Off-Whitepaper

Ethereal Explanations

Micah Dameron

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.

The Zen of Python

Abstract

The goal of this paper[a] is to create and expand concepts from Ethereum about which, notwithstanding any earlier documentation, there may be some justified confusion. We use pseudocode rather than mathematical notation to describe Ethereum's operation, because pseudocode has many advantages when describing Abstract State Machines,[b] like Ethereum. This paper takes an approach to describing Ethereum that focuses on clarity and approachability. Our prime source has been the Ethereum *Yellowpaper*, but much supplemental knowledge has been found elsewhere and crucial points from other sources have been added as well for the reader's benefit.

[a] Formally, *Blanched-Almond* Paper
[b] E. Borger and S. Robert F., *Abstract state machines: A method for high-level system design and analysis.* 1, pp. 3-8. Springer, 2003.

Acknowledgements

Thank you to the Ethereum founders for creating a product worth writing about in minute detail. Thanks to the Ethereum Foundation for maintaining this product in its basic integrity. Thanks to the ConsenSys Mesh for supporting my work on this project, by contributing your vast knowledge and expertise. Finally, thank you to Dr. Gavin Wood for your technical astuteness and creative genius; your Yellowpaper has given Ethereum a soul worth decoding.

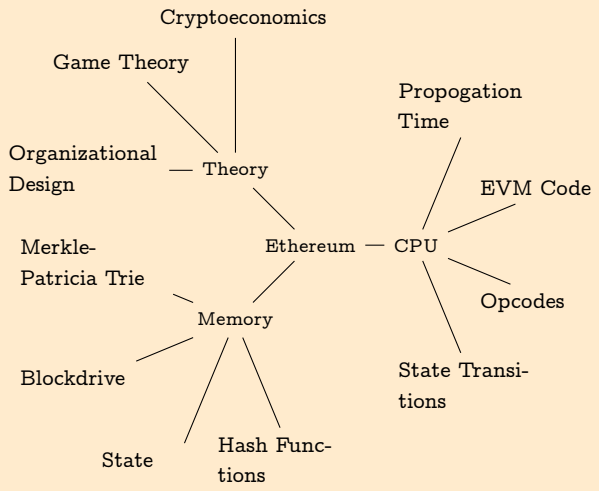
Contents

Part I.
Theory

1 The Evolution of Blockchains

Over the past decade, blockchain technology has proven its longevity and veracity through a number of systems, most notably through Bitcoin, the first electronic currency of its kind to succeed. Bitcoin was successful in its mission to create currency based on a decentralized peer-to-peer Blockchain protocol, and Ethereum takes that concept a step further by creating a *globally-distributed virtual machine*that can ad-hoc run such currency applications, along with any other conceivable applications or programs. Using well-established concepts from the relevant areas of computer science, like MESSAGE-PASSING, TRANSACTION PROCESSING, and SHARED-STATE CONCURRENCE, the *Ethereum Protocol* creates an environment for developers to execute machine instructions with the same level of veracity and certainty as monetary transactions have on more standard Blockchains.

A Conceptual Map of the Ethereum Protocol



2 Cryptography

Cryptographic hashing is what makes Blockchain technology possible. We take for granted that from a certain hash function, (say SHA-256 in the case of Bitcoin) there are a certain, limited number of hashes from the previous block's nonce function as solutions to an equation. Out there, somewhere in the world of numbers, there already exist cryptographic hashes that fit the requirements of the current block's nonce. Since it's impossible for someone to convincingly fake the current block, due to everyone running an identical protocol, bad blocks are spotted. Good solutions to the next true block are attempted instead, since it's easier (though still notably difficult) to mine for the next hash, that is, to search for the next needle-in-a-haystack that solves the equation, than it is to try and fool the peer-to-peer network into agreeing on a false block. A classic game of economics is here at play, and it runs assuming man's inherent self-interest. Not the bad kind of self-interest, but the good kind that is interested in a self's healthy growth and development. If there were not these hashing functions for us to utilize from broader studies in Mathematics, the entire cryptocurrency market and concept would fall like a house of cards. The fact that these networks are still up and running proves that the Math works. The biggest problem for hashing is the emergence of hash collisions, where two inputs produce the same output hash. These sometimes take years to find, and with a good enough hashing function usually are not found until the next more powerful hashing function has arrived on the scene.

3 Serialization

4 Hacker Ethic

5 Message-Passing Interface

6 Singleton Pattern

7 State Machines

8 Processor Technology

9 Turing Machines

9.1 Turing-Completeness

10 Intrinsic Currency

The smallest unit of currency in Ethereum is the Wei, which is $1 * 10^(18)$ Ether. All units at the machine level are counted in Wei.

11 Computation Flow

12 External Actors

A person or other entity able to interface to an Ethereum node, but not a part of Ethereum. It can interact with Ethereum through depositing signed Transactions and inspecting the blockchain and its state. By nature of the fact that external actors need an account in order to interface with the Blockchain, they necessarily (in the context of this paper, at least) have one or more intrinsic accounts.

13 ECDSA

13.1 Public Key Cryptography

Diffie Helman hypothesis–stated in their paper (citation to it)

Public Keys

Private Keys

14 Programming Languages

14.1 Lower-Level Lisp

Notation : LLL

Description : The Lisp-Like low level language: a human-writable language used for authoring simple contracts and general low-level language for trans-compiling to. and this is what turns your LLL code into EVM code and how it's executed in the evm

14.2 Solidity

and this is what turns your solidity code into EVM code and how it's executed in the EVM

[a] A database backend is accessed by users indirectly through an external application, most likely an Ethereum client; see also: state database

[b] A bytearray is specific set of bytes [data] that can be loaded into memory. It is a structure for storing binary data, e.g. the contents of a file.

[c] This permanent data structure makes it possible to easily recall any previous state with its root hash keeping the resources off-chain and minimizing on-chain storage needs.

Part II.
Practice

15 Memory and Storage

15.1 Data Structures

Merkle-Patricia Trees Merkle-Patricia Trees

World State

Also known simply as "state", this is a MAPPING of **addresses** and **account states** (RLP data structures), this is also known as *state*, or σ. This mapping is not stored on the blockchain, rather it is stored as a Merkle-Patricia trie in a DATABASE BACKEND[a] that maintains a mapping of bytearrays to bytearrays.[b] The cryptographic internal data going back to the root node represents the *State* of the Blockchain at any given root, i.e. at any given *time*.[c] As a whole, the state is the sum total of database relationships in the **state database**. The state is an inert position on the chain, a position between prior state and post state; a block's frame of reference, and a defined set of relationships to that frame of reference.

15.2 Byte Arrays

15.3 Bit Sequences

Message Calls are either bit sequences or byte arrays.

15.4 The Block

Notation : $\mathbb{B}$

Description : A block is made up of 17 different elements, all of which play a unique role in the Blockchain. The first 15 elements that make up the block are part of what is called the *block header*.

1. **Parent Hash**: The hash of the parent block's header: given in Keccak-256.
2. **Ommers Hash**: The hash of the *Ommers List* portion of this block: given in Keccak-256.
3. **Beneficiary**: The 20-character (160-bit) hash to which block rewards are transferred when the block is successfully mined.
4. **State Root**: The Keccak-256 bit hash of the root node of the state.

Block Header

Notation : H

Description : The information contained in a block besides the transactions list. This consists of:

1. PARENT HASH – This is the Keccak-256 hash of the parent block's header.
2. OMMERS HASH – This is the Keccak-256 hash of the ommer's list portion of this block.
3. BENEFICIARY – This is the 20-byte address to which all block rewards are transferred.
4. STATE ROOT – This is the Keccak-256 hash of the root node of the state trie, after a block and its transactions are finalized.
5. TRANSACTIONS ROOT – This is the Keccak-256 hash of the root node of the trie structure populated with each transaction from a Block's transaction list.
6. RECEIPTS ROOT – This is the Keccak-256 hash of the root node of the trie structure populated with the receipts of each transaction in the transactions list portion of the block.
7. LOGS BLOOM – This is the bloom filter composed from indexable information (log address and log topic) contained in the receipt for each transaction in the transactions list portion of a block.
8. DIFFICULTY – This is the difficulty of this block – a quantity calculated from the previous block's difficulty and its timestamp.
9. NUMBER – This is a quantity equal to the number of ancestor blocks behind the current block.
10. GAS LIMIT – This is a quantity equal to the current maximum gas expenditure per block.
11. GAS USED – This is a quantity equal to the total gas used in transactions in this block.

[a] Formally **world state** $= \sigma$.
[b] σ is the world state at a certain given time, and n is the number of transactions or contract creations by that account.
[c] A particular path from root to leaf in the **state database** that encodes the STORAGE CONTENTS of the account.
[d] A message call is any interaction with the account on-chain.
[e] Formal notation is $\sigma[a]_c$
[f] More formal notation is $\text{KEC}(\mathbf{b}) = \boldsymbol{\sigma}[a]_c$

12. Timestamp – This is a record of Unix's time at this block's inception.
13. Extra Data – This byte-array of size 32 bytes or less contains extra data relevant to this block.
14. Mix Hash – This is a 32-byte hash that verifies a sufficient amount of computation has been done on this block.
15. Nonce – This is an 8-byte hash that verifies a sufficient amount of computation has been done on this block.
16. Ommer Block Headers – These are the same components listed above for any ommers.
17. Transaction Series – This is the only non-header content in the block.

15.5 State Database

15.6 Merkle-Patricia Trees

RLP

Well-Formed RLP

Account State

Notation : `body`

Description : The EVM-code fragment that executes each time an account receives a message call.

Description : The account state is made up of four variables:

1. **nonce** The number of transactions sent from this address, or the number of contract creations made by the account associated with this address.
2. **balance** The number of Wei owned by this address.
3. **storage_root** A 256-bit (32-byte) hash of the root node of a Merkle Patricia tree that encodes the storage contents of the account.
4. **code_hash** The hash of the EVM code of this account's contract.

The account state is the state of any particular account during some specified world state σ.[a]

Nonce The **nonce** aspect of an account's state is the number of transactions sent from, or the number of contract-creations by, the address of that account.[b]

Storage Root The **storage root** aspect of an account's state is the hash of the trie[c]

Code Hash The **code hash** aspect of an account's state is the hash of the evm code of this account. Code hashes are stored in the **state database**. Code hashes are permanent and they are executed when the address belonging to that account receives a message call.[def]

Balance The amount of **Wei** owned by this account.

- Key/value pair stored inside the root hash.
- L_I^*, is defined as the element-wise transformation of the base function
- The *element-wise transformation of the base-function* refers to all of the key/value pairs in L_I
- L_I refers to a particular trie.

15.7 Bloom Filter

Notation : `logs_bloom`

Description : The Bloom Filter is composed from indexable information (logger address and log topics) contained in each log entry from the receipt of each transaction in the transactions list.

Transaction Receipts

16 Processing and Computation

16.1 State Transition Function

State Transitions come about through a what is known as the State Transition Function; this is an abstraction of several operations in Ethereum which comprise the overall act of computing changes to the *machine state* prior to adding them to the *world state*, that is, through them being finalized and rewards applied to a given miner. `apply_rewards` and `block_beneficiary` are here.

16.2 `machine_state`

Description : The machine state is a tuple consisting of five elements:

1. `gas_available`
2. `program_counter`
3. `memory_contents` A series of zeroes of size 2^{256}
4. `memory_words.count`
5. `stack_contents`

There is also, [to_execute]: the current operation to be executed

Exceptional Halting

An exceptional halt may be caused by a handful of boolean values:

```
forall instruction.x
if gas_empty = true
then signal halt
elif instruction.x = fake
then signal halt
elif stack = terse
then signal halt
elif jumpdest = bad
then signal halt
else exec instruction.x

forall instruction.y
[...]
[...]
```

```
[...]
[...]

forall instruction.z
[...]
[...]
[...]
[...]

then signal controlled_halt
```

No instruction can, through its execution, cause an exceptional halt. They can only happen if some instruction, for whatever reason, fails to execute.

- The amount of remaining gas in each transaction is extracted from information contained in the `machine_state`
- A simple iterative recursive loop[1] with a boolean value:

  true indicating that in the run of computation, an exception was signaled

  false indicating in the run of computation, exceptions were signaled. If this value remains false for the duration of the execution until the set of transactions becomes a series (rather than an empty set.) This means that the machine has reached a controlled halt.

16.3 Mining

Block Beneficiary The 160-bit (20-byte, or 20-character) address to which all fees collected from the successful mining of a block are transferred.

Apply Rewards The third process in `block_finalization` that sends the mining reward to an account's address. A scalar value corresponding to the difficulty level of a current block. This can be calculated from the previous block's difficulty level and the timestamp.

Ethash

GHOST Protocol

16.4 Verification

Verifies Ommer headers

Ommers

Ommershash

`Is_Sibling` **Property**

16.5 Transactions

The basic method for Ethereum accounts to interact with eachother. Transactions lie at the heart of Ethereum, and are entirely responsible for the dynamism and evolution of the platform. Transactions are the bread and butter of state transitions, that is of block additions, which are all the computation performed in one block. Each transaction applies the execution changes to the *machine state*, a temporary state which consists of all the temporary changes in computation that must be made before a block is finalized and added to the world state.

Notation : `sender`

Description : A function that maps transactions to their sender using `ECDSA` of the SECP-256k1 curve, (excepting the latter three signature fields) as the datum to sign. The sender of a given transaction can be represented: `transaction.sender`

16.6 Execution

Notation : `execution`

Description : The execution of a transaction defines the state transition function: `stf`. However, before any transaction can be executed it needs to go through the initial tests of intrinsic validity.

Intrinsic Validity

The criteria for intrinsic validity are as follows:

- The transaction follows the rules for *well-formed RLPs* (recursive length prefixes.)
- The *signature* on the transaction is valid.

- The *nonce* on the transaction is valid, i.e. it is equivalent to the sender account's current nonce.
- The gas_limit is greater than or equal to the intrinsic_gas used by the transaction.
- The sender's account balance contains the cost required in up-front payment.

Accordingly, the post-transactional state of Ethereum is expressed thus:
transaction(post.state) = stf(present.state, transaction)
While the amount of gas used in the execution is expressed: stf(gas_used) and the accrued log items belonging to the transaction are expressed: stf(logsbloom, content)(logsbloom, set) Information concering the result of a transaction's execution is stored in the transaction receipt tx_receipt. The set of log events which are created through the execution of the transaction, logs_set in addition to the bloom filter which contains the actual information from those log events logs_bloom are located in the transaction receipt. In addition, the post-transaction state post_transaction(state) and the amount of gas used in the block containing the transaction receipt post(gas_used) are stored in the transaction receipt. Thusly the transaction receipt is a record of any given execution.

A valid transaction execution begins with a permanent change to the state: the nonce of the sender account is increased by one and the balance is decreased by the collateral_gas[a] which is the amount of gas a transaction is required to pay prior to its execution. The original transactor will differ from the sender if the message call or contract creation comes from a contract account executing code.

After a transaction is executed, there comes a PROVISIONAL STATE:
post_execution(provisional.state)
Gas used for the execution of individual EVM opcodes prior to their potential addition to the world_state creates the provisional state. productive_gas, and an associated substate substate_a.

Code execution always depletes gas. If gas runs out, an out-of-gas error is signaled (oog) and the resulting state defines itself as an empty set; it has no effeffect on the world state. This describes the transactional nature of Ethereum. In order to affect the WORLD STATE, a transaction must go through completely or not at all.

[a] Designated "intrinsic_gas" in the Yellowpaper

Execution Model

Notation : EVM

Description : The stack-based *virtual machine* which lies at the heart of the Ethereum and performs the actions of a computer. This is actually an instantial runtime that executes several substates, as EVM computation instances, before adding the finished result, all calculations having been completed, to the final state via the finalization function.

In addition to the system state $\boldsymbol{\sigma}$, and the remaining gas for computation g, there are several pieces of important information used in the execution environment that the execution agent must provide; these are contained in the tuple I:

- account_address, the address of the account which owns the code that is executing.
- sender_address the sender address of the transaction that originated this execution.
- originator_price the price of gas in the transaction that originated this execution.
- input_data, a byte array that is the input data to this execution; if the execution agent is a transaction, this would be the transaction data.
- account_address the address of the account which caused the code to be executing; if the execution agent is a transaction, this would be the transaction sender.
- newstate_value the value, in Wei, passed to this account if the execution agent is a transaction, this would be the transaction value.[1]
- code.array the byte array that is the machine code to be executed.[1]
- samestate_header the block header of the present block.

- the stack depth the depth of the present message-call or contract-creation (i.e. the number of CALLs or CREATEs being executed at present).[1]

Message Calls

Description : Messages allow communication between accounts (whether contract or external,) and are a carryover from established concepts in Computer Science, most notably the *MPI: Message-Passing Framework*. Messages can come in the form of `msg_calls` which give output data. If an account has EVM code in it (a contract account,) this code gets executed when the account receives a message call. Message calls and contract creations are both *transactions*, but contract creations are never considered the same as message calls. Message calls always transfer some amount of value to an account. If the message call is an account creation transaction then the value given is takes on the role of an endowment toward the new account. Every time an account receives a message call it returns the body, something which is triggered by the `init` function. A message call can come through a transaction, or through the internal execution of code. Message call `transactions` only contain data. They are separate from regular, standard *transactions*.

Message calls always have a universally agreed-upon cost in gas. There is a strong distinction between contract creation transactions and message call transactions. Computation performed, whether it is a contract creation or a message call, represents the currently legal valid state. There can be no invalid transactions from this point.[1] There is also a message call/contract creation *stack*. This stack has a depth, depending on how many transactions are in it. Contract creations and message calls have entirely different ways of executing, and are entirely different in their roles in Ethereum. The concepts can be conflated. Message calls can result in computation that occurs in the next state rather than the current one. If an account that is currently executing receives a message call, no code will execute, because the account might exist but has no code in it yet. To execute a message call transactions are required:

- `Sender`
- `Transaction_Originator`
- `Recipient`
- `Account` (usually the same as the recipient)
- `Available_Gas`
- `Value`
- `Gas_Price`
- An arbitrary length byte-array. `arb_array`
- `Present_Depth` of the message call/contract creation stack.

Notation : `data`

Description : User data input to a `message_call`, structured as an unlimited size byte-array.

Contract Creation

Notation : `init`

Description : When INIT is executed it returns the BODY. Init is executed only once at ACCOUNT_CREATION, and permanently discarded after that. Contract creation transactions are equal the recurisive length prefix of an empty byte-sequence.

Account Creation

16.7 Halting

Execution Environment

Notation : `ERE`

Description : The environment under which an Autonomous Object executes in the EVM: the EVM runs as a part of this environment.

Notation : `big_endian_f`

Description :BIG ENDIAN FUNCTION This function expands a positive-integer value to a big-endian byte array of minimal length. When accompanied by a $\cdot$ operator, it signals sequence concatenation. The `big_endian` function accompanies RLP serialization and deserialization.

[a] A full list of Opcodes is in Appendix B

16.8 Gas

Description : The fundamental network cost unit converted to and from Ether as needed to complete the transaction while it is sent. Gas is arbitrarily determined at the moment it is needed, by the block and according to the miners decision to charge certain fees.

Miner Choice Miners choose which gas prices they want to accept.

Gasprice

Notation : `gas_limit`

Description : A value equal to the current limit of gas expenditure per block, according to the miners.

Gaslimit Any unused gas is refunded to the user.

Gasused

Description : A value equal to the total gas used in transactions in this block.

Machine State

Substate The `substate` is an emergent, ever-changing ball of computational energy that is about to be applied to the main state. It is the *meta state* by which transactions are decided valid and to be added to the blockchain.

EVM Code

The bytecode that the EVM can natively execute. Used to explicitly specify the meaning of a message to an account.

Opcodes/EVM Assembly

The human readable version of EVM code. But what exactly are these computer instructions that can be executed with the same level of veracity and certainty as Bitcoin transactions? How do they come about, what makes them up, how are they kept in order, and what makes them execute? The first part of answering this question is understanding opcodes. In traditional machine architectures, you may not be introduced to working with processor-level assembly instructions for some time. In Ethereum however, they are essential to understanding the protocol because they are the most minute and subtle (yet HUGELY important) things going on in the Ethereum Blockchain at any moment, and they are the real "currency," that Ethereum trades in. I'll explain what I mean by that in a minute. First, let's go over a few Opcodes:[a]

Data	Opcode	Gas	In	Out
0x00	STOP	0	2	1
0x01	ADD	3	2	1
0x02	MUL	5	2	1
0x03	SUB	3	2	1
0x04	DIV	5	2	1

The STOP Opcode is used in order to stop a computation once it has completed, or to halt a computation if it has run out of gas. The ADD, MUL, SUB, and DIV operations are addition, multiplication, subtraction and division operations. The In/Out columns refer to inputs (to `machine_state`), the state which decides every new `world_state`.

Part III.
Appendix

A Opcodes

Data	Opcode	Gas	Input	Output
0x00	STOP	0	0	0
0x01	ADD	3	2	1
0x02	MUL	5	2	1
0x03	SUB	3	2	1
0x04	DIV	5	2	1
0x05	SDIV	5	2	1
0x06	MOD	5	2	1
0x07	SMOD	5	2	1
0x08	ADDMOD	8	3	1
0x09	MULMOD	8	3	1
0x0a	EXP	10	2	1
0x0b	SIGNEXTEND	5	2	1
0x10	LT	3	2	1
0x11	GT	3	2	1
0x12	SLT	3	2	1
0x13	SGT	3	2	1
0x14	EQ	3	2	1
0x15	ISZERO	3	1	1
0x16	AND	3	2	1
0x17	OR	3	2	1
0x18	XOR	3	2	1
0x19	NOT	3	1	1
0x1a	BYTE	3	2	1
0x20	SHA3	30	2	1
0x30	ADDRESS	2	0	1
0x31	BALANCE	400	1	1
0x32	ORIGIN	2	0	1
0x33	CALLER	2	0	1
0x34	CALLVALUE	2	0	1
0x35	CALLDATALOAD	3	1	1
0x36	CALLDATASIZE	2	0	1
0x37	CALLDATACOPY	3	3	0
0x38	CODESIZE	2	0	1
0x39	CODECOPY	3	3	0
0x3a	GASPRICE	2	0	1
0x3b	EXTCODESIZE	700	1	1
0x3c	EXTCODECOPY	700	4	0

0x3d	RETURNDATASIZE	2	0	1
0x3e	RETURNDATACOPY	3	3	0
0x40	BLOCKHASH	20	1	1
0x41	COINBASE	2	0	1
0x42	TIMESTAMP	2	0	1
0x43	NUMBER	2	0	1
0x44	DIFFICULTY	2	0	1
0x45	GASLIMIT	2	0	1
0x50	POP	2	1	0
0x51	MLOAD	3	1	1
0x52	MSTORE	3	2	0
0x53	MSTORE8	3	2	0
0x54	SLOAD	200	1	1
0x55	SSTORE	0	2	0
0x56	JUMP	8	1	0
0x57	JUMPI	10	2	0
0x58	PC	2	0	1
0x59	MSIZE	2	0	1
0x5a	GAS	2	0	1
0x5b	JUMPDEST	1	0	0
0xa0	LOG0	375	2	0
0xa1	LOG1	750	3	0
0xa2	LOG2	1125	4	0
0xa3	LOG3	1500	5	0
0xa4	LOG4	1875	6	0
0xf0	CREATE	32000	3	1
0xf1	CALL	700	7	1
0xf2	CALLCODE	700	7	1
0xf3	RETURN	0	2	0
0xf4	DELEGATECALL	700	6	1
0xf5	CALLBLACKBOX	40	7	1
0xfa	STATICCALL	40	6	1
0xfd	REVERT	0	2	0
0xff	SUICIDE	5000	1	1

References

[1] D. G. Wood, *Ethereum: A secure decentralised generalised transaction ledger*, https://github.com/ethereum/yellowpaper, 2017 (cit. on pp. 11–13).

[2] etherscan.io, *Charts: Ethereum blocksize history*. [Online]. Available: https://etherscan.io/chart/blocksize (cit. on p. 20).

[3] S. Levy, *Hackers: Heroes of the computer revolution*. O'Reilly, 2010 (cit. on p. 21).

[4] BillWagner, *Serialization (c#)*. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/serialization/ (cit. on p. 21).

[5] *Design patterns and refactoring*. [Online]. Available: https://sourcemaking.com/design_patterns/singleton (cit. on p. 21).

[6] G. E. Ngondi and A. Butterfield, *A dictionary of computer science*. Oxford University Press, 2016 (cit. on p. 22).

[7] E. Foundation, *Ethereum whitepaper*, https://github.com/ethereum/wiki/wiki/White-Paper, 2017 (cit. on p. 22).

Glossary

abstract machine An abstract machine is a conceptual model of a computer that describes its own operations with perfect accuracy. Since abstract machines are theoretical, all possible outputs can be determined beforehand. 20

Address A 160-bit (20-byte) code used for identifying Accounts. 20

addresses 20 character strings, specifically the rightmost 20 characters of the `Keccak-256` hash of the RLP-derived mapping which contains the sender's address and the nonce of the block.. 20

Autonomous Object A notional object existent only within the hypothetical state of Ethereum. Has an intrinsic address and thus an associated account; the account will have non-empty associated EVM Code. Incorporated only as the Storage State of that account. 20

Balance A value which is intrinsic to accounts; the quantity of Wei in the account. All EVM operations are associated with changes in account balance. 20

beneficiary The 160-bit address to which all fees collected from the successful mining of this block be transferred. 20

bit The smallest unit of electronic data storage: there are eight bits in one byte. It is common to speak of cryptographic hashing algorithms using bits (e.g. 160-bits instead of 20-bytes). 20

Bit The smallest unit of electronic data storage: there are eight bits in one byte. The Yellowpaper gives certain values in bits (e.g. 160 bits instead of 20 bytes). 20

block header The information in a block besides transaction information. It consists of a dozen parts: (lists the 12 parts). 20

blockchain A consensus-based record of agreement where chunks of data[a] (called blocks) are stored with cryptographic hashes linking each Block to the next, ensuring their validity. See also: *hashing functions*.. 20

Contract A piece of EVM Code that may be associated with an Account or an Autonomous Object. 20

Cryptographic hashing functions Hash functions make secure blockchains possible by establishing universal inputs for which there can only be one given output.[b]The reason this works is because the hash of a block's data is a certainty, just like two plus two equals four is a certainty.. 20

Dapp An end-user-visible application hosted in an Ethereum browser.. 20

design pattern a pattern of design in OOP. 20

Ethereum Runtime Environment The environment which is provided to an Autonomous Object executing in the EVM. Includes the EVM but also the structure of the world state on which the relies for certain I/O instructions including CALL & CREATE. 20

Ethereum Browser [c] A cross-platform GUI of an interface similar to a simplified browser (a la Chrome) that is able to host applications, the backend of which is purely on the Ethereum protocol.. 20

Ethereum Foundation The non-profit organization in charge of executing the development processes of Ethereum in line with the Whitepaper. 20

Ethereum Virtual Machine A sub-process of the *State Transition Function* which initializes and

[a] As of November 19, 2017, roughly between 1 and 25 kilobytes in size[2]

[b] Actually, most hashing functions eventually have some collision points where two viable inputs reproduce the same output. But actual collision points are rare discoveries and tend to be followed (if not preceded by) newer more powerful hashing algorithms that are yet harder to break or find collisions in. Since the number space is infinite, we aren't likely to run out of potential new and larger hashing algorithms any time soon. Older hashing algorithms with known collisions, such as MD5 are not recommended for use in applications with stringent security requirements.

[c] a.k.a. Ethereum Reference Client

executes all of the transactions (ergo computations) in a block, prior to their finalization into the state.. 20

EVM Assembly The human readable version of EVM code. 20

EVM Code The bytecode that the EVM can natively execute. Used to formally specify the meaning and ramifications of a message to an Account. 20

External Actor A person or other entity able to interface to an Ethereum node, but external to the world of Ethereum. It can interact with Ethereum through depositing signed Transactions and inspecting the blockchain and associated state. Has one (or more) intrinsic Accounts. 20

Gas The fundamental network cost unit. Paid for exclusively by Ether (as of PoC-4), which is converted freely to and from Gas as required. Gas does not exist outside of the internal Ethereum computation engine; its price is set by the Transaction and miners are free to ignore Transactions whose Gas price is too low. 20

hacker ethic A maxim purporting that knowledge about and access to proprietary computer systems should be free and unhindered for anyone who is willing and able to explore and improve it.[3] The open-source and decentralized nature of Ethereum makes it one of the most thorough and robust implementations of the *Hacker Ethic* to date.. 20

leaf node the bottom-most node in a particular tree, of blocks, one half of the "key" the other half being the root node, which creates the path between. 20

Lower-Level Lisp The Lisp-like Low-level Language, a human-writable language used for authoring simple contracts and general low-level language toolkit for trans-compiling to. 20

Message Data (as a set of bytes) and Value (specified in Wei) that is passed between two Accounts, either through the deterministic operation of an Autonomous Object or the cryptographically secure signature of the Transaction. 20

Message The act of passing a message from one Account to another. If the destination account is associated with non-empty EVM Code, then the VM will be started with the state of said Object and the Message acted upon. If the message sender is an Autonomous Object, then the Call passes any data returned from the VM operation. 20

Object Synonym for Autonomous Object. 20

public key A term originating from *cryptography* and corresponding to **private key**, this is the 42-byte (i.e. 42-character) string of ASCII digits which transacts on the public network. 20

root node the uppermost node in a particular tree, of blocks, representing a single world state$^{\sigma}$ at a particular time. 6, 20

serialization Serialization is the process of converting an object into a stream of bytes in order to store the object or transmit it to memory, a database, or a file. Its main purpose is to save the state of an object in order to be able to recreate it when needed. The reverse process is called deserialization.[4]. 20

singleton A design pattern in Object-Oriented Programmdng which specifies a class with one instance but with a global point of access to it[5]. 20

specification Technical descriptions, instructions, and definitions from which other people can create working prototypes. 20

state A permanent, static, state `state`. 20

state-transition . 20

State Database A database backend that maintains a mapping of bytearrays to bytearrays. 20

state machine A state machine rests in a universal, stable, singular condition, called a state. State machines transition to new states given certain compatible inputs.. 20

state database A database stored off-chain, [i.e. on the computer of some user running an Ethereum client] which contains a trie structure mapping bytearrays [i.e. organized chunks of binary data] to other bytearrays [other organized chunks of binary data]. The RELATIONSHIPS between each node on this trie constitute a MAP, a.k.a. a MAPPING of all PREVIOUS STATES on the **EVM** which a client might need to reference. 6, 8, 20

Storage State The information particular to a given Account that is maintained between the times that the Account's associated EVM Code runs. 20

transaction An input message to a system that, because of the nature of the real-world event or activity it reflects, is required to be regarded as a single unit of work guaranteeing to either be processed completely or not at all.[6]. 20

Transaction A piece of data, signed by an External Actor. It represents either a Message or a new Autonomous Object. Transactions are recorded into each block of the blockchain. 20

trie A tree-structure for organizing data, the position of data in the tree contains the particular path from root to leaf node that represents the key (the path from root to leaf is "one" key) you are searching the trie structure for. The data of the key is contained in the trie relationships that emerge from related nodes in the trie structure. 6, 8, 20

Whitepaper A conceptual map, distinct from the Yellowpaper, which highlights the development goals for Ethereum as a whole[7]. 20

Yellowpaper Ethereum's primary formal specification, written by Dr. Gavin Wood, one of the founders of Ethereum.. 20

Acronyms

ERE Ethereum Runtime Environment. 20

EVM Ethereum Virtual Machine. 20

LLL Lower Level Lisp. 20

OOP Object-Oriented Programming. 20

YP Yellowpaper. 20

Index