Blockchain Technology

Dan Liu #19574

Subjects

- Cryptography
- Bitcoin
- Ethereum

1. Cryptography

- Secret Key (Symmetric Key) Cryptography
- Public Key (Asymmetric Key) Cryptography

Secret Key (Symmetric Key) Cryptography

- Block Ciphers (secure but slow)

  Major types: Substitution Ciphers/Transposition Ciphers

  Examples: DES, 3DES, AES

- Stream Ciphers (insecure but fast)

  Examples: RC4

Issues: **exchange keys and manage keys**

Lab

Key Points:

- TCP/IP Client and Server
- Encryption/Decryption with Symmetric Key

client.py

```
from socket import *
import random
import binascii
from Crypto.Cipher import AES
from Crypto.Util import Counter
from Crypto import Random
import time

HOST = 'localhost'
PORT = 21567
BUFSIZ = 1024
key_bytes = 32
ADDR = (HOST, PORT)
time.clock = time.time

tcpCliSock = socket(AF_INET, SOCK_STREAM)
tcpCliSock.connect(ADDR)
print('Connection established...')


def decrypt(key, iv, ciphertext):
    assert len(key) == key_bytes

    # Initialize counter for decryption. iv should
    # the same as the output of encrypt().
    iv_int = int(binascii.hexlify(iv), 16)
    ctr = Counter.new(AES.block_size * 8, initial

    # Create AES-CTR cipher.
    aes = AES.new(key, AES.MODE_CTR, counter=ctr)

    # Decrypt and return the plaintext.
    plaintext = aes.decrypt(ciphertext)
    return plaintext


while true:
    value = random.randint(0, 10000)
    byte_data = str(value).encode()
    # For example, if the server side
    # type Return without entering any
    # message to end communication.
    if not byte_data:
        break
    tcpCliSock.send(byte_data)
    print('A client send a random number %d to th

    received_data = tcpCliSock.recv(BUFSIZ)
    print('Read a text from server which contains
    if not received_data:
        break

    arr = received_data.split(b':')
    decrypted_data = decrypt(arr[0], arr[1], arr[
    print('The decrypted number is %s, congratula

# The client side ends communication
tcpCliSock.close()
```

server.py

```
from socket import *
import binascii
from Crypto.Cipher import AES
from Crypto.Util import Counter
from Crypto import Random
import time

HOST = ''
PORT = 21567
BUFSIZ = 1024
ADDR = (HOST, PORT)
key_bytes = 32
time.clock = time.time

tcpSerSock = socket(AF_INET, SOCK_STREAM)
tcpSerSock.bind(ADDR)
tcpSerSock.listen(5)


def encrypt(key, plaintext):
    assert len(key) == key_bytes

    # Choose a random, 16-byte IV.
    iv = Random.new().read(AES.block_size)

    # Convert the IV to a Python integer.
    iv_int = int(binascii.hexlify(iv), 16)

    # Create a new Counter object with IV = iv_int.
    ctr = Counter.new(AES.block_size * 8, initial_value=iv_int)

    # Create AES-CTR cipher.
    aes = AES.new(key, AES.MODE_CTR, counter=ctr)

    # Encrypt and return IV and ciphertext.
    ciphertext = aes.encrypt(plaintext)
    return iv, ciphertext


while True:
    print('waiting for connection...')
    tcpCliSock, addr = tcpSerSock.accept()
    print('...connected from: ', addr)

    while True:
        data = tcpCliSock.recv(BUFSIZ).decode()
        try:
            num = int(data)
            print('Get a random number %d from client.' % num)
        except ValueError:
            pass

        # For example, if the client side type Return without
        # entering any message to end communication.
        if not data:
            break

        key = '12345678901234567890123456789012'
        (iv, ciphertext) = encrypt(key, data)
        print('Encrypted plain text %s ==> %s' % (data, ciphertext))
        print('Send the key, iv and cipher text back to the client.')
        tcpCliSock.send(key.encode() + b':' + iv + b':' + ciphertext)

    # The client side ends communication
    tcpCliSock.close()
# The server side ends communication
tcpSerSock.close()
```

Desktop — mark@MacBook-Pro — -zsh — 91×24

python ~/Desktop

```
Last login: Fri Jun  3 00:17:03 on ttys000
You have new mail.
(base) ~/Desktop python client.py
Connection established...
A client send a random number 6646 to the server.
Read a text from server which contains the key, iv and cipher text: b'123456789012345678901
23456789012:\xa4\x98\x84\x03\xb1\xd9\x17\xee\x0b\x0b\xb6\x19\x0f\xfa\xfe\xc7:\xca\x8f\xb1C'
The decrypted number is 6646, congratulation!
(base) ~/Desktop
```

Desktop — python server.py — python serv

python

```
Last login: Fri Jun  3 00:13:46 on ttys001
You have new mail.
(base) ~ cd Desktop
(base) ~/Desktop python server.py
waiting for connection...
...connected from: ('127.0.0.1', 62059)
Get a random number 6646 from client.
Encrypted plain text 6646 ==> b'\xca\x8f\xb1C'
Send the key, iv and cipher text back to the client.
waiting for connection...
```

Public Key (Asymmetric Key) Cryptography

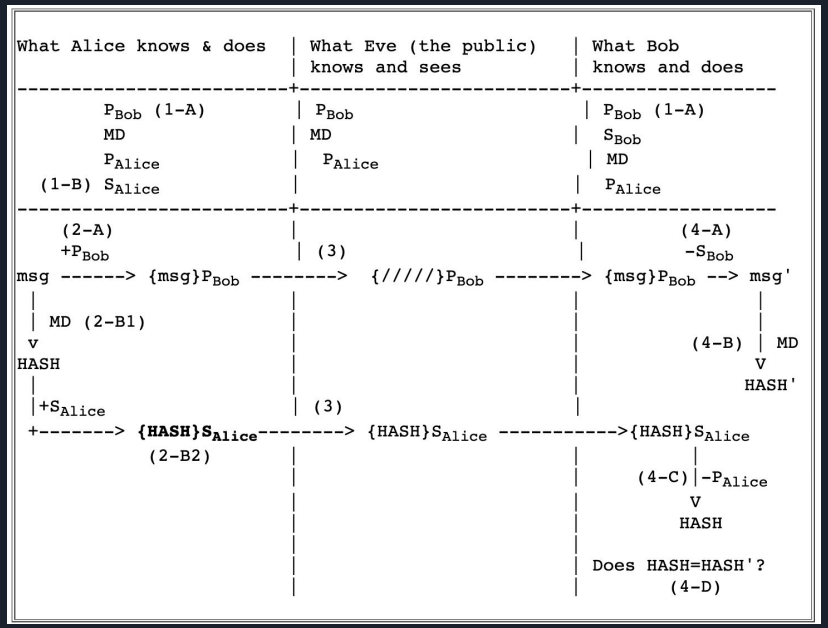
- Rules:

  Diffie Rule 1: Confidentiality

  Diffie Rule 2: Digital Signature

- Algorithms: RSA/ ECC

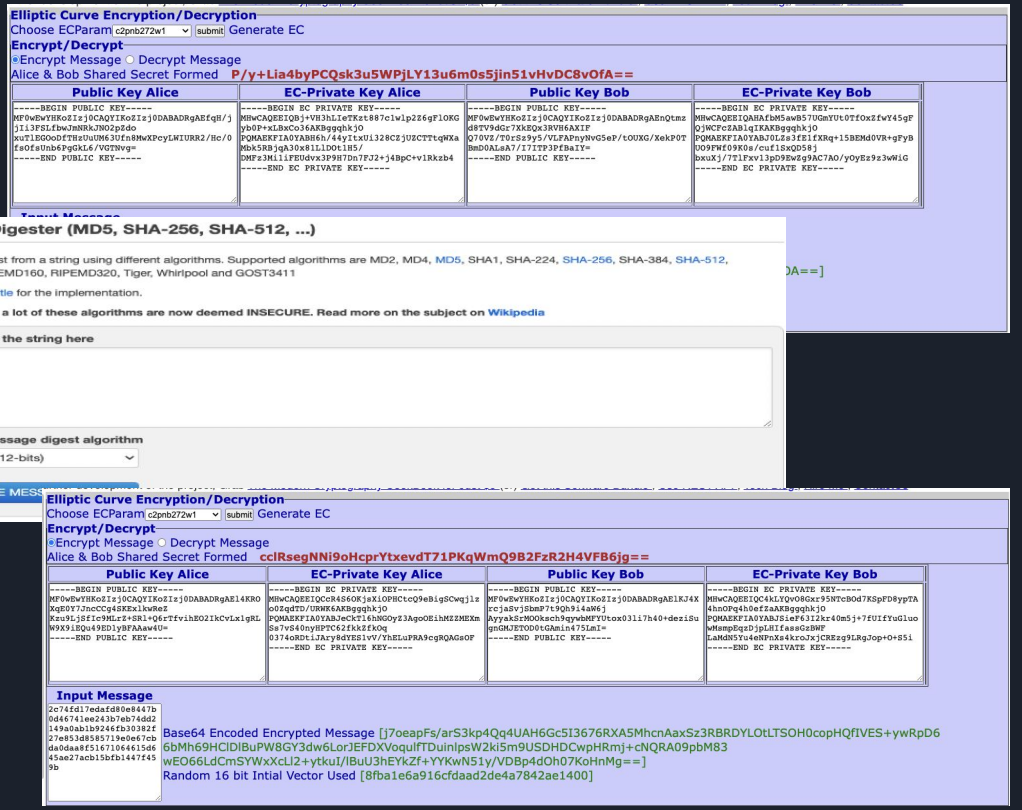
Diffie Rule



Lab

Key Points:

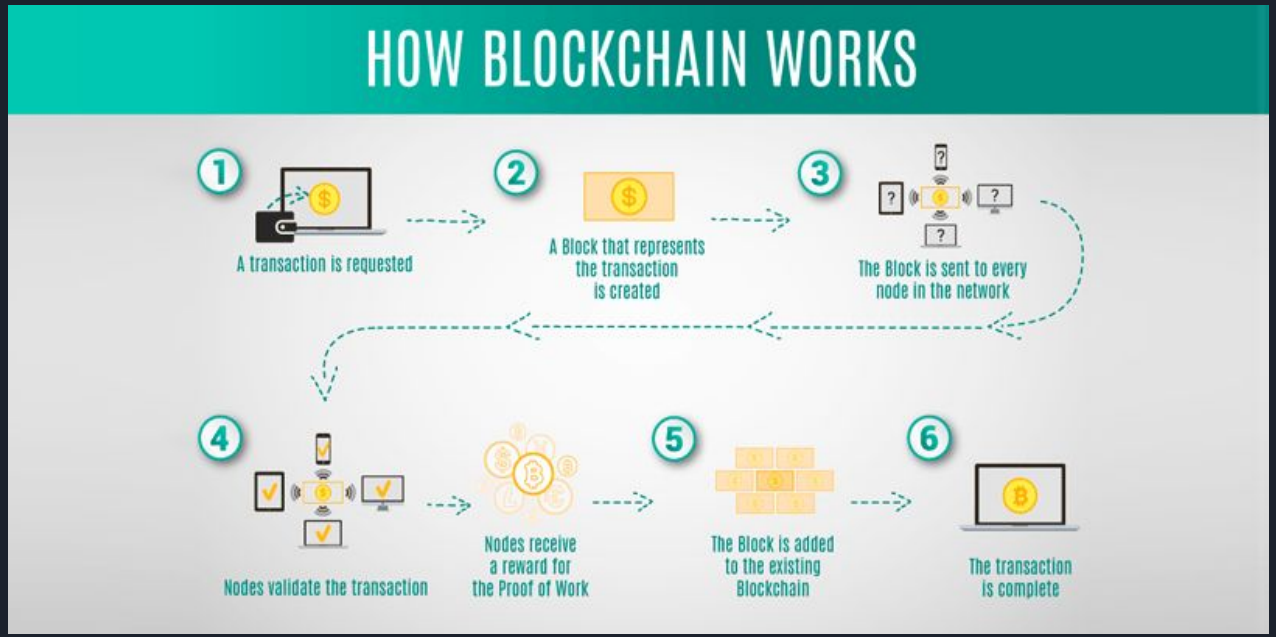
- Diffie Rule 1&2
- ECC Cryptography Tool
- Message Digester



2. Bitcoin

A global peer-to-peer currency that is designed for the Internet

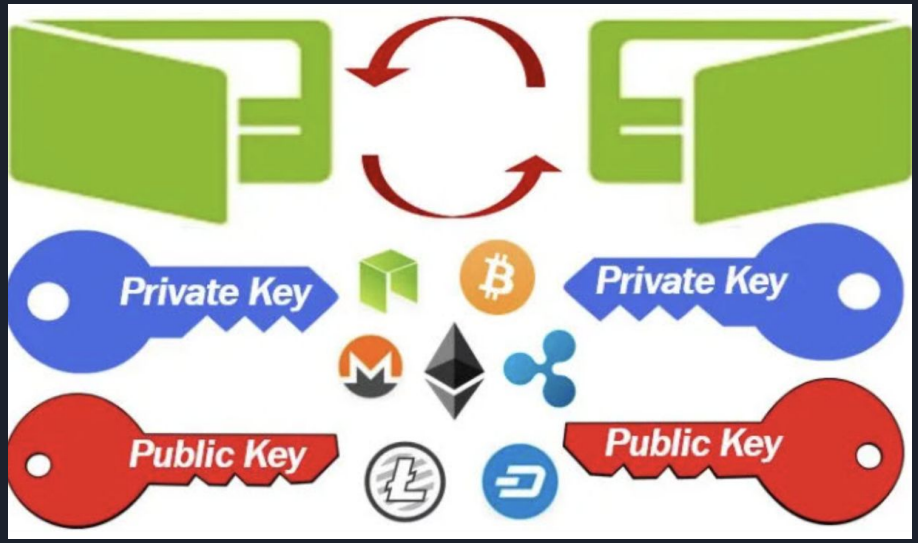
How Bitcoin works



- Construct a transaction
- Bitcoin mining
- Mine transactions in blocks
- Spend the transaction

Bitcoin wallet

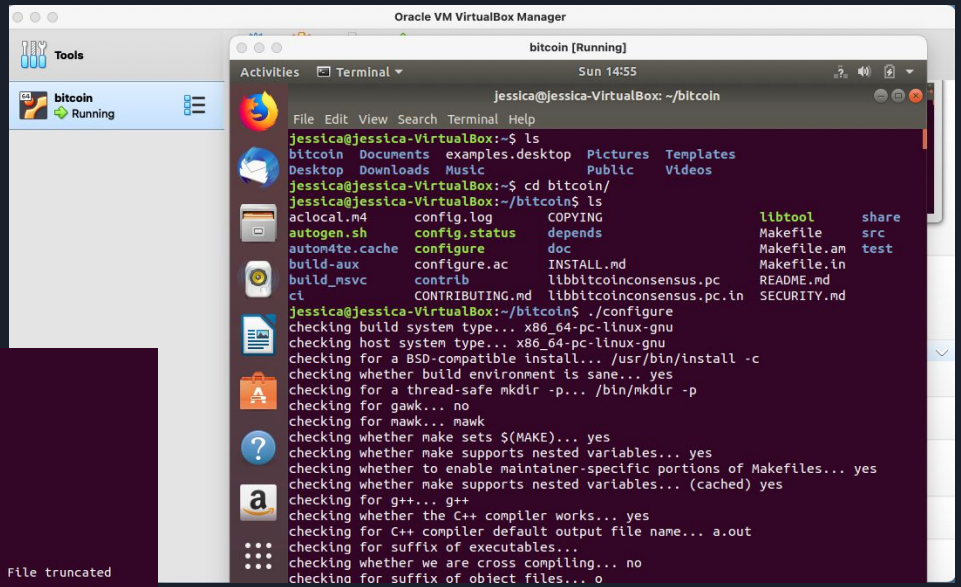
- Online Wallet Services
- Local Wallets or Offline Wallets
- Paper Wallets

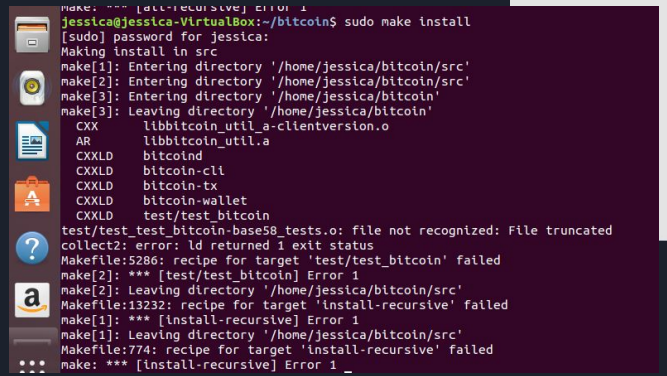


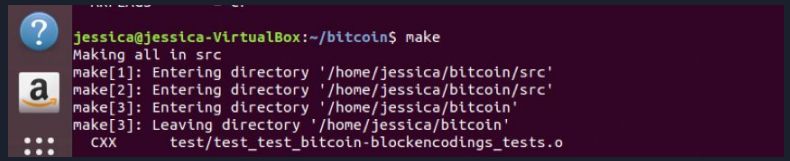
Lab 1: Access Bitcoin Network

Key Points:

- Use Bitcoin Client to access Bitcoin Network
- Bitcoin core







Lab 2: Bitcoin Transaction Chain

Key Points:

- Transaction Inputs and Outputs

Transaction ID 1111			
INPUTS From		OUTPUTS To	
From (previous transactions Joe has received)			
Address J1		Output #0 A1	0.2000 BTC (spent)
Joe	0.2003 BTC	Transaction Fees:	0.0003 BTC

Transaction ID 1234			
INPUTS From		OUTPUTS To	
From 1111		Output #0 E1	0.1700 BTC (spent)
Address A1		Output #1 A1(change)	0.0297 BTC (unspent)
Alice	0.2000 BTC	Transaction Fees:	0.0003 BTC

Transaction ID 2345			
INPUTS From		OUTPUTS To	
From 1234		Output #0 B1	0.0145 BTC (spent)
Address E1		Output #1 E1(change)	0.1552 BTC (unspent)
Eve	0.1700 BTC	Transaction Fees:	0.0003 BTC

3. Ethereum

A decentralized platform

What is Ethereum

- **Open Source**

  Like Bitcoin, Ethereum is a public blockchain, no one controls

- **Proof of Work Consensus**

  The Ethereum Whitepaper specifies the PoW rules and 4+ major clients exist and run the nodes

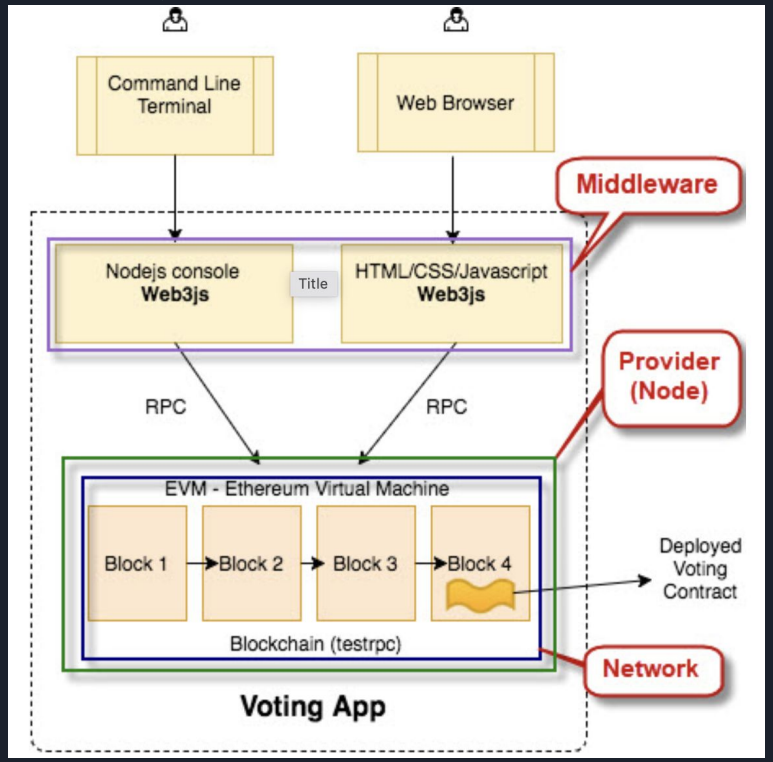
- **Ethereum Virtual Machine (EVM)**

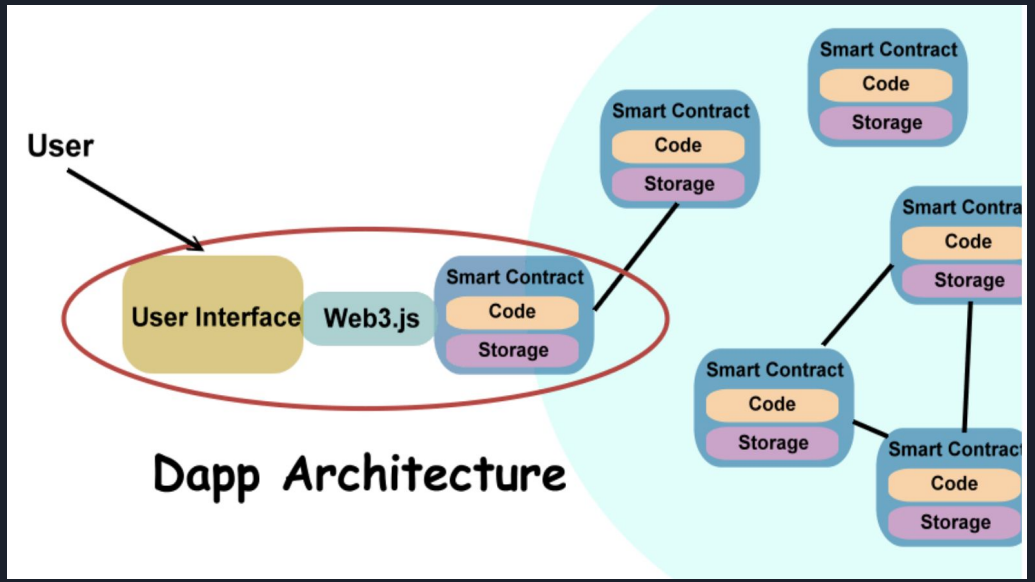
  Transactions are smart contracts, Turing-complete programs that run when blocks are processed by nodes

Ethereum Components

- **Ether:** native token
- **Solidity:** smart contract programming language
- **Whisper:** communication protocol for Dapps
- **Swarm:** the Ethereum decentralized storage protocol
- **Mist:** Dapp browser

How Ethereum Works

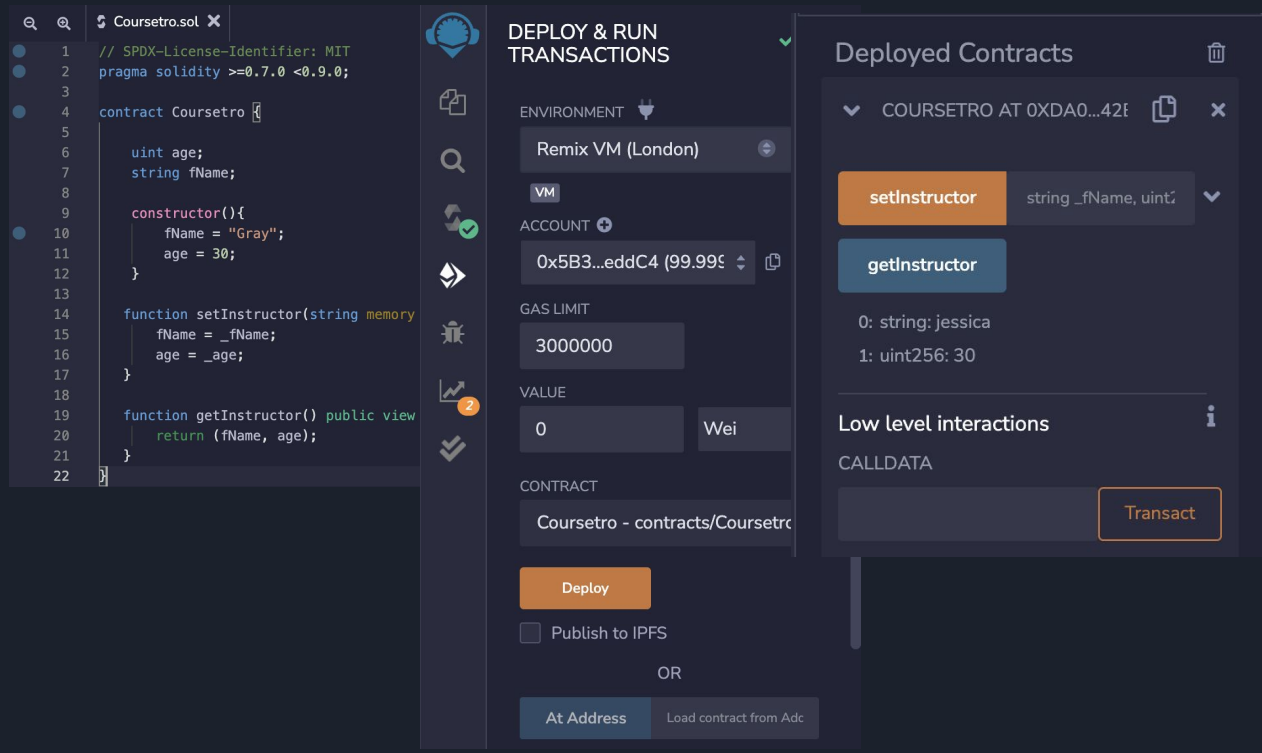




Ethereum Smart Contracts Project - Lab 1

Key Points:

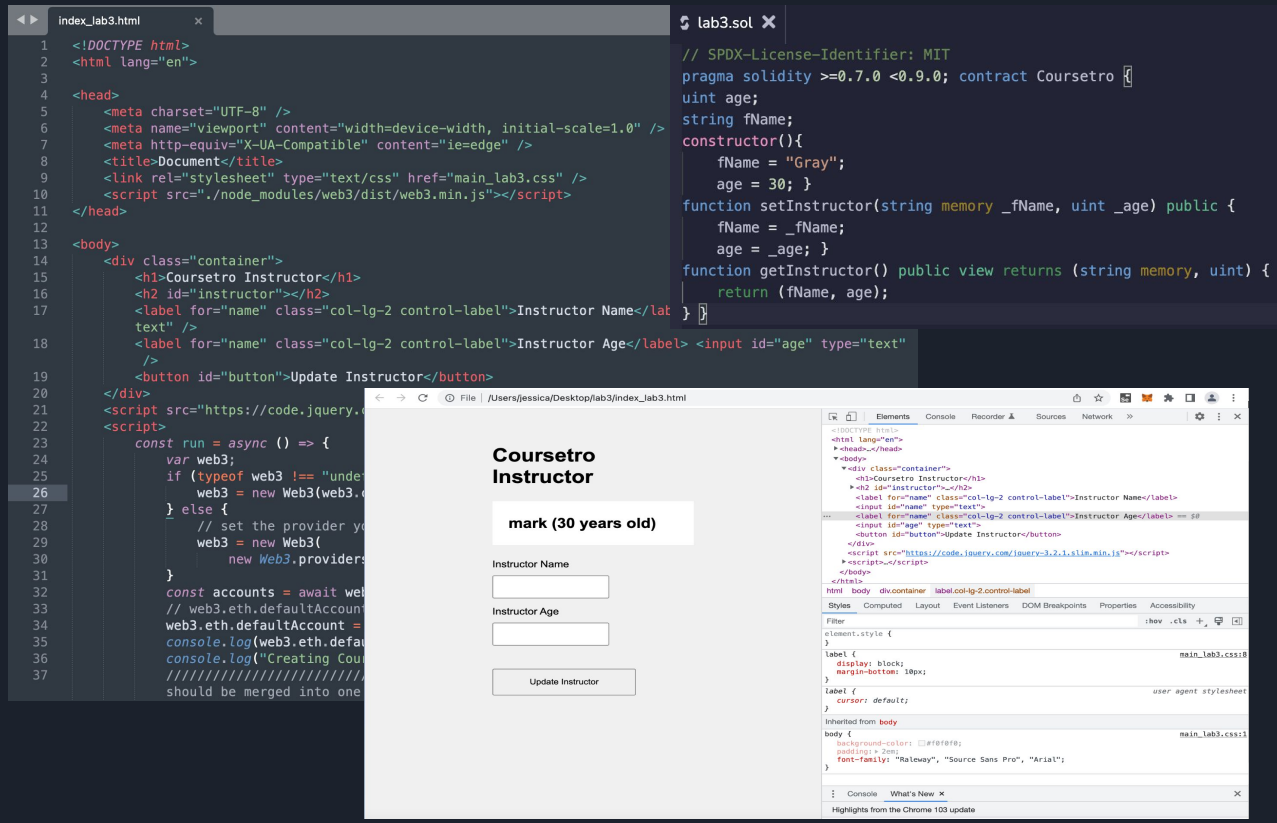
- Use Remix's Javascript VM
- Remix interacts with a smart contract
- Basic concept about Remix and Solidity



Ethereum Smart Contracts Project - Lab 2

Key Points:

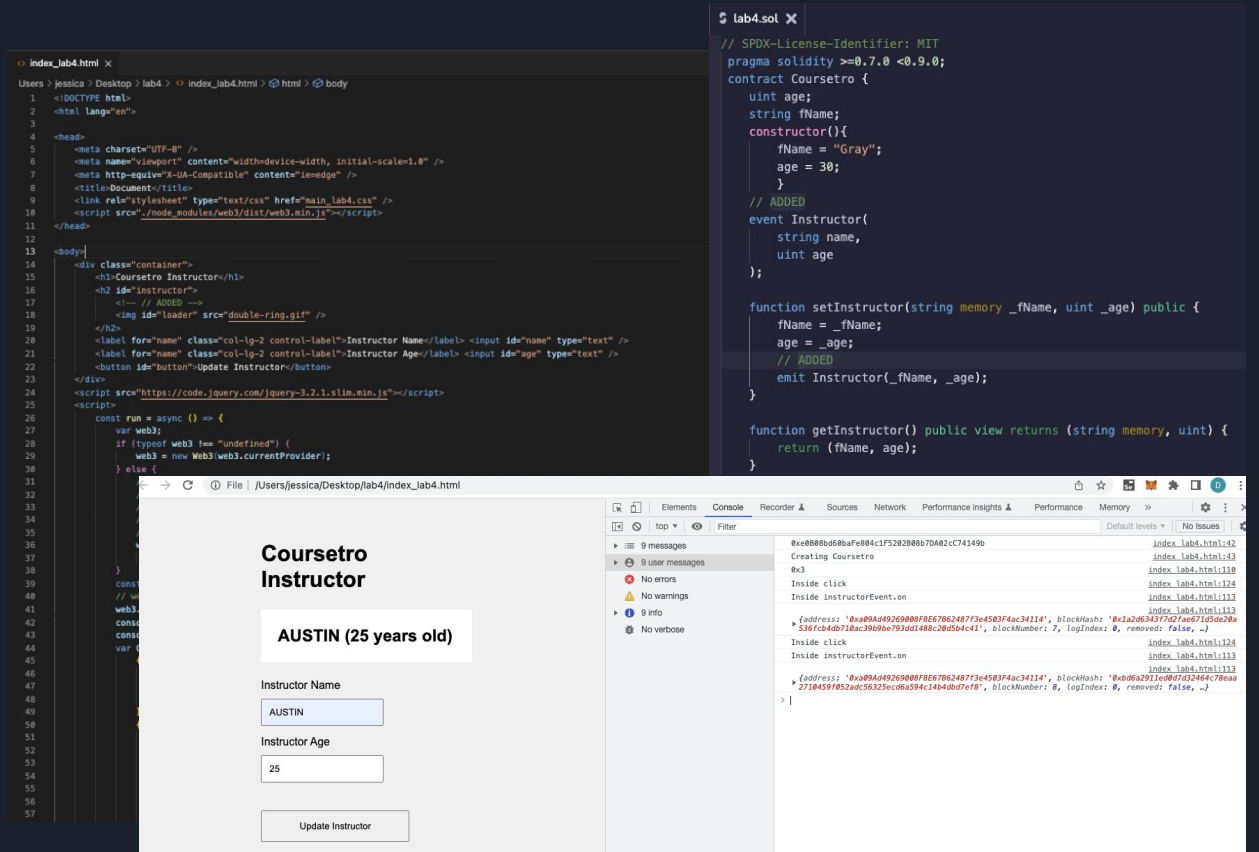
- Use Remix's Web3 Provider
- Use a web page to interact with a smart contract
- Manually reload the web page to see the result of user's action



Ethereum Smart Contracts Project - Lab 3

Key Points:

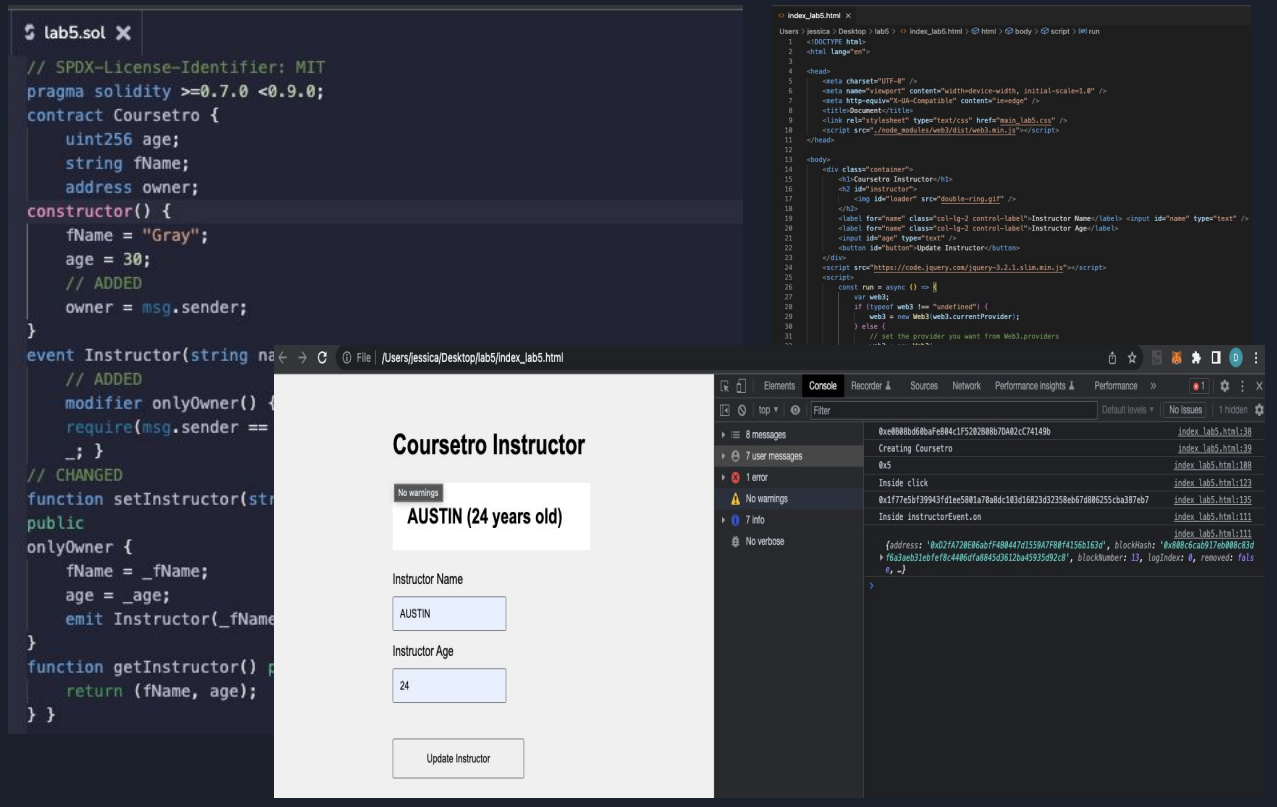
- Same as Lab2
- Use events to automatically reload the web page



Ethereum Smart Contracts Project - Lab 4

Key Points:

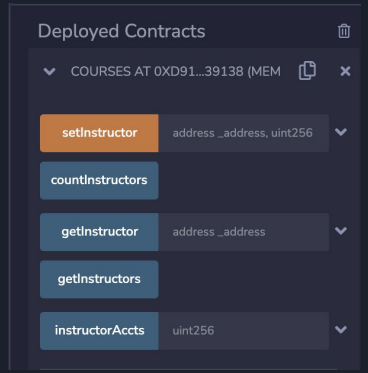
- Same as Lab 3
- Use modifier to allow only the owner can set values



Ethereum Smart Contracts Project - Lab 5

Key Points:

- Same as Lab 4
- Use Struct and Mapping in Solidity



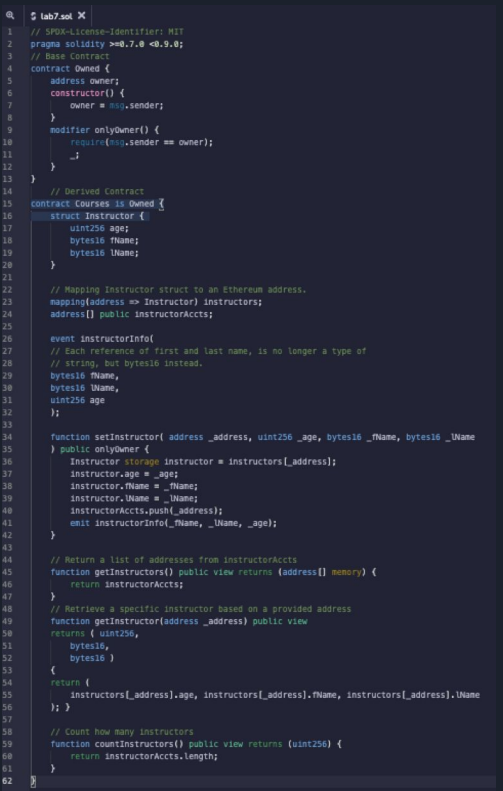
lab6.sol

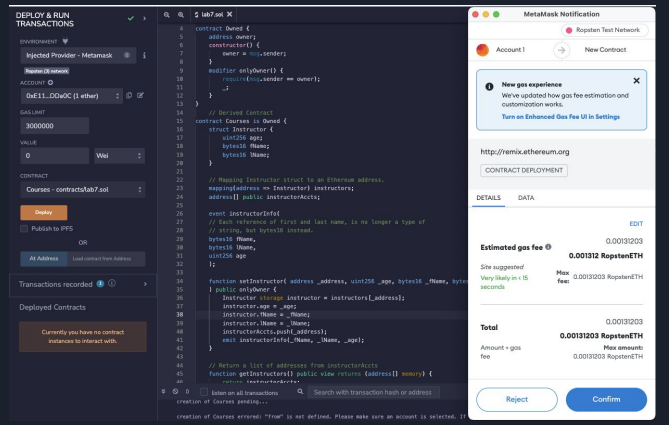
```
// SPDX-License-Identifier: MIT
pragma solidity >=0.7.0 <0.9.0;
contract Courses {
    struct Instructor {
        uint age;
        string fName;
        string lName;
        }
// Mapping an Ethereum address to Instructor struct.
mapping (address => Instructor) instructors;
address[] public instructorAccts;
function setInstructor(address _address, uint _age, string memory _fName, string memory _lName) public {
    Instructor storage instructor = instructors[_address];
        instructor.age = _age;
        instructor.fName = _fName;
        instructor.lName = _lName;
    instructorAccts.push(_address);
    }
// Return a list of addresses from instructorAccts
function getInstructors() view public returns(address[] memory) {
return instructorAccts; }

// Retrieve a specific instructor based on a provided address
function getInstructor(address _address) view public returns (uint, string memory, string memory) {
return (instructors[_address].age, instructors[_address].fName, instructors[_address].lName); }

// Count how many instructors
function countInstructors() view public returns (uint) {
return instructorAccts.length; }
}
```

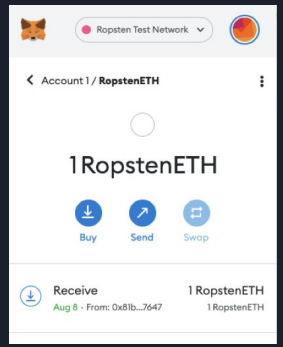
Ethereum Smart Contracts Project - Lab 6

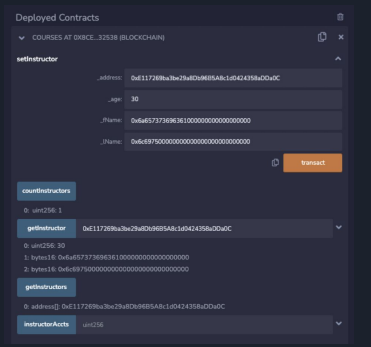
Key Points:

- Use Remix's Injected Provider - MetaMask to interact with Ropsten Test Net





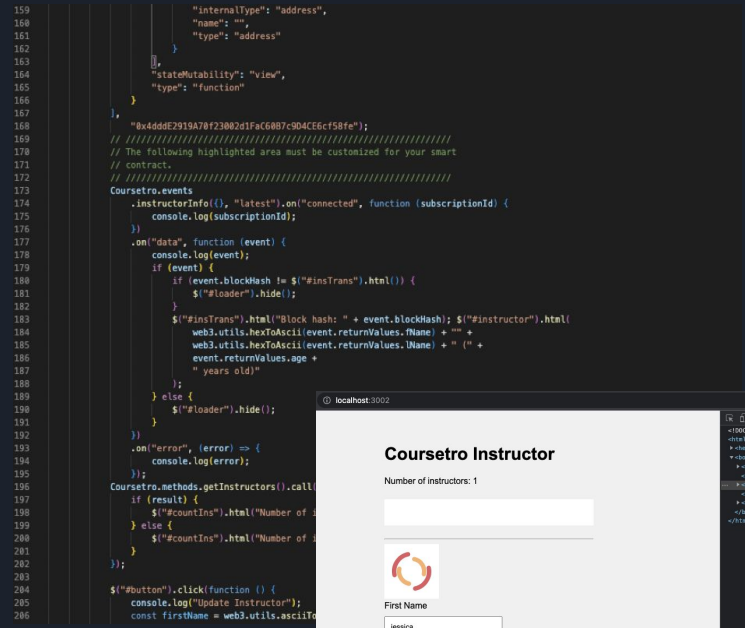


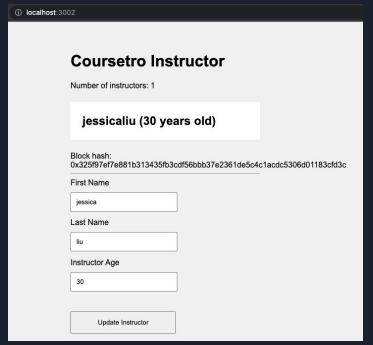


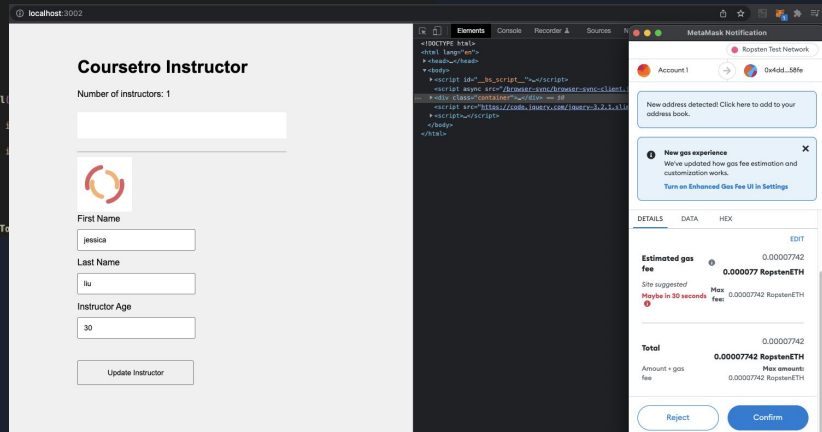
Ethereum Smart Contracts Project - Lab 7

Key Points:

- Same as Lab 6
- Update Web3 UI to work with the updated smart contract







Major Ethereum Use Cases

- Decentralized finance (DeFi)
- Non-fungible tokens (NFT)
- Decentralized autonomous organizations (DAO)
- Decentralized social networks (DeSO)
- Decentralized identity (DI)

Reference

- https://hc.labnet.sfbu.edu/~henry/npu/classes/security/symmetric/slide/index_slide.html
- https://en.wikipedia.org/wiki/Block_cipher
- https://hc.labnet.sfbu.edu/~henry/npu/classes/building_blockchain_projects/smart_contract/smartcard_for_beginner/index_slide.html
- Network Security Essentials, Applications and Standards, Second Edition
- https://medium.com/blockchannel/life-cycle-of-an-ethereum-transaction-e5c66bae0f6e
- https://consensys.net/

Thank You!