smart contracts
spending the money of a utxo
the easy story

• this is similar to a challenge response protocol
• txin of a transaction tx provides...
  – public key whose hash should match the address in txout
  – signature of a string X
• X is a string derived from...
  – tx where signatures are omitted
  – the destination address contained in referred txout
the reality: the conditions for unlocking funds can vary

- only one subject can spend
- anyone can spend
- nobody can spend (logging)
- M-of-N subjects should agree to spend
- one subject can spend after a certain amount of funds are accumulated (crowdfunding)
- one (or many) can spend after a certain time
- etc...
- a combination of the above
scripts

• **locking** script (a.k.a. scriptPubKey)
  – associated with txout
  – states conditions to spend the output (a “question”)
    • usually it specifies the (hash of) a public key

• **unlocking** script (a.k.a. scriptSig)
  – associated with txin
  – should «match» the conditions of the corresponding txout (the “answer”)
    • usually it contains a signature

• the output of the unlocking script (answer) is used as input for the locking script (question)
  – essentially: (1) exec the unlocking script (2) exec the unlocking without resetting the stack (3) success if top of the stack is not zero and no operation failed

• executed as part of consensus checks
the language

• proprietary
• simple
• stack-based
• no state
• same execution on all nodes
• no iteration instructions
  – Turing incomplete
the language

- read and executed from left to right
- **constants** push themselves onto the stack
- arithmetic: ADD, SUB, …
- stack: DUP, DROP, ROT, 2DUP, …
- flow: IF, ELSE, ENDIF, VERIFY, RETURN, …
- crypto: HASH160, SHA1, CHECKSIG, CHECKMULTISIG …
- time: CHECKLOCKTIME,

https://en.bitcoin.it/wiki/Script
examples

• pay-to-public-key-hash (the “standard” one)
  unlock: <sig> <pubKey>
  lock: DUP HASH160 <pubKeyHash> EQUALVERIFY CHECKSIG

• anyone-can-spend
  unlock: (empty)
  lock: TRUE

• provably-unspendable, just to store data
  lock: RETURN <data max 80 bytes> (never considered an UTXO for efficiency)

• A or B can spend
  unlock: <sig> <A-or-BpubKey> <1 for A, 0 for B>
  lock: IF DUP HASH160 <ApubKeyHash>
          ELSE DUP HASH160 <BpubKeyHash> ENDIF
          EQUALVERIFY CHECKSIG

• freezing funds until a time in the future
  unlock: <sig> <pubKey>
  lock: <expiry time> CHECKLOCKTIMEVERIFY DROP
        DUP HASH160 <pubKeyHash> EQUALVERIFY CHECKSIG
smart contracts

• each one of these scripts is called *smart contract*
  – it is not a “legal contract”, it is just a script!
  – they may realize/support legal contracts
    • it might be recognized as a contract, if parties agree that “code is law”, since the execution is checked by consensus

• enable to use the bitcoin blockchain for other purposes:
  – colored coins
    • *tokens*, that are distinct from bitcoin, whose transactions are recorded in the bitcoin blockchain
    • obsoleted by the rising of the transaction fees
  – record of transaction for generic assets
  – settlement of off-chain transactions
    • so called “payment channels”, see the Lightning Network
bitcoin for smart contracts: limits

• high fees
• limited expressiveness
  – Turing incomplete
• slow
• the blockchain records any “state change”
  – unhandy for software execution
# Ethereum

- Ethereum is targeted to smart contracts

<table>
<thead>
<tr>
<th></th>
<th>Bitcoin</th>
<th>Ethereum</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Turing completeness</strong></td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td><strong>persistent values for scripts</strong></td>
<td>not supported, complex, just UTXO, usually need external code</td>
<td>contracts accounts <strong>can store variables</strong>, easy to retrieve</td>
</tr>
<tr>
<td><strong>blockchain contains</strong></td>
<td>just transactions</td>
<td><strong>current status</strong></td>
</tr>
<tr>
<td><strong>language</strong></td>
<td>simple stack based</td>
<td><strong>high level language</strong> compiled in a bytecode for the Ethereum Virtual Machine</td>
</tr>
<tr>
<td><strong>block time</strong></td>
<td>10 minutes</td>
<td><strong>20 seconds</strong></td>
</tr>
<tr>
<td><strong>consensus</strong></td>
<td>PoW</td>
<td>PoW-&gt;PoS?</td>
</tr>
<tr>
<td><strong>block size limit</strong></td>
<td>1MB</td>
<td>adjusted dynamically, no limit</td>
</tr>
</tbody>
</table>
accounts

• in bitcoin the lock script states what should be provided to unlock funds
  – it is a feature of every UTXO
  – some standard scripts (P2PKH, 2-of-3, etc.)
  – potentially infinite kinds of UTXO

• in Ethereum just two kinds of accounts
  – Externally Owned Accounts (EOA)
  – contract accounts
## accounts

<table>
<thead>
<tr>
<th></th>
<th><strong>EOA</strong></th>
<th><strong>contract</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>associated keys</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>balance</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>other persistent</td>
<td>no</td>
<td>yes, it also stores EVM bytecode</td>
</tr>
<tr>
<td>values/variables</td>
<td></td>
<td></td>
</tr>
<tr>
<td>as a transaction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sender…</td>
<td>• can send ETH • can call operations on a contract</td>
<td>• can send ETH • can call operations on another contract (in the same transaction)</td>
</tr>
<tr>
<td>as a transaction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>recipient…</td>
<td>• can receive ETH</td>
<td>• can receive ETH • always executes an operation (possibly the fallback one)</td>
</tr>
</tbody>
</table>
transactions fields

- (sender address)
- recipient address
- value (exchanged ETH)
- data
- nonce (increasing to avoid replay attack)
- gas price
- gas limit

- max fee = gas price * gas limit
  - actual fee depends on the executed code
  - if a tx run “out of gas”, it is reverted, but fee is taken anyway
contract lifecycle

- written in a high-level language
- compiled to EVM bytecode
- deployed
  - transaction sent to special address 0x0 and bytecode as data
- operations are called on the contract
  - as part of transactions, which may update its state
- cannot be deleted, but the contract can selfdestruct itself
a solidity example

• anyone can withdraw funds from this contract

```solidity
// Our first contract is a faucet!
contract Faucet {

    // Give out ether to anyone who asks
    function withdraw(uint withdraw_amount) public payable {

        // Limit withdrawal amount
        require(withdraw_amount <= 1000000000000000000);

        // Send the amount to the address that requested it
        msg.sender.transfer(withdraw_amount);

    }

    // Accept any incoming amount
    function () public payable {}
contract owned {
  address owner;
  // Contract constructor: set owner
  constructor() {
    owner = msg.sender;
  }
  // Access control modifier
  modifier onlyOwner {
    require(msg.sender == owner,
      "Only the contract owner can call this function");
  }
}

contract mortal is owned {
  // Contract destructor
  function destroy() public onlyOwner {
    selfdestruct(owner);
  }
}

contract Faucet is mortal {
  event Withdrawal(address indexed to, uint amount);
  event Deposit(address indexed from, uint amount);
  // Give out ether to anyone who asks
  function withdraw(uint withdraw_amount) public {
    // Limit withdrawal amount
    require(withdraw_amount <= 0.1 ether);
    require(this.balance >= withdraw_amount,
      "Insufficient balance in faucet for withdrawal request");
    // Send the amount to the address that requested it
    msg.sender.transfer(withdraw_amount);
    emit Withdrawal(msg.sender, withdraw_amount);
  }
  // Accept any incoming amount
  function () public payable {
    emit Deposit(msg.sender, msg.value);
  }
}
simple things might be complex

• for example, multisignature
libraries

• libraries can be imported in a project as included code…
• …or from the blockchain!
  – …if you trust it!
remix

- a basic web based editor, emulator, debugger
- https://remix.ethereum.org
contracts security

- contracts are usually not very long
- writing contracts is easy
- **writing secure contracts is difficult**
  - solidity/EVM semantic may be subtle
  - mistakes may cost a lot of money!
references

• A. M. Antonopoulos – Mastering Bitcoin
• A. M. Antonopoulos, G. Wood - Mastering Ethereum