1 Week 1

1.1 Why Study Blockchains

High level idea: Blockchains are a new paradigm for distributed secure systems.

They combine:

- Cryptography (hashes, signatures, PoW).
- Distributed systems (consensus, P2P networks).
- Economics and game theory (incentives, equilibria).

Why they matter:

- Help understand modern security mechanisms:
  - Key management and PKI.
  - Software integrity and update mechanisms.
  - Privacy enhancing technologies.
- Enable new organizational forms:
  - Cryptocurrency and DeFi.
  - DAOs, on chain governance.
  - Token based coordination and funding.
- Bitcoin is a concrete proof that a large scale open system can run for years without a central operator.

1.2 Blockchains and Distributed Ledgers

1.2.1 Concepts

Blockchain: a distributed, append only data structure that maintains a consistent log of transactions across many nodes.

Distributed ledger: a general term for systems that maintain shared state among multiple parties without a single trusted authority.

Desired properties:

- **Safety:** all honest nodes agree on the same history (no conflicting logs).
- **Liveness:** valid transactions are eventually included and confirmed.

Bitcoin is the first widely deployed blockchain protocol that achieves these properties in a permissionless setting.

1.3 Endless Ledger Parable

1.3.1 Book and Scribes

Intuition: model Bitcoin as an endlessly growing ledger maintained by many *scribes*.

- There is a shared ledger (a book) with many numbered pages.
- Anyone can become a scribe and propose a new page.
- Each page records a batch of transactions.
- The ledger never stops growing: new pages are added over time.

Constraint: adding a page requires expensive work.

- To write page i, the scribe must solve a hard puzzle, like throwing many dice until a rare pattern appears.
- This represents Proof of Work (PoW).

1.3.2 Forks and Longest Chain Rule

Multiple copies of the ledger may exist:

- Different scribes work in parallel and may produce conflicting next pages.
- Question: which ledger is the "correct" one?

Rule: everyone follows the ledger with the largest number of valid pages.

- If several ledgers have the same maximum length, pick the first one you received and keep writing on top of it.
- Pages not on the longest ledger become *orphan* pages.

This is the **longest chain rule**: choose the chain with the greatest cumulative work.

1.3.3 Randomness and Symmetry Breaking

Each page is produced by a random process (dice throwing, PoW search):

- With many scribes, the chances that two of them keep finding pages in perfect lockstep are tiny.
- Eventually one scribe gets ahead, creating a longer ledger.
- Other scribes then switch to this longer ledger.

Randomness breaks symmetry and lets the system converge on a single chain.

1.3.4 Incentives for Scribes

To motivate scribes to do the costly work:

- The rules allow the scribe who creates a valid new page to insert a special record awarding them a reward.
- In Bitcoin this is the block reward (newly minted coins) plus transaction fees.

Key points:

- Anyone with computing resources can become a miner.
- More computing power implies higher probability of winning the next block.

1.4 Scalable Service Provision Problem

General IT question:

How can we scale an online service to the whole world when participants do not trust each other and there is no central authority?

Traditional answers:

- **Federation:** multiple providers cooperate (e.g. email, XMPP).
- **Centralization:** one dominant provider (e.g. large social networks, cloud services).

Blockchains show a third option: decentralized provision by *resource owners* instead of fixed organizations.

1.4.1 Software Only Launch (SOL)

Goal: deploy a system purely by publishing software.

- Release an open source program.
- Announce a start time.
- Anyone can download the program and run it.
- When enough nodes run the software, the system "self boots" and becomes operational.

Bitcoin is a successful example of such a software only launch.

1.5 Hash Functions

1.5.1 Definition and Basic Properties

A hash function H maps inputs of arbitrary length to fixed length outputs.

Requirements:

- Efficient to compute.
- Output looks random and is well spread over the output space.

Cryptographic security properties:

- **Pre image resistance:** given y, it is hard to find any x with $H(x) = y$.
- **Second pre image resistance:** given x, it is hard to find $x' \neq x$ with $H(x') = H(x)$.
- **Collision resistance:** it is hard to find any pair $x \neq x'$ such that $H(x) = H(x')$.

1.5.2 Birthday Paradox

If there are n possible hash outputs, collisions appear surprisingly early.

- Approximate number of random samples needed for a collision with probability $\approx 50\%$: $k \approx 1.177\sqrt{n}$.
- For hash outputs of t bits, $n = 2^t$, so attacks based on collisions cost about $2^{t/2}$ operations.

1.5.3 Examples

- Broken: MD5, SHA 1 (known collisions).
- Current families: SHA 2 and SHA 3 with 224, 256, 384, 512 bit outputs.
- Bitcoin uses SHA 256 (from SHA 2 family).

1.6 Digital Signatures

1.6.1 API

A signature scheme $\Sigma = (\text{KeyGen}, \text{Sign}, \text{Verify})$:

- **KeyGen**:
$$(sk, vk) \leftarrow \text{KeyGen}(1^\lambda)$$
where sk is the secret signing key, vk is the public verification key.
- **Sign**:
$$\sigma \leftarrow \text{Sign}(sk, m)$$
where m is the message.
- **Verify**:
$$b \leftarrow \text{Verify}(vk, m, \sigma) \in \{0, 1\}$$
output 1 (accept) or 0 (reject).

1.6.2 Security Intuition

Existential unforgeability under chosen message attack (EU CMA):

- Attacker can obtain signatures on messages of their choice from a signing oracle.
- Even with this advantage, attacker should not be able to produce a valid signature on a new message that was never signed before.

1.6.3 Constructions

- Based on RSA (integer factorization hard).
- Based on discrete logarithms (DSA).
- Elliptic curve variants (ECDSA, Schnorr).

Bitcoin uses ECDSA originally and later also supports Schnorr style signatures.

1.7 Proof of Work (PoW)

1.7.1 Definition

A PoW scheme allows a prover to demonstrate that a certain amount of computational work has been done.

Typical hash based PoW:

$$\text{Find } w \text{ such that } H(\text{data} \| w) \leq T$$

where T is a difficulty target.

1.7.2 Simple Algorithm

```
ctr = 0
while H(data || ctr) > T:
    ctr = ctr + 1
return ctr
```

Properties:

- **Fast verification:** given w, one hash evaluation checks whether the condition holds.
- **No shortcuts:** for a well designed hash function, there is no significantly faster way than brute forcing different w.

1.7.3 Variants

- Standard Hashcash style PoW (used in Bitcoin).
- Memory hard PoW (e.g. scrypt, Equihash) to force large RAM usage.
- ASIC resistant PoW to reduce the advantage of specialized hardware.

1.8 Resource Based Systems

1.8.1 Resource Types

In resource based systems participation is tied to control of some scarce resource:

- **Proof of Work (PoW):** computational power.
- **Proof of Stake (PoS):** ownership of currency or tokens.
- **Proof of Space / Capacity:** available storage.
- **Proof of Time / Identity:** trusted hardware or other timing assumptions.

The system is not pinned to a fixed set of identities. Instead, any entity that can show a valid proof of resource can participate in maintaining the ledger.

1.8.2 PoW vs PoS

PoW:

- Pros: simple design, well studied, direct link between cost and security.
- Cons: high energy usage, hardware centralization (ASIC farms), environmental concerns.

PoS:

- Pros: lower energy usage, security based on economic value at stake.
- Cons: subtle security issues (nothing at stake, long range attacks), more complex protocol design.

1.9 Tokenomics

1.9.1 Basic Idea

Tokenomics studies how to use tokens and rewards to align incentives of participants.

Typical cycle:

- Users pay fees to use the service (transactions, smart contracts).
- The protocol distributes rewards to resource providers (miners, validators).
- Providers sell some of their tokens to cover costs and profit.

Goal: set parameters so that:

- Providing honest service is economically attractive.
- Attacking or misbehaving is economically disfavored.

1.10 Decentralized Service Provision

To run a decentralized service in an open network, the protocol must handle:

- **DoS resistance:** prevent abuse by spamming transactions or connections.
- **Consistency:** all honest participants eventually agree on the same state.
- **Liveness and censorship resistance:** valid transactions should not be permanently excluded.
- **Fairness of rewards:** contributions of resource providers should be measured and rewarded in a predictable way.

1.10.1 Reward Sharing

Rewards can be distributed:

- Per block or per action (e.g. each mined block gets a fixed reward).
- Per epoch (e.g. aggregate rewards over a time window then share according to contribution).

Design challenge:

- Ensure that rational, self interested participants collectively form a robust and secure system.
- Avoid centralization and cartel behavior where possible.

1.11 Quick Summary

- Blockchains provide a new way to build global services without central operators.
- The "endless ledger" parable explains how longest chain and randomness yield consensus.
- Hash functions and digital signatures are the basic cryptographic tools for integrity and authenticity.
- Proof of Work ties block production to computational effort and is easy to verify.
- Resource based systems use PoW, PoS, or other proofs to select and reward maintainers.
- Tokenomics and reward sharing mechanisms align incentives so that honest behavior is profitable.

2 Week2

2.1 Authenticated File Storage

Goal: Store a file on an untrusted server but keep only a short local state so that later you can check whether the server returned the correct data.

Client has identifier F and data D. It sends (F, D) to the server, and wants to delete D locally while still being able to verify any future response from the server.

2.1.1 Naive solution (does not help)

Client keeps a full local copy of D and checks equality with any D' returned by the server. This gives integrity but saves no storage.

2.2 Basic Cryptographic Tools

2.2.1 Hash-based authentication

Hash function H is collision resistant.

- Upload: client sends (F, D) to server.
- Commit: client stores only $h = H(D)$, deletes D.
- Retrieval: server returns D'.
- Verify: client accepts if $H(D') = h$, rejects otherwise.

Properties:

- Client keeps a short fixed-size value.
- Integrity relies on collision resistance of H.

2.2.2 Digital signatures

Signature scheme $\Sigma = (\text{KeyGen}, \text{Sign}, \text{Verify})$.

- Key generation: client runs KeyGen to get (sk, vk).
- Upload: compute $\sigma = \text{Sign}(sk, \langle F, D\rangle)$, send (F, D, σ) to server.
- Client keeps only vk.
- Retrieval: server returns (D', σ').
- Verify: accept if $\text{Verify}(vk, \langle F, D'\rangle, \sigma') = 1$.

Difference from pure hashing:

- Signatures are publicly verifiable (third parties can check).
- Useful for transferable proofs of origin and integrity.

2.3 Merkle Trees

2.3.1 Structure

Goal: authenticate large data split into blocks, and allow efficient verification of individual blocks.

- Split data D into blocks $D_1, \ldots, D_n$.
- Compute leaf hashes $H_1 = H(D_1), \ldots, H_n = H(D_n)$.
- Build a binary tree:

$$H_{i,j} = H(H_i \| H_j)$$

  up to a single *Merkle root* MTR.

Client stores only MTR as the commitment to the entire file.

2.3.2 Merkle-based storage protocol

- Upload: client sends full D to server, builds Merkle tree locally and computes root MTR, then keeps only MTR.
- Retrieval of a block D_x: server returns D_x and a *proof of inclusion* π.
- Verification: client uses D_x, π, and H to recompute a root and checks that it equals stored MTR.

2.3.3 Proof of inclusion

For a block D_x:

- Proof consists of all sibling hashes along the path from the leaf $H(D_x)$ to the root.
- Verifier:
  1. Starts from $H(D_x)$.
  2. Iteratively combines with sibling hashes and hashes upward.
  3. Checks whether the final value equals MTR.

Tree height is $O(\log n)$ for n leaves, so proof size and verification time are $O(\log n)$.

2.3.4 Applications

- BitTorrent: verify file chunks during download.
- Bitcoin: Merkle tree of transactions inside each block.
- Ethereum: variants of Merkle trees for state and transactions.

2.4 Merkle Trees for Sets

Goal: store a set S on a server and later prove membership or non-membership of any element x.

Construction:

- Sort the elements of S.
- Build a Merkle tree where leaves are sorted elements.

2.4.1 Membership proof

If $x \in S$, the server provides a normal proof of inclusion for the leaf corresponding to x.

2.4.2 Non-membership proof

If $x \notin S$:

- Find neighbors $H_<$ and $H_>$ in the sorted order such that $H_< < x < H_>$.
- Provide inclusion proofs for $H_<$ and $H_>$.
- Show they are adjacent in the sorted set representation.
- Conclude that x is not present.

2.5 Tries and Patricia Tries

2.5.1 Trie (prefix tree)

Data structure for a set of key-value pairs $\{(key, value)\}$ where keys are strings.

- Each edge is labeled with a character.
- A path from the root spells out a key.
- Nodes may store values for keys ending at that node.

Operations:

- **add(key,value)**: follow or create edges for each character, then store the value at the final node.
- **query(key)**: follow edges by characters, and check whether the final node has a value.

2.5.2 Patricia Trie

Compressed version of a Trie:

- Any chain of nodes where each node has a single child and no value can be merged into a single edge labeled with a substring.
- Saves space and reduces tree height.

Patricia tries are widely used in blockchain systems for efficient key-value storage.

2.6 Merkle Patricia Trie (MPT)

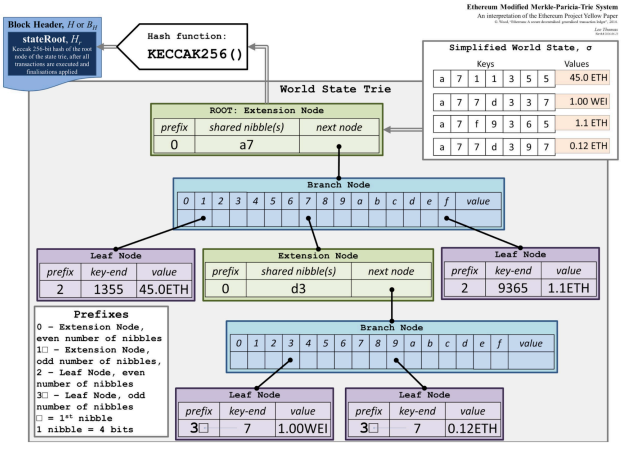
Ethereum combines Merkle hashing with Patricia tries.

2.6.1 Node types

Keys are encoded in hexadecimal nibbles. There are three logical node types:

- **Leaf node**: stores remaining key fragment and the value.
- **Extension node**: stores a shared key prefix and a pointer to another node.
- **Branch node**: has up to 16 child pointers (for hex digits) plus an optional value.

Each logical node is serialized and hashed. Child pointers store the hash of the child node, so the root hash commits to the entire key-value map.



2.6.2 Properties

- Root hash acts as a commitment to the whole dictionary.
- Inclusion proofs: show the path from the root down to a leaf and the content of intermediate nodes.
- Non-inclusion proofs: show that the search path terminates at a node that proves no matching key exists.
- Ethereum uses an MPT for the global world state, and the state root hash is stored in the block header.

2.7 Blockchain Data Structures

2.7.1 Block structure

A block consists of:

- Random nonce ctr.
- Data x (e.g. transactions, root hashes).
- Pointer s to the previous block (usually a hash of the previous header).

The header typically includes (ctr, x, s). The pointer field s creates a hash chain of blocks back to the genesis block.

2.7.2 Proof of Work (PoW)

In PoW systems a block header must satisfy:

$$H(\text{ctr}\|x\|s) \leq T$$

where T is a global difficulty target.

- Miners fix x and s, iterate over ctr, and search for a header whose hash is below T.
- The same hash is also used as the block identifier.

2.8 Bitcoin Overview

2.8.1 High-level protocol

1. New transactions are broadcast to the network.
2. Nodes collect transactions into candidate blocks.
3. Nodes perform PoW to find valid blocks.
4. A node that finds a valid block broadcasts it.
5. Other nodes validate all transactions and the PoW before accepting.
6. Nodes start mining on top of the longest valid chain.

2.8.2 UTXO model

Bitcoin represents ownership via unspent transaction outputs (UTXOs).

- A transaction has multiple inputs and outputs.
- Each **input** references a previous output (by transaction hash and index) and provides a script that authorizes spending.
- Each **output** specifies a value and a script defining how it may be spent in the future.

2.8.3 Scripts

Typical pay-to-public-key-hash (P2PKH) transaction:

- Output script (`scriptPubKey`):

$$\text{OP\_DUP OP\_HASH160 } \langle \text{pubKeyHash} \rangle \text{ OP\_EQUALVERIFY OP\_CHECKSIG}$$

- Input script (`scriptSig`) when spent:

$$\langle sig \rangle \ \langle pubKey \rangle$$

During validation, the combined script is executed to check that the spender owns the corresponding private key.

2.8.4 Merkle tree of transactions in a block

- All transactions in a block are organized as a Merkle tree.
- The Merkle root is stored in the block header.
- Simplified payment verification (SPV) clients download only block headers and proofs of inclusion for specific transactions.

2.9 Bitcoin Network

2.9.1 P2P topology

- All nodes run the same open-source protocol.
- Each node maintains connections to a set of peers.
- The network is permissionless: nodes may join or leave at any time.

Bootstrapping:

- Peer-to-peer nodes come "pre-installed" with some peers by IP / host.
- A user can also manually configure known peers.

2.9.2 Gossip protocol

- When a node learns about a new transaction or block, it forwards it to its peers.
- Each peer that sees a new item forwards it to its own peers.
- Nodes ignore items they have already seen.
- This peer-to-peer diffusion eventually spreads data to most honest nodes.

2.9.3 Eclipse attacks and connectivity assumption

Connectivity assumption: every honest node can reach every other honest node through some path in the network.

Eclipse attack:

- Attacker surrounds a victim with malicious peers.
- Victim only connects to attacker-controlled nodes and is cut off from the honest network.
- This can delay or hide blocks and transactions, enabling double spending or inconsistent views.

Maintaining good connectivity and diverse peer sets is critical for blockchain security.

2.10 Quick Summary

- Hashes and signatures support basic authenticated storage.
- Merkle trees give efficient proofs of inclusion with size $O(\log n)$.
- Merkle trees extend to sets and enable non-membership proofs.
- Tries store key-value maps by sharing prefixes; Patricia tries compress long paths.
- Merkle Patricia tries combine hashing and compressed tries; Ethereum uses them for state.
- Blockchain data structures link blocks using hash pointers and PoW.
- Bitcoin uses UTXOs, scripts, Merkle trees of transactions, and a P2P gossip network.