P2P Systems and Blockchains

Spring 2021,
instructor: Laura Ricci
laura.ricci@unipi.it

Lesson 16:
ETHEREUM ACCOUNTS, TRANSACTIONS, GAS

21/04/2021
OUTLINE

- the idea of smart contracts
- Ethereum accounts
- Ethereum transactions
- the concept of gas
- programming smart contracts: Solidity
- Ethereum mining and data structures
A **smart contract** is a computerized transaction protocol that executes the terms of a contract. The general objectives are to satisfy common contractual conditions (such as payment terms, liens, confidentiality, and even enforcement), minimize exceptions both malicious and accidental, and minimize the need for trusted intermediaries. Related economic goals include lowering fraud loss, arbitrations and enforcement costs, and other transaction costs.

THE ORIGINAL IDEA OF SMART CONTRACTS

- smart contract
  - some code automating the “if this happens then do that” part of traditional contracts.

- what is better with respect to normal contract?
  - computer code behaves in expected ways and doesn’t have the linguistic nuances of human languages.
SMART CONTRACTS AND DISTRIBUTED LEDGERS

- contract
  - formalizes a relationship and contains promises made between principals.

- smart contract
  - based on the translation of contractual clauses into code
  - more functional compared to paper-based
    - can reduce costs
  - aim to remove the need for trusted intermediaries
    - make it more difficult for malicious parties to undermine compliance with the contract terms
  - uses cryptography and other security mechanisms
    - secure algorithmically specifiable relationships from being breached and ensure the agreed upon terms are satisfied.
SMART CONTRACTS: AN EXAMPLE

- A bank account may behave like a smart contract
  - My bank account has a balance
  - Every month, I commission an automated payment that deducts a fixed amount and sends it to my landlady
  - If there is not enough money in my bank account, the payment fails
    - I get fined, and another procedure is triggered.

- These are the instructions I have set up which are associated with the account
  - A contract between me and the bank.
  - Can be easily enforced by a computer
  - But.....
**SMART CONTRACTS: AN EXAMPLE**

- my bank is the ultimate guardian of my bank account and has complete control
  - can arbitrarily add money to my account (unlikely) or subtract it
  - even if coded, it sits on one computer and is executed by one party (the bank) which must be trusted

- idea: making contracts decentralized
  - no centralized trusted authority
  - put the contract on the blockchain
    - consensus of the nodes on the results of the execution of the contract
    - a distributed state machine

- extend what Bitcoin
  - does for *distributed data storage*
  - to a blockchain supporting *distributed data storage + computations*
SMART CONTRACTS IN BITCOIN

- use scripting language of Bitcoin to write smart contracts?

- the scripting language has several limitations:
  - not Turing-complete (e.g., no loops)
  - lack of arbitrary state variables:
    - scripts on Bitcoins cannot hold internal state
    - a script consumes its input to produce the output
  - blockchain-blindness: cannot access block header values such as nonce, timestamp and previous hash block

- Ethereum is based on the possibility of extending Bitcoin scripts and defining real smart contracts.
Thanks to this guy!

Vitalik Buterin (January 31th 1994)
ETHEREUM IN A NUTSHELL

- a blockchain platform for building decentralized applications
  - application code and state is stored on a blockchain
  - transactions cause code execution and update state, emit events, and write logs
  - frontend web interfaces can respond to events and read logs
- most popular platform for creating new tokens (ICO)
  - each ICO implements an ERC-20 token contract
    - investments in ICOs was about $7 billion in 2017
    - about $12 billion in 2018
ETHEREUM APPLICATIONS

• not just money!
  • asset issuance
  • crowdfunding
  • domain registration
  • title registration, gambling
  • prediction markets
  • internet of things
  • voting
  • hundreds of applications!

• some examples
  • Ethereum Name Service (https://ens.domains/)
  • Cryptokitties (https://www.cryptokitties.co/)
  • Decentralized exchanges (https://idex.market)
ETHEREUM IN A NUTSHELL

- Ethereum: **smart contracts** that are
  - decentralized
  - replicated
  - processed on all the nodes on the network, without a central coordinator
  - consensus mechanisms assure that all the nodes agree on the results of the execution and update the state in the same way
  - each node changes its own version of the ledger with the result of the smart contract evaluation

- like a distributed computer to execute code

- a distributed state machine:
  - global state: status of all the smart contracts
  - transactions change global state
ETHEREUM SMART CONTRACTS

transparency

- all participants in a blockchain run the same code, each verifying the other
  - smart contract must be deterministic
- the logic of the smart contract is visible to all
- privacy may be an issue
  - solutions based on zero-knowledge proofs may be used in some cases

flexibility with respect to Bitcoin's scripts

- smart contract are written in a “Turing complete” language
- can do anything that a normal computer can do
- but you need to pay for all nodes on the network to run the code in parallel.
  - nodes must be rewarded for executing smart contracts
  - pay for the execution cost
Ethereum’s Brief History

- November 2013: Whitepaper
- April 2014: Yellowpaper
- July 2015: Frontier
- March 2016: Homestead
- June 2016: The DAO attack
- February 2019: Constantinople & St. Petersburg
- October 2017: Byzantium Hard Fork
- May 2019 - Present: Istanbul & Serenity
WHERE WE WERE: BITCOIN STATE MACHINE

- Bitocoin's state stored in UTXO
- user’s available balance: sum of his/her unspent transaction outputs
- UXT0 unspent transaction outputs: Bitcoin implicit state

![State transition diagram]

**State**
- 8f762bb1:0
- d2e712ac:2
- 35ce3fa2:0

**Transaction**
- Spend: 6c38ff21:1
- Sig: 3fa45ee178ff 41c240ee51ee
- Create: dd57e801:0  dd57e801:1  dd57e801:2

**State'**
- 8f762bb1:0
- 91bcf43f:0
- 35ce3fa2:0
- dd57e801:0
- dd57e801:1
- dd57e801:2
ETHEREUM STATE MACHINE

- like Bitcoin is a **transaction-based deterministic** state machine
  - a virtual machine that applies changes to a global, replicated state
- like in Bitcoin transactions, in Ethereum, trigger state change
  - but anyone can create its **own state transition functions**
  - not limited to scripts
  - smart contracts
- different from Bitcoin, Ethereum uses **accounts**
  - like a bank account, keeps track of balance
- the global shared state of Ethereum is stored
  - in a multitude of accounts
  - small objects interacting with one another through a message-passing paradigm
ETHEREUM COMPARED TO BITCOIN

• a public and permissionless blockchain
  • addresses are generated by keys
  • transactions are signed through a digital signatures
  • in Ethereum blocks contain data and smart contracts, in Bitcoin blocks contain data and scripts

• consensus algorithm
  • currently Proof of Work
  • Proof of Stake proposed for Casper, we are still waiting....

• a native cryptocurrency: ether (ETH)

• deployed on a peer to peer network
  • Ethereum uses Kademlia at the P2P level to discover peers
ETHEREUM ACCOUNTS

• each account has an identifier of 20 byte/160 bit + a state

• two types of accounts
  • externally owned
    • controlled by private keys
    • have no code associated with them.

• contract accounts
  • have code associated with them
  • controlled by the associated code.
EXTERNALLY OWNED ACCOUNTS

• “personal accounts” owned by some external entity, a person or an organization
  • controlled by private keys (users)
    • having the private key allows access to funds and to contracts invocation
• hold
  • address
  • Ether balance
  • nonce: total transactions emitted from that account
• can
  • send transactions to transfer Ether
  • send transactions to trigger a smart contract
EOA TO EOA TRANSACTIONS

- transfer money, like Bitcoin transaction
- account based, not UXTO based
EOA TO EOA TRANSACTION FORMAT

(amount transferred by the transaction in wei)

receiver of the transaction

digital signature created by the sender's private key

(amount transferred by the transaction in wei)

receiver of the transaction

digital signature created by the sender's private key

(amount transferred by the transaction in wei)

receiver of the transaction

digital signature created by the sender's private key

(amount transferred by the transaction in wei)

receiver of the transaction

digital signature created by the sender's private key
(SMART) CONTRACT ACCOUNTS

Contract Account

- contain
  - contract code
  - persistent storage for contract variables
  - like EOAs, contract accounts have an Ether balance and can receive/transfer Ethers
  - nonce: number of messages sent from this account
  - do not have a private key
WHAT IS A SMART CONTRACT?

• a computer program
  • code of smart contract cannot change
  • executed by full nodes
• deterministic computation
  • the outcome of a smart contract must be the same for every node
• execution context of a smart contract
  • transaction context: data taken from the transaction that has activated the contract
  • internal storage
  • may take information from block headers of the blockchain
pragma solidity ^0.4.7;

contract Coin {
    address public minter;
    mapping (address => uint) public balances;

    event Sent(address from, address to, uint amount);

    constructor() public {
        minter = msg.sender;
    }

    function mint(address receiver, uint amount) public {
        if (msg.sender != minter) return;
        balances[receiver] += amount;
    }

    function send(address receiver, uint amount) public {
        if (balances[msg.sender] < amount) return;
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        emit Sent(msg.sender, receiver, amount);
    }
}
SMART CONTRACT LIFECYCLE

- Creation
- Interaction
- Destruction
SMART CONTRACT CREATION

External Account

Contract Account

• who creates a smart contract?
  • you need a EOA
  • the EOA may deploy smart contracts on the Ethereum blockchain
TRANSACTION TO CREATE CONTRACTS

(code of the smart contract)

empty: no address until the contract is deployed
EOA TO SMART CONTRACT INTERACTION

EOA calls a method of the smart contract through a transaction
CONTRACT TO CONTRACT INTERACTION

Call Transaction

Run your code with these arguments

Oh yes!

Call Transaction
TRANSACTION FOR CONTRACT INTERACTION

- Address of the contact
- Which method to call + parameters

Diagram:
- (other fields)
- Signature
- To
- Amount
- Data
smart contracts are triggered by transactions from EOA or from messages from other smart contracts which

- call functions inside a contract, specifying the address of the contract
- specify parameters of the function
- may contain Ether to transfer to the smart contract
SMART CONTRACT CHARACTERISTICS

- contract account cannot initiate a transaction by themselves
- when activated may call other contracts, but not themselves
- may build complex execution paths
- can generate messages as a reply to transactions they have received from
  - an EOA
  - from another contract account
SMART CONTRACT CHARACTERISTICS

- when receive a transaction/message, are executed by the Ethereum Virtual Machine (EVM)
- can perform different actions
  - computation
  - write to internal storage
  - send messages to another smart contract
  - create new contracts

Run your code with these arguments

Oh yes!
CALLING SMART CONTRACTS

Externally owned account → Transaction → Contract account

Internal transaction → Contract account

(contract code gets executed)

Externally owned Account → Transaction → Contract account

Internal transaction → Contract account

(contract code gets executed)

Internal transaction → Contract account

(contract code gets executed)
DESTROYING CONTRACTS

- Creation
- Interaction
- Destruction
TRANSACTION FOR CONTRACT DESTRUCTION

Contract Address

name of a method that calls the selfdestruct operation
**INTERACTING WITH ETHEREUM**

- any action occurring on the Ethereum blockchain is always set in motion by **transactions** fired from externally controlled accounts.

- **EOAs** are a bridge between the external world to the internal state of Ethereum.

- a **transaction** is a signed data package storing a message
  - serialized and sent from an EOA to another account
  - may trigger subsequent messages from contracts to other contracts
  - generates a change in the state of the blockchain if included in a block
  - used also to generate new contracts
<table>
<thead>
<tr>
<th></th>
<th>Externally owned account</th>
<th>Contract account</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identified by an address</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Holds the account’s Ether balance</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Hold contract code</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>Holds the account’s storage</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Associated private-public key</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>Can send signed transactions</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>Can create contracts</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Can send unsigned transactions</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>Holds a nonce</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

- Note that contract can send messages and create another contract(s)!
- A signed transaction is sent only by externally owned account and sending it will cost the sender.
MORE INSIGHTS IN ETHEREUM ACCOUNTS

- **nonce**
  - number of transactions sent/contracts created

- **balance**
  - wei owned

- **storageRoot**
  - digest of the Ethereum's state
  - hash of the root node of a Merkle Patricia trie

- **codeHash**
  - hash of the code of contract account account.
  - hash(“ ”) for external accounts.
WHAT IS A TRANSACTION NONCE?

- from the yellow paper:
  
  a scalar value equal to the number of transactions sent from this address or, in the case of accounts with associated code, the number of contract-creations made by this account

- an attribute of the address originating the transaction, not of the transaction itself

- records the order of the transactions
  
  - protection against transaction duplication and reply attacks
  
  - avoid double spending from the same address

- remark: two types of nonces in Ethereum
  
  - transaction nonce: not Bitcoin nonce!
  
  - block nonce: used by PoW: like Bitcoin nonce
THE REPLY ATTACK

- Bitcoin stores in the blockchain the unspent output and transactions consume those outputs
  - spent output no longer exist
  - only a transaction for each single UTXO
- Ethereum instead stores account balances
  - more transactions start for the same balance
- “replay attack”
  - the attacker takes any existing transaction and resends it on the network more and more times
  - all these replayed transactions
    - withdraw funds from the same account
    - are valid (signer from the owner of the account)
  - if the account has enough Ethers, the attack is successfully
THE REPLAY ATTACK

- Alice signs a transaction to send 10 ETH to Bob
- an Ethereum miner
  - picks the transaction,
  - writes it into a block
  - adjusts the balances associated to Alice and Bob
- Bob replies the transaction
  - sends again the same transaction in the network and repeatedly submits it
  - drains Alice's account doing this over and over until there are Ethers in Alice's account
THE REPLAY ATTACK

- Ethereum prevents this with the using the *account/transaction nonce*
  - suppose that the nonce is 22, when Alice sends her transaction
  - the network does not accept another transaction from Alice with nonce 22
  - Bob cannot change the nonce, because this invalidates the signature of the transaction
- each time an account sends a transaction, the nonce increases by 1.
- the nonce is used to enforce the rules to consider a transaction valid
  - transactions must be in order
    - a transaction with a nonce of 1 cannot be mined before one with a nonce of 0.
  - transaction cannot be skipped.
    - a transaction with a nonce of 2 cannot be mined if the miner has not already sent transactions with a nonce of 1 and 0.
TRANSACTION PROPAGATION

- Ethereum uses flooding to propagate the transactions.
- Transaction propagation starts with the node creating a signed transaction.
  - The transaction is validated by each node.
  - Then, it is transmitted to all the other nodes that are directly connected to the originating node.
  - On average, each Ethereum node maintains connections to at least 13 other nodes.
- Each neighbor node validates the transaction as soon as it receives it.
- If it is valid:
  - They store a copy.
  - Propagate it to all their neighbors (except the one it came from).
- when a miner mines a block, the global state of Ethereum is modified
- the balances of the involved accounts are changed
- all contracts activated by transactions/messages are executed
  - state of the accounts are updates and stored in a Patricia Merkle tree
  - New State Root is stored in the block header
  - Receipt Root: a tree recording generated events

\[ \text{SHA3(} \text{Block}) < D \]
• the winning miner will publish the block to the rest of the network,
• the other nodes execute the contracts and check the result
• if the result is valid, they add the block to their own blockchains.
• the state of Ethereum’s blockchain gets updated.
• the entire Ethereum network agrees on the current balance, storage state, contract code, etc. of every single account.
THE ETHEREUM STATE MACHINE

- Ethereum network state: state of all its accounts
- a simplified account state
  - account address
  - balance
  - code and state of the variables for contract accounts

Diagram:

- External Account
  - 14c5f8ba: -1024 eth
  - bb75a980: -5202 eth
    - if contract.storage[tx.data[0]]: contract.storage[tx.data[0]] = tx.data[1]
    - [0, 235235, 0, ALICE...]
  - 892bf92f: -0 eth
    - send(tx.value / 3, contract.storage[0])
    - send(tx.value / 3, contract.storage[1])
    - send(tx.value / 3, contract.storage[2])
    - [ALICE, BOB, CHARLIE ]
  - 4096ad65: -77 eth

- Contract Account
a simplified transaction
THE ETHEREUM STATE MACHINE

- an example of state transition fired by a transaction
THE HALTING PROBLEM

• every node in the network
  • evaluates all transactions and store all the contract state
• halting problem
  • it is not possible to tell, just by looking at the program, whether it will take forever or not to execute.
  • actually, the program must be executed to verify this
  • due to Turing completeness, this may happen in Ethereum, not in Bitcoin
• what if malicious users or accidents ask to execute a never stopping code?

```
function foo()
{
    while (true) {
        /* Loop forever! */
    }
}
```

since EVM is a single-threaded machine, without any scheduler
• denial of service!
GAS: A NECESSARY EVIL

- Ethereum introduces the concept of gas

- idea: pay for contract execution giving payed “gas” to your smart contract
  - make DoS attacks expensive
  - make executing a smart contract not free
  - the EVM halts the execution of the program when contract goes out of gas

- each computational step has a fixed “gas fee”
  - also storage resources required to perform contract actions have gas fees

- EVM: a quasi-Turing-complete machine
  - can run any program, but only if the program has paid enough gas
  - at any one time, there is a defined computational limit given by the amount of gas of the contract
GAS AND GAS PRICE

• need to “buy gas” to run a smart contract
  • purchasing gas is like purchasing distributed, trustless computational power

• gas price in Ether is up to the caller!
  • the amount of Ether you are willing to spend on every unit of gas
  • low price means low priority, and viceversa
  • measured in “gwei” \( 1 \text{ gwei} = 1,000,000,000 \text{ wei} \)
  • price is variable

• unit of gas spent for each instruction is fixed
  
  adding two numbers \( : 3 \text{ gas} \)
  computing a Keccak-256 hash \( : 30 \text{ gas} + 6 \text{ gas for each 256 bits of data being hashed} \)
  sending a transaction \( : 21,000 \text{ gas} \)
ETHER AND ITS FRACTION

- internal currency of Ethereum used
- to transfer values in transactions
- to pay gas: computation fees

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

each name “a piece of computer science”
GAS AND CONTRACT COMPUTATION

- gas price and gas limit are set for each transaction
  - gas limit
    - maximum amount of gas the sender is willing to pay for this transaction.
  - fee = gas price * gas limit,
    - max amount of wei sender is willing to pay for transaction
    - fee is computed for each transaction
- the fees are rewards for miners for the effort to run computations and validate transactions
  - the higher the gas price, the more likely miners will select the transaction.

\[
\text{Gas Limit} \times \text{Gas Price} = \text{Max transaction fee}
\]

\[
\begin{align*}
\text{Gas Limit} & = 50,000 \\
\text{Gas Price} & = 20 \text{ gwei} \\
\text{Max transaction fee} & = 0.001 \text{ Ether}
\end{align*}
\]
when specifying a gas limit, the following points must be considered:

- different operations will have different gas costs
- the miners will stop executing the moment the gas runs out.
- if there is any gas left over, it will be immediately refunded to the operation generator.
- Ethereum has also a limit on block gas
- miners consider a block full when transactions's gas costs reach limit
GAS COMPUTATION: NO EXCEPTION

1. if $\text{gas\_limit} \times \text{gas\_price} > \text{balance}$ then halt
2. deduct $\text{gas\_limit} \times \text{gas\_price}$ from balance
3. set $\text{gas} = \text{gas\_limit}$
4. run code deducting from $\text{gas}$ the amount required to run code
5. after termination return remaining $\text{gas}$ to balance
if the sender does not provide the necessary gas to execute the transaction, the transaction is "out of gas" and is considered invalid.

- the transaction aborts
- state reverts to previous state
- gas_limit * gas_price is still deducted from balance
  - because the node has spent the effort to run the calculations before running out of gas
## GAS COST

<table>
<thead>
<tr>
<th>Operation</th>
<th>Gas</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD/SUB</td>
<td>3</td>
<td>Arithmetic operation</td>
</tr>
<tr>
<td>MUL/DIV</td>
<td>5</td>
<td>Arithmetic operation</td>
</tr>
<tr>
<td>ADDMOD/MULMOD</td>
<td>8</td>
<td>Arithmetic operation</td>
</tr>
<tr>
<td>AND/OR/XOR</td>
<td>3</td>
<td>Bitwise logic operation</td>
</tr>
<tr>
<td>LT/GT/SLT/SGT/EQ</td>
<td>3</td>
<td>Comparison operation</td>
</tr>
<tr>
<td>POP</td>
<td>2</td>
<td>Stack operation</td>
</tr>
<tr>
<td>PUSH/DUP/SWAP</td>
<td>3</td>
<td>Stack operation</td>
</tr>
<tr>
<td>MLOAD/MSTORE</td>
<td>3</td>
<td>Memory operation</td>
</tr>
</tbody>
</table>
# GAS COST

<table>
<thead>
<tr>
<th>Operation</th>
<th>Gas</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JUMP</td>
<td>8</td>
<td>Unconditional jump</td>
</tr>
<tr>
<td>JUMPI</td>
<td>10</td>
<td>Conditional jump</td>
</tr>
<tr>
<td>SLOAD</td>
<td>200</td>
<td>Read from storage</td>
</tr>
<tr>
<td>SSTORE</td>
<td>20.000</td>
<td>Write to storage</td>
</tr>
<tr>
<td>BALANCE</td>
<td>400</td>
<td>Get balance of an account</td>
</tr>
<tr>
<td>CREATE</td>
<td>32.000</td>
<td>Create a new account using CREATE</td>
</tr>
<tr>
<td>CALL</td>
<td>25.000</td>
<td>Message-call into an account</td>
</tr>
</tbody>
</table>
### STORAGE PRICE IN ETHERUM

ETH Price: $166.41 (Apr 17, 2019) - Gas Price: 3 Gwei  \( \$ 172.16 \) (Apr 21, 2020)

<table>
<thead>
<tr>
<th>Size</th>
<th>Gas</th>
<th>Cost (ETH)</th>
<th>Cost ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>32 bytes</td>
<td>21.000</td>
<td>0.000063</td>
<td>$0.36088</td>
</tr>
<tr>
<td>1KB</td>
<td>724.664</td>
<td>0.002174</td>
<td>$0.01046</td>
</tr>
<tr>
<td>1MB</td>
<td>(~697.325.562)</td>
<td>2.09198</td>
<td>$347.268</td>
</tr>
<tr>
<td>10MB</td>
<td>(~7.000.000.000)</td>
<td>~21</td>
<td>$3,486</td>
</tr>
<tr>
<td>100MB</td>
<td>(~70.000.000.000)</td>
<td>~210</td>
<td>$34,860</td>
</tr>
<tr>
<td>1GB</td>
<td>(~700.000.000.000)</td>
<td>~2100</td>
<td>$348,600</td>
</tr>
</tbody>
</table>
MESSAGES AND GAS

- no gas limit in internal transactions: why?
- gasLimit is determined by the external creator of the original transaction, by some EOA
  - must be enough to carry out all the transactions
    - including any sub-executions that occur as a result of that transaction, such as contract-to-contract messages.
  - if, in the chain of transactions and messages, a particular message execution runs out of gas
    - that message’s execution will revert, along with any subsequent messages triggered by the execution.
  - however, the parent execution may not need to revert.
the transaction is serialized using the *Recursive Length Prefix (RLP)* encoding scheme
ETHEREUM TRANSACTION: RECIPIENT

• **TO** field
  • a 20-byte Ethereum address
  • EOA or contract address

• no validation of the **to** field
  • any 20-byte value is valid.
  • if it is invalid, the Ether sent is burnt.

• comparison with Bitcoin transactions
  • only one output address and no script in it
  • value directly inserted in the transaction, no reference to a previous transaction output
ETHEREUM TRANSACTION: DATA AND VALUE

- the "payload" of a transaction contains the fields
  - **VALUE** and **DATA**
- fields be significative or null
  - only value significative: a payment
  - only data significative: a function invocation
  - both value and data significative: payment and invocation
- **VALUE** trasmitted to
  - EOA: added to the balance of the target address account
  - contract accounts
    - if no function is found, increase the balance of the contract
    - otherwise, the function named in the data payload must be payable: can accept Ether from the caller.
ETHEREUM TRANSACTIONS: DATA AND VALUE

• DATA
  • transmitted to EOA: allowed, but not specified in the Ethereum protocol
  • to contracts accounts:
    • function selector:
      • first 4 bytes of the Keccak-256 hash of the function’s prototype
      • allows to unambiguously identify which function to invoke.
    • function arguments:
      • encoded according to the rules for the various elementary types
  • for contract creation transactions
    • deploy a new contract on the blockchain
    • data payload contains the compiled bytecode which will create the contract
• GAS PRICE

• GAS LIMIT

• V, R, S

  • signature components

  • three components of an ECDSA digital signature of the originating EOA

  • allow to compute the address of the account sending the transaction
<table>
<thead>
<tr>
<th></th>
<th>Bitcoin</th>
<th>Ethereum</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>First issued</strong></td>
<td>January 2009</td>
<td>July 2015</td>
</tr>
<tr>
<td><strong>Distributed Consensus</strong></td>
<td>Proof-of-work</td>
<td>Proof-of-work, moving to proof-of-stake (Casper)</td>
</tr>
<tr>
<td><strong>Block mining rate</strong></td>
<td>( \approx 10 ) minutes</td>
<td>10 - 20 seconds</td>
</tr>
<tr>
<td><strong>Mining reward</strong></td>
<td>12.5 BTC / block mined, halves every 210k blocks ( (\approx 4 ) years)</td>
<td>5 ETH / block mined, constant (^1)</td>
</tr>
<tr>
<td><strong>Total Supply</strong></td>
<td>Finite; capped at 21 M</td>
<td>Infinite</td>
</tr>
<tr>
<td><strong>Monetary Policy</strong></td>
<td>deflationary</td>
<td>inflationary</td>
</tr>
<tr>
<td><strong>Tx. Processing</strong></td>
<td>Rests on miners (+ tip)</td>
<td>Rests on everyone (+ tip)</td>
</tr>
<tr>
<td><strong>Price</strong></td>
<td>( 1 ) BTC = 6.366,51 Euro</td>
<td>( 1 ) ETH = 159,10 Euro</td>
</tr>
</tbody>
</table>
ETHEREUM VERSUS BITCOIN

- Ethereum extends the blockchain concepts from Bitcoin
  - runs computer code equivalently on many computers around the world.

- Ethereum characteristics
  - account-based
  - Turing complete smart contracts
  - uncles
  - gas
  - different PoW
  - evolving toward other consensus mechanisms (proof of stake?)
STILL TO BE DISCUSSED

- Programming Smart Contracts: Solidity
- Ethereum Block Structure:
  - logs
  - ETHASH
  - owners and uncles in Ethereum
    - GHOST protocols
  - Merkle Patricia Tries
  - block difficulty
  - Proof of Stake
  - Ethereum Virtual Machine