title: Ethereum Virtual Machine (EVM) description: An introduction to the Ethereum virtual machine and how it relates to state, transactions, and smart contracts. lang: en

The EVM's physical instantiation can't be described in the same way that one might point to a cloud or an ocean wave, but it does *exist* as one single entity maintained by thousands of connected computers running an Ethereum client.

The Ethereum protocol itself exists solely for the purpose of keeping the continuous, uninterrupted, and immutable operation of this special state machine. It's the environment in which all Ethereum accounts and smart contracts live. At any given block in the chain, Ethereum has one and only one 'canonical' state, and the EVM is what defines the rules for computing a new valid state from block to block.

Prerequisites {#prerequisites}

Some basic familiarity with common terminology in computer science such as bytes, memory, and a stack are necessary to understand the EVM. It would also be helpful to be comfortable with cryptography/blockchain concepts like hash functions and the Merkle tree.

From ledger to state machine {#from-ledger-to-state-machine}

The analogy of a 'distributed ledger' is often used to describe blockchains like Bitcoin, which enable a decentralized currency using fundamental tools of cryptography. The ledger maintains a record of activity which must adhere to a set of rules that govern what someone can and cannot do to modify the ledger. For example, a Bitcoin address cannot spend more Bitcoin than it has previously received. These rules underpin all transactions on Bitcoin and many other blockchains.

While Ethereum has its own native cryptocurrency (Ether) that follows almost exactly the same intuitive rules, it also enables a much more powerful function: smart contracts. For this more complex feature, a more sophisticated analogy is required. Instead of a distributed ledger, Ethereum is a distributed state machine. Ethereum's state is a large data structure which holds not only all accounts and balances, but a *machine state*, which can change from block to block according to a pre-defined set of rules, and which can execute arbitrary machine code. The specific rules of changing state from block to block are defined by the EVM.

Diagram adapted from Ethereum EVM illustrated

The Ethereum state transition function {#the-ethereum-state-transition-function}

The EVM behaves as a mathematical function would: Given an input, it produces a deterministic output. It therefore is quite helpful to more formally describe Ethereum as having a **state transition function**:

```
Y(S, T)= S'
```

Given an old valid state `(S)` and a new set of valid transactions `(T)`, the Ethereum state transition function `Y(S, T)` produces a new valid output state `S'`

State {#state}

In the context of Ethereum, the state is an enormous data structure called a modified Merkle Patricia Trie, which keeps all accounts linked by hashes and reducible to a single root hash stored on the blockchain.

Transactions {#transactions}

Transactions are cryptographically signed instructions from accounts. There are two types of transactions: those which result in message calls and those which result in contract creation.

Contract creation results in the creation of a new contract account containing compiled smart contract bytecode. Whenever another account makes a message call to that contract, it executes its bytecode.

EVM instructions {#evm-instructions}

The EVM executes as a stack machine with a depth of 1024 items. Each item is a 256-bit word, which was chosen for the ease of use with 256-bit cryptography (such as Keccak-256 hashes or secp256k1 signatures).

During execution, the EVM maintains a transient *memory* (as a word-addressed byte array), which does not persist between transactions.

Contracts, however, do contain a Merkle Patricia *storage* trie (as a word-addressable word array), associated with the account in question and part of the global state.

Compiled smart contract bytecode executes as a number of EVM opcodes, which perform standard stack operations like `XOR`, `AND`, `ADD`, `SUB`, etc. The EVM also implements a number of blockchain-specific stack operations, such as `ADDRESS`, `BALANCE`, `BLOCKHASH`, etc.

Diagrams adapted from Ethereum EVM illustrated

EVM implementations {#evm-implementations}

All implementations of the EVM must adhere to the specification described in the Ethereum Yellowpaper.

Over Ethereum's nine year history, the EVM has undergone several revisions, and there are several implementations of the EVM in various programming languages.

Ethereum execution clients include an EVM implementation. Additionally, there are multiple standalone implementations, including:

- Py-EVM - *Python*
- evmone - *C++*
- ethereumjs-vm - *JavaScript*
- eEVM - *C++*
- revm - *Rust*

Further Reading {#further-reading}

- Ethereum Yellowpaper
- Jellopaper aka KEVM: Semantics of EVM in K
- The Beigepaper
- Ethereum Virtual Machine Opcodes
- Ethereum Virtual Machine Opcodes Interactive Reference
- A short introduction in Solidity's documentation

Related Topics {#related-topics}

- Gas