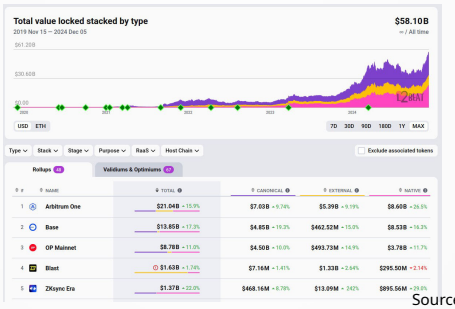
- Currently, **partly centralized**
- Ownership through a DAO governed by tokens
- Validators through an allow list
- The sequencer is centralized
  - It can only delay transactions, not prevent them since they can be posted on the L1 blockchain

L2 Panorama

Zero-Knowledge Rollups

- A promising alternative is based on **crypto proofs** of correctness
- The crypto machinery belongs to the category of **non-interactive zero-knowledge proofs**
  - Proof of correctness that do not reveal any specific information beyond the validity of the statement
  - **zk-SNARK**: zero-knowledge succinct non-interactive argument of knowledge
- Computation-intensive, difficult to prove generic computation

Existing L2 Chains



Source: l2beat.com

Decentralized Systems

Anonimity

Financial Privacy—the Good

- Which **goods one buys**
- **Supply chains** for businesses
- Financial **status**
  - Criminals target wealthy people
- Protecting against **oppressive governments**

Financial Privacy—the Bad

- Tax evasion
- Money laundering
- Criminal activities
- Financing sanctioned governments
- Unfortunately, very often, the good and the bad are **technically undistinguishable**

References

- Narayanan et al. Bitcoin and Cryptocurrency Technologies, Chapter 6 (free draft version).
- Narayanan et al. BTC-Tech: Bitcoin and Cryptocurrency Technologies. Princeton University online course, lecture 6 (also on Coursera).
- Andrei Savdeiev. Monero explanatory videos.

Anonymity

Pseudonimity and Anonymity

- Literally, “anonymous” = **without a name**
- We have seen **public key hashes,** not real names
  - In computer science, we call this **pseudonimity**
  - You can have as many pseudonyms as you want
- **Unlinkability**: different actions of the same user shouldn’t be **linkable to each other**
- **Anonymity = pseudonimity + unlinkability**

Is Unlinkability Needed?

- Pseudonimity can be **fragile**
- Many cryptocurrency services require **real identities**
  - Know-Your-Consumer (KYC)
  - People interacting with you can get personal information
- Side channels, based on **extra information** leaked
  - E.g., you send payments when you're awake and online, and you also post on social media in the same periods
  - In the long run, this may uncover who you are

Unlinkability in Cryptocurrencies

- It should be **hard** to link
  - Different **addresses** of the same user
  - Different **transactions** of the same user
  - The **sender** of a payment/message to its **recipient**

Anonymity Set

- The **crowd** one is **hiding into**
- We need to define an **adversary model**
- What they
  - **Know**
  - **Don't know**
  - **Can't know**

So, Bitcoin and Ethereum?

- Transactions are pseudonymous
  - They are **public forever**
  - We'll see the problem of **linkability**
- Privacy bottleneck in **exchanges**
  - **Converting** cryptocurrency to **fiat currency** (€, $)
  - And vice versa

Deanonymizing Bitcoin

Unlinkability

- Best practice: **always receive payments at a fresh address**
- Does this choice guarantee unlinkability?
- **Not necessarily**. This helps, but we can recover **patterns to link addresses**
  - When receiving (time, price, ...)
  - When spending

Alice Buys a Teapot

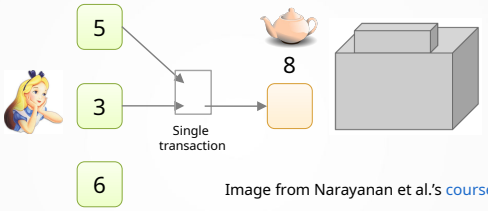


Image from Narayanan et al.'s course

- **Shared spending** may indicate **joint control** (i.e., the **same owner**)
- Addresses can be linked **transitively**

Alice Needs Change

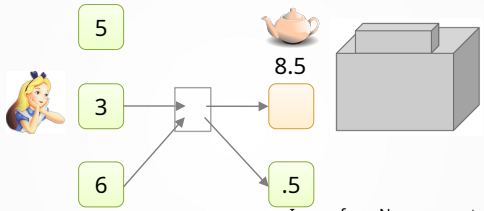


Image from Narayanan et al.'s course

- How to determine **which address is change?**
- In this case, it wouldn't make sense to combine two inputs if the cost is 0.5
- Other features depend on **how wallets are programmed**
  - E.g., last output in the transaction

Transaction Graph Analysis (1)

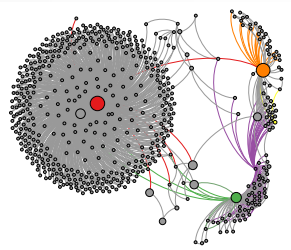


Image from Reid and Harrigan 2012

- Associate addresses that spent currency together
- In this cases, if some nodes (colored) are identified you can spot histories of payments
- This paper checks the story of Bitcoin coming from a theft

Transaction Graph Analysis (2)

- Nodes are **clusters of Bitcoin addresses**, size proportional to the transaction volumes
- Edges are **transactions**
- Authors deanonymized clusters by **transacting** with actors

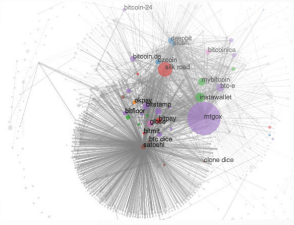


Image from Meiklejohn et al., IMC 2013

Deanonymizing Regular Users

- If one interacts with **popular services**
  - Authorities can **subpoena** them
  - Malicious entities can **attack** or **corrupt** them
- If one **publicly posts** one of their addresses
  - They can be linked with the others

Network Layer Deanonymization

Image from Narayanan et al.'s course

- Connect to **as many machines as possible** in the Bitcoin network
- Way easier than an Eclipse attack
- "The first node to inform you of a transaction is probably the source of it" (Dan Kaminsky, BlackHat 2011 talk (slides))
- Countermeasure: use Tor or similar software

Mixing

Centralized Mixers

- Services that **receive money from a given set of addresses** and return them to **other addresses**
- Different from exchanges in that they **promise not to record identities**
- Rely on **trust** and **reputation**

Decentralized Mixing: CoinJoin

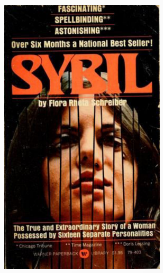
- Users find each other and build a transaction that sends money to **fresh addresses**
- They can sign if they see that it sends "their" money to the right address
- Anybody can send the tx

From	To
1 coin address 17	1 coin address 33
1 coin address 42	1 coin address 67
1 coin address 73	1 coin address 73

- This is **one** mixing round
- One generally wants to use **multiple ones** to increase anonymity set size

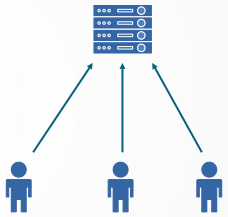
CoinJoin: Problems

- How to **find peers**
- Peers know the **mapping between inputs and outputs**
  - With a Sybil attack, you can learn it even on **multiple rounds**
- **Denial of Service**
  - A node disappears before signing
  - A node double spends the input before it passes to CoinJoin



CoinJoin: Finding Peers

- Easy! Just use an **untrusted server**
- If you think about it, the worst thing the server can do is **stop working**



CoinJoin: Mapping Inputs and Outputs

- Nodes can **use Tor** (or another anonymity solution)
- They need to use **different circuits** when communicating the inputs and the outputs
  - That way, they should be unlinkable

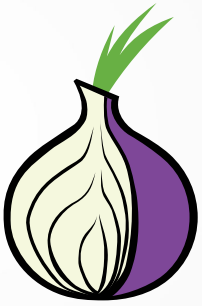


Image extracted from the Tor logo

CoinJoin: Denial of Service

- **Proof of work**
  - You must compute some hashes to talk to peers
- Proof of **burn**
  - You must destroy a small amount of currency (e.g. send to unspendable address)
- There are **cryptographic alternatives** that allow kicking non-cooperating users without revealing them

High-Level Flows

- Say Alice gets a weekly salary of 127.1425152 coins
- She puts 10% of it in a savings account right away
- This is a pattern that can be noticed **no matter what**

Merge Avoidance

Rather than a single transaction

- The receiver provides **multiple output addresses**
- The sender **avoids combining** different inputs

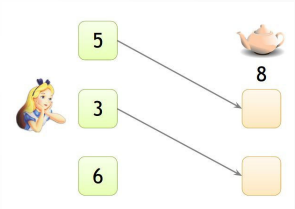


Image from the Princeton text book

Monero

Monero

- Esperanto for “coin”
- A proof-of-work cryptocurrency (XMR) designed for **anonymity** and **fungibility**
- **Fungible** goods are **interchangeable**
  - One may not accept bitcoins that are **tainted** because they come from theft, or a mix
  - Hence, Bitcoins may be **not fungible**
- Unsurprisingly, **liked by criminals**

Keys

- Each user has two asymmetric keypairs: **view** and **send**, which are not published on the blockchain
- To send XMR to Bob, Alice has to obtain **both public keys** of the recipient
- Bob’s private **view key** allows **reading** all transactions sent to Bob
- Bob’s private **send key** allows **spending** his XMR

Stealth Addresses

Crypto magic!

- With Bob's public and view keys plus some random data, Alice generates a **stealth address** for Bob
- Blockchain transactions are sent to this **stealth address**
  - She can later **prove** she **sent money to Bob**
  - The stealth address is **unlinkable** to Bob
- Bob scans the whole blockchain using his view key to **find which transactions are for him**
  - To spend them, he can compute a **one-time secret** for each of them to spend them that will be used **together with his spend key**

Ring Signatures

Crypto magic!

- Originally called **group signatures** (Rivest et al. 2001)
- Meaning: "this document has been signed by X, Y or Z"
  - You can't know who among them, though
- "This transaction is using funds from one output among A, B, C or D"
- How to prevent double spend? Using a **key image**
  - Unique crypto key derived from an output (and the send key)
  - Miners check it's never reused

Ring Confidential Transactions

Crypto magic!

- Hide the **amount** of the transaction
  - Before 2017, transactions could only have fixed amounts
  - Think cash: only 1, 2, 5, 10, 20, 50 euro notes…
- Old or newly-minted XMR need to be converted to RingCT outputs
- Miners verify a crypto proof that
  - The **sum of inputs** is **equal to the sum of outputs**
  - Every output is larger than zero

In Summary

- For a Monero transaction,
  - Ring signatures hide the **sender**
  - Ring confidential transactions hide the **amount**
  - Stealth addresses hide the **recipient**
- Moreover, Kovri (a Tor-like system) hides IP addresses

Decentralized Systems

Are Blockchains Good?

The Skeptics' Point of View

Politics and Ideology

- Some cryptocurrency proponents share **political views**
- **Crypto-anarchism**
  - Distrust institutions
  - **Anonymity**
  - **Freedom of speech**
  - **Freedom of trade**

(Monero community, CC-BY-SA 4.0)

Some Objections From a Big Debate

- Institutions are useful
  - **Public services** (e.g., health, instruction, security)
- There are reasons to **limit freedoms**, for example:
  - Trading weapons and illegal goods
  - Hate speech
  - Child abuse

Centralization: Economies of Scale

David Rosenthal makes this economic point:

- **Income** should be **linearly proportional** to contribution
  - If not, one can mount a **Sybil attack** to appear as several smaller entities
- **Cost** is **less then linear** due to **economies of scale**
  - Producing X objects costs less than X times the cost of producing one
- Hence, **P2P systems** inevitably **tend towards centralization**

User Tendency to Centralization

Moxie Marlinspike (creator of Signal):

- People **don't want to run their own server**
- **Protocols move** much more **slowly** than a (centralized) platform: less features, less convenient
- This is why we don't use our own mail and web server
- This is why **centralized services** like Infura, OpenSea, Coinbase, Etherscan show up
  - Even Metamask uses them to show e.g. the NFT in your wallet

Sybil Attack Vs. Democracy

- "One head, one vote" only works if we can **count heads**
  - Permissionless systems: subverted by creating **fake identities** at will with a Sybil attack
- Solutions:
  - Proof of work: "**one hash, one vote**"
    - Costs **infrastructure and energy**, hence **money**
  - Proof of stake: "**one coin, one vote**"
- They point towards **plutocracy** (government of the rich) rather than democracy

Inflation and Deflation

- An effective currency should have a **stable value**
  - Handling this is the job of central banks
  - Low but positive **inflation**—incentive to spend/invest
- Deflation: **value goes up** in time
  - Encourages people to invest: buy coin because it will be worth more in the future (the more people buy it, the higher the value)
  - If it is deflationary, one **doesn't want to spend it**
  - Paradox: a coin's success makes it a **bad currency**

Scams and Greater Fool

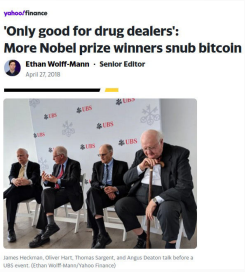
- The space is filled with scams of all kind
- **Rug pull**: somebody raises funds and runs away
- **Wash trade**: somebody buying from themselves
- **Greater fool** (e.g., Ponzi or piramid schemes): investments on a scheme that yields no returns
  - As long as new investors arrive, old investors are paid with the new investors' money
  - At some point the scheme inevitably collapses

Red Flags

- **Ignoring criticisms**
  - Marking them as FUD, "Fear Uncertainty and Doubt"
- **Proselitism** to recruit fresh money and advice to **never sell**
  - HODL ("Hold on for dear life"), HFSP ("Have fun staying poor"), "Diamond hands", "To the moon!"
- **Unclear business plan**

Are All Cryptocurrencies Scams?

- The **debate is up**
- According to some economists, they are pretty much all **greater fool** schemes
- It boils down to which **real-world applications**, beside speculations, they will have in the long run


yahoo/finance

'Only good for drug dealers': More Nobel prize winners snub bitcoin

Ethan Wolff-Mann · Senior Editor

April 27, 2018

James Heckman, Oliver Hart, Thomas Sargent, and Angus Deaton talk before a UBS event. (Ethan Wolff-Mann/Yahoo Finance)

Source: Yahoo! Finance