smart contracts

smart contracts

- a smart contract allows the users to customize the consensus rules to support specific applications
  - i.e., the consensus rules ask for the successful execution of all smart contracts executed within each transaction.

- it is not a “legal contract”
  it is just a software executed by nodes during the validation
  - they may realize/support legal contracts
  - it might be recognized as a contract, if parties agree that “code is law”
  - however, agreement between parties is essential for a legal contract since code is usually hard to understand, the agreement can hardly be based on it
    - in Italy they are regulated by Legge 11 feb. 2019 n.12

intrinsic security features of SCs

- when you ask a centralized server to execute something, you have to trust the server
- when you ask a blockchain for a smart contract execution, **correctness of execution is guaranteed**
  under the assumptions of used consensus approach
- however, input and output (and state, for Ethereum) are essentially public
- Integrity: OK, availability: OK, **confidentiality: NO**
  – **lack of confidentiality should be carefully considered when compliance with privacy regulation is needed**

smart contracts flavors

- in Bitcoin they were attached to UTXOs and used to check if related cryptocurrency can be spent
  - just true/false predicates
  - spent UTXO the contract cannot be used
  - no state persistency
- SCs are stored as bytecode in the blockchain as an independently addressable account
  - they do have state and perform computations (part of the consensus rules), not just checks
  - many take this approach (e.g., Ethereum, EOS, Cardano, etc.)
- just the bytecode hash is stored
  - to save space
  - in this case the transaction should provide the bytecode which is checked against the hash (used in Algorand ASC1)
- the whole source code may be stored
  - this make code verification easier, but the blockchain should take in charge of compilation as part of consensus

smart contract problems

- SCs are written by users and run by nodes
- errors in SCs and abuse may impact the whole network
  - the halting problem
    - what if the execution does not finish?
  - resource consumption
    - what if the execution consume a large amount of memory or takes a lot of time?
- in permission**ed** DLT the subject that own the contract is known
  - a timeout on the execution is usually enough to avoid faulty code to impact the whole blockchain network

smart contracts in permission**less** DLTs

two approaches:

- limit the expressiveness of the smart contracts

  a typical choice:

  - smart contracts are attached to accounts to allow just the customization of transaction acceptance
  - Turing incomplete (no loops)
  - no persistent state

  this is the Bitcoin approach

- ask a payment for resource consumption and limit it
  - smart contracts are independent object with their own account and balance
  - they allow persistent state (paid)
  - Turing completeness (pay each executed instruction),
    i.e., **the fees of the transaction depends on the resources consumed by the execution** (part of the consensus rules)
  - fees are usually transferred to block producers

  this is the Ethereum approach

the Ethereum smart contract model

- in Ethereum we have two kinds of accounts
  - Externally Owned Accounts (EOA): owned by users
    - a user has a private key to unlock founds, EOA are identified by a public key (address)
  - contract accounts: associated with a smart contract
    - no private key (no user), just an identifier of the account
    - this also stores the state of the SC
- the execution consumes **gas** which is paid with Ethereum cryptocurrency (ETH)
  - execution is performed on the Ethereum Virtual Machine (EVM)
    - which executes bytecode
  - each instruction has its gas consumption

Ethereum SC and contracts accounts

- each contract account is associated with a software object, which is an instance of the smart contract
  - very much like a software object of OOP
- each smart contract has a **state** stored in its account
  - persisted in the blockchain
- smart contract has **operations**
  - to be called externally by a transaction or by another smart contract (in the same transaction)

operations

- an operation is executed within a transaction
- it can…
  - change the state of the object
  - take parameters
  - return values
- essentially, they are the methods of the object/contract

accounts recap

	EOA	contract accounts
associated private keys	**yes**	**no**
balance	yes	yes
other persistent values/variables	no	yes it also stores EVM bytecode
as a transaction recipient…	• can receive ETH	• can receive ETH • **always executes an operation** (possibly the fallback one)
as a transaction sender…	• can send ETH • can call operations on a contract	contracts cannot really send transaction but • can call operations on another contract **in the same received transaction** • can send ETH

Ethereum transactions fields

- (sender address)
- recipient address
- value (exchanged ETH)
  - this is transferred from the balance of the sender address to the one of the recipient address
- data
- nonce (increasing, to avoid replay attack)
- gas limit
  - this limit the execution, it should be high enough for a regular execution to complete but not too high, since this is the limit of what is paid if the stimulated operation loops indefinitely
- gas price
  - this is the amount paid for one unit of consumed gas in ETH. The node that makes the block could deem it too low and never pick the transaction from the pool.
- max fee = gas price * gas limit
  - actual fee depends on the consumed gas which depends on the executed code
  - if a tx runs “out of gas”, state changes are reverted, but fee is taken from the sender balance anyway

Ethereum: contract lifecycle

- written in a high-level language
  - Solidity (javascript-like, statically typed)
- compiled to EVM bytecode
  - EVM: Ethereum Virtual Machine (used also by other DLTes)
- deployed
  - transaction sent to special address 0x0 and bytecode in the data field of tx
  - Ethereum returns the address of the contract
- operations are called on the contract
  - as part of tx's, which may update its state, change balance, call other contracts (within the same tx, tx sender pays), etc.
- cannot be deleted, but the contract can destruct itself
  - the current balance must be sent to some address

a solidity example

- anyone can withdraw funds from this contract

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

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

        // 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 {}

}
```

```
// Version of Solidity compiler this program was written for
pragma solidity ^0.4.22;

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);
  }
 }
```

evolution

- state variables
- constructors
- inheritance
- custom modifiers
- assertions
- events

simple things might be complex

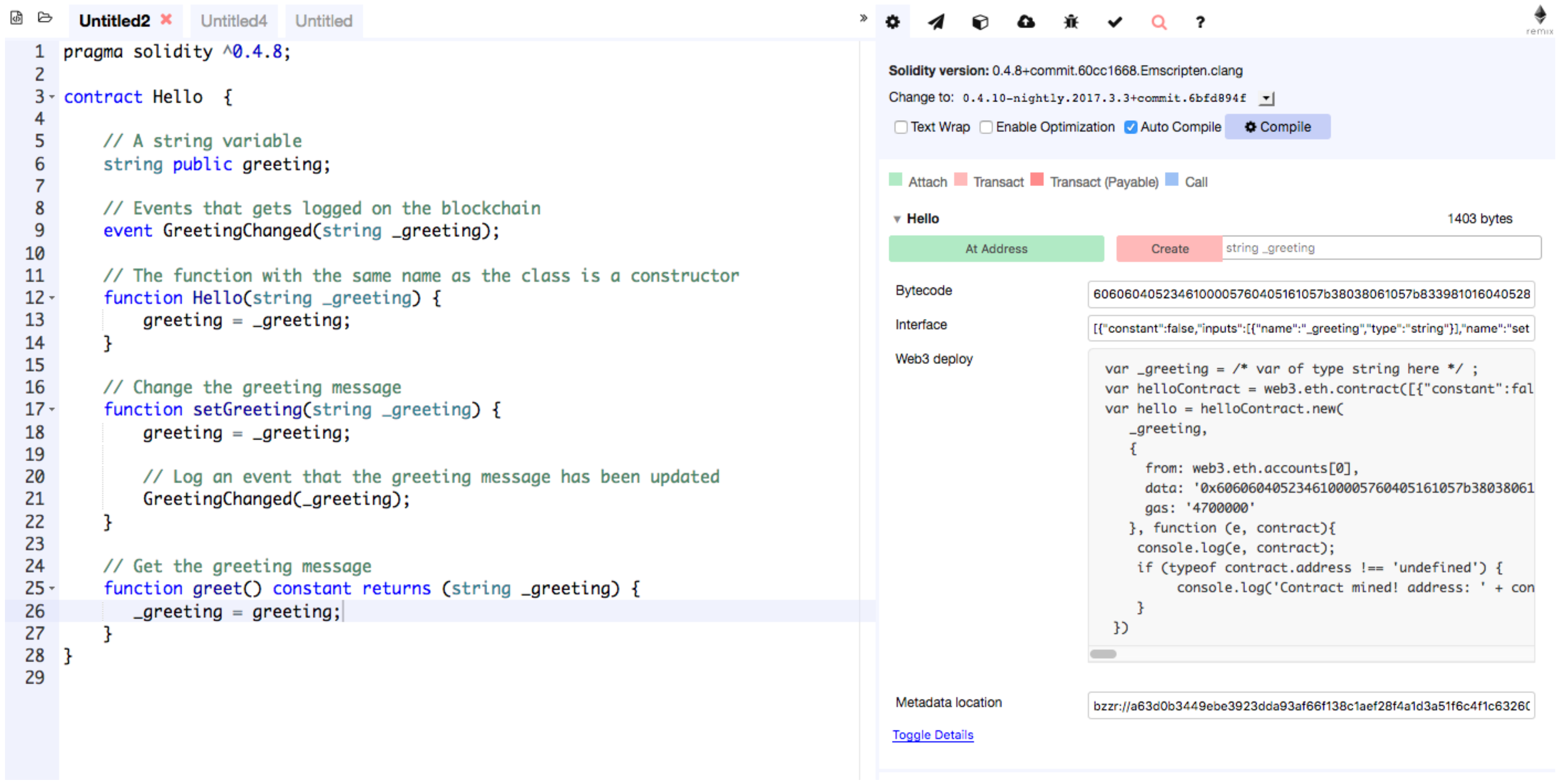
- for example, requiring a multisignature to unlock funds

libraries

- libraries can be imported in a project as included code…
- …or from the blockchain!
  - …if you trust it!
  - operations can call other operations in other smart contracts
  - the execution occurs in the same transaction, paid with the gas for that transaction

remix

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



Untitled2 Untitled4 Untitled

```
pragma solidity ^0.4.8;

contract Hello  {

    // A string variable
    string public greeting;

    // Events that gets logged on the blockchain
    event GreetingChanged(string _greeting);

    // The function with the same name as the class is a constructor
    function Hello(string _greeting) {
        greeting = _greeting;
    }

    // Change the greeting message
    function setGreeting(string _greeting) {
        greeting = _greeting;

        // Log an event that the greeting message has been updated
        GreetingChanged(_greeting);
    }

    // Get the greeting message
    function greet() constant returns (string _greeting) {
        _greeting = greeting;
    }
}

```

Solidity version: 0.4.8+commit.60cc1668.Emscripten.clang

Change to: `0.4.10-nightly.2017.3.3+commit.6bfd894f`

☐ Text Wrap ☐ Enable Optimization ☑ Auto Compile

Hello 1403 bytes

Bytecode: 6060604052346100005760405161057b38038061057b833981016040528

Interface: [{"constant":false,"inputs":[{"name":"_greeting","type":"string"}],"name":"set

Web3 deploy:

```
var _greeting = /* var of type string here */ ;
var helloContract = web3.eth.contract([{"constant":fal
var hello = helloContract.new(
   _greeting,
   {
     from: web3.eth.accounts[0],
     data: '0x6060604052346100005760405161057b38038061
     gas: '4700000'
   }, function (e, contract){
    console.log(e, contract);
    if (typeof contract.address !== 'undefined') {
         console.log('Contract mined! address: ' + con
    }
 })
```

Metadata location: bzzr://a63d0b3449ebe3923dda93af66f138c1aef28f4a1d3a51f6c4f1c6326(

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!

  Atzei N. et al. **A survey of attacks on ethereum smart contracts**. International Conference on Principles of Security and Trust 2017

reducing the costs

- executing a smart contract on a permissionless blockchain is expensive
- approaches to cost reduction
  - make smart contracts as simple as possible
  - move as much as possible of the system off-chain
    - usually centralizing part of the system
- *level 2* solutions (a.k.a. *roll-ups*)
  - move computation and storage off-chain
  - use the original blockchain to ensure integrity
  - may coordinate off-chain p2p infrastructures
  - may involve additional blockchains (*side-chains*)

a comparison of some real blockchains

	Bitcoin	Ethereum	EOS	Algorand	Cardano	Solana	Avalanche
consensus	PoW	PoS	PoS	PoS	PoS	PoH	Snowflake
blocktime	~10min	~12sec	~1sec	~4sec	~20sec	~0.5sec	~3sec
language	stack based, assembly-like	solidity, js-like, EVM	C++	TEAL, assembly-like	Plutus, Haskell-like	Rust, C, C++, compiled into Solana Bytecode Format	solidity, js-like, see Ethereum (EVM)
pros	famous	famous, well supported, easy to program	cheap	cheap	cheap	scalability	modular architecture, see Ethereuem for pros
cons	slow, costly	hard to write secure smart contracts, costly	not much decentralized	hard to program	you have to learn Haskell functional language	PoH has received some critics and attacks	new