

Ethereum

Zichao Yang

Zhongnan University of Economics & Law

Date: October 29, 2025

Part I. Ethereum Introduction

Introduction

Ethereum was conceived in 2013 by **Vitalik Buterin**. Other founders include Gavin Wood, Charles Hoskinson, Anthony Di Iorio, and Joseph Lubin.

The Ethereum network went live on July 30, 2015.

Ethereum is a decentralized blockchain with smart contract functionality.

Unlike Bitcoin, the network is called **Ethereum**, and the token used inside the network is called **Ether**.

Similarities between Bitcoin and Ethereum

- **Blockchain Technology:** Both Bitcoin and Ethereum are based on blockchain technology, which ensures a decentralized ledger system. Transactions are recorded in blocks, allowing secure and transparent record-keeping.
- **Cryptocurrency Tokens:** Bitcoin and Ethereum each have a native cryptocurrency — Bitcoin (BTC) and Ether (ETH), respectively.
- **Decentralization and Peer-to-Peer Networks:** Both systems operate on peer-to-peer networks, meaning transactions occur directly between users without intermediaries.

Major Differences between Bitcoin and Ethereum

- **Consensus Mechanism:** Bitcoin uses **Proof of Work** (PoW). Ethereum has shifted from PoW to **Proof of Stake** (PoS) in Ethereum 2.0.
- **Token Supply:** Bitcoin has a strict supply cap of **21 million bitcoins**. Ethereum has an **elastic supply** policy thanks to the combination of PoS issuance and the **EIP-1559** fee-burning mechanism.
- **Smart Contracts:** Bitcoin programming language is **not Turing-complete**, and can only implement basic smart contracts on it. Ethereum is equipped with **Turing-complete** programming language and makes smart contract functionality a core feature.
- **Blockchain Structure:** Ethereum has a more complex blockchain structure. More on this in the following part.

Ethereum Basics

Ethereum's currency unit is called *ether*, identified also as "ETH".

The smallest unit in Ethereum is called *wei*, $1 \text{ ether} = 10^{18} \text{ wei}$.

Table 2-1. Ether denominations and unit names

Value (in wei)	Exponent	Common name	SI name
1	1	wei	Wei
1,000	10^3	Babbage	Kilowei or femtoether
1,000,000	10^6	Lovelace	Megawei or picoether
1,000,000,000	10^9	Shannon	Gigawei or nanoether
1,000,000,000,000	10^{12}	Szabo	Microether or micro
1,000,000,000,000,000	10^{15}	Finney	Milliether or milli
1,000,000,000,000,000,000	10^{18}	Ether	Ether
1,000,000,000,000,000,000,000	10^{21}	Grand	Kiloether
1,000,000,000,000,000,000,000,000	10^{24}		Megaether

Source: Antonopoulos, Andreas M., and Gavin Wood. *Mastering ethereum: building smart contracts and dapps*. O'reilly Media, 2018.

Ethereum State

In Ethereum, the **state** encompasses all data that is required to describe the current conditions of the blockchain. Key components of Ethereum's state include:

- ① Account Balances
- ② Smart Contract Code and Storage
- ③ Nonces: not the same concept in Bitcoin blockchain!
- ④ Smart Contract Execution Context
- ⑤ Global State Tree
- ⑥ Transaction History

Whenever a transaction occurs, it typically results in a change in the Ethereum state.

Ethereum Virtual Machine (EVM)

The **Ethereum Virtual Machine** (EVM) is the decentralized computing environment that allows developers to execute smart contracts and run decentralized applications (DApps) on the Ethereum blockchain.

Key Features of the EVM:

- ① **Execution of Smart Contracts:** ensures that all nodes in the network agree on the outcome of a contract's execution
- ② **Isolated Execution Environment:** ensures contracts run in a contained environment, and can't directly interact with external systems or databases, making the system more secure and predictable.
- ③ **Turing Completeness:** capable of performing any computation

Ethereum Virtual Machine (EVM)

Key Features of the EVM:

- ④ **State Machine:** processes the state of the Ethereum blockchain by applying a sequence of transitions based on the transactions and smart contracts being executed.
- ⑤ **Decentralization:** EVM is not a single centralized server or system; it is executed by all Ethereum nodes in the network. Every node in the Ethereum network runs the EVM, ensuring that all participants have the same view of the current state of the blockchain.
- ⑥ **Interoperability:** EVM allows for interoperability between different Ethereum-based applications and tokens. For example, a smart contract written for one DApp can call another DApp's contract, and tokens (like ERC-20 tokens) can be exchanged or used within other applications seamlessly.

Ethereum Account

Bitcoin and Ethereum use different models for managing and tracking balances on their respective blockchains, and this distinction is often described as **token-based** for Bitcoin (i.e., UTXO) and **account-based** for Ethereum.

Ethereum has two types of accounts: (1) **Externally Owned Accounts** (EOAs) and (2) **Contract Accounts**.

EOAs represent user wallets, while Contract Accounts represent smart contracts.

EOA VS. Contract Account

A contract account has smart contract code, which a simple EOA can't have.

A contract account does not have a private key, and it is owned by the logic of its smart contract code.

Only EOAs can initiate transactions, but the destination can be either EOAs or contract addresses.

When a transaction destination is a contract address, it causes that contract to run in the EVM, using the transaction, and the transaction's data, as its input.

Contracts can react to transactions by calling other contracts, building complex execution paths.

Account Structure

Field	Description	Applicability
Nonce	Counter for tracking transactions or contracts created.	EOAs and Contract Accounts
Balance	Amount of Ether held by the account, stored in Wei.	EOAs and Contract Accounts
Storage Root (State Root)	Root hash of a trie that stores contract data. Empty for EOAs.	Contract Accounts
Code Hash	Hash of the bytecode for contracts. Points to the code's location; empty for EOAs.	Contract Accounts

Table 1: Ethereum Account Fields

Ethereum Explorer

Etherscan is one of the most widely used and trusted block explorers for Ethereum.

There are other Ethereum explorers, like Ethplorer, Blockscout.

Let's take a look at:

(1) one transaction

0x2e8425ec699d2c3b1dbbadbf8cdb75bc755ca8bac22e0c3400b89490acbd5a71

(2) one account

0xb23360CCDd9Ed1b15D45E5d3824Bb409C8D7c460

Part II. Ethereum Transactions

Ethereum Transactions

Transactions are signed messages originated by an externally owned account, transmitted by the Ethereum network, and recorded on the Ethereum blockchain.

Transactions are stored as serialize binary messages in the blockchain.

Transactions are the only things that can trigger a change of state, or cause a contract to execute in the EVM.

The Structure of a Transaction

A transaction contains the following fields:

- **Nonce:** A sequence number, issued by the originating EOA, used to prevent message replay.
- **Gas price:** The price of gas the originator is willing to pay
- **Gas limit:** (a.k.a. STARTGAS) The maximum amount of gas the originator is willing to buy for this transaction
- **Recipient:** The destination Ethereum address
- **Value:** The amount of ether to send to the destination
- **Data:** The variable-length binary data payload
- **v,r,s:** The three components of an ECDSA digital signature of the originating EOA

The Structure of a Transaction

Q: why is there no “from” data in the address identifying the originator EOA?

Answer: That is because the EOA’s public key can be derived from the `v`, `r`, `s` components of the ECDSA signature. The address can, in turn, be derived from the public key.

When you see a transaction showing a “from” field in an Ethereum explorer, that piece of information was added by the explorer.

Transaction Nonce

The transaction **nonce** is a unique counter assigned to each transaction sent from an **Externally Owned Account** (EOA).

Note: nonces are only used to track transactions from Externally Owned Accounts (EOAs) and are not required for Contract Accounts.

The nonce is a critical field in Ethereum transactions because it ensures transaction order, prevents double-spending, and mitigates replay attacks.

Transaction Nonce

- **Double-Spending Prevention:** each transaction's nonce must be unique. Re-using a nonce would create an invalid transaction, so sending the same transaction twice with the same nonce is not allowed.
- **Replay Attack Mitigation:** a replay attack is when a transaction is maliciously repeated to cause unintended outcomes. Because each transaction has a unique nonce, replaying a transaction with the same nonce now would be invalidated by the network.

Q: why we never discuss the *Replay Attack* when we study Bitcoin?

Transaction Nonce

Q: why we never discuss the *Replay Attack* when we study Bitcoin?

Answer: because Bitcoin is a token-based system, not an account-based system. Replay attack cannot work in a token-based system.

Analogously, you can swipe a stolen credit card multiple times, but you cannot spend a stolen paper bill more than once.

Transaction Order

The Ethereum network processes transactions sequentially, based on the nonce.

If you transmit a transaction with **nonce 0** and then transmit a transaction with **nonce 2**, the second transaction will not be included in any blocks.

The transaction with nonce 2 will be stored in the **mempool**, while the Ethereum network waits for the missing transaction with nonce 1 to appear.

After the transaction with nonce 1 is received, miners will process transaction with nonce 1 and with nonce 2 simultaneously, **higher priority gives to the transaction with nonce 1**.

Transaction Order

Q: what if I accidentally transmitted two transactions with the **same nonce** but different recipients or values? Which transaction will be confirmed?

Answer: One transaction will be confirmed at random. However, if one of these transactions includes a higher transaction fee, it is more likely to be confirmed.

Transaction Order

Q: What if I want to supersede a previous transaction before it gets confirmed, what can I do?

Transaction Order

Q: What if I want to supersede a previous transaction before it gets confirmed, what can I do?

Answer: send out a transaction with the same nonce but with a higher transaction fee. This strategy is called **Replace-By-Fee** (RBF) mechanism.

Transaction Gas

We have casually mentioned the term “transaction fee”. Now let us discuss the two components that decide the transaction fee inside Ethereum: `gasPrice` and `gasLimit`.

Gas is the fuel of Ethereum. Ethereum uses gas to control the amount of resources that a transaction can use to avoid denial-of-service attacks in a Turing-complete system.

Gas is not ether—it’s a separate virtual currency with its own exchange rate against ether. This design aims to protect the system from the volatility that might arise along with rapid changes in the value of ether.

Transaction Gas

The **gasPrice** field in a transaction allows the transaction originator to set the price they are willing to pay in exchange for gas. The price is measured in *wei* per gas unit.

The higher the **gasPrice**, the faster the transaction is likely to be confirmed.

The **gasLimit** gives the maximum number of units of gas the transaction originator is willing to buy in order to complete the transaction.

In the EVM, each **OPcode** has a fixed gas cost specified by Ethereum's protocol. For example, transfer ether from one EOA to another EOA, the gas amount needed is fixed at 21,000 gas units.

Transaction Gas

If your transaction's destination address is a contract, then the amount of gas needed can be estimated but cannot be determined with accuracy.

A contract can evaluate different conditions that lead to different execution paths, with different total gas costs.

If the code execution ran out of gas, EVM reverts all state changes except the transaction fee (`gasPrice * gasLimit`), which is collected by the miner.

Q: Do you remember how transaction fee is calculated in Bitcoin?
Another difference between these two chains.

Transaction Gas

Q: why do we need two variables: `gasPrice` and `gasLimit`? Can't we indicate how much we are willing to pay using one variable?

Transaction Gas

Q: why do we need two variables: `gasPrice` and `gasLimit`? Can't we indicate how much we are willing to pay using one variable?

Answer: these two variables control separate aspects of transaction execution:

- **Transaction Complexity (`gasLimit`)**: Sets a ceiling on the transaction's complexity or resource demand.
- **Fee per Unit of Complexity (`gasPrice`)**: Determines how much the user is willing to pay for each computational unit, affecting prioritization by miners.

A user may want to set a high gas limit for a complex transaction but specify a low gas price to avoid high fees, or vice versa. Separating these controls allows users to balance transaction cost with the desired execution priority and complexity.

Transaction Value and Data

The main payload* of a transaction is contained in two fields: **value** and **data**. Transactions can have both value and data, only value, only data, or neither value nor data.

A transaction with only value is a **payment**.

A transaction with only data is an **invocation**.

A transaction with both value and data is both a payment and an invocation.

A transaction with neither value nor data is... a waste of gas! But it is still possible!

***payload**: (computing term) the actual information or message in transmitted data, as opposed to automatically generated metadata.

Transaction Value and Data

(1) A transaction with value, and the destination addresses are:

- **EOA addresses:** Ethereum will record a state change, adding the value you sent to the balance of the address.
- **Contract address:** EVM will execute the contract and will attempt to call the function named in the data payload of your transaction.
 - If no data in your transaction: EVM will call a *fallback* function to determine what to do next.
 - If the fallback function does not exist: this transaction will essentially become a payment to the wallet.
 - A contract can also reject incoming payments by throwing an exception immediately when a function is called.

Transaction Value and Data

(2) A transaction with data, and the destination addresses are:

- **Contract address:** the data will be interpreted by the EVM as a **contract invocation**. Most contracts use this data to call the named function and pass any encoded arguments to the function. The data payload consist of:
 - **A function selector:** This allows the contract to identify which function you wish to invoke.
 - **The function arguments:** The function's arguments, encoded according to the rules defined in the ABI (Application Binary Interface) specification.
- **EOA addresses:** EOAs will ignore the data.

Special Transaction: Contract Creation

Contract creation transactions are sent to a special destination address called the **zero address**; the **to** field in a contract registration transaction contains the address 0x0.

This address represents neither an EOA nor a contract. It can never spend ether or initiate a transaction. It is only used as a destination, with the special meaning “create this contract.”

Special Transaction: Contract Creation

A contract creation transaction need only contain a **data** payload that contains the compiled bytecode which will create the contract.

The transaction can include an ether amount in the **value** field if you want to set the new contract up with a starting balance, but this is optional.

If you send a **value** to the contract creation address without a **data** payload, then the effect is the same as sending to a burn address, the ether is lost for good.

Transaction Signing in Practice

To sign a transaction in Ethereum, the originator must:

- ➊ Create a transaction data structure, containing nine fields: `nonce`, `gasPrice`, `gasLimit`, `to`, `value`, `data`, `chainID`, 0, 0.
- ➋ Produce an RLP-encoded serialized message of the transaction data structure.
- ➌ Compute the Keccak-256 hash of this serialized message.
- ➍ Compute the ECDSA signature, signing the hash with the originating EOA's private key.
- ➎ Append the ECDSA signature's computed v , r , and s values to the transaction.

Transaction Signing in Practice

Variable v indicates two things: the chain ID and the recovery identifier:

- **recovery identifier:** in previous ECDSA part, we learned that we can calculate two public key candidates based on r , and s , and we choose the right one based on v is odd or even.
- **chain ID:** 27 or 28 means this transaction is for the Ethereum main chain. Other values mean this transaction is for other chains.

Part III. Smart Contracts

What is a Smart Contract?

A **smart contract** in Ethereum is a self-executing program stored on the blockchain. It operates according to predefined rules written in code and is executed by the Ethereum Virtual Machine (EVM).

Smart contracts have these features:

- **Execution on the Blockchain:** When deployed, the smart contract code is stored on the Ethereum blockchain and can be executed by interacting with it via transactions.
- **Immutable:** Once deployed, the code of a smart contract cannot be changed. But the code can be replaced.

What is a Smart Contract?

Smart contracts have these features:

- **Deterministic:** A smart contract produces the same result every time, given the same inputs.
- **Trustless Execution:** Smart contracts are executed automatically by the Ethereum network. No central authority controls them, and their execution is verifiable by anyone.
- **Transparent:** The code of a smart contract is publicly visible on the blockchain, ensuring transparency and allowing anyone to audit its logic. (Smart contract auditing companies, like OpenZeppelin)

How Smart Contracts Work?

- ➊ **Creation:** a smart contract is deployed to the Ethereum blockchain via a transaction initiated by an EOA.
- ➋ **Interaction:**
 - Users interact with the contract by sending transactions that call specific functions in the contract.
 - These transactions may include data (e.g., inputs for the function) and Ether (to pay for the gas fees or as part of the contract's logic).
- ➌ **Execution:** The Ethereum network validates and executes the transaction using the EVM, updating the state of the blockchain accordingly.

Smart Contract Limitations

- **Gas Costs:** Every operation in a smart contract requires computational resources, paid for in gas.
- **Immutability:** Bugs in smart contract code cannot be fixed once deployed, unless designed with upgradeability mechanisms.
- **Security Risks:** Poorly written contracts can be exploited, leading to loss of funds (e.g., the DAO hack in 2016).

Build A Smart Contract

Smart contracts support multiple high-level programming languages. In this class, we will use the most popular one, **Solidity**, to code a simple smart contract.

Now, let's code it out!