Course Introduction

Course Plan

Week 1

Course Introduction / Blockchain / Cryptography review
Solidity Review and New Features
EVM Deep Dive
Advanced Solidity / Design patterns / Dev tools

Week 2

Solidity Assembly part 1
Solidity Assembly part 2
Layer 2 and Rollups
Gas Optimisation part 1

Week 3

Gas Optimisation part 2
MEV
Security
Auditing

Week 4

Stablecoins
Useful Libraries
Research
Review Day

BUT
We can cover other topics, please show your interest using the Sli.do polls

Practical Details

All lessons will be conducted via zoom.
The format will usually be 45 mins of theory followed by 45 mins practical

You can work in teams if you wish
We use Sli.do to provide Q&A and polls : [link]

Homeworks

We call the practical sessions `homeworks` , you do not need to submit these.
We also have a `CTF` system and will ask you to complete some levels, more details later this week when we introduce it.

IDEs

We will be using mostly Foundry / Hardhat (in VSCode) plus Remix

About us

Free Developer Workshops

- Basic
- Enterprise
- Advanced EVM
- Zero Knowledge Proofs

Business Workshops

Website : https://extropy.io

Email : info@extropy.io

Twitter : @extropy

Modular Blockchains

If we follow the principle of separation of concerns, we can use a combination of blockchains to provide the functionality of a L1 and increase scalability.
For further details see Volt [article]

The Blockspace Market

See article

Demand Side

The demand side is formed from the collection of transactions supplied by users (via DApps)

Supply Side

The supply side is provided by the validators proposing and voting on blocks, adding to the Ethereum blockchain.
The 2 sides are mediated by congestion and fees.
The mechanics are complex and give rise to features such as

- MEV
- ASIC development

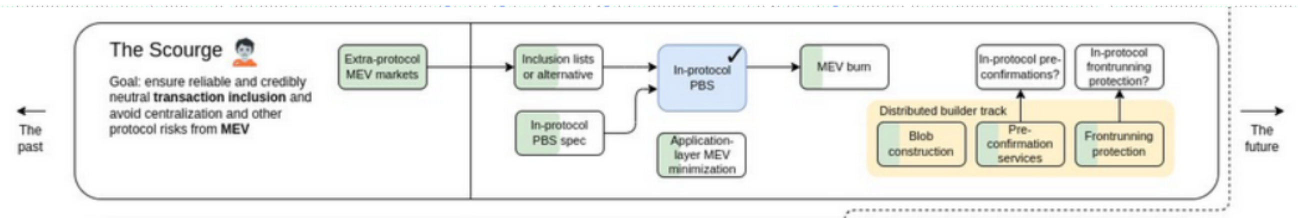
ETH2

ETH 2 - where has the term gone ?

"To limit confusion, the community has updated these terms:"

- 'Eth1' is now the 'execution layer', which handles transactions and execution.
- 'Eth2' is now the 'consensus layer', which handles proof-of-stake consensus.

The Ethereum roadmap



When finished this above system should be capable of 100k tx/sec (including rollups).
Source: https://twitter.com/VitalikButerin/status/1466411377107558402/photo/1

Consensus on Ethereum

From Ethereum developer documentation
"Now technically, proof-of-work and proof-of-stake are not consensus protocols by themselves, but they are often referred to as such for simplicity. They are actually Sybil resistance mechanisms and block author selectors; they are a way to decide who is the author of the latest block.
It is this Sybil resistance mechanism combined with a chain selection rule that makes up a true consensus mechanism."

There are 2 parts to block addition :

- block producer selection
- block acceptance

From yellow paper
" Since the system is decentralised and all parties have an opportunity to create a new block on some older pre-existing block, the resultant structure is necessarily a tree of blocks. In order to form a consensus as to which path, from root (the genesis block) to leaf (the block containing the most recent transactions) through this tree structure, known as the blockchain, there must be an agreed-upon scheme. If there is ever a disagreement between nodes as to which root-to-leaf path down the block tree is the ‘best’ blockchain, then a fork occurs."

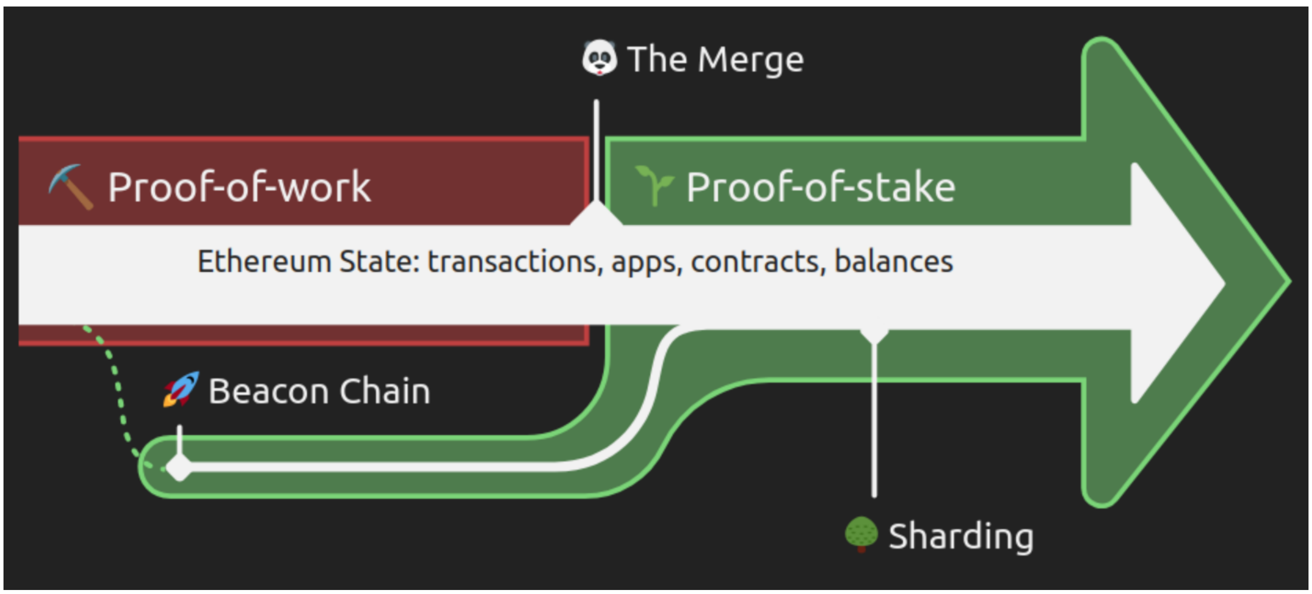
Consensus Mechanism Families

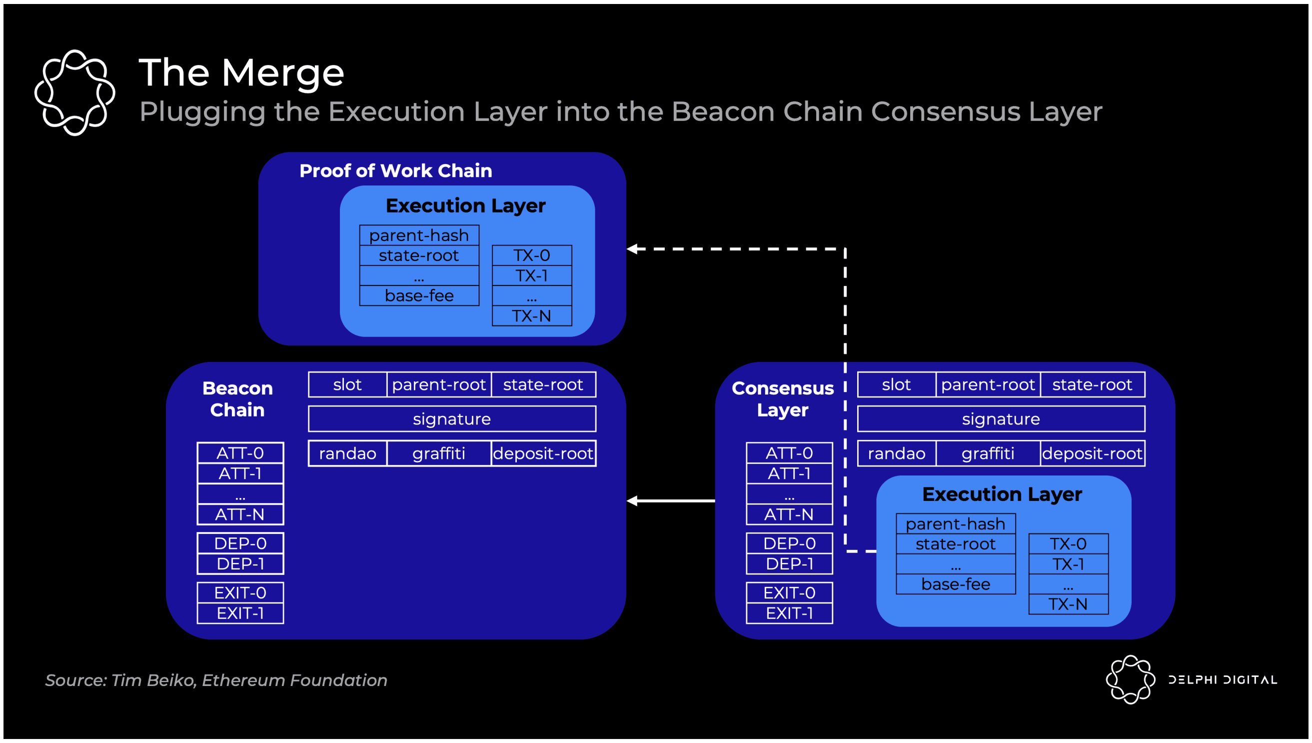
- Nakamoto style - Bitcoin, Ethereum pre merge
- BFT style - Ethereum post merge
- Avalanche Snow

The Merge - Proof of stake update

- Replaced proof of work with the proof of stake beacon chain.
  i.e. merging existing beacon chain into ethereum.
- Pre merge the Beacon Chain had not been processing Mainnet transactions. Instead, it has been reaching consensus on its own state by agreeing on active validators and their account balances.
- POS specs: https://github.com/ethereum/consensus-specs#phase-0

"The number `58750000000000000000000` joins the list of the most important integers of our time. The moment Ethereum's proof of work chain accumulates that much difficulty (read, hashes done), the entire network will switch over to proof of stake"

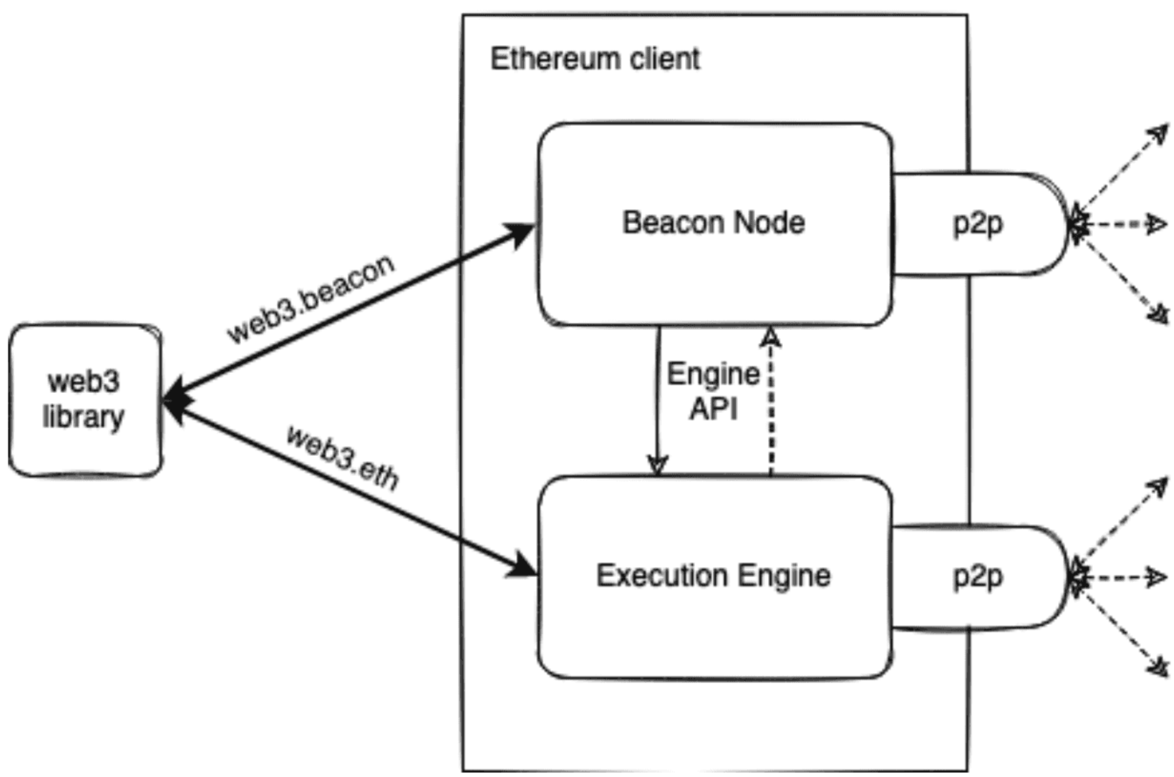




The Merge
Plugging the Execution Layer into the Beacon Chain Consensus Layer
Proof of Work Chain
Execution Layer
parent-hash
state-root
...
base-fee
TX-0
TX-1
...
TX-N
Beacon Chain
slot
parent-root
state-root
signature
randao
graffiti
deposit-root
ATT-0
ATT-1
...
ATT-N
DEP-0
DEP-1
EXIT-0
EXIT-1
Consensus Layer
slot
parent-root
state-root
signature
randao
graffiti
deposit-root
ATT-0
ATT-1
...
ATT-N
DEP-0
DEP-1
EXIT-0
EXIT-1
Execution Layer
parent-hash
state-root
...
base-fee
TX-0
TX-1
...
TX-N
Source: Tim Beiko, Ethereum Foundation
DELPHI DIGITAL

Ethereum clients after the merge

- Current Eth 1.0 clients continue to handle execution. They process blocks, maintain mempools, and manage and sync state. The PoW stuff has been taken out.
- Consensus client – Current Beacon Chain clients continue to handle PoS consensus. They track the chain's head, gossip and attest to blocks, and receive validator rewards.



Execution Clients See [Docs](https://ethereum.org/en/developers/docs/nodes-and-clients/#execution-clients)

Client	Language	Operating systems	Networks	Sync strategies	State pruning
Geth ↗	Go	Linux, Windows, macOS	Mainnet, Sepolia, Goerli	Snap, Full	Archive, Pruned
Nethermind ↗	C#, .NET	Linux, Windows, macOS	Mainnet, Sepolia, Goerli, and more	Snap (without serving), Fast, Full	Archive, Pruned
Besu ↗	Java	Linux, Windows, macOS	Mainnet, Sepolia, Goerli, and more	Snap, Fast, Full	Archive, Pruned
Erigon ↗	Go	Linux, Windows, macOS	Mainnet, Sepolia, Goerli, and more	Full	Archive, Pruned
Akula ↗	Rust	Linux	Mainnet, Sepolia, Goerli	Full	Archive, Pruned

(Open Ethereum has been deprecated)

Consensus Clients

See Docs

Client	Language	Operating systems	Networks
Lighthouse ↗	Rust	Linux, Windows, macOS	Beacon Chain, Goerli, Pyrmont, Sepolia, Ropsten, and more
Lodestar ↗	TypeScript	Linux, Windows, macOS	Beacon Chain, Goerli, Sepolia, Ropsten, and more
Nimbus ↗	Nim	Linux, Windows, macOS	Beacon Chain, Goerli, Sepolia, Ropsten, and more
Prysm ↗	Go	Linux, Windows, macOS	Beacon Chain, Gnosis, Goerli, Pyrmont, Sepolia, Ropsten, and more
Teku ↗	Java	Linux, Windows, macOS	Beacon Chain, Gnosis, Goerli, Sepolia, Ropsten, and more

A number of different synchronisation modes are available see docs

Client diversity in the network is very much encouraged

Consensus after the Merge

Ethereum has moved to Gasper (Casper FFG + LMD GHOST (Latest Message Driven Greediest Heaviest Observed SubTree))

Consensus relies on both LMD-GHOST – which adds new blocks and decides what the head of the chain is – and Casper FFG which makes the final decision on which blocks *are* and *are not* a part of the chain.
GHOST's favourable liveness properties allow new blocks to quickly and efficiently be added to the chain, while FFG follows behind to provide safety by finalising epochs.

The two protocols are merged by running GHOST from the last finalised block as decided upon by FFG.
By construction, the last finalised block is always a part of the chain which means GHOST doesn't need to consider earlier blocks.

Safety favouring protocols such as Tendermint can halt, if they don't get enough votes.
Liveness favouring protocols such as Nakamoto continue to add blocks, but they may not come to finality.
Ethereum will achieve finality by checkpointing

Epochs and checkpointing

Epochs of about 6 mins have 32 slots with all validators attesting to one slot (~12K attestations per block)
The fork-choice rule LMD GHOST then determines the current head of chain based on these attestations.
Finality is achieved when sufficient votes are reached, generally after 2 epochs.

Validator Selection and consensus in more detail

You can find stats about the validators here

Most recent epochs View more

Epoch	Time	Final	Eligible (ETH)	Voted
178,858	3 mins ago	No	Calculating...	Calculating...
178,857	9 mins ago	No	16,379,848	15,807,513 (96.51%)
178,856	16 mins ago	Yes	16,379,624	16,316,025 (99.61%)
178,855	22 mins ago	Yes	16,379,400	16,304,281 (99.54%)
178,854	28 mins ago	Yes	16,379,400	16,323,577 (99.66%)
178,853	35 mins ago	Yes	16,379,400	15,816,985 (96.57%)
178,852	41 mins ago	Yes	16,379,400	16,312,281 (99.59%)
178,851	48 mins ago	Yes	16,379,400	16,321,497 (99.65%)
178,850	54 mins ago	Yes	16,379,400	16,280,569 (99.40%)
178,849	1 hr ago	Yes	16,379,400	16,323,225 (99.66%)

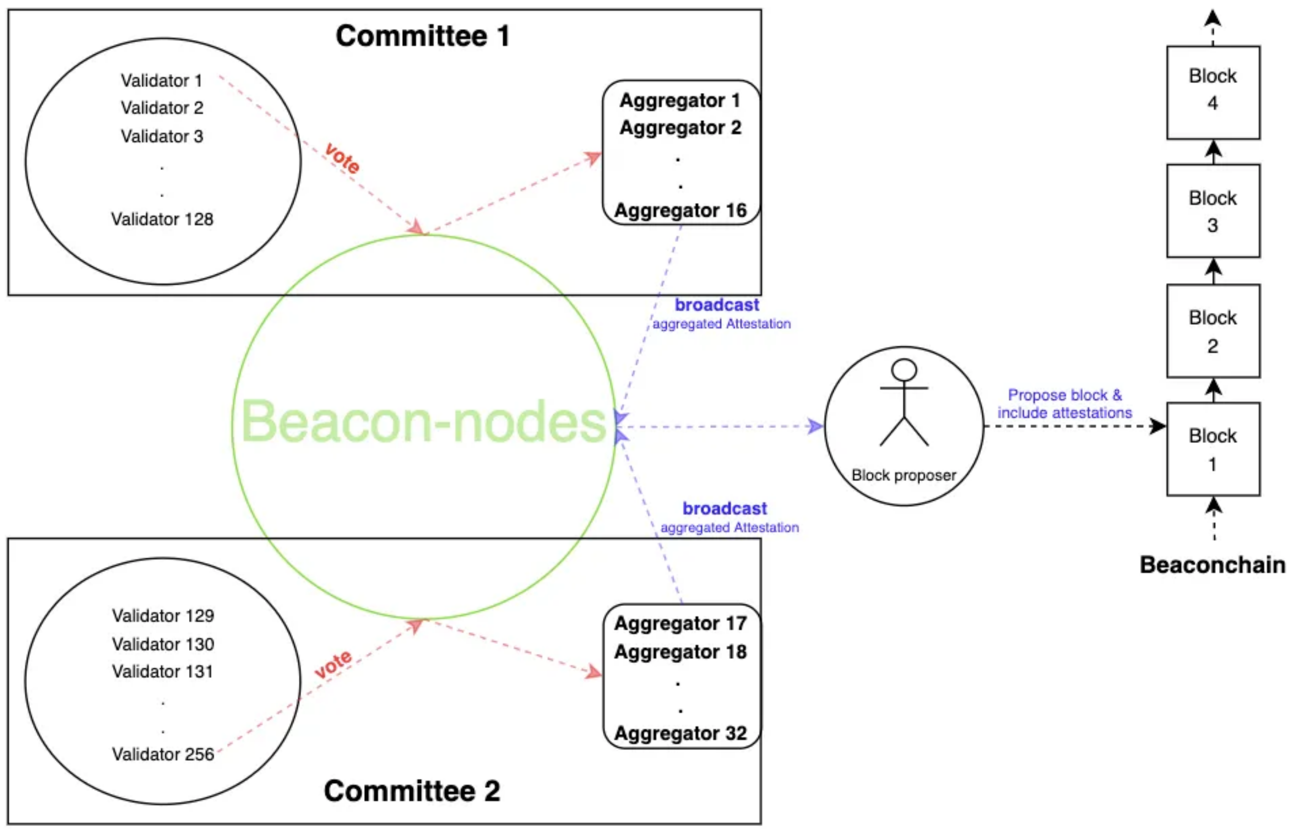
Most recent blocks View more

Epoch	Slot	Block	Status	Time	Proposer
178,858	5,723,471	16,554,754	Proposed	17 secs ago	437785
178,858	5,723,470	16,554,753	Proposed	29 secs ago	137513
178,858	5,723,469	16,554,752	Proposed	41 secs ago	89305
178,858	5,723,468	16,554,751	Proposed	53 secs ago	81882
178,858	5,723,467	16,554,750	Proposed	1 min ago	30424
178,858	5,723,466	16,554,749	Proposed	1 min ago	99864
178,858	5,723,465	16,554,748	Proposed	1 min ago	254509
178,858	5,723,464	16,554,747	Proposed	1 min ago	49345
178,858	5,723,463	16,554,746	Proposed	1 min ago	164741
178,858	5,723,462	16,554,745	Proposed	2 mins ago	162

A slot occurs every 12s and one validator is chosen to submit a block within that slot.
If the validator fails to do so, the slot is let empty.
The first block within an epoch (~6.4 mins) is a checkpoint block.

Coming to consensus about the block

If a validator is not chosen to produce a block, it will instead vote on what it regards as the current head of the chain and the checkpoint block.
Within an epoch a validator will only vote once.



Validators are grouped into committees at random, their votes are aggregated and published in the block header.

This article gives an in depth view of validator rewards

Fork choice rule

When faced with a potential fork, we choose the fork that has the most votes, but when counting the votes, we only include the last one from any validator.

Voting rules

The validator must vote for the chain they consider to be correct
The validator cannot vote for 2 blocks in any one time slot.

Finalisation

Validators vote on a pair of checkpoint blocks, to indicate that they are valid.
Once a checkpoint block gets sufficient votes, it is regarded as 'justified'
Once its child checkpoint block becomes justified, then the parent is regarded as final.

ETH 2 Validator rewards

Staking Rewards

In order to incentivize those that have ETH to stake in the network, validators will be rewarded for performing their assigned duties. Every 6 minutes, a validator is assigned a duty and is rewarded if it is performed. This reward is a sliding scale based on total network stake. So if total ETH staked is very low, the return rate per validator increases, but as stake rises, total annual issuance increases to fund those validators, while they individually will receive less rewards. The current suggested payouts are as follows:

ETH validating	Max annual issuance	Max annual network issuance %	Max annual return rate (for validators)
1,000,000	181,019	0.17%	18.10%
3,000,000	313,534	0.30%	10.45%
10,000,000	572,433	0.54%	5.72%
30,000,000	991,483	0.94%	3.30%
100,000,000	1,810,193	1.71%	1.81%

Rewards and penalties at the beginning of an epoch		
	Reward	Penalty
Matching the **source** vote	$base\ reward * \frac{attesting\ balance}{total\ active\ balance}$	$base\ reward$
Matching the **source** and **target** vote	$base\ reward * \frac{attesting\ balance}{total\ active\ balance}$	$base\ reward$
Matching the **source** and **head** vote	$base\ reward * \frac{attesting\ balance}{total\ active\ balance}$	$base\ reward$
Being a **proposer** in the votes that matched **source**	$\frac{base\ reward}{8} * block\ attesters$	N/A
Being an **attester** in the earliest vote that matched **source**	$\frac{7}{8} * base\ reward * \frac{1}{inclusion\ delay}$	N/A
Inactivity Penalty - System failed to finalize for 4 epochs	N/A	$4 * base\ reward$
Inactivity Penalty - If triggered, and the validator failed to both attest and match target	N/A	$eff\ bal * \frac{finality\ delay}{2^{25}}$

$$base\ reward = effective\ balance * \frac{BASE\_REWARD\_FACTOR}{BASE\_REWARDS\_PER\_EPOCH\ \sqrt{\Sigma\ active\ balance}}$$

- *Finality Delay = Previous epoch − Finalized epoch*
- *Attesting Balance = Sum of unslashed attester balance*
- *Constant* BASE_REWARD_FACTOR = 64
- *Constant* BASE_REWARDS_PER_EPOCH = 4
- *Constant* PROPOSER_REWARD_QUOTIENT = 8
- *Constant* MIN_EPOCHS_TO_INACTIVITY_PENALTY = 4
- *Constant* INACTIVITY_PENALTY_QUOTIENT = 2**25

Ethereum State, Transactions, and Blocks

World State. The world state is a mapping between addresses (160-bit identifiers) and account states
Though not stored on the blockchain, it is assumed that the implementation will maintain this mapping in a modified Merkle Patricia tree

The account state, comprises the following four fields:

- nonce: A scalar value equal to the number of transactions sent from this address or, in the case of accounts with associated code, the number of contract-creations made by this account.
- balance: A scalar value equal to the number of Wei owned by this address.
- storageRoot: A 256-bit hash of the root node of a Merkle Patricia tree that encodes the storage contents of the account (a mapping between 256-bit integer values), encoded into the trie as a mapping from the Keccak 256-bit hash of the 256-bit integer keys to the RLP-encoded 256-bit integer values.
- codeHash: The hash of the EVM code of this account—this is the code that gets executed should this address receive a message call; it is immutable and thus, unlike all other fields, cannot be changed after construction.

EIP-2930

Adds a transaction type which contains an access list, a list of addresses and storage keys that the transaction plans to access. Accesses outside the list are possible, but become more expensive. Intended as a mitigation to contract breakage risks introduced by EIP 2929 and simultaneously a stepping stone toward broader use of access lists in other contexts.

Digression about gas costs of opcodes
From [EIP2929]
Generally, the main function of gas costs of opcodes is to be an estimate of the time needed to process that opcode, the goal being for the gas limit to correspond to a limit on the time needed to process a block.
However, storage-accessing opcodes (`SLOAD`, as well as the `CALL`, `BALANCE` and `EXT` opcodes) have historically been underpriced.
In the 2016 Shanghai DoS attacks, once the most serious client bugs were fixed, one of the more durably successful strategies used by the attacker was to simply send transactions that access or call a large number of accounts.

Transactions

A transaction is a single cryptographically-signed instruction constructed by an actor externally to the scope of Ethereum. The sender of a transaction cannot be a contract.
As of the Berlin version of the protocol, there are two transaction types: 0 (legacy) and 1 (EIP-2930)

Fields :

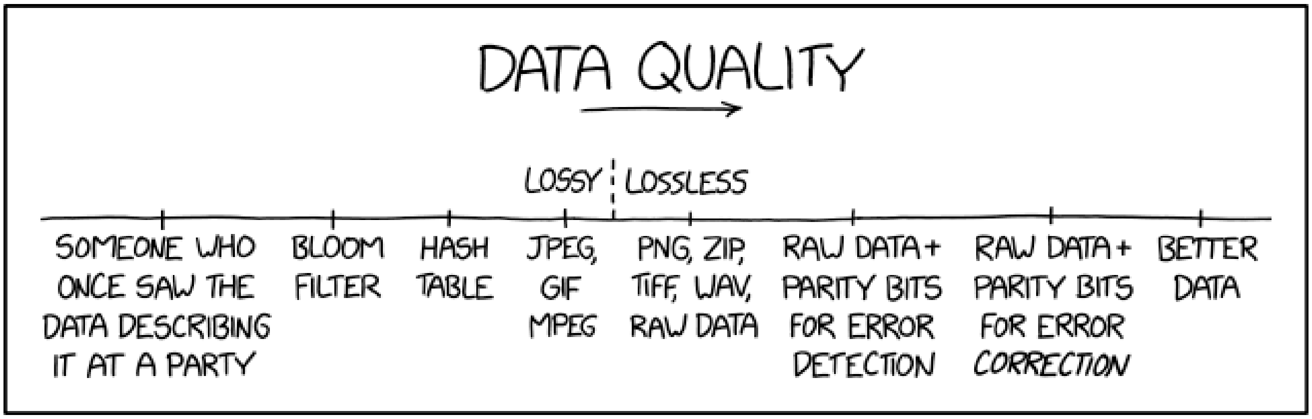
- type: EIP-2718 transaction type; formally Tx.
- nonce: A scalar value equal to the number of transactions sent by the sender; formally Tn.
- gasPrice: A scalar value equal to the number of Wei to be paid per unit of gas for all computation costs incurred as a result of the execution of this transaction; formally Tp.
- gasLimit: A scalar value equal to the maximum amount of gas that should be used in executing this transaction. This is paid up-front, before any computation is done and may not be increased later; formally Tg.
- to: The 160-bit address of the message call's recipient or, for a contract creation transaction, ∅, used here to denote the only member of B0 ; formally Tt.
- value: A scalar value equal to the number of Wei to be transferred to the message call's recipient or, in the case of contract creation, as an endowment to the newly created account; formally Tv.
- r, s: Values corresponding to the signature of the transaction and used to determine the sender of the transaction; formally Tr and Ts.

EIP-2930 (type 1) transactions also have:

- accessList: List of access entries to warm up. Each access list entry E is a tuple of an account address and a list of storage keys: E ≡ (Ea,Es).
- chainId
- yParity (in legacy transactions this is combined with the chain ID)

Bloom Filters

Oblig xkcd



A Bloom filter is a probabilistic, space-efficient data structure used for fast checks of set membership.
The Bloom filter principle: Wherever a list or set is used, and space is at a premium, consider using a Bloom filter if the effect of false positives can be mitigated.

Bloom filter answers are 'no' and 'maybe'
Bloom filters have the inverse behavior of caches

- bloom filter: miss == definitely not present, hit == probably present
- cache: miss == probably not present, hit == definitely present

The basic idea behind the Bloom filter is to hash each new element that goes into the data set, take certain bits from this hash, and then use those bits to fill in parts of a fixed-size bit array (e.g. set certain bits to 1). This bit array is called a bloom filter.

Later, when we want to check if an element is in the set, we simply hash the element and check that the right bits are 1 in the bloom filter. If at least one of the bits is 0, then the element definitely isn't in our data set! If all of the bits are 1, then the element might be in the data set, but we need to actually query the database to be sure. So we might have false positives, but we'll never have false negatives. This can greatly reduce the number of database queries we have to make.

For an interactive example, see

https://llimllib.github.io/bloomfilter-tutorial/

How are bloom filters used ?

Bloom filters are used with logs (events)
Instead of storing events from a transaction, we store the bloom filter, We store a

bloom filter for

- each transaction (if needed)
- for each block (for all transactions)

Thus a light client can check for an event in the block bloom filter, and if it probably exists, then check the transaction bloom filters.
A client storing the transaction receipts could then check those for the event.

Remember a Transaction receipt is a tuple of five items comprising:
-the type of the transaction
-the status code of the transaction
-the cumulative gas used in the block containing the transaction receipt as of immediately after the transaction has happened
-the set of logs created through execution of the transaction
-the Bloom filter composed from information in those logs

References

Ethereum yellow paper
Ethereum white paper

Account Abstraction

See docs from Nethermind
Also this blog from Ethereum and this blog from Gnosis
See this article from Binance Academy

Why Account Abstraction matters

The shift from EOAs to smart contract wallets with arbitrary verification logic paves the way for a series of improvements to wallet designs, as well as reducing complexity for end users. Some of the improvements Account Abstraction brings include:

- Paying for transactions in currencies other than ETH
- The ability for third parties to cover transaction fees
- Support for more efficient signature schemes (Schnorr, BLS) as well as quantum-safe ones (Lamport, Winternitz)
- Support for multisig transactions
- Support for social recovery

Previous solutions relied on centralised relay services or a steep gas overhead, which inevitably fell on the users' EOA.

EIP-4337 is a collaborative effort between the Ethereum Foundation, OpenGSN, and Nethermind to achieve Account Abstraction in a user-friendly, decentralised way.

Features

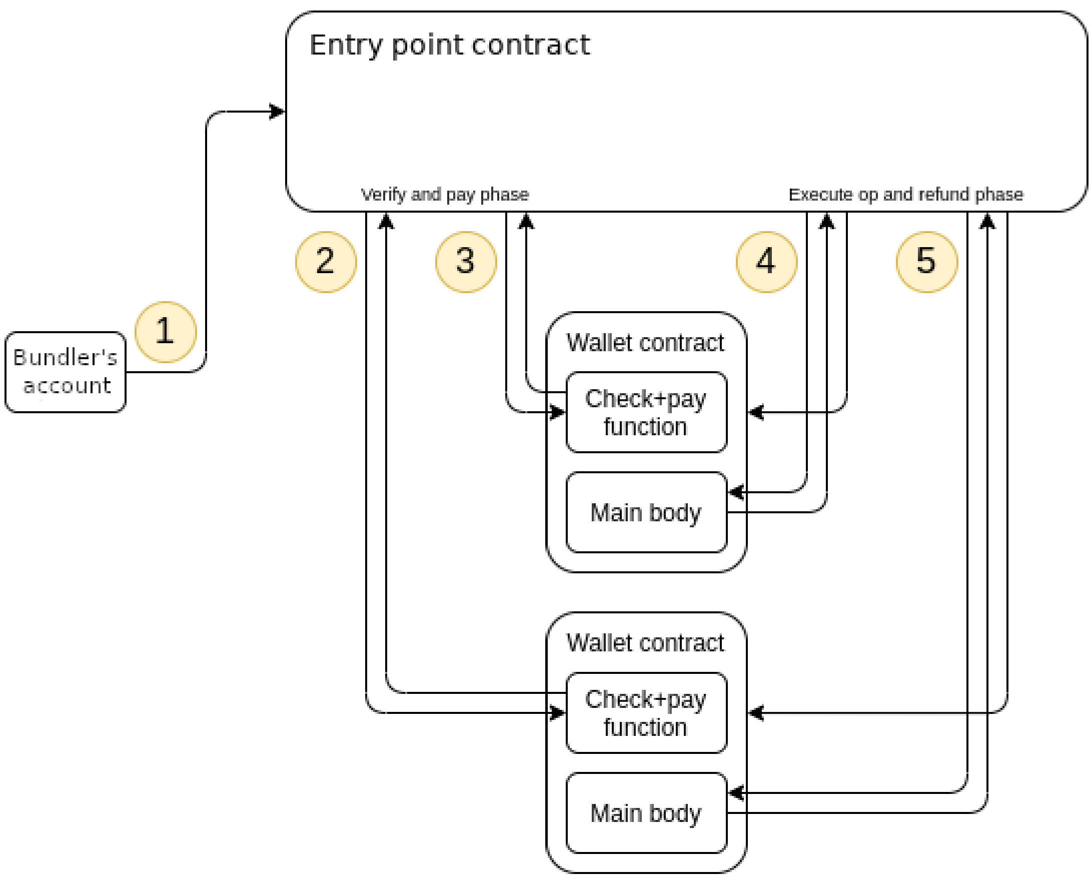
- **Achieve the key goal of account abstraction**: allow users to use smart contract wallets containing arbitrary verification logic instead of EOAs as their primary account. Completely remove any need at all for users to also have EOAs (as status quo SC wallets and EIP-3074 both require)
- **Decentralization**
    - Allow any bundler (think: block builder) to participate in the process of including account-abstracted user operations
    - Work with all activity happening over a public mempool; users do not need to know the direct communication addresses (eg. IP, onion) of any specific actors
    - Avoid trust assumptions on bundlers
- **Do not require any Ethereum consensus changes**: Ethereum consensus layer development is focusing on the merge and later on scalability-oriented features,

and there may not be any opportunity for further protocol changes for a long time. Hence, to increase the chance of faster adoption, this proposal avoids Ethereum consensus changes.

- **Try to support other use cases**
  - Privacy-preserving applications
  - Atomic multi-operations (similar goal to EIP-3074)
  - Pay tx fees with ERC-20 tokens, allow developers to pay fees for their users, and EIP-3074-like **sponsored transaction** use cases more generally
  - Support aggregated signature (e.g. BLS)

Instead of transactions, users send 'UserOperation' objects into a separate mempool, then actors called 'bundlers' package these up as transactions to a special contract. The entry point's `handleOps` function must perform the following steps
It must make two loops, the **verification loop** and the **execution loop**. In the verification loop, the `handleOps` call must perform the following steps for each `UserOperation`:

- **Create the account if it does not yet exist**
- **Call `validateUserOp` on the account**, passing in the `UserOperation`, the required fee and aggregator (if there is one).

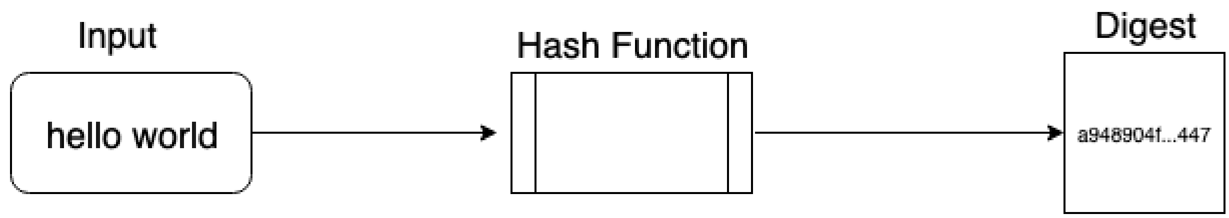


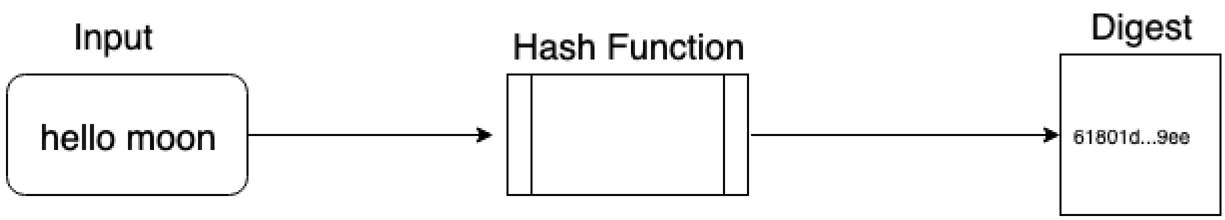
First-time account creation

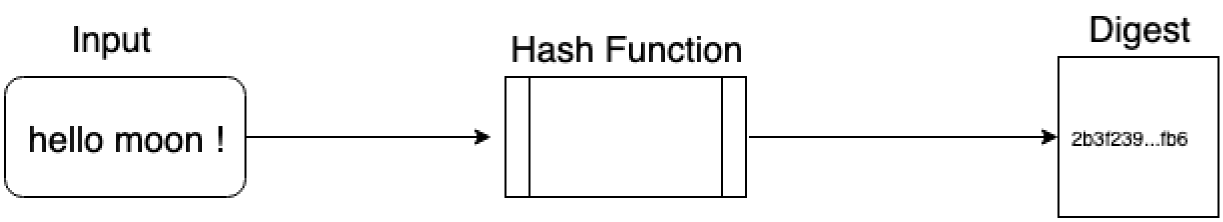
> It is an important design goal of this proposal to replicate the key property of EOAs that users do not need to perform some custom action or rely on an existing user to create their wallet; they can simply generate an address locally and immediately start accepting funds.
>
> The wallet creation itself is done by a “factory” contract, with wallet-specific data. The factory is expected to use CREATE2 (not CREATE) to create the wallet, so that the order of creation of wallets doesn’t interfere with the generated addresses.
>
> There is a certain synergy here with some of the MEV solutions.

Cryptography Review

Hash Functions

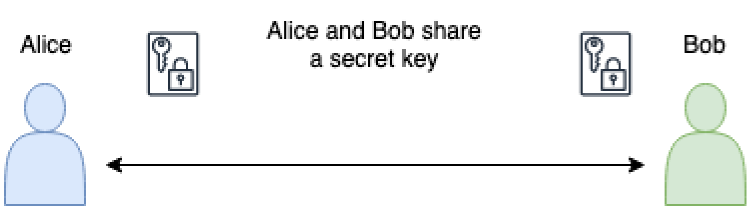


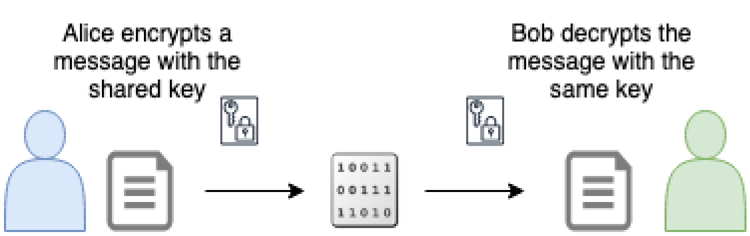




Encryption

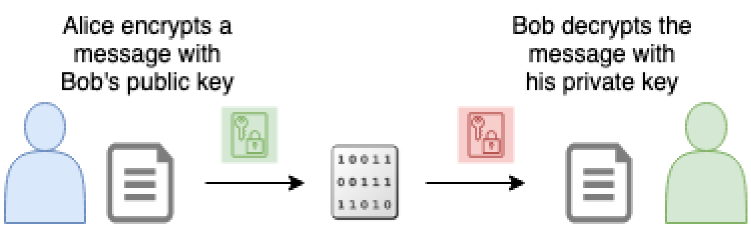
Symmetric Encryption



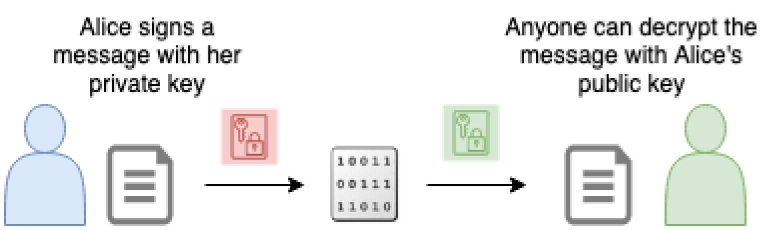


Asymmetric Encryption

Sending a secret message

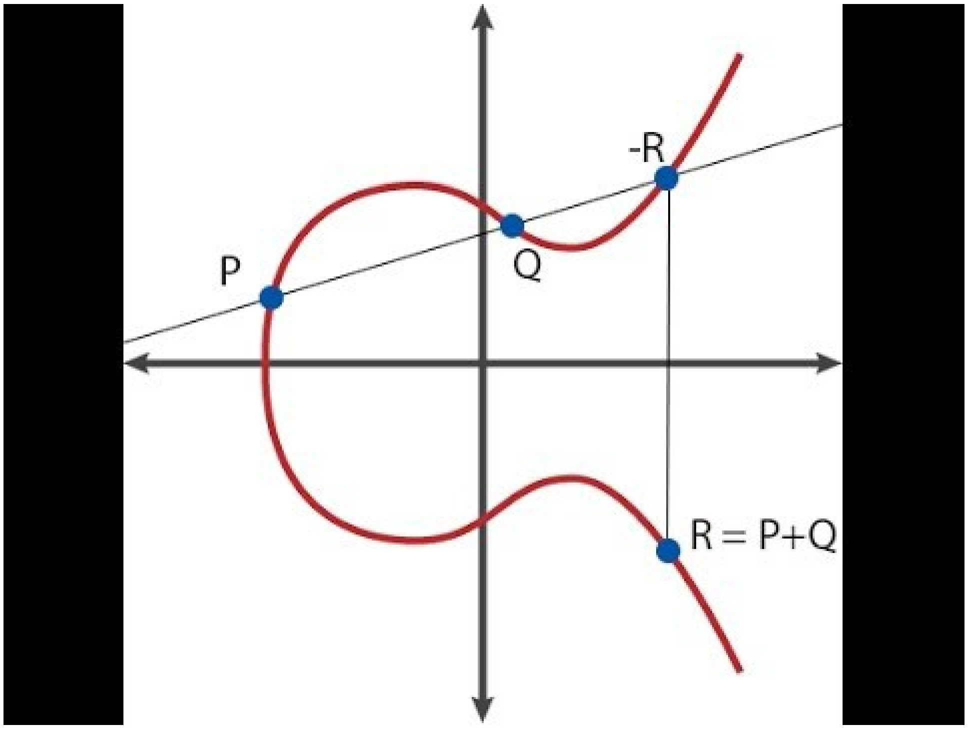


Proving ownership (knowledge of) of a private key



Elliptic Curves

The defining equation for an elliptic curve is for example $y^2 = x^3 + ax + b$



Ethereum uses the SECP-256k1 curve.

"LatestCourse/img/Screenshot 2022-02-19 at 16.01.36.png" is not created yet. Click to create.

Cryptographic Signatures

See appendix F of the Ethereum Yellow Paper

Inputs

- Input message
- Private Key
- Random secret

Output

- Digital signature

The process can be checked even though the private key and random secret remain unknown.

Ethereum uses Elliptic Curve Digital Signature Algorithm, or ECDSA.
Note that ECDSA is only a signature algorithm. Unlike RSA and AES, it cannot be used for encryption.

Signing and verifying using ECDSA

ECDSA signatures consist of two numbers (integers): `r` and `s`.
Ethereum also uses an additional `v` (recovery identifier) variable. The signature can be notated as `{r, s, v}`.

To create a signature you need the message to sign and the private key (d_a) to sign it with. The “simplified” signing process looks something like this:

1. Calculate a hash (`e`) from the message to sign.
2. Generate a secure random value for `k`.
3. Calculate point (x_1, y_1) on the elliptic curve by multiplying `k` with the `G` constant of the elliptic curve.
4. Calculate $r = x_1 \bmod n$. If `r` equals zero, go back to step 2.
5. Calculate $s = k^{-1}(e + rd_a) \bmod n$. If `s` equals zero, go back to step 2.

In Ethereum in step 1 we usually add
`Keccak256("\x19Ethereum Signed Message:\n32")`
to the beginning of the hashed message.

To verify the message you need

- the original message
- the address associated with the private key
- the signature `{s,r,v}`

v is either 27 or 28 in Bitcoin and Ethereum before EIP 155, since then, the chain ID is used in the calculation of v, to give protection against replaying transactions
`v = {0,1} + CHAIN_ID * 2 + 35`

Why do we need the v value ?
There can be up to 4 different points for a particular x coordinate modulo n
2 because each X coordinate has two possible Y coordinates (reflection in x axis), and
2 because r+n may still be a valid X coordinate
The v value is used to determine which one of the 4.

From the yellow paper
Recovery :
`ECDSARECOVER(e , v , r , s ) ≡ pu`
Where pu is the public key, assumed to be a byte array of size 64
e is the hash of the transaction, h(T).
v, r, s are the values taken from the signature as above.