Introduction to Blockchain, Ethereum and Smart Contracts — Chapter 1

Ritesh Modi
May 16, 2018 · 27 min read

This decade has already witnessed an extraordinary evolution in the technology and computing ecosystem. Technology innovation and its impact is already running very high. From IoT to Artificial Intelligence to Blockchain. Each of them have a disruptive force within multiple industries and Blockchain is termed as one of the most disruptive technology of today. So much so, Blockchain has potential to change almost every industry today and its working. The applicability of Blockchain because of its advantages and pervasiveness has already picked up stream and seems like it will continue to long time to come. Blockchain is not a new technology however it has gained super momentum in last couple of years. It is a big leap forward in terms of things about decentralized and distributed applications. It is about thinking of current architectural landscape and strategize to move towards immutable distributed databases. The advantages and many and helping organisations reach out to their stakeholders without requiring any central authority and intermediaries.

In this first chapter, you will quickly learn understand the basic and foundational concepts of Blockchain and Ethereum. I will also discuss the important components and their interaction to make Blockchain and Ethereum work. It will also touch briefly on the topic of smart contracts and how to author them using solidity.

It is to be noted that this chapter explain important Blockchain and Ethereum concepts briefly for writing sound Solidity contracts. It does not explain concepts in details and complete book can be written for that purpose. Since, Ethereum is an implementation of Blockchain, they have been used interchangeable in this book.

This chapter will focus on

· Architecture of Blockchain · Understanding Storage

· Understanding Blocks

· Understanding Transactions

· Understand Mining

· Understanding Accounts

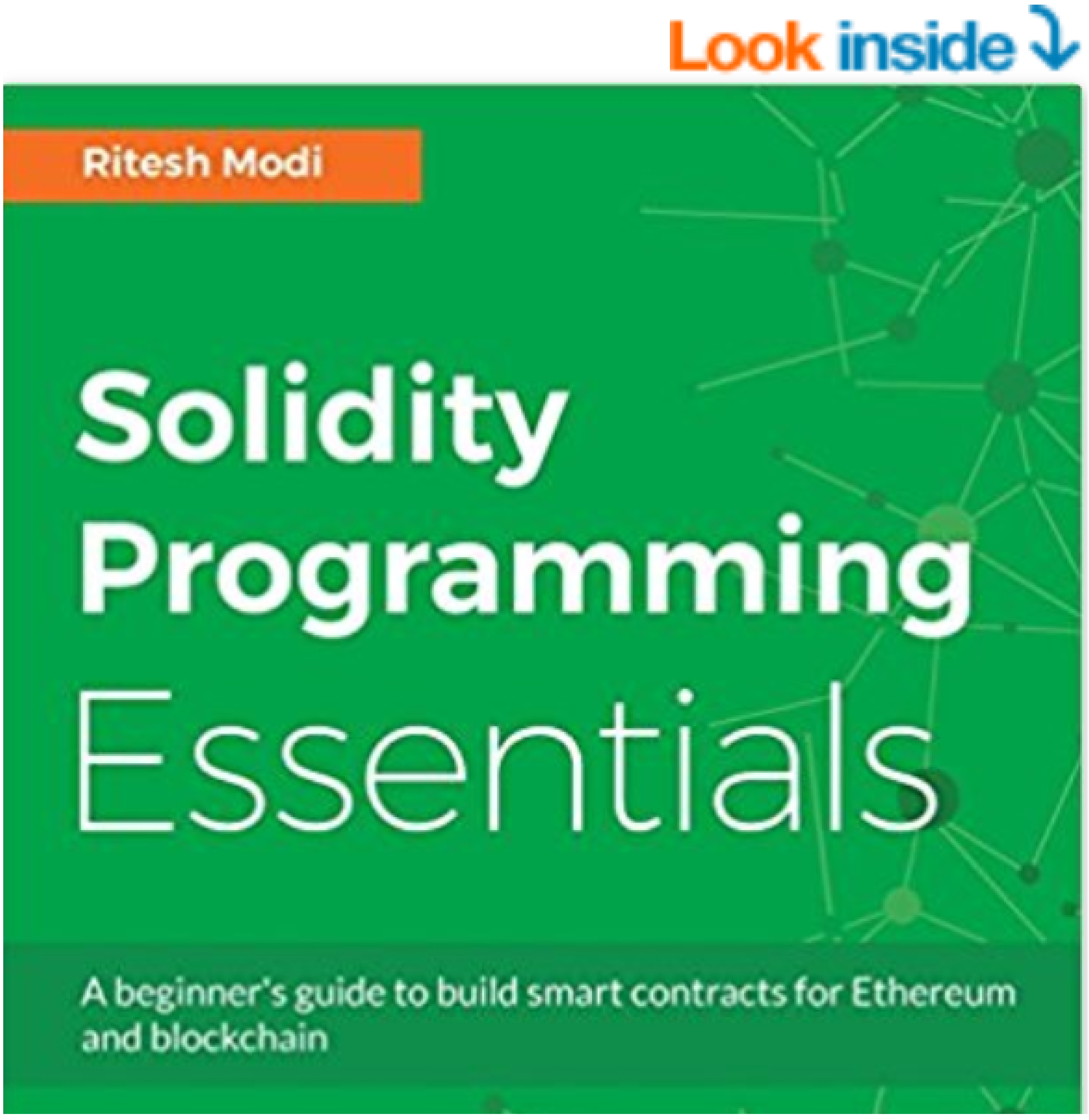
· Understanding Cryptography

· Gas

· Ether

· Sending transaction

· Smart Contracts


Look inside
Ritesh Modi
Solidity Programming Essentials
A beginner's guide to build smart contracts for Ethereum and blockchain

What is Blockchain?

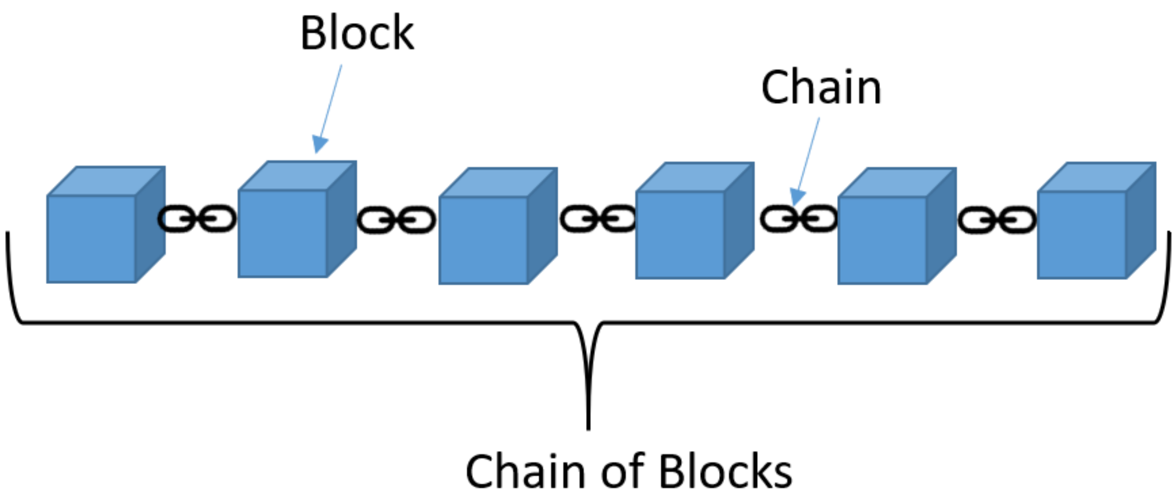
Blockchain is essentially a decentralized distributed database or a ledger. There are some very important keywords like Decentralized, distributed, database and ledger used in defining a Blockchain.

Decentralization in simple term means that the application or service continues to be available and usable even if a server or a group of servers on a network crashes or are not available. The service or application is deployed on a network in a way that no server has absolute control over data and execution rather each server has current copy of data and execution logic with them.

Distributed means that any server or node on a network is connected to every other node on the network directly or indirectly. Rather than having one to one or one to many connectivity between servers, servers have many to many connections with other servers.

Database refers to location for storing durable data that can be accessed at any point in time. Database allows storage and retrieval of data and provides management functionalities to manage data efficiently like export, import, backup and restoration.

Ledger is an accounting term and think of it as specialized storage and retrieval of data. Think of ledgers that are available with banks. When a transaction is executed with a Bank say Tom deposits 100 dollars in his account, the bank enters this information in ledger as a credit. At some point in time in future Tom withdraws 25 dollars. The bank does not modify the existing entry and stored data from 100 to 75. Instead it adds another entry in the same ledger as a debit of 25 dollars. It means a ledger is a specialized database that do not allow modification of existing data. It allows creation and appending of new transaction to modify the current balance in ledger. Blockchain is a database that has the same characteristics of a ledger. It allows newer transaction to be stores in append only pattern without any scope to modify past transaction. It is important here to understand that existing data can be modified be using a new transaction, but past transactions cannot be modified. A balance of 100 dollar can be modified anytime by executing a new debit or credit transaction but previous transactions cannot be modified.



Blockchain as the word refers means a chain of Blocks. Blockchain means having multiple blocks chained together and each block stores transactions in a way that it is not possible to modify these transactions. We will discuss in later section about storage of transactions and how immutability is achieved in Blockchain.

Not being able to change and modify past transactions makes Blockchain solution highly trustworthy, transparent and incorruptible.

It is important to understand that blocks and its chain is just one of the facets of blockchain. There are other important concept like mining, miners, consensus and protocol that works along with chain of blocks to make blockchain work flawlessly.

Ethereum is an implementation of Blockchain and allows extending its functionality with the help of smart contracts. Smart contracts will be addressed in detail through out this book.

Why Blockchain

The main objective of Ethereum is to accept transactions from accounts, update their state and maintain this state as current state till another transaction updates it again. The whole process of accepting, executing and writing transactions can be divided into two phases in Ethereum. There is a decoupling between when a transaction is accepted by Ethereum and when the transaction is executed and written to the ledger. This decoupling is quite important for decentralization and distributed architecture to work as expected.

Blockchain helps primarily in three different ways

1. Trust — Blockchain helps in creating applications that are decentralized and collectively owned by multiple people. No body within this group has the power to change or delete previous transactions. Even if someone tries to do so, it will not be accepted by other stakeholders.

2. Autonomy — There is no single owner for Blockchain based applications. No one controls the blockchain, but everyone participates into its activities. This helps in creating solutions that cannot be manipulated or induce corruption.

3. Integrity — The state and transactions are secured cryptographically and cannot be modified easily.

4. Intermediaries — Blockchain based application can help remove the intermediaries from existing processes. Generally, there is a central body like Vehicle registration, licence issuing etc who acts as registrar for registering vehicles as well as issuing driver licences. Without Blockchain based systems, there is no central body and if a licence is issues or vehicle is registration after Blockchain mining process, that will remain a fact for epoch time-period without the need of any central authority vouching for it.

Blockchain is heavily depended on Cryptography technologies and next section is a brief on it.

Cryptography

There are primarily two types of cryptography in computing:

Symmetric Encryption and Decryption

Symmetric cryptography refers to process of using a single key both encryption and decryption. It means the same key should be available to multiple people if they want to exchange messages using this form of cryptography

aSymmetric Encryption and Decryption

Asymmetric cryptography refers to process of using two keys for encryption and decryption. Any key can be used for encryption and decryption. Messages encryption with public key can be decrypted using private key and messages encryption by private key can be decrypted using public key. Let's understand this with the help of an example. Tom using Alice's public key to encrypts messages and sends it to Alice. Alice can use her private key to decrypt the message and extract contents out of it. Messages encrypted with Alice's public key can only be decrypted by Alice as she only holds her private key and no one else. This is the general use case of Asymmetric keys. There is another use which we will see while discussing Digital signatures.

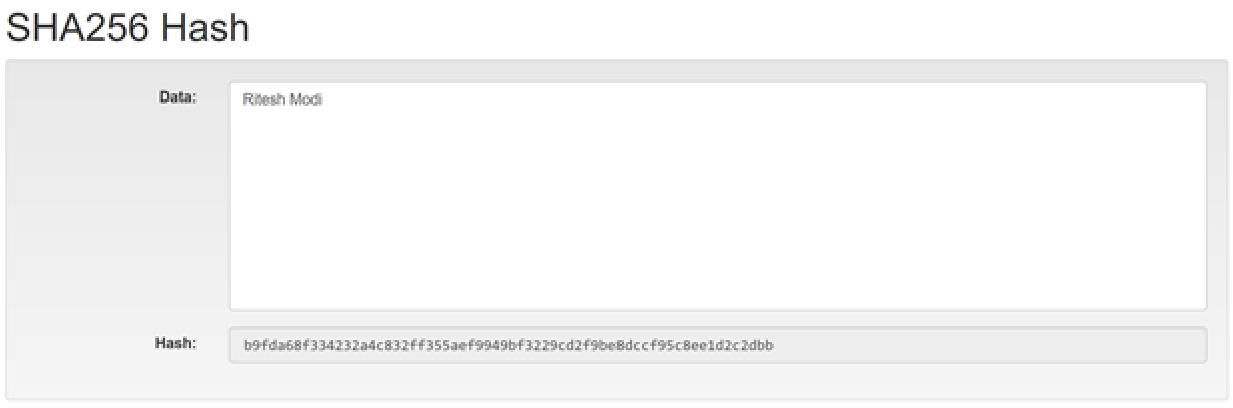
Hashing

Hashing is the process of transforming string data into another fixed length string data and it is not possible to re-generate or identify the original data from resultant string data. Hashing ensures that even a slight change in input data will completely change the output data and no one can ascertain the change in the original data. There is another important property of hashing is that no matter the size of input string data the length of its output is always fixed. For example, using SHA256 hashing algorithm and function with any length of input will always generate 256-bit output data. This can especially become useful when large amount of data can be stored as 256-bit output data. Ethereum uses Hashing quite extensively. It hashes all the transaction data, hashes multiple transaction hashes to generate single root transaction hash and in fact the blocks in Ethereum are also represented as hash.

Another important property of hashing is that it is mathematically not feasible to identify two different input strings that will output the same hash. Similarly, it is not possible to computationally and mathematically find the input from the hash itself.

Ethereum uses SHA256 for all its hashing needs.

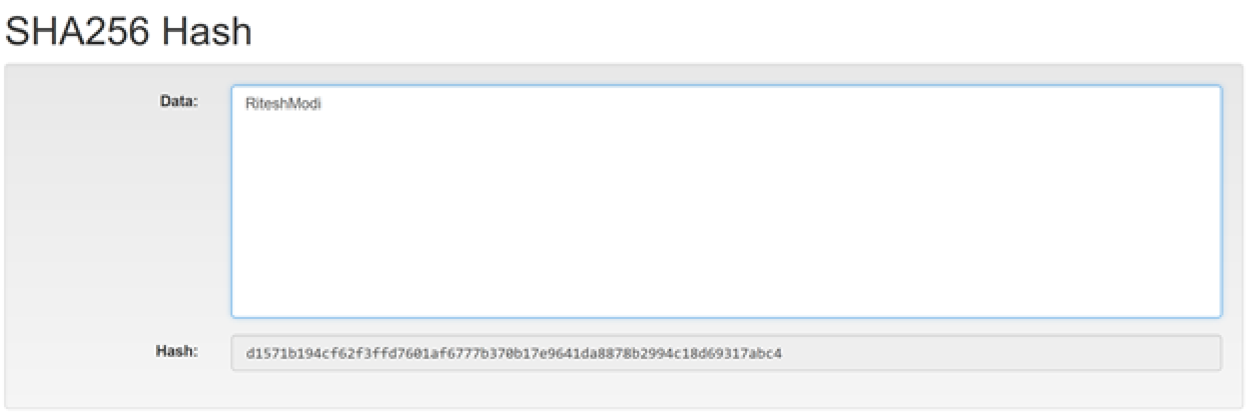
The next image shows an example of Hashing. The input Ritesh Modi generates a hash "b9fda68f334232a4c832ff355aef9949bf3229cd2f9be8dccf95c8ee1d2c2dbb"


SHA256 Hash

Data: Ritesh Modi

Hash: b9fda68f334232a4c832ff355aef9949bf3229cd2f9be8dccf95c8ee1d2c2dbb

And even a small modification to input generates a completely different hash.



SHA256 Hash

Data: RiteshModi

Hash: d1571b194cf62f3ffd7601af6777b370b17e9641da8878b2994c18d69317abc4

Digital Signatures

Earlier, we discussed cryptography using asymmetric keys. One of the important uses case of using Asymmetric keys in in creation and verification of Digital signature. Digital signatures are very similar to signature done by an individual on a piece of paper. Similar to a paper signature, digital signature helps in identifying an individual. It also helps in ensuring that messages are not tampered with in transit. Lets understand Digital signature with the help of an example.

Alice wants to send a message to Tom. How can Tom identify and ensure that the message has come from Alice only and that the message has not been changed or tampered with in transit? Alice takes the message she wants to send to Tom and generates a hash of it and then using her private key (yes, private key) encrypts the hash and appends the resultant cipher data to the original message.

Once the resultant message reaches to Tom, he segregates the messages into the original message and cipher data. He decrypts the cipher data using Alice's public key and extracts the hash out of it. He further hashes just the original message and compares both the hashes. If the hashes are same, it means that the message is not tampered with. It also establishes the fact that the message is originated by Alice as only she can encrypt the hash with her private key.

Digital signatures are used to sign transaction data by the owner of asset or crypto-currency like ether.

Ether

Ether is the currency of Ethereum. Every activity on Ethereum that modifies its state costs Ether as fee and miners who are successful in generating and writing a block in chain are also rewards Ether. Ether can easily be converted to dollars or other traditional currencies through Crypto-exchanges.

Ethereum has a metric system of denominations used as units of ether. The smallest denomination aka base unit of ether is called Wei. Below is a list of the named denominations and their value in Wei. This information is available at https://github.com/ethereum/web3.js/blob/0.15.0/lib/utils/utils.js#L40

‘wei’ : ‘1’

‘kwei’: ‘1000’,

‘ada’: ‘1000’,

‘femtoether’: ‘1000’,

‘mwei’: ‘1000000’,

‘babbage’: ‘1000000’,

‘picoether’: ‘1000000’,

‘gwei’: ‘1000000000’,

‘shannon’: ‘1000000000’,

‘nanoether’: ‘1000000000’,

‘nano’: ‘1000000000’,

‘szabo’: ‘1000000000000’,

‘microether’: ‘1000000000000’,

‘micro’: ‘1000000000000’,

‘finney’: ‘1000000000000000’,

‘milliether’: ‘1000000000000000’,

‘milli’: ‘1000000000000000’,

‘ether’: ‘1000000000000000000’,

‘kether’: ‘1000000000000000000000’,

‘grand’: ‘1000000000000000000000’,

‘einstein’: ‘1000000000000000000000’,

‘mether’: ‘1000000000000000000000000’,

‘gether’: ‘1000000000000000000000000000’,

‘tether’: ‘1000000000000000000000000000000’

Gas

In last section, it was mentioned that Ether is paid as fees for any execution that changes state in Ethereum. Ether is traded on public exchanges and its price fluctuate daily. If Ether is used for paying fees, then the cost of using the same service could be very high on certain days and low on other days. People will wait for price of Ether to fall to execute their transactions. This is not ideal for a platform like Ethereum. Gas helps in alleviating this problem. Gas is the internal currency of Ethereum. The execution and resource utilization cost is predetermined in Ethereum in terms of Gas units. This is also known as Gas Cost. There is also Gas price that can be adjusted to lower price when price of Ether increases and higher price when price of Ether decreases. For example, to invoke a function in a contract that modifies a string will cost Gas which is pre-determined, and Users should pay in terms of Gas to ensure smooth execution of this transaction.

Blockchain and Ethereum Architecture

Blockchain is an architecture comprising of multiple components and what makes Blockchain unique is the way these components function and interact with each other. Some of the important Ethereum components are Ethereum virtual machine, Miner, Blocks, Transactions, Consensus algorithm, Accounts, Smart contracts, mining nodes, Ether and Gas. We are going to discuss each of this component in this chapter.

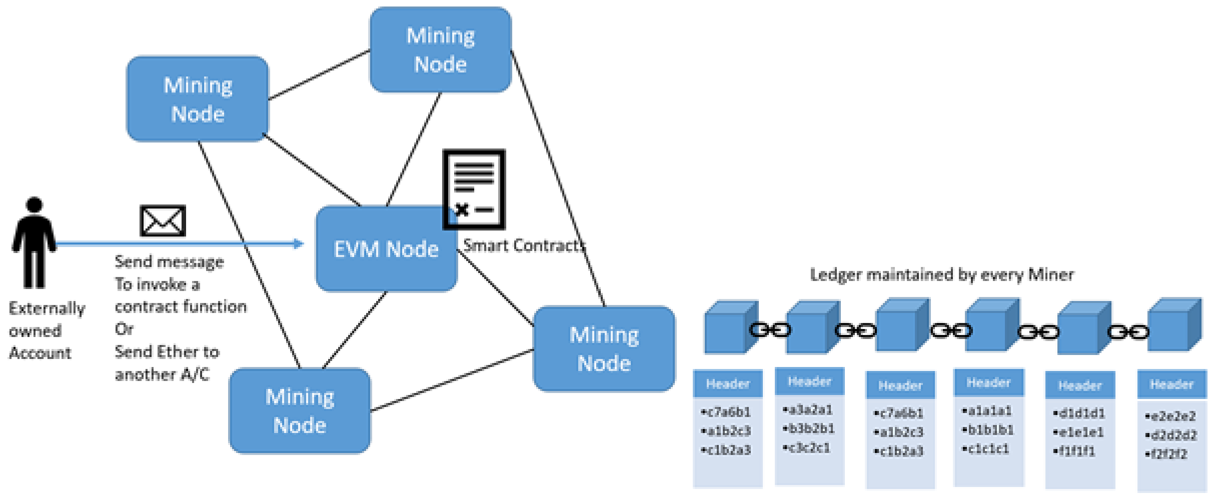
A Blockchain network consists of multiple nodes belonging to miners and some nodes that do not mine but helps in execution of smart contracts and transactions. These are known as Ethereum virtual machines. Each node is connected to other node on the network. These nodes use peer-to-peer protocol to talk to each other. They by default use 30303 port number to talk among themselves.

Each miner maintains an instance of ledger. Ledger contains all blocks in the chain. With multiple miners it is quite possible that each miner's ledger instance might have different blocks than other. The miners synchronize their blocks on an on-going basis to ensure that every miner's ledger instance is same as other.

Details about ledger, Blocks and transactions are discussed in detail in subsequent sections in this chapter.

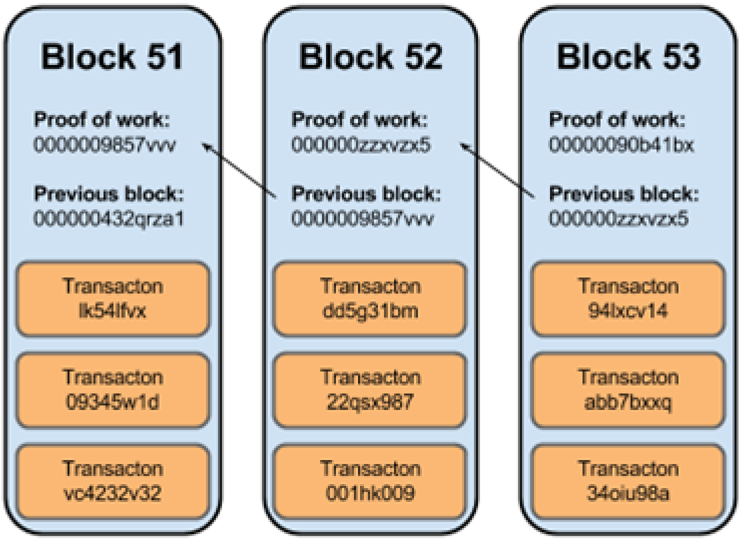
The EVM also hosts Smart contracts. Smart contracts help in extending Ethereum by writing custom business functionality into it. These smart contracts can be executed as part of transaction and it follows the process of mining as discussed earlier.

A person having an account on network can send a message for transfer of Ether from his account to another or he can send a message to invoke a function within a contract. Ethereum do not distinguish them as far as transactions are considered. The transaction must be digitally signed an account holder's private key. This is to ensure that identity of sender can be established while verifying the transaction and changing balances of multiple accounts.


Mining Node
Mining Node
EVM Node
Smart Contracts
Mining Node
Mining Node
Externally owned Account
Send message To invoke a contract function Or Send Ether to another A/C
Ledger maintained by every Miner
Header •c7a6b1 •a1b2c3 •c1b2a3
Header •a3a2a1 •b3b2b1 •c3c2c1
Header •c7a6b1 •a1b2c3 •c1b2a3
Header •a1a1a1 •b1b1b1 •c1c1c1
Header •d1d1d1 •e1e1e1 •f1f1f1
Header •e2e2e2 •d2d2d2 •f2f2f2

How are Blocks related to each other?

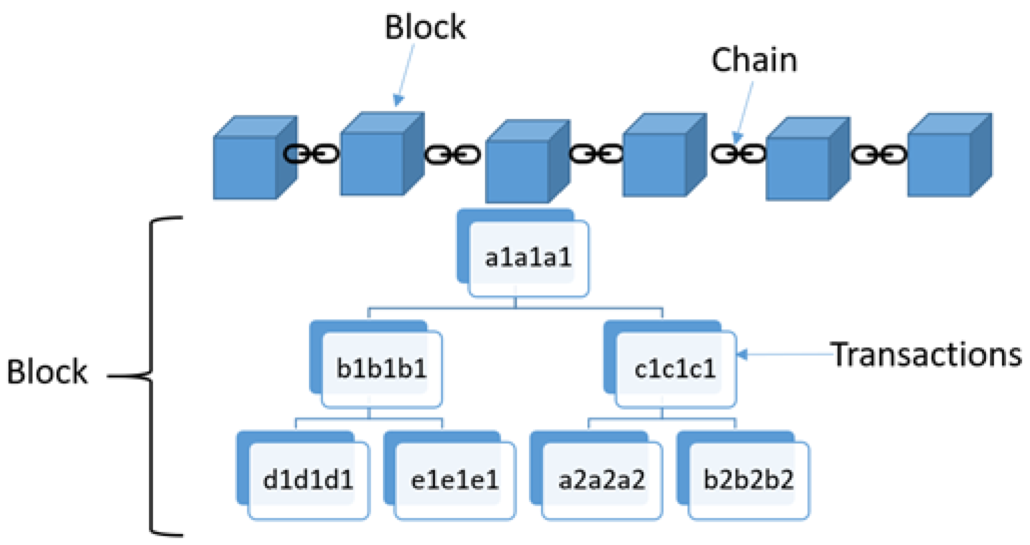
In Blockchain and Ethereum every block is related to another Block. There is a parent-child relationship between two blocks. There can be only one child to a parent and a child can have a single parent. This helps in forming a chain in Blockchain. Blocks are explained in later section in this chapter. In next image, three blocks are shown — Block 51, Block 52 and Block53. Block 51 is the parent of Block 52 and Block 52 is parent of Block 53. The relationship is established by storing the parent block's hash in child's block header. Block 52 stores the hash of Block 51 in its header and Block 53 stored the hash of Block 52 in its header. So, the question arises — Who is the parent of the first Block. Ethereum has a concept of Genesis Block also known as first block. This block is created automatically when the chain is first initiated. You can say that a chain is initiated with the first block known as Genesis block and the formation of this block is driven through Genesis.json file. Next chapter will show how to use Genesis.json file to create first block while initializing the Blockchain.



How are transactions and Blocks related to each other?

Now that we know that Blocks are related to each other, You would be interested in known how transactions are related to Blocks. Ethereum stores transactions within Blocks. Each block has a upper Gas limit and each transaction needs certain amount of Gas to be consumed as part of its execution. The cumulative gas from all transactions that are not yet written in ledger cannot surpass the Block Gas limit. This ensures that all transactions do not get stored within a single Block. As soon as the Gas limit is reached, other transaction is removed from block and mining begins thereafter.

The transactions are hashed and stored in the block. The hashes of two transactions are taken and hashes further to generate another hash. This process eventually provides a single hash from all transactions stored within the block. This hash is known as Transaction Merkle root hash and stored in Block's header. A change in any of a transaction will result in change in its hash and eventually change in root transaction hash. It will have cumulative effect because the hash of the block will change, and the child block has to change his hash because it stores its parent hash. This helps in making transactions immutable. This is also shown in next image.



Nodes

There are two types of nodes in Ethereum.

· Ethereum virtual machines and

· Mining Nodes

It is to be noted that this distinction is made to clarify concepts of Ethereum. In most scenarios' there is no dedicated EVM machines instead all nodes acts as miner as well as EVM node.

Ethereum virtual machines (EVM)

Think of EVM as the execution runtime for Ethereum network. EVM's are primarily responsible to provide a runtime that can execute code written in smart contracts. It can access accounts — contract and externally owned, its own storage data. It does not have access to ledger but has limited information about current transaction.

EVM are the execution component in Ethereum. The purpose of EVM is to execute the code in smart contract line by line. However, when a transaction is submitted, the transaction is not executed immediately instead is it pooled in a transaction pool. These transactions are not yet executed and not yet written to the Ethereum ledger. EVM nodes are similar to Mining nodes however they do not do mining.

Mining Nodes

A miner is responsible for writing transactions to the Ethereum chain. A miner job is very similar to that of an accountant. As an accountant is responsible for writing and maintain the ledger similarly, a Miner is solely responsible for writing transaction to Ethereum ledger. A miner is interested in writing transactions to ledger because of the reward associated with it. Miners get two types of reward — reward for writing a block to the chain and cumulative gas fees from all transactions in the block. There are generally many miners available within a blockchain network each trying and competing to write transactions. However only one miner can write the block to the ledger and rest will not able to write the current block and determination of a miner who will write the block happens using a challenge. The challange is given to every node and every miner tries to solve the puzzle using its compute power. The miner who solves the puzzle first write block containing transactions to ledger and also receiver 5 ether as reward. Every Mining node maintains its own instance of Ethereum ledger and the ledger is same ultimately across all miners. It is job of miners to ensure that their ledger is updated with latest blocks. There are three important functions performed by Miners or Mining Nodes.

· Mine or create a new block with transaction and write the same to Ethereum Ledger

· Advertise and send a newly mined block to other miners.

· To accept new blocks mined by other miners and keep its own ledger instance up-to-date

Mining Nodes refers the nodes that belong to Miners. These nodes are part of the same network where EVM is hosted. At some point of time, the miners would create a new Block, collect all transaction from transaction pool and adds them to the newly created block. Finally, this Block is added to the chain. There are additional concepts like consensus, solving of target puzzle before writing the block and will be explained in section “How mining works”.

How does mining works

The process of mining explained here is applicable to every miner on the network and every miner keeps executing tasks mentioned here regularly.

Miners are always looking forward to mine new block and are also listening actively to receive new blocks from other miners. As mentioned before, at some point, the miner collects all transactions from the transaction pool. This activity is done by all miners.

The miner constructs a new block and adds all transactions to it. Before adding these transaction, it will check if any of the transaction is not already written in a block that it might receive from other miners. If so, it will discard those transactions.

The miner will add his own coinbase transaction for getting rewards of mining the block.

The next task for miner is to generate the Block header and performs following task.

· The miner hashes all the transactions in the block, these hashes are further combined in pairs to generate a new hash. The process continues until there is just one hash for all transactions in the block. The hash is referred as Root transaction hash or Merkel Root transaction hash. This hash is added to the block header.

· The miner also identifies the hash of the previous block. The previous block will become parent to the current block and its hash will also be added to the block header.

· The miner in similar way calculates the State and Receipts transaction root hashes and add them to the block header.

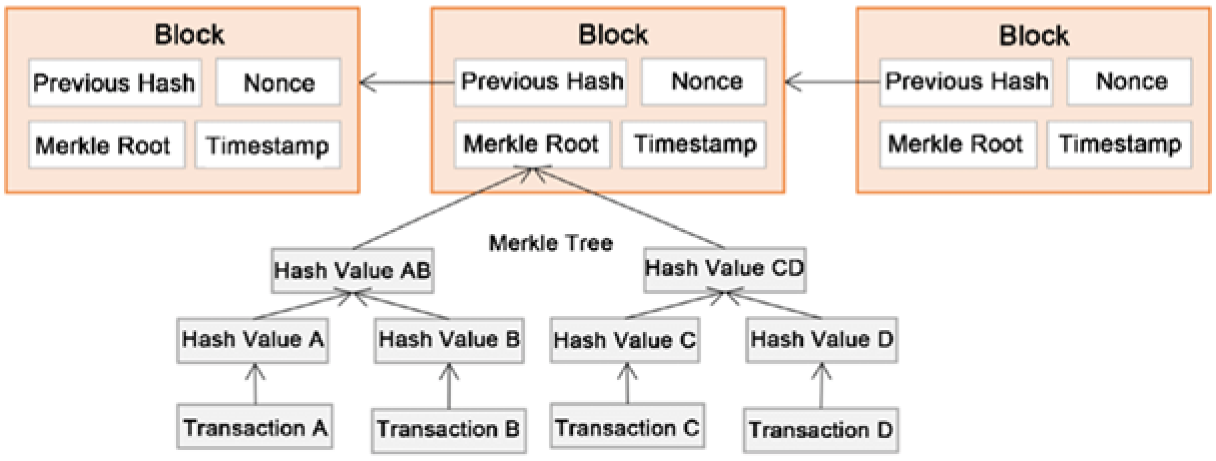
· A nonce and timestamp is also added to the block header.

The mining process starts where the miner keeps changing the nonce value and try to find a hash that will satisfy as an answer to the given puzzle. It is to be kept in mind that everything mentioned here is executed by every miner in the network.

Eventually, one of the miner would be able to solve the puzzle and advertise the same to other miners in the network. The other miners would verify the answer and if found correct would further verify every transaction while accept the block and append the same to their ledger instance.

This entire process is also known as Proof of Work wherein a miner provides proof that is has worked on computing the final answer that could satisfy as solution to the puzzle. There are other algorithms like Proof of Stake and Proof of authority and they are not used or discussed in this book.

The block header and its content is shown in next image.



Accounts

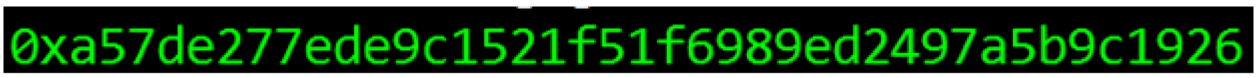
Accounts are main building block for Ethereum ecosystem. It is the interaction between accounts that Ethereum wants to store as transaction in its ledger. Ethereum supports two types of accounts. Each account has a balance property that returns the current value stored in it.

Externally owned accounts

Externally owned accounts are accounts that are owned by people on Ethereum. Accounts are not referred by names in Ethereum. When an externally owned account is created on Ethereum by an individual, a public-private key is generated. The private key is kept safe with the individual while the public key becomes the identity of this externally owned account. This public key is generally of 256 characters however Ethereum uses the first 160 characters to represent the identity of an account.

If Bob creates an account on Ethereum network — whether private or public, he will have his private key available to himself while his first 160 characters of public key will become his identity. Other accounts on the network can then send ether or other crypto-currencies based on ether to this account.

An account on Ethereum looks like the one shown here.


0xa57de277ede9c1521f51f6989ed2497a5b9c1926

An externally owned account can hold Ether in its balance and do not have any code associated with them. This of these accounts like a Bank account. They can execute transactions with other externally owned accounts and they can also execute transactions by invoking functions within contracts.

Contracts Accounts

Contracts accounts are very similar to externally owned accounts. They are identified using their public address. They do not have any private key. They can hold ether similar to externally owned accounts however they contain code — code for smart contracts consisting of functions are state variables.

Transaction

A transaction is an agreement between a buyer and seller, a supplier and a consumer or a provider and a consumer that there would be exchange of assets, products or services in lieu of currency, crypto-currency or some other asset either in present or in future. Ethereum helps in executing transaction. There are four types of transaction that can be executed in Ethereum.

1. Transfer of Ether from one account to another. The accounts can be externally owned accounts or contract account. Following are the possible cases

a. An externally owned account sending ether to another externally owned account in a transaction.

b. An externally owned account sending ether to a contract account in a transaction.

c. A contract account sending ether to another contract account in a transaction.

d. A contract account sending ether to an externally owned account in a transaction

2. Deployment of Smart contract — An externally owned account can deploy a contract using a transaction in Ethereum virtual machine.

3. Using or invoking a function within a contract — Executing a function in a contract that changes state are considered as transactions in Ethereum. If executing a function does not change state, it does not require a transaction

A transaction has some important properties related to it.

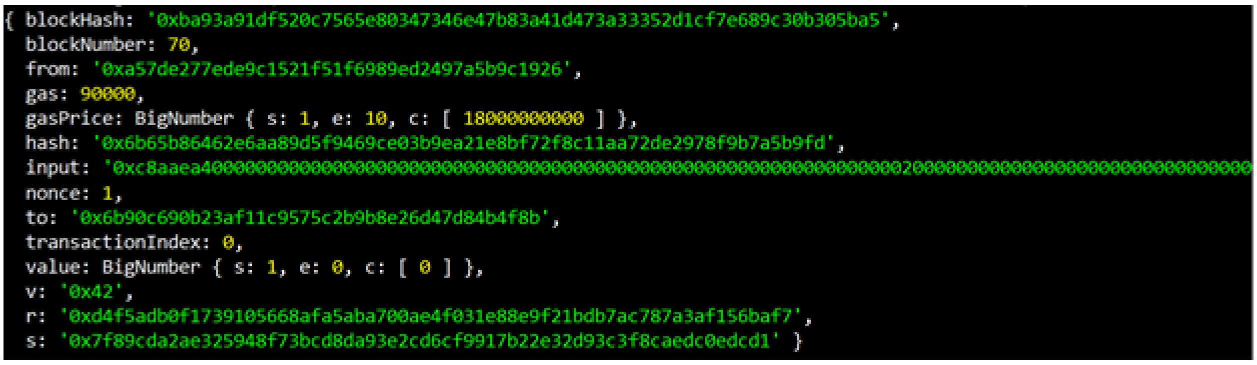
From Account property denotes the account that is originating the transaction and represents an account who is ready to send some gas or ether. We will consider concepts related to Gas and Ether later section in this chapter. From account can be externally owned or a contract account.

To account property refers to an account that is receiving ethers or benefits in lieu of an exchange. In case of transaction related to deployment of contract, the To field is empty. It can be externally owned or a contract account.

Value refers to the amount of ether that is transferred from one account to another.

Input refers to the compiled contract bytecode and is used during contract deployment in EVM. It is also used for storing data related to smart contract function calls along with its parameters.

A typical transaction in Ethereum where a contract function is invoked is shown here. Notice the input field containing the function call to contract along with its parameters.



```
{ blockHash: '0xba93a91df520c7565e80347346e47b83a41d473a33352d1cf7e689c30b305ba5',
  blockNumber: 70,
  from: '0xa57de277ede9c1521f51f6989ed2497a5b9c1926',
  gas: 90000,
  gasPrice: BigNumber { s: 1, e: 10, c: [ 18000000000 ] },
  hash: '0x6b65b86462e6aa89d5f9469ce03b9ea21e8bf72f8c11aa72de2978f9b7a5b9fd',
  input: '0xc8aaea400000000000000000000000000000000000000000000000000000000000000020000000000000000000000000000000
  nonce: 1,
  to: '0x6b90c690b23af11c9575c2b9b8e26d47d84b4f8b',
  transactionIndex: 0,
  value: BigNumber { s: 1, e: 0, c: [ 0 ] },
  v: '0x42',
  r: '0xd4f5adb0f1739105668afa5aba700ae4f031e88e9f21bdb7ac787a3af156baf7',
  s: '0x7f89cda2ae325948f73bcd8da93e2cd6cf9917b22e32d93c3f8caedc0edcd1' }
```

Blockhash refers to the hash of Block to which this transaction belongs to.

BlockNumber is the block in which this transaction belongs to.

Gas refers to amount of gas supplied by sender which executing this transaction

GasPrice refers to the price per gas the sender was willing to pay in Wei (Wei is explained in section related to Ether). Total Gas is computed at Gas units * Gas price

Hash refers to the hash of transaction

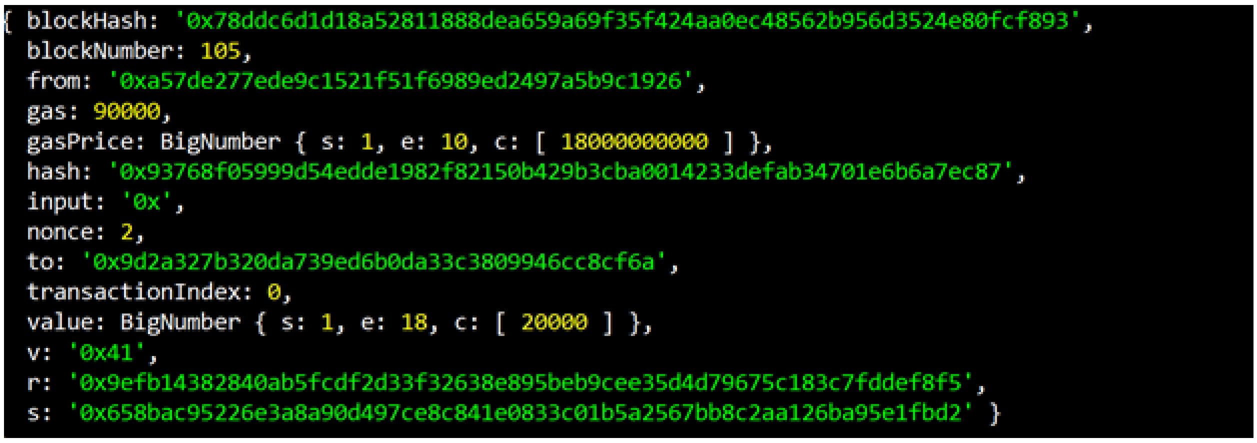
Nonce refers to number of transactions made by the sender prior to current transaction.

TransactionIndex refers to the serial number of current transaction in block

Value refers to amount of ether transferred in Wei

V, R and S relates to digital signature and signing of transaction.

A typical transaction in Ethereum where an externally owned account sends some ether to another externally owned account is shown here. Notice the input field is not used here. Since 2 ethers were send in transaction, the value field is showing the value accordingly in Wei.



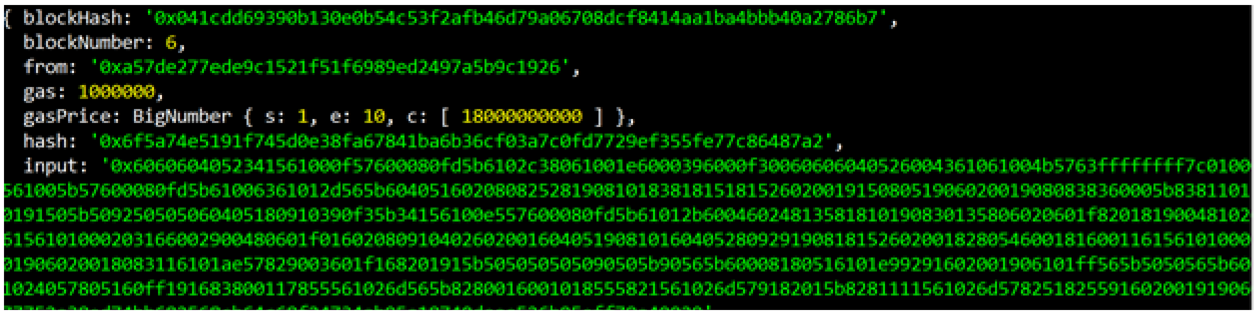
```
{ blockHash: '0x78ddc6d1d18a52811888dea659a69f35f424aa0ec48562b956d3524e80fcf893',
  blockNumber: 105,
  from: '0xa57de277ede9c1521f51f6989ed2497a5b9c1926',
  gas: 90000,
  gasPrice: BigNumber { s: 1, e: 10, c: [ 18000000000 ] },
  hash: '0x93768f05999d54edde1982f82150b429b3cba0014233defab34701e6b6a7ec87',
  input: '0x',
  nonce: 2,
  to: '0x9d2a327b320da739ed6b0da33c3809946cc8cf6a',
  transactionIndex: 0,
  value: BigNumber { s: 1, e: 18, c: [ 20000 ] },
  v: '0x41',
  r: '0x9efb14382840ab5fcdf2d33f32638e895beb9cee35d4d79675c183c7fddef8f5',
  s: '0x658bac95226e3a8a90d497ce8c841e0833c01b5a2567bb8c2aa126ba95e1fbd2' }
```

One of the way to send ether from an externally owned account to another externally owned account is shown here using web3 which will be covered extensively in this book in rest of the chapters.

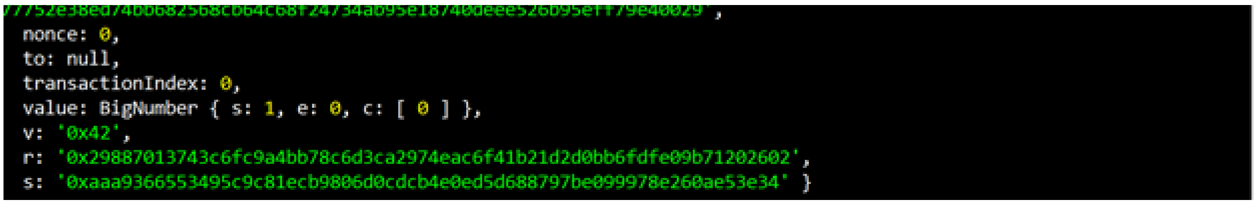
web.eth.sendTransaction({from: web.eth.accounts[0], to: “0x9d2a327b320da739ed6b0da33c3809946cc8cf6a”, value: web.toWei(2, ‘ether’)})

A typical transaction in Ethereum where a contract is deployed is shown here.

Notice the input field containing the bytecode of contract.



```
{ blockHash: '0x041cdd69390b130e0b54c53f2afb46d79a06708dcf8414aa1ba4bbb40a2786b7',
  blockNumber: 6,
  from: '0xa57de277ede9c1521f51f6989ed2497a5b9c1926',
  gas: 1000000,
  gasPrice: BigNumber { s: 1, e: 10, c: [ 18000000000 ] },
  hash: '0x6f5a74e5191f745d0e38fa67841ba6b36cf03a7c0fd7729ef355fe77c86487a2',
  input: '0x6060604052341561000f57600080fd5b6102c38061001e6000396000f3006060604052600436106100 4b5763ffffffff7c0100
561005b57600080fd5b61006361012d565b60405160208082528190810183818151815260200191508051906020019080838360005b8381101
0191505b509250505050604051809103 90f35b34156100e557600080fd5b61012b6004602481358181019083013580602060 1f82018190048102
6156101000203166002900480601f0160208091040260200160405190810160405280929190818152602001828054600181600116156101000
01906020018083116101ae57829003601f168201915b505050505090505b90565b6000818051610 1e99291602001906101ff565b5050565b60
1024057805160ff19168380011785556102 6d565b828001600101855582156102 6d579182015b82811115610 26d57825182559160200191906
```

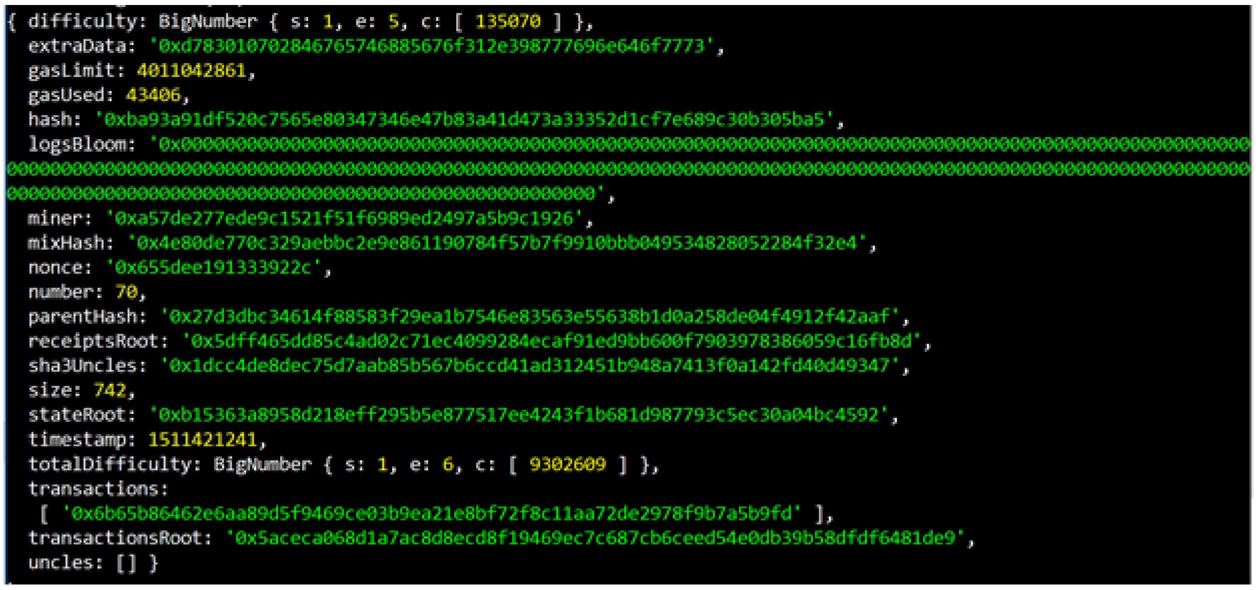


```
77752e38ed74bb682568cb64c68f24734ab95e18740deee526b95eff79e40029',
  nonce: 0,
  to: null,
  transactionIndex: 0,
  value: BigNumber { s: 1, e: 0, c: [ 0 ] },
  v: '0x42',
  r: '0x29887013743c6fc9a4bb78c6d3ca2974eac6f41b21d2d0bb6fdfe09b71202602',
  s: '0xaaa9366553495c9c81ecb9806d0cdcb4e0ed5d688797be099978e260ae53e34' }
```

Block

Block is an important concept in Ethereum. Block are containers for transaction. A block contains multiple transactions. Each block has different number of transactions based on Gas limit. Gas limit will be explained in detail in later sections. The blocks are chained together to form blockchain. Each Block has a parent and it stored the hash of parent block in its header. Only the first Block known as Genesis block does not have a parent.

A typical Block data in Ethereum is shown here.



```
{ difficulty: BigNumber { s: 1, e: 5, c: [ 135070 ] },
  extraData: '0xd783010702846765746885676f312e398777696e646f7773',
  gasLimit: 4011042861,
  gasUsed: 43406,
  hash: '0xba93a91df520c7565e80347346e47b83a41d473a33352d1cf7e689c30b305ba5',
  logsBloom: '0x00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000',
  miner: '0xa57de277ede9c1521f51f6989ed2497a5b9c1926',
  mixHash: '0x4e80de770c329aebbc2e9e861190784f57b7f9910bbb049534828052284f32e4',
  nonce: '0x655dee191333922c',
  number: 70,
  parentHash: '0x27d3dbc34614f88583f29ea1b7546e83563e55638b1d0a258de04f4912f42aaf',
  receiptsRoot: '0x5dff465dd85c4ad02c71ec4099284ecaf91ed9bb600f7903978386059c16fb8d',
  sha3Uncles: '0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347',
  size: 742,
  stateRoot: '0xb15363a8958d218eff295b5e877517ee4243f1b681d987793c5ec30a04bc4592',
  timestamp: 1511421241,
  totalDifficulty: BigNumber { s: 1, e: 6, c: [ 9302609 ] },
  transactions:
   [ '0x6b65b86462e6aa89d5f9469ce03b9ea21e8bf72f8c11aa72de2978f9b7a5b9fd' ],
  transactionsRoot: '0x5aceca068d1a7ac8d8ecd8f19469ec7c687cb6ceed54e0db39b58dfdf6481de9',
  uncles: [] }
```

The **Difficulty** property determines the complexity of the puzzle/challenge given to miners for this block.

GasLimit determines the maximum gas allowed. This helps in determining how many transaction can be part of the block.

GasUsed refers to the actual gas used for this block for executing all transactions in it.

Hash refers to the hash of the block.

Nonce refers to the number that helping in solving the challenge.

Miner property is the account identifier of miner also known as coinbase or Etherbase

Number is the sequential number of this block on the chain

ParentHash refers to parents blocks hash.

ReceiptsRoot, StateRoot and transactionRoot refers to Merkel trees discussed during mining process.

Transactions refers to array of transactions that are part of this block.

TotalDifficulty refers to total difficulty of the chain until this block

An end to end Transaction

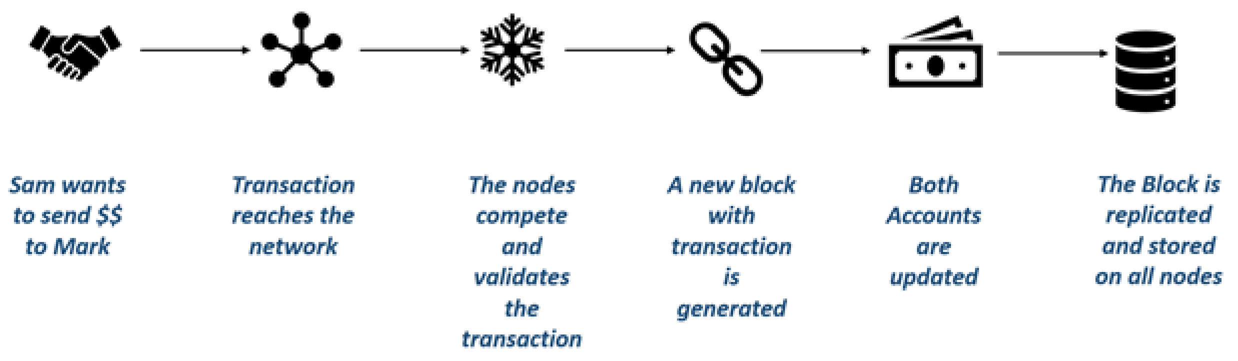
Armed with the understanding of foundational concepts of Blockchain and Ethereum, it time to see a complete end to end transaction and how it flows through multiple components and get stored in the ledger.

In this example, Sam wants to send few dollars to Mark. Sam generates a transaction message as shown before containing From, To, Value fields and sends it across to Ethereum network. The transaction is not written to the ledger immediately and instead it is placed in a transaction pool.

The mining node creates a New Block and takes all transactions from the pool honouring the Gas limit criteria and adds them to the block. This activity is done by all miners on the network. Sam's transaction will also be part of this process.

The miners compete trying to solve the challenge thrown to them. The winner is a miner who can solve the challenge first. After a period (10 seconds in Ethereum) one of the miner will advertise that he has found solution to the challenge and that he is the winner and should write the block to the chain. The winner sends the challenge solution along with the new Block to all other miners. Rest of the miners validates and verifies the solution and once satisfied with the solution, they accept the new Block containing Sam's transaction to append in their instance of ledger. This generates a new block on the chain that is persisted across time and space. During this time the accounts of both the parties are updated with new balance. Finally, the Block is replicate across every node in the network.

The same process is also shown in next image.


Sam wants to send $$ to Mark
Transaction reaches the network
The nodes compete and validates the transaction
A new block with transaction is generated
Both Accounts are updated
The Block is replicated and stored on all nodes

What is a contract

A contract is a legal document that binds two or more parties who agrees to execute a transaction immediately or in future. Since Contracts are legal documents, they are enforced and implemented by law. Examples of contract are an individual getting into a contract with Insurance company for covering his health or an individual buying a piece of land from another individual or a company selling its shares to another company.

What is a smart contract

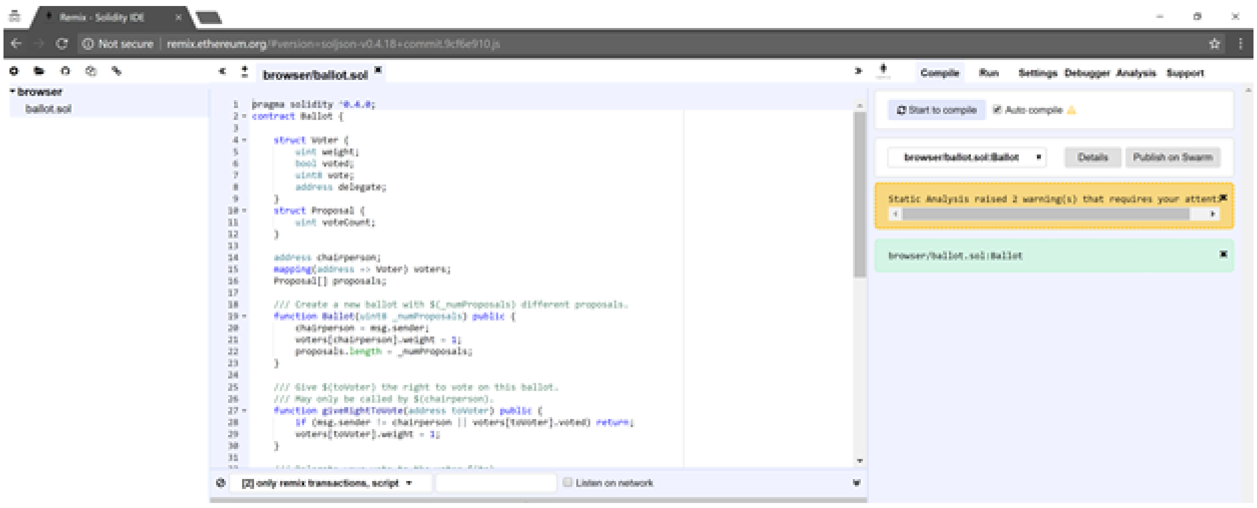
A smart contract is a contract implemented, deployed and executed within Ethereum environment. Smart contracts are digitization of the legal contracts. Smart contracts are deployed, stored and executed within the Ethereum Virtual machine. Smart contracts can store data. The data stored can be used to record information, fact, associations, balances and any other information needed to implement logic for real world contracts. Smart contracts are very similar to Object oriented classes. A smart contract can call another smart contract just like an Object-oriented object to create and use objects of another class. Think of smart contract as a small program consisting of functions. You can create an instance of the contract and invoke functions to view and update contract data along with execution of some logic

How to write Smart Contracts

There are multiple smart Contract authoring tools including Visual Studio. However, the easiest and fastest way to develop smart contracts is to use a browser based tool known as Remix. Remix is available at http://remix.ethereum.org. Remix is a new name and was earlier known as browser solidity. Remix provides a rich integrated development environment on browser for authoring, developing, deployment and troubleshooting of contracts written using solidity language. All Contract management related activities like authoring, deployment and troubleshooting can be performed from the same environment without moving to other windows or tabs.

Since Remix is on internet and not everybody would be comfortable to author their smart contracts, Remix is an open source tool that can be downloaded from GitHub at https://github.com/ethereum/browser-solidity and compiled to run a private version sing browser. Another advantage of running remix locally is that it can connect to local private chain networks directly otherwise users will first have to author the contact at online Remix and then copy the same in a file and compile and deploy manually to private network.

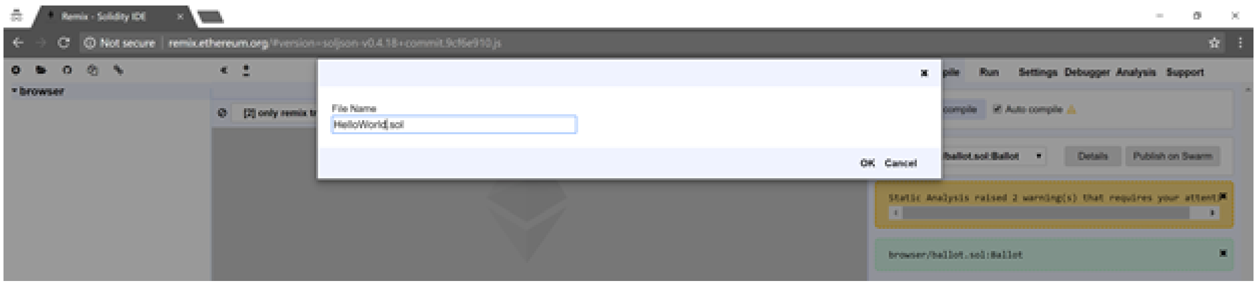
Navigate to remix.ethereum.org and the site will open in browser with a default contract as shown in next figure. This contract can be deleted. Delete this contract as we won't be using it.



```
pragma solidity ^0.4.0;
contract Ballot {

    struct Voter {
        uint weight;
        bool voted;
        uint8 vote;
        address delegate;
    }
    struct Proposal {
        uint voteCount;
    }

    address chairperson;
    mapping(address => Voter) voters;
    Proposal[] proposals;

    /// Create a new ballot with $(_numProposals) different proposals.
    function Ballot(uint8 _numProposals) public {
        chairperson = msg.sender;
        voters[chairperson].weight = 1;
        proposals.length = _numProposals;
    }

    /// Give $(toVoter) the right to vote on this ballot.
    /// May only be called by $(chairperson).
    function giveRightToVote(address toVoter) public {
        if (msg.sender != chairperson || voters[toVoter].voted) return;
        voters[toVoter].weight = 1;
    }

```

```
Static Analysis raised 2 warning(s) that requires your attent
browser/ballot.sol:Ballot
```

The first thing we need to do is to create a new contract by selecting "+" from Remix left menu bar. Then provide a name for new solidity file that has an extension .sol. Name the contract "HelloWorld" and click OK to continue. This will create a blank contract.



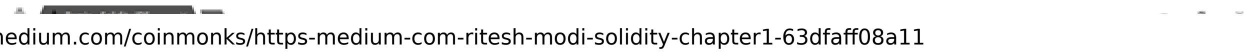
```
Static Analysis raised 2 warning(s) that requires your attent
browser/ballot.sol:Ballot
```

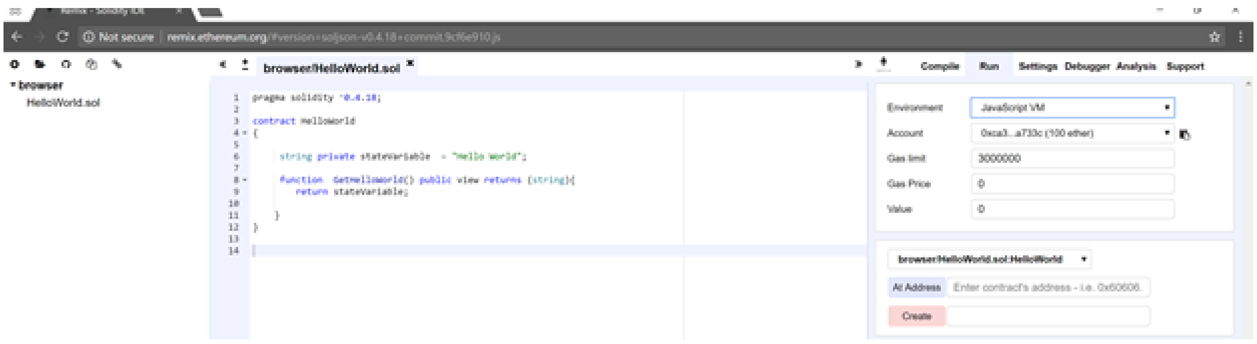
Type the following code in the empty authoring pane to create your first contract. This contract will be explained in detail in third chapter. For now, it is sufficient to understand that contract is created using contract keyword, you can declare global state variables and functions and contracts are saved with .sol file extension. The HelloWorld contracts returns “HelloWorld” string when GetHelloWorld function is called.

```
pragma solidity ^0.4.18;

contract HelloWorld

{

string private stateVariable = “Hello World”;

function GetHelloWorld() public view returns (string){

return stateVariable;

}

}
```

Look at the action window to the right of Remix. It has got several tabs — Compile, Run, Settings, debugger, analysis and support. The action tabs helps in compiling, deploying, troubleshooting and invoking contracts. The compile tab compiles the contract into bytecode — code that is understood by Ethereum. It displays warnings and errors as you author and edit the contract. These warnings and errors are to be taken seriously and they really help in creating robust contracts.

Run tab is the place where you would spend maximum time apart from writing the contract. Remix comes bundled with Ethereum runtime within the browser. Run tab allows to deploy the contract to this runtime using “Javascript VM” environment option. “Injected Web3” is used along with tools like Mist and MetaMask which will be covered in next chapter and “Web3 provider” can be used when using Remix in a local environment connecting to private network. In our case for this chapter, the default, “Javascript VM” is sufficient. Rest of the options will be discussed later in chapter 3. However the important action is deployment of contract and that can be done using the “Create” button.

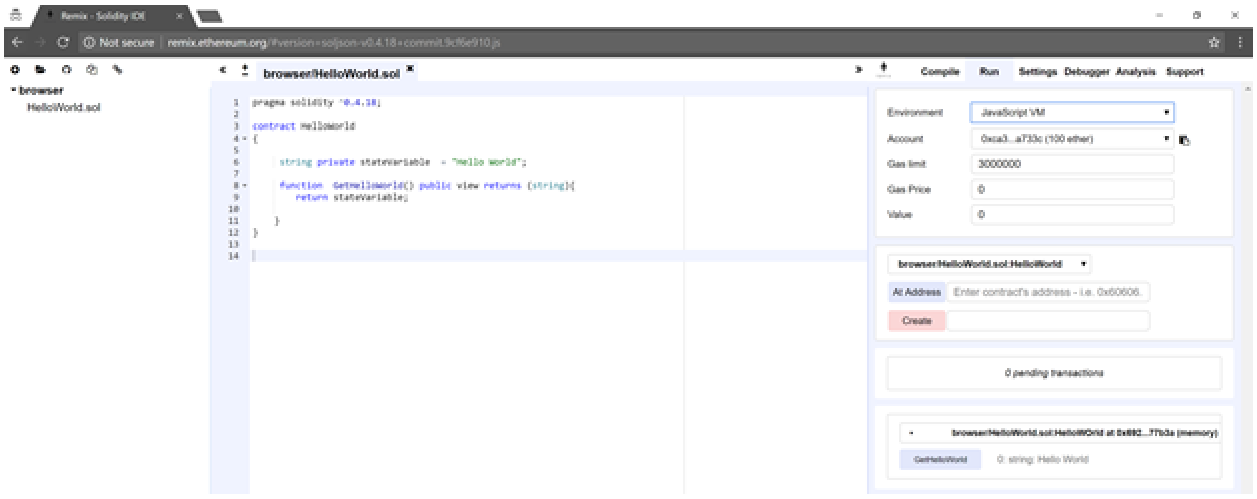




```
pragma solidity ^0.4.18;

contract HelloWorld
{

    string private stateVariable  = "Hello World";

    function  GetHelloWorld() public view returns (string){
       return stateVariable;

    }
}
```

Environment: JavaScript VM
Account: 0xca3...a733c (100 ether)
Gas limit: 3000000
Gas Price: 0
Value: 0
browser/HelloWorld.sol:HelloWorld
Enter contract's address - i.e. 0x60606...

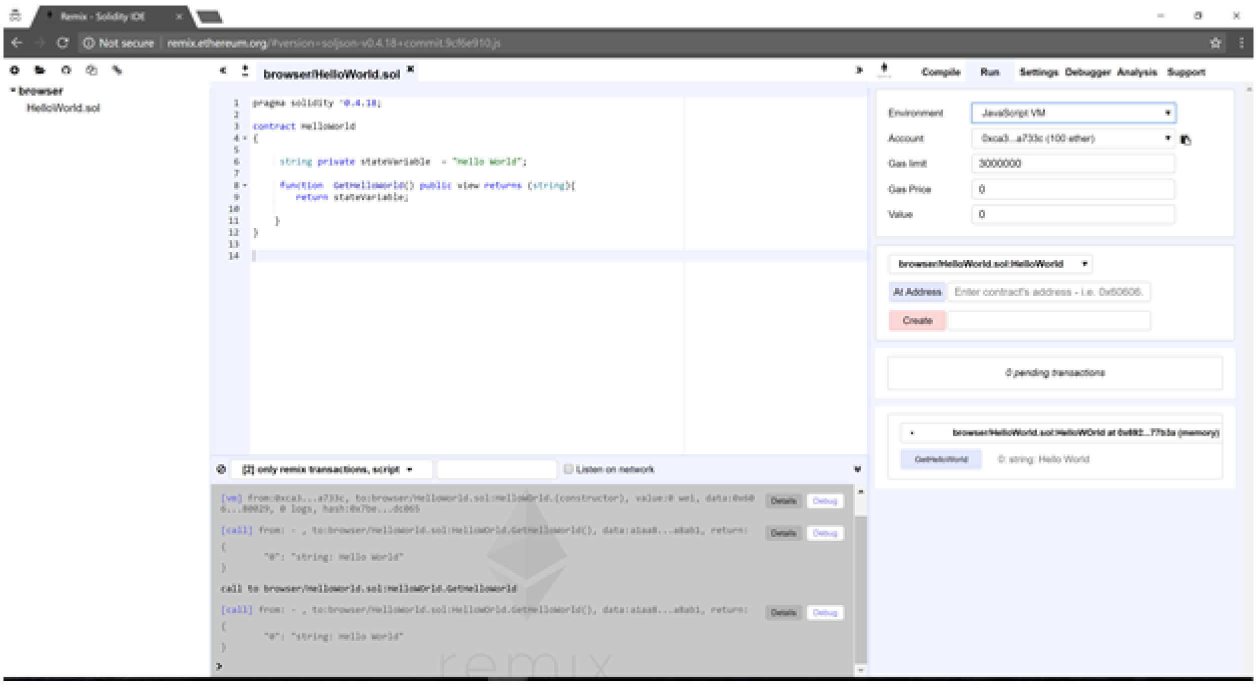
Click on create button to deploy the contract to browser Ethereum runtime and this will list all the functions available within the contract below the create button. Since, we only had a single function GetHelloWorld, the same is displayed.



```
pragma solidity ^0.4.18;

contract HelloWorld
{

    string private stateVariable  = "Hello World";

    function  GetHelloWorld() public view returns (string){
       return stateVariable;

    }
}
```

Environment: JavaScript VM
Account: 0xca3...a733c (100 ether)
Gas limit: 3000000
Gas Price: 0
Value: 0
browser/HelloWorld.sol:HelloWorld
Enter contract's address - i.e. 0x60606...
0 pending transactions
browser/HelloWorld.sol:HelloWOrld at 0x692...77b3a (memory)
0: string: Hello World

Click on GetHelloWorld button to invoke and execute the function. The lower pane of Remix will show the results of execution.



```
pragma solidity ^0.4.18;

contract HelloWorld
{

    string private stateVariable  = "Hello World";

    function  GetHelloWorld() public view returns (string){
       return stateVariable;

    }
}
```

Environment: JavaScript VM
Account: 0xca3...a733c (100 ether)
Gas limit: 3000000
Gas Price: 0
Value: 0
browser/HelloWorld.sol:HelloWorld
Enter contract's address - i.e. 0x60606...
0 pending transactions
browser/HelloWorld.sol:HelloWOrld at 0x692...77b3a (memory)
0: string: Hello World

```
[vm] from:0xca3...a733c, to:browser/HelloWorld.sol:HelloWOrld.(constructor), value:0 wei, data:0x60
6...80029, 0 logs, hash:0x7be...dc065
[call] from: - , to:browser/HelloWorld.sol:HelloWOrld.GetHelloWorld(), data:a1aa8...a8ab1, return:
{
        "0": "string: Hello World"
}
call to browser/HelloWorld.sol:HelloWOrld.GetHelloWorld
[call] from: - , to:browser/HelloWorld.sol:HelloWOrld.GetHelloWorld(), data:a1aa8...a8ab1, return:
{
        "0": "string: Hello World"
}
>
```

Congratulations, you have created, deployed and also executed a function on your first contract. The code for HelloWorld contract is accompanied with this chapter and can be used in Remix if you are not interested in typing the contract.

How the contracts are deployed

Remix makes deployment of contracts a breeze however it is performing a lot of steps behind the scenes. It is always useful to understand the process of deploying contracts to have more finer control over deployment process.

The first step is compilation of contracts. The compilation is done using solidity compiler. Next chapter will show how to download and compile a contract using solidity compiler.

The compiler generates two major artifacts

· **ABI definition and**

· **Contracts byte code**

Think of ABI (application binary interface) as an interface consisting of all external and public function declarations along with their parameters and return types. The ABI defines the contract and any caller wanting to invoke any contract function can use the ABI to do so.

The bytecode is what represents the contract and is deployed in Ethereum ecosystem. The bytecode is needed during deployment and ABI is needed for invoking functions in contract.

A new instance of contract is created using the ABI definition.

A new transaction is created that utilizes the newly instance of contract passing in the contract bytecode and appropriate quantity of gas for execution of transaction. After the transaction is mined, the contract is available at an address determined by Ethereum.

Now, that contract is deployed within the Ethereum virtual machine, its time to invoke the contract.

Using the newly generated address, an instance of contracts can be created and its functions can be invoked.

Summary

This was the first chapter of the book and we have already covered lot of ground. This chapter was an introduction to Blockchain and more specific to Ethereum. Having a good understanding of the big picture about how Blockchain and Ethereum works will go a long way in understanding and Writing robust, secure and cost effective smart contracts using solidity. This chapter covered the basics of blockchain, explained what is Blockchain, why is Blockchain important and how does it help in building decentralized, distributed applications. The architecture of Ethereum was discussed in brief along with some of the important concepts like Transactions, Blocks, Gas, Ether, Accounts, cryptography and Mining. This chapter also touched briefly on the topic of Smart contracts, using Remix to author smart contracts and how to execute them using Remix itself. I've kept this chapter brief since the rest of the book will explain these concepts further and allow you to quickly develop Solidity based smart contracts. You'll notice that this chapter does not contain any mention of Ethereum tools and utilities. This is what we will cover in the next chapter, by diving straight in and installing Ethereum and its toolset.

Coinmonks

Blockchain Ethereum Solidity Learning Smart Contracts