



Seventh Semester B.E./B.Tech. Degree Examination, Dec.2024/Jan.2025
Block Chain Technology

Time: 3 hrs.

Max. Marks: 100

Note: Answer any FIVE full questions, choosing ONE full question from each module.

Module-1

- 1 a. Define Block Chain. With the neat diagram explain the Network view of a block chain. (07 Marks)
b. Explain the different types of block chain. (08 Marks)
c. Mention the benefits and limitations of block chain. (05 Marks)

OR

- 2 a. Explain the methods that are used to achieve decentralization. (04 Marks)
b. Describe the Decentralized organizations in block chain. (06 Marks)
c. Define the following :
i) Decentralization ii) Centralized systems iii) Distributed systems
iv) Atomicity v) Cap theorem. (10 Marks)

Module-2

- 3 a. Discuss about the cryptographic hash functions and its properties. (10 Marks)
b. Write a note on Merkle trees. (05 Marks)
c. What is double-spending problem? How to overcome this problem in crypto currency? (05 Marks)

OR

- 4 a. Discuss about the distributed consensus protocol with an example. (07 Marks)
b. Explain the incentive mechanisms used in Bitcoin. (06 Marks)
c. Explain the three important properties of hash puzzles. (07 Marks)

Module-3

- 5 a. Illustrate the concept on Bitcoin transactions along with its syntax. (07 Marks)
b. Explain how to execute a script in a stack-based programming language. (06 Marks)
c. Summarize the applications of Bitcoin scripts. (07 Marks)

OR

- 6 a. Explain hot and cold storage for Bitcoin along with its solutions, in detail. (10 Marks)
b. Discuss about the payment service for Bitcoin in detail. Also explain its steps involved in Bitcoin Mines. (10 Marks)

Module-4

- 7 a. Discuss the tasks involved in bitcoin miners. (10 Marks)
b. Explain the mining incentives and strategies to be followed by miners. (10 Marks)

Important Note : 1. On completing your answers, compulsorily draw diagonal cross lines on the remaining blank pages.
2. Any revealing of identification, appeal to evaluator and /or equations written eg. 42+8 = 50, will be treated as malpractice.

21CS734

OR

- 8 a. Explain the functions of decentralized mixing in Bitcoin. (05 Marks)
b. Discuss in detail how to De-anonymize Bitcoin. (10 Marks)
c. Write a note on Zerocoin. (05 Marks)

Module-5

- 9 a. Define Smart Contract. Explain the properties of Ricardian contract Bowtie model with a neat diagram. (08 Marks)
b. Explain Ethereum block chain with a state transition function. (06 Marks)
c. Explain the types of accounts for ethereum. (06 Marks)

OR

CMRIT LIBRARY
BANGALORE - 560 037

- 10 a. Write a note on precompiled contracts. (08 Marks)
b. Explain EVM (Ethereum Virtual Machine) with a neat diagram. (08 Marks)
c. Mention the various functions of the Iterator functions. (04 Marks)

* * * * *

1 a	Definition: Blockchain is a distributed, decentralized digital ledger that records transactions across many computers in such a way that the registered transactions cannot be altered retroactively. Each set of transactions is grouped into a “block,” and these blocks are linked or “chained” together using cryptographic hashes, forming a secure and transparent chain of data.
-----	--

* Blockchain at its core is 'Distributed ledger' that is cryptographically secure, append-only, immutable and updateable only via consensus or agreement among peers.

Peer to peer: This means that there is no central controller in the network and all the participants talk to each other directly.

* This property allows for cash transactions to be exchanged directly among the peers without a third-party involvement such as a bank.

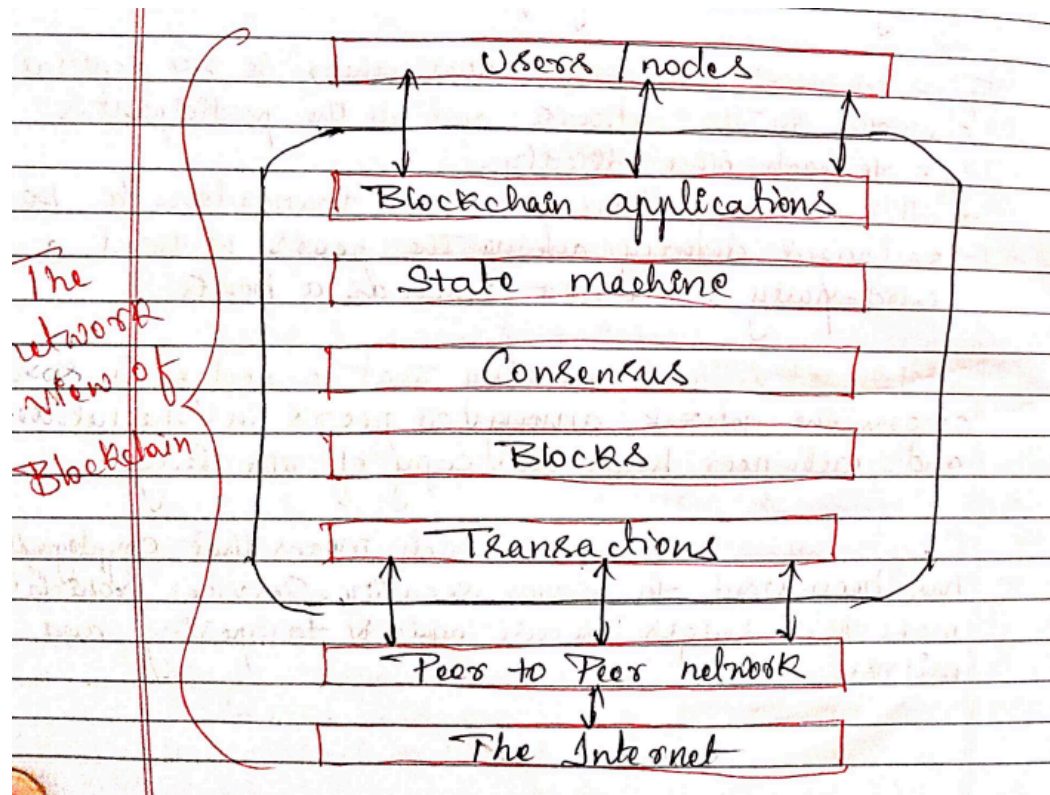
Distributed ledger: meaning that a ledger is spread across the network among all peers in the network and each peer holds the copy of the ledger.

Cryptographically Secure: which means that cryptography has been used to provide security services which

Append-only: the data can only be added to the blockchain in time-ordered sequential order.

It indicates once data is added to the blockchain it is almost impossible to change that data and considered practically immutable.

Updateable: updateable via consensus, which enables the power of decentralization.



- * At the bottom layer, there is the Internet, which provides a basic communication layer for any n/w.
- * Above a peer-to-peer network runs on top of the Internet, which hosts the another layer of Blockchain. This layer contains transactions, blocks, consensus mechanism, state machines and blockchain smart contracts.
- * All of these components are shown as a single logical entity in a box, representing blockchain above the peer-to-peer network.
- * Finally at the top, there are users or nodes that connect to the blockchain and perform various operations.

* A blockchain can be defined as a platform where peers can exchange value / electronic cash using transactions without the need for a centrally-trusted arbitrator.

* A block is merely a selection of transactions bundled together and organized logically.

* A transaction is a record of an event. for example transferring amount from a Sender's account to beneficiary account.

1b.

Types of Block chain

1. Public Blockchain

✓ Description:

A **public blockchain** is a **completely decentralized**, open-source network. Anyone can **join the network**, **participate in consensus**, and **view the ledger**. These blockchains are transparent and trustless — meaning no central authority is needed.

🔧 Features:

- **Permissionless:** Anyone can read, write, or audit the blockchain.
- **Decentralized governance.**
- **Immutability:** Once data is written, it cannot be changed.
- **Incentive-based:** Participants (miners/validators) are rewarded.

🔄 Consensus Mechanisms:

- Proof of Work (PoW)
- Proof of Stake (PoS)
- Delegated PoS, etc.

👛 Use Cases:

- Cryptocurrencies (Bitcoin, Ethereum)
- Decentralized Finance (DeFi)
- Non-Fungible Tokens (NFTs)
- Public voting systems

Examples:

- **Bitcoin** – first and most popular cryptocurrency
 - **Ethereum** – smart contract platform
 - **Litecoin, Dogecoin, etc.**
-

2. Private Blockchain

Description:

A **private blockchain** is a **permissioned network** where access is restricted to a certain organization or a group. It is **centrally controlled**, and only authorized participants can read or write data.

Features:

- **Permissioned:** Only selected users can access or contribute.
- **Faster performance** due to fewer nodes and controlled environment.
- **Low energy consumption.**
- **Customizable privacy and security settings.**

Consensus Mechanisms:

- Practical Byzantine Fault Tolerance (PBFT)
- RAFT, Istanbul BFT, etc.

Use Cases:

- Enterprise-level applications
- Internal auditing
- Supply chain management
- Healthcare data sharing

Examples:

- **Hyperledger Fabric** (IBM)
 - **Multichain**
 - **R3 Corda**
-

3. Consortium Blockchain (Federated Blockchain)

Description:

A **consortium blockchain** is a hybrid between public and private blockchains. It's **permissioned** but not controlled by a single entity — rather, a **group of organizations** manages the network.

Features:

- **Partially decentralized.**
- **Shared responsibility and governance.**
- Faster than public blockchains but more secure than private ones.
- Requires **predefined consensus** among participants.

Consensus Mechanisms:

- Voting-based consensus
- Multi-party signature-based

Use Cases:

- Cross-institutional collaboration (banks, governments)
- Trade finance
- Insurance claims management
- Supply chain consortiums

Examples:

- **Energy Web Foundation**
 - **IBM Food Trust**
 - **Marco Polo Network**
-

4. Hybrid Blockchain

Description:

A **hybrid blockchain** combines elements of both public and private blockchains. It allows **controlled access to certain data**, while **other data can be public**. It offers the **flexibility** of both types.

Features:


- **Customizable transparency** and security.
- Enables **internal and public interactions**.
- Allows **selective sharing** of data.
- Useful in scenarios where certain data must be **publicly verifiable** while others stay **private**.

Consensus Mechanisms:

- Custom or combined mechanisms, depending on architecture

Use Cases:

- Government systems

	<ul style="list-style-type: none">● Real estate (public property records + private buyer info)● Enterprise blockchain solutions● Retail and banking systems <p> Examples:</p> <ul style="list-style-type: none">● Dragonchain – built by Disney for enterprise use● XinFin – hybrid blockchain for global trade and finance● IBM Blockchain Platform
1c	<p>Benefits, features, and limitations of blockchain</p> <p>Numerous advantages of blockchain technology have been discussed in many industries and proposed by thought leaders around the world who are participating in the blockchain space. The notable benefits of blockchain technology are as follows:</p> <ol style="list-style-type: none">1. Decentralization: This is a core concept and benefit of blockchain. There is no need for a trusted third party or intermediary to validate transactions; instead, a consensus mechanism is used to agree on the validity of transactions.2. Transparency and trust: As blockchains are shared and everyone can see what is on the blockchain, this allows the system to be transparent. As a result, trust is established. This is more relevant in scenarios such as the disbursement of funds or benefits where personal discretion in relation to selecting beneficiaries needs to be restricted.3. Immutability: Once the data has been written to the blockchain, it is extremely difficult to change it back. It is not genuinely immutable, but because changing data is so challenging and nearly impossible, this is seen as a benefit to maintaining an immutable ledger of transactions.

	<ol style="list-style-type: none"> 4. High availability: As the system is based on thousands of nodes in a peer-to-peer network, and the data is replicated and updated on every node, the system becomes highly available. Even if some nodes leave the network or become inaccessible, the network as a whole continues to work, thus making it highly available. This redundancy results in high availability. 5. Highly secure: All transactions on a blockchain are cryptographically secured and thus provide network integrity. Any transactions posted from the nodes on the blockchain are verified based on a predetermined set of rules. Only valid transactions are selected for inclusion in a block. The blockchain is based on proven cryptographic technology that ensures the integrity and availability of data. Generally, confidentiality is not provided due to the requirements of transparency. This limitation is the leading barrier to its adoption by financial institutions and other industries that require the privacy and confidentiality of transactions. As such, the privacy and confidentiality of transactions on the blockchain are being researched very actively, and advancements are already being made. It could be argued that, in many situations, confidentiality is not needed and transparency is preferred. For example, with Bitcoin, confidentiality is not an absolute requirement; however, it is desirable in some scenarios. A more recent example is Zcash (https://z.cash), which provides a platform for conducting anonymous transactions. Other security services, such as non-repudiation and authentication, are also provided by blockchain, as all actions are secured using private keys and digital signatures. 6. Simplification of current paradigms: The current blockchain model in many industries, such as finance or health, is somewhat disorganized. In this model, multiple entities maintain their own databases and data sharing can become very difficult due to the disparate nature of the systems. However, as a blockchain can serve as a single shared ledger among many interested parties, this can result in simplifying the model by reducing the complexity of <u>managing the separate systems maintained</u> by each entity.
2a	<p>Methods of decentralization</p> <p>Two methods can be used to achieve decentralization: disintermediation and competition. These methods will be discussed in detail in the sections that follow.</p>

	<p>Disintermediation</p> <p>The concept of disintermediation can be explained with the aid of an example. Imagine that you want to send money to a friend in another country. You go to a bank, which, for a fee, will transfer your money to the bank in that country. In this case, the bank maintains a central database that is updated, confirming that you have sent the money. With blockchain technology, it is possible to send this money directly to your friend without the need for a bank. All you need is the address of your friend on the blockchain. This way, the intermediary (that is, the bank) is no longer required, and decentralization is achieved by disintermediation. It is debatable, however, how practical decentralization through disintermediation is in the financial sector due to the massive regulatory and compliance requirements. Nevertheless, this model can be used not only in finance but in many other industries as well, such as health, law, and the public sector. In the health industry, where patients, instead of relying on a trusted third party (such as the hospital record system) can be in full control of their own identity and their data that they can share directly with only those entities that they trust. As a general solution, blockchain can serve as a decentralized health record management system where health records can be exchanged securely and directly between different entities (hospitals, pharmaceutical companies, patients) globally without any central authority.</p> <p>Contest-driven decentralization</p> <p>In the method involving competition, different service providers compete with each other in order to be selected for the provision of services by the system. This paradigm does not achieve complete decentralization. However, to a certain degree, it ensures that an intermediary or service provider is not monopolizing the service. In the context of blockchain technology, a system can be envisioned in which smart contracts can choose an external data provider from a large number of providers based on their reputation, previous score, reviews, and quality of service.</p>
2b.	<p>Decentralized autonomous organizations</p> <p>Just like DOs, a decentralized autonomous organization (DAO) is also a computer program that runs on top of a blockchain, and embedded within it are governance and business logic rules. DAOs and DOs are fundamentally the same thing. The main difference, however, is that DAOs are autonomous, which means that they are fully automated and contain artificially intelligent logic. DOs, on the other hand, lack this feature and rely on human input to execute business logic.</p> <p>Ethereum blockchain led the way with the introduction of DAOs. In a DAO, the code is considered the governing entity rather than people or paper contracts. However, a human curator maintains this code and acts as a proposal evaluator for the community. DAOs are capable of hiring external contractors if enough input is received from the token holders (participants).</p> <p>Decentralized autonomous corporations</p> <p>Decentralized autonomous corporations (DACs) are similar to DAOs in concept, though considered to be a subset of them. The definitions of DACs and DAOs may sometimes overlap, but the general distinction is that DAOs are usually considered to be nonprofit, whereas DACs can earn a profit via shares offered to the participants and to whom they can pay dividends. DACs can run a business automatically without human intervention based on the logic programmed into them.</p> <p>Decentralized applications</p> <p>All the ideas mentioned up to this point come under the broader umbrella of decentralized applications, abbreviated to DApps. DAOs, DACs, and DOs are DApps that run on top of a blockchain in a peer-to-peer network. They represent the latest advancement in decentralization technology.</p> <p>DApps at a fundamental level are software programs that execute using either of the following methods. They are categorized as Type 1, Type 2, or Type 3 DApps:</p>

1. **Type 1:** Run on their own dedicated blockchain, for example, standard smart contract based DApps running on Ethereum. If required, they make use of a native token, for example, ETH on Ethereum blockchain.



For example, Ethlance is a DApp that makes use of ETH to provide a job market. More information about Ethlance can be found at <https://ethlance.com>.

2. **Type 2:** Use an existing established blockchain. that is, make use of Type 1 blockchain and bear custom protocols and tokens, for example, smart contract based tokenization DApps running Ethereum blockchain.
An example is DAI, which is built on top of Ethereum blockchain, but contains its own stable coins and mechanism of distribution and control. Another example is Golem, which has its own token GNT and a transaction framework built on top of Ethereum blockchain to provide a **decentralized marketplace** for computing power where users share their computing power with each other in a peer-to-peer network.

3. **Type 3:** Use the protocols of Type 2 DApps; for example, the SAFE Network uses the OMNI network protocol.



More information on the SAFE Network can be found at <https://safenetwork.tech>.

Another example to understand the difference between different types of DApps is the USDT token (Tethers). The original USDT uses the OMNI layer (a Type 2 DApp) on top of the Bitcoin network. USDT is also available on Ethereum using ERC20 tokens. This example shows that a USDT can be considered a Type 3 DApp, where the OMNI layer protocol (a Type 2 DApp) is used, which is itself built on Bitcoin (a Type 1 DApp). Also, from an Ethereum point of view USDT can also be considered a Type 3 DApp in that it makes use of the Type 1 DApp Ethereum blockchain using the ERC 20 standard, which was built to operate on Ethereum.

Decentralized organizations

DOs are software programs that run on a blockchain and are based on the idea of actual organizations with people and protocols. Once a DO is added to the blockchain in the form of a smart contract or a set of smart contracts, it becomes decentralized and parties interact with each other based on the code defined within the DO software.

2C	<p>Decentralization is the process of distributing and delegating power, authority, and decision-making away from a central authority to multiple individuals, groups, or nodes within a system. In technology (especially blockchain), it means that no single entity controls the entire network — instead, control is shared across all participants.</p> <p>Centralized System refers to a structure where control, authority, and decision-making are concentrated in a single central entity or organization. All operations, data storage, and governance are managed from this central point.</p> <p>Centralized System refers to a structure where control, authority, and decision-making are concentrated in a single central entity or organization. All operations, data storage, and governance are managed from this central point.</p> <p>Atomicity is one of the key properties of database transactions (the "A" in ACID) and refers to the concept that a transaction must be all or nothing — it either completes entirely or does not happen at all. If any part of the transaction fails, the entire operation is rolled back, leaving the system in its original state.</p> <p>The CAP Theorem (also known as Brewer's Theorem) states that in a distributed data system, it is impossible to simultaneously guarantee all three of the following properties:</p> <ol style="list-style-type: none"> 1. Consistency (C) 2. Availability (A) 3. Partition Tolerance (P) <p>A distributed system can only achieve two out of the three at any given time — not all three together</p>
3A	<p>1.1 Cryptographic Hash Functions</p> <p>The first cryptographic primitive that we'll need to understand is a cryptographic hash function. A hash function is a mathematical function with the following three properties:</p> <ul style="list-style-type: none"> • Its input can be any string of any size. • It produces a fixed size output. For the purpose of making the discussion in this chapter concrete, we will assume a 256-bit output size. However, our discussion holds true for any output size as long as it is sufficiently large. • It is efficiently computable. Intuitively this means that for a given input string, you can figure

Property 1: Collision-resistance. The first property that we need from a cryptographic hash function is that it's collision-resistant. A collision occurs when two distinct inputs produce the same output. A hash function $H(\cdot)$ is collision-resistant if nobody can find a collision. Formally:

Collision-resistance: A hash function H is said to be collision resistant if it is infeasible to find two values, x and y , such that $x \neq y$, yet $H(x)=H(y)$.



Figure 1.1 A hash collision. x and y are distinct values, yet when input into hash function H , they produce the same output.

Notice that we said *nobody can find* a collision, but we did not say that no collisions exist. Actually, we know for a fact that collisions do exist, and we can prove this by a simple counting argument. The input space to the hash function contains all strings of all lengths, yet the output space contains only strings of a specific fixed length. Because the input space is larger than the output space (indeed, the input space is infinite, while the output space is finite), there must be input strings that map to the same output string. In fact, by the Pigeonhole Principle there will necessarily be a very large number of possible inputs that map to any particular output.

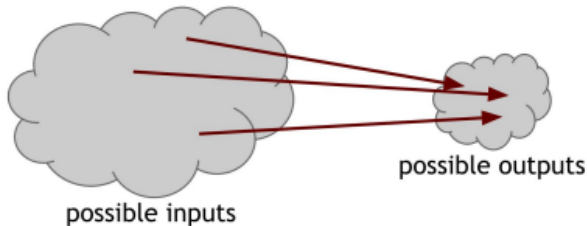


Figure 1.2 Because the number of inputs exceeds the number of outputs, we are guaranteed that there must be at least one output to which the hash function maps more than one input.

	<p>Property 2: Hiding The second property that we want from our hash functions is that it's hiding. The hiding property asserts that if we're given the output of the hash function $y = H(x)$, there's no feasible way to figure out what the input, x, was. The problem is that this property can't be true in the stated form. Consider the following simple example: we're going to do an experiment where we flip a coin. If the result of the coin flip was heads, we're going to announce the hash of the string "heads". If the result was tails, we're going to announce the hash of the string "tails".</p> <p>We then ask someone, an adversary, who didn't see the coin flip, but only saw this hash output, to figure out what the string was that was hashed (we'll soon see why we might want to play games like this). In response, they would simply compute both the hash of the string "heads" and the hash of the string "tails", and they could see which one they were given. And so, in just a couple steps, they can figure out what the input was.</p> <p>The adversary was able to guess what the string was because there were only two possible values of x, and it was easy for the adversary to just try both of them. In order to be able to achieve the hiding property, it needs to be the case that there's no value of x which is particularly likely. That is, x has to be chosen from a set that's, in some sense, very spread out. If x is chosen from such a set, this method of trying a few values of x that are especially likely will not work.</p> <p>The big question is: can we achieve the hiding property when the values that we want do not come from a spread-out set as in our "heads" and "tails" experiment? Fortunately, the answer is yes! So perhaps we can hide even an input that's not spread out by concatenating it with another input that is spread out. We can now be slightly more precise about what we mean by hiding (the double vertical bar \parallel denotes concatenation).</p> <div><p>Hiding. A hash function H is hiding if: when a secret value r is chosen from a probability distribution that has <i>high min-entropy</i>, then given $H(r \parallel x)$ it is infeasible to find x.</p></div> <p>work, in the sense that it will have the necessary security properties.</p> <p>Property 3: Puzzle friendliness. The third security property we're going to need from hash functions is that they are puzzle-friendly. This property is a bit complicated. We will first explain what the technical requirements of this property are and then give an application that illustrates why this property is useful.</p> <div><p>Puzzle friendliness. A hash function H is said to be puzzle-friendly if for every possible n-bit output value y, if k is chosen from a distribution with high min-entropy, then it is infeasible to find x such that $H(k \parallel x) = y$ in time significantly less than 2^n.</p></div> <p>Intuitively, what this means is that if someone wants to target the hash function to come out to some particular output value y, that if there's part of the input that is chosen in a suitably randomized way, it's very difficult to find another value that hits exactly that target.</p>
3b	<p>Merkle tree is a cryptographic data structure used to efficiently and securely verify the integrity of data. It consists of a binary tree where each leaf node is the hash of a data block, and each non-leaf node is the hash of the concatenation of its two child nodes. The root of the tree, called the Merkle root, represents a cryptographic fingerprint of all the data in the tree. This structure is particularly useful in distributed systems and blockchain technology, as it allows for efficient verification of data without needing to access the entire dataset. In a blockchain, for example, a Merkle tree helps to verify that a transaction is included in a block by using a Merkle proof, which provides only a small subset of the tree to confirm the validity of a transaction. Merkle trees are tamper-resistant, meaning that even small changes in the data will cause significant changes to the root hash, making it easy to detect unauthorized modifications. This structure is widely used in applications such as cryptocurrencies, file sharing systems, and digital signatures to ensure data integrity and efficiency.</p>

DOUBLE SPENDING PROBLEM

Double-spend attack. Alice may try to launch a double-spend attack. To understand how that works, let's assume that Alice is a customer of some online merchant or website run by Bob, who provides some online service in exchange for payment in bitcoins. Let's say Bob's service allows the download of some software. So here's how a double-spend attack might work. Alice adds an item to her shopping cart on Bob's website and the server requests payment. Then Alice creates a Bitcoin transaction from her address to Bob's and broadcasts it to the network. Let's say that some honest node creates the next block, and includes this transaction in that block. So there is now a block that was created by an honest node that contains a transaction that represents a payment from Alice to the merchant Bob.

Recall that a transaction is a data structure that contains Alice's signature, an instruction to pay to Bob's public key, and a hash. This hash represents a pointer to a previous transaction output that Alice received and is now spending. That pointer must reference a transaction that was included in some previous block in the consensus chain.

Note, by the way, that there are two different types of hash pointers here that can easily be confused. Blocks include a hash pointer to the previous block that they're extending. Transactions include one or more hash pointers to previous transaction outputs that are being redeemed.

Let's return to how Alice can launch a double spend attack. The latest block was generated by an honest node and includes a transaction in which Alice pays Bob for the software download. Upon seeing this transaction included in the block chain, Bob concludes that Alice has paid him and allows Alice to download the software. Suppose the next random node that is selected in the next round happens to be controlled by Alice. Now since Alice gets to propose the next block, she could propose a block that ignores the block that contains the payment to Bob and instead contains a pointer to the previous block. Furthermore, in the block that she proposes, Alice includes a transaction that transfers the very coins that she was sending to Bob to a different address that she herself controls. This is a classic double-spend pattern. Since the two transactions spend the same coins, only one of them can be included in the block chain. If Alice succeeds in including the payment to her own address in the block chain, then the transaction in which she pays Bob is useless as it can never be included later in the block chain.

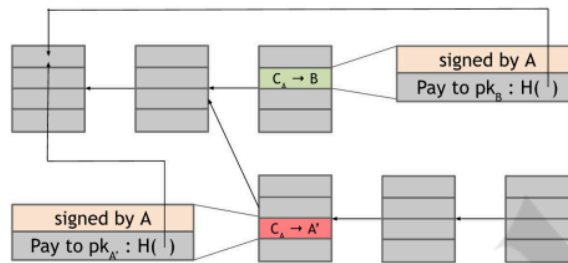


Figure 2.2 A double spend attempt. Alice creates two transactions: one in which she sends Bob Bitcoins, and a second in which she double spends those Bitcoins by sending them to a different address that she controls. As they spend the same Bitcoins, only one of these transactions can be included in the block chain. The arrows are pointers from one block to the previous block that it extends including a hash of that previous block within its own contents. C_A is used to denote a coin owned by Alice.

And how do we know if this double spend attempt is going to succeed or not? Well, that depends on which block will ultimately end up on the long-term consensus chain — the one with the Alice → Bob transaction or the one with the Alice → Alice transaction. What determines which block will be included? Honest nodes follow the policy of extending the longest valid branch, so which branch will they extend? There is no right answer! At this point, the two branches are the same length — they only differ in the last block and both of these blocks are valid. The node that chooses the next block then may decide to build upon either one of them, and this choice will largely determine whether or not the double-spend succeeds.

A subtle point: from a moral point of view, there is a clear difference between the block containing the transaction that pays Bob and the block containing the transaction in which Alice double spends those coins to her own address. But this distinction is only based on our knowledge of the story that Alice first paid Bob and then attempted to double spend. From a technological point of view, however, these two transactions are completely identical and both blocks are equally valid. The nodes that are looking at this really have no way to tell which is the morally legitimate transaction.

4A

2.2 Distributed consensus

We've discussed, in a generic manner, centralization and decentralization. Let's now examine decentralization in Bitcoin at a more technical level. A key term that will come up throughout this discussion is **consensus**, and specifically, **distributed consensus**. The key technical problem to solve in building a distributed e-cash system is achieving distributed consensus. Intuitively, you can think of our goal as decentralizing ScroogeCoin, the hypothetical currency that we saw in the first chapter.

Distributed consensus has various applications, and it has been studied for decades in computer science. The traditional motivating application is reliability in distributed systems. Imagine you're in charge of the backend for a large social networking company like Facebook. Systems of this sort typically have thousands or even millions of servers, which together form a massive distributed database that records all of the actions that happen in the system. Each piece of information must be recorded on several different nodes in this backend, and the nodes must be in sync about the overall

state of the system.

The implications of having a distributed consensus protocol reach far beyond this traditional application. If we had such a protocol, we could use it to build a massive, distributed key-value store, that maps arbitrary keys, or names, to arbitrary values. A distributed key-value store, in turn, would enable many applications. For example, we could use it to build a distributive domain name system, which is simply a mapping between human understandable domain names to IP addresses. We could build a public key directory, which is a mapping between email addresses (or some other form of real-world identity) to public keys.

That's the intuition of what distributed consensus is, but it is useful to provide a technical definition as this will help us determine whether or not a given protocol meets the requirements.

Distributed consensus protocol. There are n nodes that each have an input value. Some of these nodes are faulty or malicious. A distributed consensus protocol has the following two properties:

- It must terminate with all honest nodes in agreement on the value
- The value must have been generated by an honest node

What does this mean in the context of Bitcoin? To understand how distributed consensus could work in Bitcoin, remember that Bitcoin is a peer-to-peer system. When Alice wants to pay Bob, what she actually does is broadcast a transaction to all of the Bitcoin nodes that comprise the peer-to-peer network. See Figure 2.1.

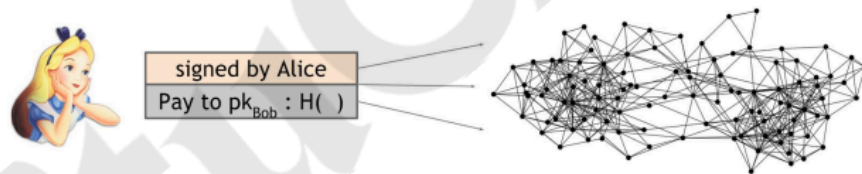


Figure 2.1 Broadcasting a transaction In order to pay Bob, Alice broadcasts the transaction to the entire Bitcoin peer-to-peer network.

Incidentally, you may have noticed that Alice broadcasts the transaction to all the Bitcoin peer-to-peer nodes, but Bob's computer is nowhere in this picture. It's of course possible that Bob is running one of the nodes in the peer-to-peer network. In fact, if he wants to be notified that this transaction did in fact happen and that he got paid, running a node might be a good idea. Nevertheless, there is no requirement that Bob be listening on the network; running a node is not necessary for Bob to receive the funds. The bitcoins will be his whether or not he's operating a node on the network.

Property 3: Puzzle friendliness. The third security property we're going to need from hash functions is that they are puzzle-friendly. This property is a bit complicated. We will first explain what the technical requirements of this property are and then give an application that illustrates why this property is useful.

Puzzle friendliness. A hash function H is said to be puzzle-friendly if for every possible n -bit output value y , if k is chosen from a distribution with high min-entropy, then it is infeasible to find x such that $H(k \parallel x) = y$ in time significantly less than 2^n .

Intuitively, what this means is that if someone wants to target the hash function to come out to some particular output value y , that if there's part of the input that is chosen in a suitably randomized way, it's very difficult to find another value that hits exactly that target.

Application: Search puzzle. Now, let's consider an application that illustrates the usefulness of this property. In this application, we're going to build a **search puzzle**, a mathematical problem which requires searching a very large space in order to find the solution. In particular, a search puzzle has no shortcuts. That is, there's no way to find a valid solution other than searching that large space.

Search puzzle. A search puzzle consists of

- a hash function, H ,
- a value, id (which we call the **puzzle-ID**), chosen from a high min-entropy distribution
- and a target set Y

A solution to this puzzle is a value, x , such that

$$H(id \parallel x) \in Y.$$

The intuition is this: if H has an n -bit output, then it can take any of 2^n values. Solving the puzzle requires finding an input so that the output falls within the set Y , which is typically much smaller than the set of all outputs. The size of Y determines how hard the puzzle is. If Y is the set of all n -bit strings the puzzle is trivial, whereas if Y has only 1 element the puzzle is maximally hard. The fact that the puzzle id has high min-entropy ensures that there are no shortcuts. On the contrary, if a particular value of the ID were likely, then someone could cheat, say by pre-computing a solution to the puzzle with that ID .

If a search puzzle is puzzle-friendly, this implies that there's no solving strategy for this puzzle which is much better than just trying random values of x . And so, if we want to pose a puzzle that's difficult to solve, we can do it this way as long as we can generate puzzle-IDs in a suitably random way. We're going to use this idea later when we talk about Bitcoin mining, which is a sort of computational puzzle.

SHA-256. We've discussed three properties of hash functions, and one application of each of those. Now let's discuss a particular hash function that we're going to use a lot in this book. There are lots of hash functions in existence, but this is the one Bitcoin uses primarily, and it's a pretty good one to use. It's called **SHA-256**.

Recall that we require that our hash functions work on inputs of arbitrary length. Luckily, as long as we can build a hash function that works on fixed-length inputs, there's a generic method to convert it into a hash function that works on arbitrary-length inputs. It's called the **Merkle-Damgard transform**. SHA-256 is one of a number of commonly used hash functions that make use of this method. In common terminology, the underlying fixed-length collision-resistant hash function is called the **compression function**. It has been proven that if the underlying compression function is collision resistant, then the overall hash function is collision resistant as well.

In a Bitcoin transaction, the concept of transferring ownership of Bitcoin from one user to another is built on cryptographic principles. Let's walk through the process step by step:

1. The Participants

- **Sender (Alice):** The person who wants to send Bitcoin.
- **Receiver (Bob):** The person who will receive the Bitcoin.

Both Alice and Bob have Bitcoin wallets containing private and public keys. The public key is like an account number, while the private key is like a password used to authorize transactions.

2. Creating the Transaction

When Alice wants to send Bitcoin to Bob, she creates a transaction that includes:

- **Sender's Public Key (Alice's address):** This identifies who is sending the Bitcoin.
- **Receiver's Public Key (Bob's address):** This identifies who is receiving the Bitcoin.
- **Amount of Bitcoin:** How much Bitcoin Alice wants to send to Bob.
- **Transaction Fee:** An additional fee Alice may add to incentivize miners to process her transaction faster.

The transaction also references inputs and outputs:

- **Inputs:** These are the previous Bitcoin transactions Alice has received, and she is now spending. Essentially, Alice is using "previous" Bitcoin as funds for the new transaction.
 - **Outputs:** These specify where the Bitcoin should go. In this case, the output will direct the Bitcoin to Bob's address. If Alice sends less than what she has, the change is sent back to her as a new output to her own address.
-

3. Signing the Transaction

- Alice then signs the transaction with her private key, which proves that she has control over the Bitcoin she is sending. This is a cryptographic signature that guarantees authenticity and prevents anyone else from altering the transaction.
-

4. Broadcasting the Transaction

Once signed, Alice's transaction is broadcast to the Bitcoin network, where it is picked up by miners. Miners are participants in the Bitcoin network who verify and record transactions in the blockchain.

5. Verification and Mining

Miners take the transaction and validate it by checking that:

- Alice has enough Bitcoin to make the transaction (i.e., the inputs exist and are valid).
- The signature is correct and matches Alice's public key.
- The transaction adheres to the rules of the network.

Once verified, miners include Alice's transaction in a block and compete to solve a complex mathematical puzzle (this is the Proof of Work mechanism). The first miner to solve the puzzle gets to add the block to the blockchain and is rewarded with newly minted Bitcoin (called the block reward) and transaction fees.

6. Confirming the Transaction

Once the block containing Alice's transaction is added to the blockchain, the transaction is considered confirmed. The more blocks that are added after it, the more confirmed the transaction becomes, making it increasingly difficult to reverse.

7. Finality

When the transaction has enough confirmations (usually six for Bitcoin), it is

	<p>deemed final, and Bob now has ownership of the Bitcoin that Alice sent.</p>
5C	<p>Bitcoin uses a Forth-like, stack-based, and non-Turing complete language to process and validate transactions. The script is executed by nodes to determine if a transaction is valid — especially when unlocking funds (i.e., spending from an output).</p> <p>There are two parts to the script:</p> <ol style="list-style-type: none"> 1. ScriptPubKey (Locking script) – in the UTXO (output of the previous transaction) 2. ScriptSig (Unlocking script) – in the input of the new transaction <p>Together, they are executed to validate spending.</p> <p>When a node or wallet processes a Bitcoin transaction:</p> <ol style="list-style-type: none"> 1. Push ScriptSig onto the stack 2. Push ScriptPubKey onto the stack 3. Execute the combined script 4. If the stack ends with true, the transaction is valid <p>OP_DUP OP_HASH160 <PubKeyHash> OP_EQUALVERIFY OP_CHECKSIG</p> <p><Signature> <PublicKey></p> <p><Signature> <PublicKey> OP_DUP OP_HASH160 <PubKeyHash> OP_EQUALVERIFY OP_CHECKSIG</p> <p>The Bitcoin node executes these instructions one-by-one using a stack:</p> <ol style="list-style-type: none"> 1. Push <Signature> 2. Push <PublicKey> 3. OP_DUP duplicates the public key 4. OP_HASH160 hashes the duplicated public key

	<ol style="list-style-type: none"> 5. Push <PubKeyHash> (from the locking script) 6. OP_EQUALVERIFY checks the hash matches (and removes both if true) 7. OP_CHECKSIG verifies the signature with the public key
5C	<p>Bitcoin Script is a simple, stack-based programming language used to define conditions under which a Bitcoin transaction output can be spent. Each transaction includes two scripts: the ScriptSig (unlocking script) provided by the sender, and the ScriptPubKey (locking script) embedded in the UTXO from the previous transaction. During validation, Bitcoin nodes concatenate the ScriptSig and ScriptPubKey and execute them sequentially using a stack. The language follows a last-in-first-out (LIFO) stack model, where data and operations (called opcodes) are pushed onto or popped from the stack. Common operations include stack manipulation (OP_DUP, OP_DROP), cryptographic functions (OP_HASH160, OP_CHECKSIG), logical checks (OP_EQUAL, OP_VERIFY), and simple control flow (OP_IF, OP_ELSE). For a transaction to be valid, the final result of script execution must leave true at the top of the stack. Bitcoin Script is intentionally not Turing-complete to ensure security and predictability—there are no loops or complex control structures. This scripting system enables functionalities like standard payments, multi-signature wallets, and time-locked transactions, forming the backbone of Bitcoin's decentralized transaction validation.</p>
6A	<p>HOT AND COLD STORAGE</p>

As we just saw, storing bitcoins on your computer is like carrying money around in your wallet or your purse. This is called “hot storage”. It’s convenient but also somewhat risky. On the other hand, “cold

103

storage” is offline. It’s locked away somewhere. It’s not connected to the internet, and it’s archival. So it’s safer and more secure, but of course, not as convenient. This is similar to how you carry some money around on your person, but put your life’s savings somewhere safer.

To have separate hot and cold storage, obviously you need to have separate secret keys for each — otherwise the coins in cold storage would be vulnerable if the hot storage is compromised. You’ll want to move coins back and forth between the hot side and the cold side, so each side will need to know the other’s addresses, or public keys.

Cold storage is not online, and so the hot storage and the cold storage won’t be able to connect to each other across any network. But the good news is that cold storage doesn’t have to be online to receive coins — since the hot storage knows the cold storage addresses, it can send coins to cold storage at any time. At any time if the amount of money in your hot wallet becomes uncomfortably large, you can transfer a chunk of it over to cold storage, without putting your cold storage at risk by connecting to the network. Next time the cold storage connects it will be able to receive from the block chain information about those transfers to it and then the cold storage will be able to do what it wants with those coins.

But there’s a little problem when it comes to managing cold storage addresses. On the one hand, as we saw earlier, for privacy and other reasons we want to be able to receive each coin at a separate address with different secret keys. So whenever we transfer a coin from the hot side to the cold side

Hierarchical wallets. A more effective solution is to use a hierarchical wallet. It allows the cold side to use an essentially unbounded number of addresses and the hot side to know about these addresses, but with only a short, one-time communication between the two sides. But it requires a little bit of cryptographic trickery.

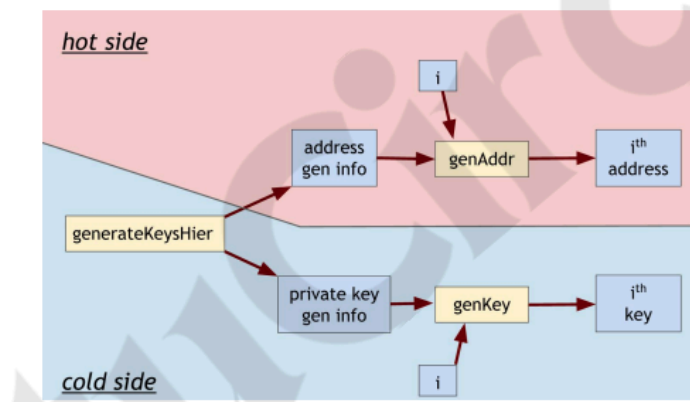
To review, previously when we talked about key generation and digital signatures back in chapter 1, we looked at a function called `generateKeys` that generates a public key (which acts as an address) and a secret key. In a hierarchical wallet, key generation works differently. Instead of generating a single address we generate what we'll call address generation info, and rather than a private key we generate what we'll call private key generation info. Given the address generation info, we can generate a sequence of addresses: we apply an address generation function that takes as input the address generation info and any integer i and generates the i 'th address in the sequence. Similarly we can generate a sequence of private keys using the private key generation info.

104

The cryptographic magic that makes this useful is that for every i , the i 'th address and i 'th secret key "match up" — that is, the i 'th secret key controls, and can be used to spend, bitcoins from the i 'th address just as if the pair were generated the old fashioned way. So it's as if we have a sequence of regular key pairs.

The other important cryptographic property here is security: the address generation info doesn't leak any information about the private keys. That means that it's safe to give the address generation info to anybody, and so that anybody can be enabled to generate the i 'th key.

Now, not all digital signature schemes that exist can be modified to support hierarchical key generation. Some can and some can't, but the good news is that the digital signature scheme used by Bitcoin, ECDSA, does support hierarchical key generation, allowing this trick. That is, the cold side generates arbitrarily many keys and the hot side generates the corresponding addresses.



6B

4.5 Payment Services

So far we've talked about how you can store and manage your bitcoins. Now let's consider how a merchant — whether an online merchant or a local retail merchant — can accept payments in bitcoins in a practical way. Merchants generally support Bitcoin payments because their customers want to be able to pay with bitcoins. The merchant may not want to hold on to bitcoins, but simply receive dollars or whatever is the local fiat currency at the end of the day. They want an easy way to do this without worrying too much about technology, changing their website or building some type of point of sale technology.



The merchant also wants low risk. There are various possible risks: using new technology may cause their website to go down, costing them money. There's the security risk of handling bitcoins — someone might break into their hot wallet or some employee will make off with their bitcoins. Finally there's the exchange rate risk: the value of bitcoins in dollars might fluctuate from time to time. The merchant who might want to sell a pizza for twelve dollars wants to know that they're going to get twelve dollars or something close to it, and that the value of the bitcoins that they receive in exchange for that pizza won't drop drastically before they can exchange those bitcoins for dollars.



Payment services exist to allow both the customer and the merchant to get what they want, bridging the gap between these different desires.

Choose A Way To Accept Bitcoin or [see examples](#) of each payment method.

Type ☒ Button ☐ Hosted Page ☐ iFrame ☐ Email invoice

Payment ☒ Buy now ☐ Donation ☐ Subscription

Button Style ☒ Pay with Bitcoin  ☐ Pay with Bitcoin 

☐  Pay With Bitcoin ☐  Pay With Bitcoin

Item Name Amount

Item Description

Send Funds To

[Show Advanced Options](#)

[Generate Button Code](#)

Figure 4.7: Example payment service interface for generating a pay-with-Bitcoin button. A merchant can use this interface to generate a HTML snippet to embed on their website.

The process of receiving Bitcoin payments through a payment service might look like this to the merchant:

1. The merchant goes to payment service website and fills out a form describing the item, price, and presentation of the payment widget, and so on. Figure 4.7 shows an illustrative example of a form from Coinbase.
2. The payment service generates HTML code that the merchant can drop into their website.
3. When the customer clicks the payment button, various things happen in the background and eventually the merchant gets a confirmation saying, "a payment was made by customer ID [customer-id] for item [item-id] in amount [value]."

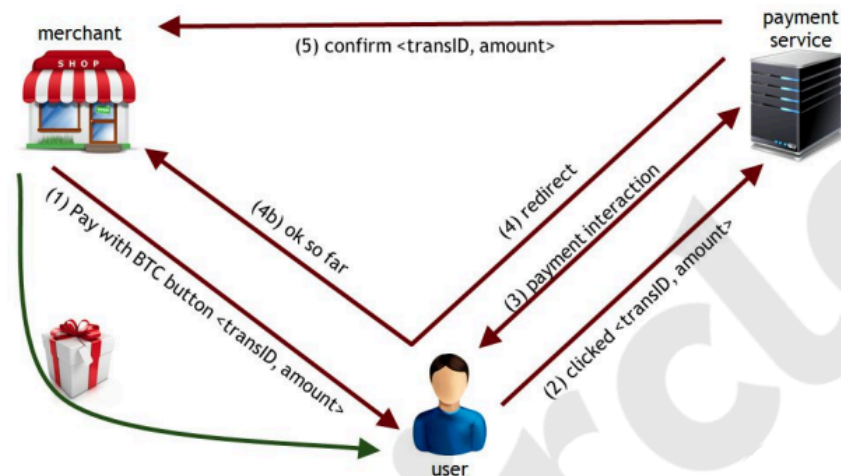


Figure 4.8: Payment process involving a user, merchant, and payment service.

Now let's look at the payment process in more detail to see what happens when the customer makes a purchase with Bitcoin. The steps below are illustrated in Figure 4.8.


1. The user picks out an item to buy on the merchant website, and when it comes time to pay, the merchant will deliver a webpage which will contain the Pay with Bitcoin button, which is the HTML snippet provided by the payment service. The page will also contain a transaction ID — which is an identifier that's meaningful to the merchant and allows them to locate a record in their own accounting system — along with an amount the merchant wants to be paid.
2. If the user wants to pay with bitcoins, they will click that button. That will trigger an HTTPS request to the payment service saying that the button was clicked, and passing on the identity of the merchant, the merchant's transaction ID, and the amount.
3. Now the payment service knows that this customer — whoever they are — wants to pay a certain amount of bitcoins, and so the payment service will pop up some kind of a box, or initiate some kind of an interaction with the user. This gives the user information about how to pay, and the user will then initiate a bitcoin transfer to the payment service through their preferred wallet.
4. Once the user has created the payment, the payment service will redirect the browser to the merchant, passing on the message from the payment service that it looks okay so far. This might mean, for example, that the payment service has observed the transaction broadcast to the peer-to-peer network, but the transaction hasn't received enough (or any) confirmations so far. This completes the payment as far as the user is concerned, with the merchant's shipment of goods pending a final confirmation from the payment service.

6C

Bitcoin miners are specialized participants in the Bitcoin network who perform the critical job of validating transactions and securing the blockchain. Their primary role is to group valid, unconfirmed transactions into blocks and then add those blocks to the Bitcoin blockchain. This process is known as mining.

To add a block, miners must solve a complex mathematical problem called a proof-of-work. This involves finding a special number called a nonce such that when it's combined with the block's data and passed through the SHA-256 hashing algorithm, it produces a hash that starts with a certain number of zeros (determined by the current network difficulty). Since hashing is unpredictable, miners must try many different nonces—often trillions—before finding a valid one. This process requires significant computational power.

Once a valid hash is found, the miner broadcasts the new block to the network. If

	<p>other nodes verify it as valid, the block is added to the blockchain, and the miner is rewarded. The reward includes two parts: the block subsidy (a fixed number of new bitcoins created, which halves approximately every four years) and the transaction fees from all transactions included in the block.</p> <p>Miners play a crucial role in keeping Bitcoin decentralized and secure. By investing energy and hardware to solve proof-of-work puzzles, they make it costly to tamper with the blockchain, ensuring that past transactions remain immutable and trustworthy.</p>
7a	<p>Bitcoin miners are specialized participants in the Bitcoin network who perform the critical job of validating transactions and securing the blockchain. Their primary role is to group valid, unconfirmed transactions into blocks and then add those blocks to the Bitcoin blockchain. This process is known as mining.</p> <p>To add a block, miners must solve a complex mathematical problem called a proof-of-work. This involves finding a special number called a nonce such that when it's combined with the block's data and passed through the SHA-256 hashing algorithm, it produces a hash that starts with a certain number of zeros (determined by the current network difficulty). Since hashing is unpredictable, miners must try many different nonces—often trillions—before finding a valid one. This process requires significant computational power.</p> <p>Once a valid hash is found, the miner broadcasts the new block to the network. If other nodes verify it as valid, the block is added to the blockchain, and the miner is rewarded. The reward includes two parts: the block subsidy (a fixed number of new bitcoins created, which halves approximately every four years) and the transaction fees from all transactions included in the block.</p> <p>Miners play a crucial role in keeping Bitcoin decentralized and secure. By investing energy and hardware to solve proof-of-work puzzles, they make it costly to tamper with the blockchain, ensuring that past transactions remain immutable and trustworthy.</p>
7b	<p>Bitcoin Mining Incentives and Strategy – Explained</p> <p> Mining Incentives</p> <p>Bitcoin miners are economically motivated to secure the network by receiving rewards for their work. These incentives come from two main sources:</p> <ol style="list-style-type: none"> 1. Block Reward (Subsidy): When a miner successfully adds a block to the blockchain, they receive a

fixed amount of newly created bitcoins. This is called the coinbase reward.

- **Initially, it was 50 BTC per block, but it halves every 210,000 blocks (~every 4 years).**
- **As of the 2024 halving, the reward is 3.125 BTC per block.**

2. Transaction Fees:

Every transaction in a block includes a small fee, paid by users to incentivize miners to include their transactions. These fees become more important over time as block rewards shrink.

Together, these form the total mining reward:

Total Reward = Block Subsidy + Sum of Transaction Fees

To maximize profit and stay competitive, miners adopt a variety of strategies:

1. Pool Mining:

Individual miners often join mining pools to combine computational power. Rewards are split among members based on how much work each contributed. This ensures more frequent, smaller payouts rather than waiting for a solo block win.

2. Choosing Transactions (Fee Optimization):

Miners prioritize transactions offering higher fees per byte. They select which transactions to include in a block to maximize total transaction fees.

3. Hardware Optimization:

Miners use ASICs (Application-Specific Integrated Circuits) designed specifically for Bitcoin's SHA-256 algorithm. These are much faster and more energy-efficient than general-purpose CPUs or GPUs.

4. Energy Strategy:



Since mining consumes a lot of electricity, miners seek cheap, renewable, or excess energy sources to reduce operational costs — like hydroelectric, geothermal, or solar energy.

5. Block Timing & Orphan Risk:

Miners aim to find blocks quickly, but sometimes two miners find blocks at nearly the same time. One will eventually be rejected (orphaned). To reduce this risk, miners prefer to mine on the longest valid chain and propagate blocks quickly through the network.

6. Geographical Considerations:

Location matters for access to cheap electricity and stable internet. Many mining farms are located in regions with low power costs and cool climates

	(to reduce cooling expenses).		
8A	<p>Bitcoin mixing, also known as coin mixing or coin tumbling, is a technique used to improve transaction privacy by obscuring the traceability of coins on the public blockchain. While Bitcoin itself is pseudonymous (addresses aren't tied to identities), all transactions are public, which means anyone can analyze them to try and trace ownership. Mixing helps prevent that.</p> <p>Decentralized Bitcoin mixing is a privacy-enhancing process where multiple users cooperate to break the link between input and output addresses — but without relying on a centralized party. Instead of trusting a single entity (which could steal funds or leak data), decentralized mixing uses cryptographic protocols and coordinated transactions that make it impossible to link who sent what to whom.</p> <p> How It Works (Simplified)</p> <ol style="list-style-type: none">1. Multiple participants agree to mix coins of the same amount (e.g., 0.1 BTC).2. Each participant provides a fresh output address (never used before).3. A coordinated transaction is constructed that:<ul style="list-style-type: none">○ Has multiple equal-sized inputs (from different users)○ Has matching number of outputs (to fresh addresses)4. The transaction is signed by all participants and broadcasted to the Bitcoin network.5. Once confirmed, no one can tell which input corresponds to which output. <hr/> <p> Key Techniques in Decentralized Mixing</p> <table><tr><th>Technique</th><th>Description</th></tr></table>	Technique	Description
Technique	Description		

CoinJoin

A privacy method where multiple users combine their inputs and outputs into a single Bitcoin transaction. Popular and efficient.

JoinMarket

A decentralized CoinJoin implementation where makers and takers participate in a market-like system to coordinate mixes.

Wasabi Wallet

A privacy-focused Bitcoin wallet using CoinJoin with enhanced privacy features and Tor integration.

Samourai Wallet & Whirlpool



Another decentralized mixing tool using CoinJoin with a strong focus on anonymity and post-mix coin management.

**Benefits of Decentralized Mixing**

- **Enhanced Privacy:** Breaks on-chain links between sender and receiver.
 - **No Central Trust Required:** Unlike centralized mixers, no one entity can steal funds or log data.
 - **Resistance to Censorship:** Harder for governments or attackers to shut down decentralized protocols.
-

**Risks and Considerations**

- **Legal Grey Areas:** Some jurisdictions may treat mixing services suspiciously (associated with money laundering).
- **Timing Attacks:** If users don't use fresh addresses or coordinate well, mixing might be deanonymized.
- **Fee Overhead:** Mixing involves transaction fees and sometimes coordination fees.

8B	<p>While Bitcoin is often thought to be anonymous, in reality, it is only pseudonymous. This means that users are identified by alphanumeric public addresses rather than real-world identities. However, every transaction is permanently stored on the public blockchain, making it possible to analyze and trace the flow of funds. Over time, with the right tools and data, Bitcoin activity can be de-anonymized — that is, linked to real identities.</p> <hr/>
	<p> How De-anonymization Happens</p> <p>Here are the main methods used to de-anonymize Bitcoin transactions:</p> <hr/>
	<p>1.  Blockchain Analysis</p> <p>Specialized firms and governments use blockchain forensics to analyze patterns in the public ledger. This includes:</p> <ul style="list-style-type: none">● Transaction Graph Analysis: Mapping how coins move between addresses over time.● Address Clustering: Identifying which addresses belong to the same user using clues like:<ul style="list-style-type: none">○ Multi-input transactions (if multiple inputs are used, they're likely owned by the same person).○ Change address detection (in typical Bitcoin use, one output is the payment, and one is the "change" sent back to the sender).● Heuristics: Patterns like round amounts, timing, and reused addresses help cluster identities.

Example: If Address A often transacts with Addresses B, C, and D, and Address D is linked to an exchange account with KYC, analysts can infer that all addresses might belong to the same individual.

2. 🔍 KYC/AML Data from Exchanges



Most modern exchanges (like Coinbase, Binance) require Know Your Customer (KYC) and Anti-Money Laundering (AML) compliance. This means they collect:



- **Real names**
- **Government-issued IDs**
- **IP addresses**
- **Bank account info**



If a Bitcoin address interacts with a KYC exchange, that address can now be linked to a real-world identity. Once that happens, all associated transaction history becomes visible.

3. 🕵️ Network-Level Surveillance

- **Entities can monitor network traffic when transactions are broadcast.**
 - **By analyzing the origin IP address of a transaction, they can sometimes associate it with a user (especially if not using Tor or VPN).**
 - **Tools like Bitcoin Node Network Crawlers watch for transaction propagation to identify the source.**
-

	<p>4.  Behavioral Patterns and Metadata</p> <p>Even without KYC, patterns in how and when someone uses Bitcoin can reveal identity:</p> <ul style="list-style-type: none">● Timing correlations (e.g., transactions happen during work hours in a certain time zone)● Amount patterns (e.g., always sending round figures)● Social and online activity (e.g., sharing addresses on forums or donation pages)
8C	<p>Zerocoin is a cryptographic protocol designed to provide strong anonymity and privacy in cryptocurrency transactions. It was proposed in 2013 as an extension to Bitcoin by researchers at Johns Hopkins University to address Bitcoin's lack of transaction privacy.</p> <p>In Bitcoin, all transactions are publicly visible and traceable, but Zerocoin introduces a system where coins can be "minted" and "spent" anonymously, breaking the link between sender and receiver.</p> <p> Key Features:</p> <ul style="list-style-type: none">● Minting: Users convert regular coins into Zerocoins, which are recorded on the blockchain but do not reveal user identity.● Spending: Zerocoins can later be spent without linking them to the original minting transaction, using zero-knowledge proofs.● Anonymity Set: The more people use Zerocoin, the greater the anonymity, since your transaction is hidden among many others. <p>Zerocoin was implemented in cryptocurrencies like Zcoin (now Firo), but it had limitations like large proof sizes, slow performance, and some vulnerabilities. It was eventually succeeded by more advanced protocols like Zerocash and zk-SNARKs, which offer better efficiency</p>

	<p>and privacy.</p> <p>In summary, Zerocoin was a major step forward in the evolution of privacy-centric cryptocurrencies, offering true anonymity through advanced cryptographic techniques.</p>
9A	<p>A smart contract is a self-executing digital contract with the terms and rules directly written into code. It automatically enforces, verifies, and executes agreements between parties without the need for intermediaries. Smart contracts run on blockchain networks (like Ethereum or Solana), making them immutable, transparent, and trustless.</p> <p>Key properties:</p> <ul style="list-style-type: none"> • Automation: Executes when predefined conditions are met. • Security: Once deployed, it cannot be changed (unless designed to be upgradeable). • Trustless: No central authority needed—code is the law. • Transparency: Anyone can view the contract logic and activity on the blockchain. <hr/> <p> Radiant Contract (Assuming you meant “Radiant”)</p> <p>If you're referring to Radiant Capital (a DeFi protocol) or a "Radiant Contract" used within that ecosystem, here's an explanation based on that context:</p> <p> Radiant Capital and Its Smart Contract System</p> <p>Radiant is a DeFi lending protocol built on LayerZero and Arbitrum, enabling users to lend and borrow assets across chains using omnichain smart contracts. The Radiant smart contracts manage operations like:</p>

	<ul style="list-style-type: none"> ● Depositing collateral ● Earning interest ● Borrowing against collateral ● Liquidations if collateral value falls <p>These contracts are programmed to automatically enforce loan terms, calculate interest, and trigger liquidation without needing a bank or authority.</p> <p> Key Functions of a Radiant Smart Contract:</p> <ul style="list-style-type: none"> ● Cross-chain lending/borrowing using LayerZero (Omnichain Interoperability Protocol) ● Yield optimization via automated strategies ● Risk management with real-time oracle pricing and liquidation bots <p>If you're referring to a different term like “Radium contract” or a specific contract name, let me know and I’ll clarify accordingly.</p>
9B	<p>he Ethereum blockchain is a decentralized, open-source platform that allows developers to build and deploy smart contracts and decentralized applications (DApps). It was proposed by Vitalik Buterin in 2013 and went live in 2015. Ethereum is not just a cryptocurrency (Ether, ETH), but a programmable blockchain that allows users to interact with code directly, making it a foundational part of Decentralized Finance (DeFi), Non-Fungible Tokens (NFTs), and Smart Contracts.</p> <hr/> <p> Key Features of Ethereum Blockchain:</p>

- 1. Smart Contracts:** Ethereum allows the creation of smart contracts, which are self-executing contracts with predefined rules directly written in code. These contracts are automatically executed when conditions are met, without the need for intermediaries. Ethereum's ability to execute smart contracts gives it significant advantages over Bitcoin, which only processes transactions.
- 2. Decentralization:** Like Bitcoin, Ethereum operates on a decentralized network of nodes (computers that validate and store the blockchain). This ensures that no single party controls the Ethereum network, making it resistant to censorship and fraud.
- 3. Ether (ETH):** Ether (ETH) is the native cryptocurrency of the Ethereum blockchain. It is used to pay for transaction fees (known as gas), incentivize miners (or validators in Ethereum 2.0), and for other network-related functions.
- 4. Gas Fees:** Every operation on Ethereum, such as sending transactions or interacting with smart contracts, requires a gas fee. Gas is a measure of computational work needed for an operation. Gas fees are paid in Ether (ETH) and vary depending on the complexity of the operation and network congestion.
- 5. Ethereum Virtual Machine (EVM):** The EVM is the runtime environment for smart contracts in Ethereum. It is responsible for executing all transactions and smart contracts on the network. The EVM makes Ethereum Turing-complete, meaning it can execute any computation, making it capable of handling complex decentralized applications.



How Ethereum Blockchain Works

- 1. Transactions and Smart Contract Execution:**
 - A user submits a transaction or a contract execution request (e.g., transferring ETH or interacting with a

DApp).

- **The transaction is broadcast to the network.**
- **Miners (or validators in Ethereum 2.0) verify the transaction and execute the contract code (if it's a smart contract).**
- **Once validated, the transaction is added to a new block, which is then appended to the blockchain.**

2. Proof-of-Work (PoW) vs Proof-of-Stake (PoS):

- **Ethereum 1.0 uses Proof-of-Work (PoW), where miners compete to solve complex mathematical puzzles to validate transactions and add new blocks.**
- **Ethereum 2.0 is transitioning to Proof-of-Stake (PoS), where validators are selected to create new blocks based on the amount of ETH they hold and are willing to "stake" as collateral.**

Use Cases of Ethereum Blockchain:

- 1. Decentralized Finance (DeFi):** Ethereum supports a wide variety of DeFi platforms, where users can lend, borrow, trade, and earn interest on assets without relying on traditional financial institutions.
- 2. NFTs (Non-Fungible Tokens):** Ethereum is the primary blockchain for creating and trading NFTs, digital assets that represent ownership or proof of authenticity for unique items like art, collectibles, and real estate.
- 3. DAOs (Decentralized Autonomous Organizations):** Ethereum allows the creation of DAOs, which are organizations that operate through smart contracts, with decision-making processes based on the consensus of their members.


	<p>4. Supply Chain and Tokenization: Ethereum is used for tokenizing real-world assets and creating transparent supply chain solutions, allowing stakeholders to track products from origin to final sale.</p> <hr/>
9C	<p>In Ethereum, there are two primary types of accounts that interact with the network:</p> <p>1. Externally Owned Accounts (EOAs)</p> <ul style="list-style-type: none"> ● Definition: Externally Owned Accounts (EOAs) are controlled by private keys, which are owned by individuals or entities. These accounts do not contain any code but hold and manage Ether (ETH) and can send transactions to other accounts or interact with smart contracts. ● Features: <ul style="list-style-type: none"> ○ Controlled by a private key (the owner has full control). ○ Can send transactions (ETH transfers or smart contract interactions). ○ Cannot execute any code themselves. ○ Have an associated nonce, which is a counter used to prevent double-spending (each transaction from an EOA increments the nonce by 1). ● Use Cases: <ul style="list-style-type: none"> ○ Personal wallets: Used by individuals to store ETH or interact with DApps. ○ DApp users: Interact with decentralized applications by sending transactions and invoking smart contract

functions.

- **Example:** If you hold an Ethereum wallet (like MetaMask or a hardware wallet), you're using an EOA.
-

2. Contract Accounts (Smart Contracts)

- **Definition:** Contract Accounts are controlled by the code of smart contracts deployed on the Ethereum blockchain. These accounts are not controlled by a private key but are instead governed by the smart contract's code. When a contract account receives a transaction, it executes the code associated with that contract, depending on the function called and the input provided.
- **Features:**
 - **Controlled by smart contract code, not a private key.**
 - **Can hold ETH and interact with other contracts or EOAs.**
 - **Have no private key and can only be interacted with through transactions from EOAs or other contracts.**
 - **Can execute code (smart contract logic) when receiving a transaction.**
 - **Do not have a nonce in the same sense as EOAs, but they have internal state and are executed based on the contract's code.**
- **Use Cases:**
 - **Decentralized Applications (DApps):** Used to build and run applications that do not require intermediaries.
 - **Token Contracts:** ERC-20 and ERC-721 token contracts are examples of contract accounts that manage token

	<p>transfers and ownership.</p> <ul style="list-style-type: none"> ○ DeFi protocols: Smart contracts that enable decentralized finance operations like lending, borrowing, and trading. ● Example: A contract that controls the issuance of a token like Uniswap's or Compound's smart contracts. <hr/>
10A	<p>Precompiled contracts in Ethereum are special contracts that are already deployed on the Ethereum network and can be called directly by other contracts or externally owned accounts (EOAs). These contracts are built into the Ethereum Virtual Machine (EVM) and offer optimized, low-level functions that can be executed with greater efficiency compared to regular smart contract code.</p> <p>Precompiled contracts are essentially native functions provided by the Ethereum protocol to carry out certain operations that would otherwise require more complex code. They are implemented in low-level languages, often written in C or Rust, and their execution is optimized to minimize gas costs and maximize performance.</p> <hr/> <p> Key Features of Precompiled Contracts:</p> <ol style="list-style-type: none"> 1. Efficiency: Since precompiled contracts are written in low-level code and executed directly by the EVM, they are much faster and more gas-efficient than implementing the same functionality using regular smart contract code in Solidity. 2. Gas Cost: Precompiled contracts typically have a much lower gas cost than executing equivalent operations in a smart contract. This makes them ideal for operations that need to be executed frequently or involve heavy computations. 3. Fixed Set of Functions: The set of precompiled contracts in

	<p>Ethereum is fixed, meaning only specific functions are available as precompiled. For instance, cryptographic functions like hashing and signature verification.</p> <p>4. No Source Code: Users or developers cannot modify or access the source code of precompiled contracts. They are hardcoded into the Ethereum protocol.</p>
10B	<p>The Ethereum Virtual Machine (EVM)</p> <p>The EVM is a simple stack-based execution machine that runs bytecode instructions to transform the system state from one state to another. The word size of the EVM is set to 256-bit. The stack size is limited to 1,024 elements and is based on the Last In, First Out (LIFO) queue. The EVM is a Turing-complete machine but is limited by the amount of gas that is required to run any instruction. This means that infinite loops that can result in denial-of-service attacks are not possible due to gas requirements. The EVM also supports exception handling should exceptions occur, such as not having enough gas or providing invalid instructions, in which case the machine would immediately halt and return the error to the executing agent.</p> <p>The EVM is an entirely isolated and sandboxed runtime environment. The code that runs on the EVM does not have access to any external resources such as a network or filesystem. This results in increased security, deterministic execution, and allows untrusted code (code that can be run by anyone) to be executed on Ethereum blockchain.</p> <p>As discussed earlier, the EVM is a stack-based architecture. The EVM is big-endian by design, and it uses 256-bit-wide words. This word size allows for Keccak 256-bit hash and ECC computations.</p>

There are three main types of storage available for contracts and the EVM:

- **Memory:** The first type is called memory or volatile memory, which is a word-addressed byte array. When a contract finishes its code execution, the memory is cleared. It is akin to the concept of RAM. write operations to the memory can be of 8 or 256 bits, whereas read operations are limited to 256-bit words. Memory is unlimited but constrained by gas fee requirements.
- **Storage:** The other type is called storage, which is a key-value store and is permanently persisted on the blockchain. Keys and values are each 256 bits wide. It is allocated to all accounts on the blockchain. As a security measure, storage is only accessible by its own respective CAs. It can be thought of as hard disk storage.
- **Stack:** EVM is a stack-based machine, and thus performs all computations in a data area called the stack. All in-memory values are also stored in the stack. It has a maximum depth of 1024 elements and supports the word size of 256 bits.

The storage associated with the EVM is a word-addressable word array that is non-volatile and is maintained as part of the system state. Keys and values are 32 bytes in size and storage. The program code is stored in **virtual read-only memory (virtual ROM)** that is accessible using the CODECOPY instruction. The CODECOPY instruction copies the program code into the main memory. Initially, all storage and memory are set to zero in the EVM.

The following diagram shows the design of the EVM where the virtual ROM stores the program code that is copied into the main memory using the CODECOPY instruction. The main memory is then read by the EVM by referring to the program counter and executes instructions step by step. The program counter and EVM stack are updated accordingly with each instruction execution:

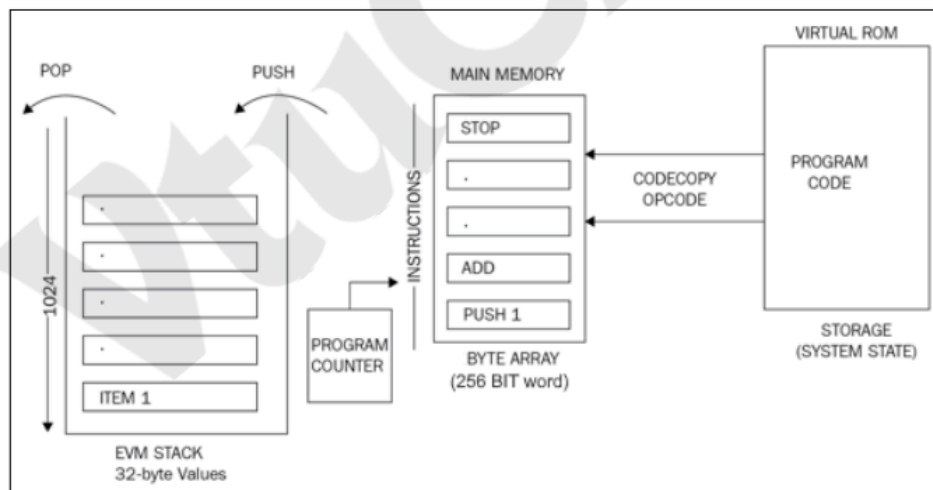


Figure 11.11: EVM operation

10C

n Ethereum, iterator functions are commonly used in smart contracts to loop through data structures such as arrays or mappings. Since Solidity, the language used for Ethereum smart contracts, doesn't have a built-in way to iterate over mappings (because mappings don't maintain an order of keys), iteration is usually implemented through arrays or by maintaining an additional list of keys separately. For example, an iterator function can be used to sum the elements of an array by iterating over it in a loop. This function typically involves a for-loop that starts from the first element and runs through each element until it reaches the end. When working with mappings,

	<p>developers often use an auxiliary array that stores the keys, allowing iteration over the keys to access their corresponding values. However, it's important to note that iterating over large datasets in smart contracts can be gas-expensive, so it's crucial to optimize such operations to avoid high transaction costs.</p>
--	---