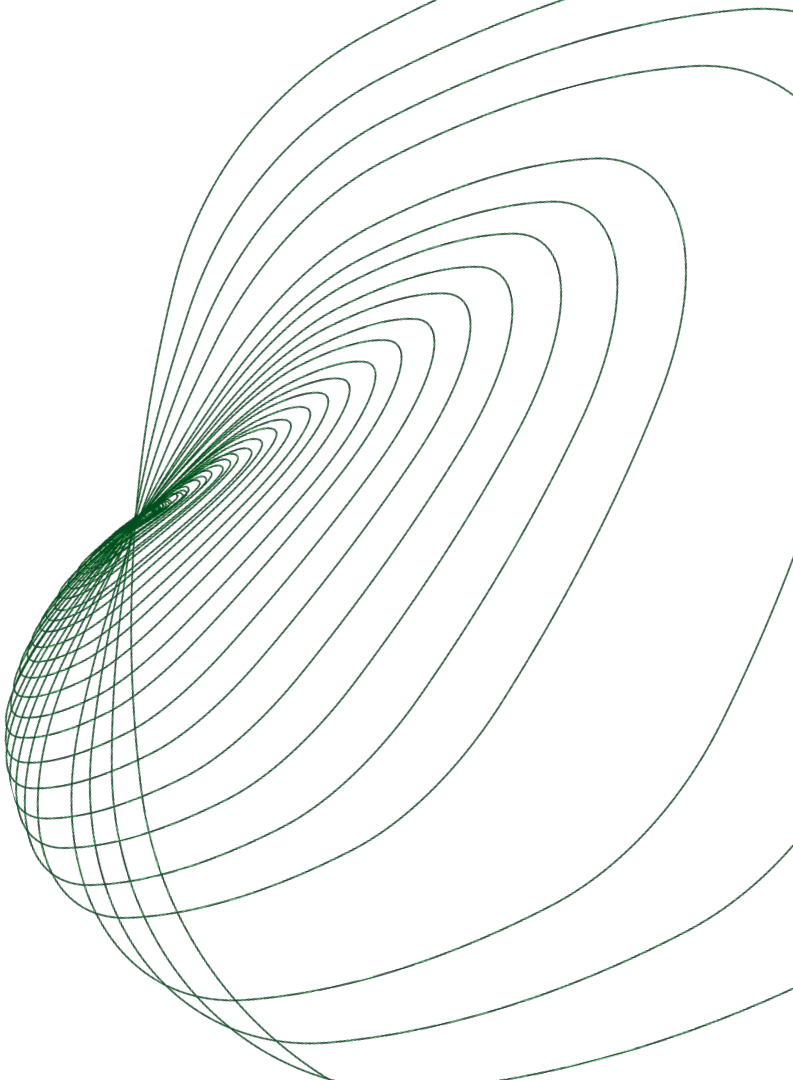
PAXOS

Deploying Dollars on the Blockchain

Fintech Devcon 2021

https://git.io/digitaldollars

September 8th, 2021



Today's Topics

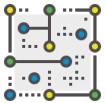
- Overview of what we'll learn
- Repo Setup
- Application Overview
- Deploying the Smart Contract
- Mint/Withdrawals
- Signing a Transaction

Deploying Dollars on the Blockchain

A whirlwind tour of the Ethereum blockchain, crypto signing, tokenized assets, and web3

What We'll Learn

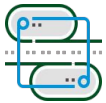
By the end of this workshop you should be able to explain to your friends about...



Ethereum Basics

What is a blockchain?

How does Ethereum differ from Bitcoin?



Smart Contracts

What does it mean to deploy code to Ethereum?

What functionality does this unlock?

Transactions

What does it mean to sign a transaction?

How do you call smart contract functions?

Web3 Integrations

As a bonus, we will preview how you can interact with live blockchain based applications.

Repo Setup

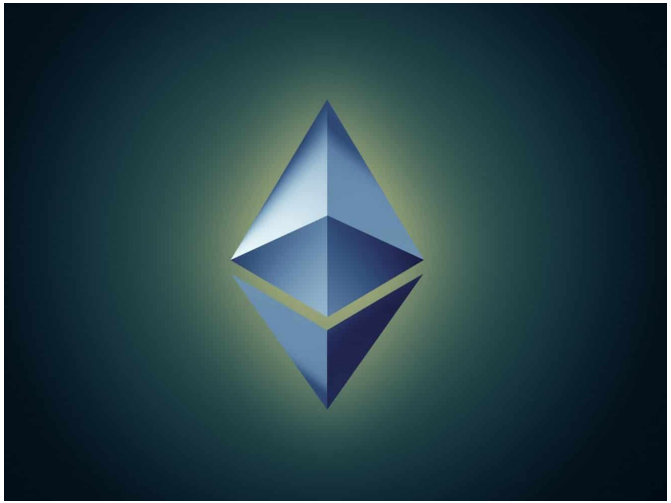
To get started with this workshop

1. Fork the workshop repo: https://git.io/digitaldollars (https://github.com/paxosglobal/fintech-devcon-2021-stablecoin-workshop)
2. Install the dependencies (go, docker, and node)
3. Checkout the `exercise-1` branch
4. Open the code in your IDE

Application Features

We'll build an app that can do the following:

- **Deposit** USD into the application
- **Mint** USDK as part of the withdrawal flow
- **Withdraw** USDK to a provided Ethereum address
- **Reconcile** expected USDK balances with on-chain amounts



1. Fork https://git.io/digitaldollars 2. Install Dependencies 3. Checkout the exercise-1 branch

Application Overview

Main areas to focus on in the repo

Webapp

The webapp exists to make interacting with the application easier.

We will not focus on the code here.

Smart Contract

The smart contract is in `contracts`. USDK uses standard ERC20 implementations by openzeppelin.

API Server

The server code is in `pkg/server`.

Most of the workshop will focus on this package.

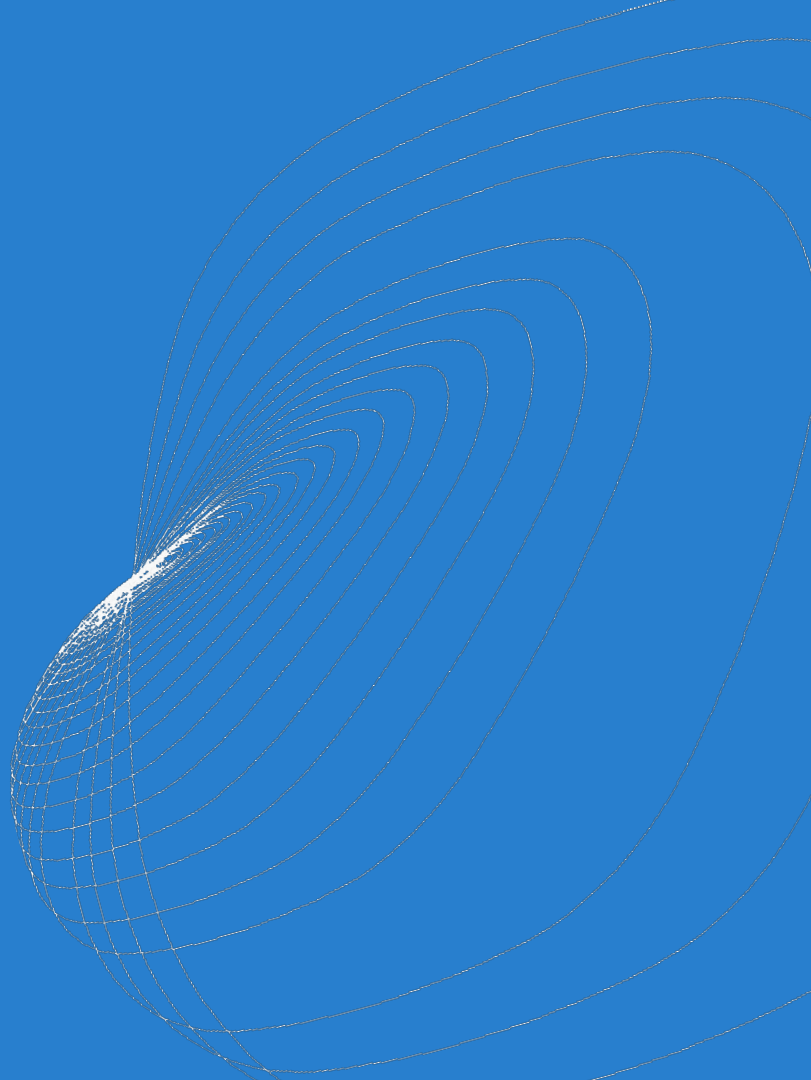
Local Blockchain

Ganache is deployed via Docker for a local Ethereum blockchain.

Our application will interact with Ganache.

1. Fork https://git.io/digitaldollars 2. Install Dependencies 3. Checkout the exercise-1 branch

Bitcoin vs. Ethereum



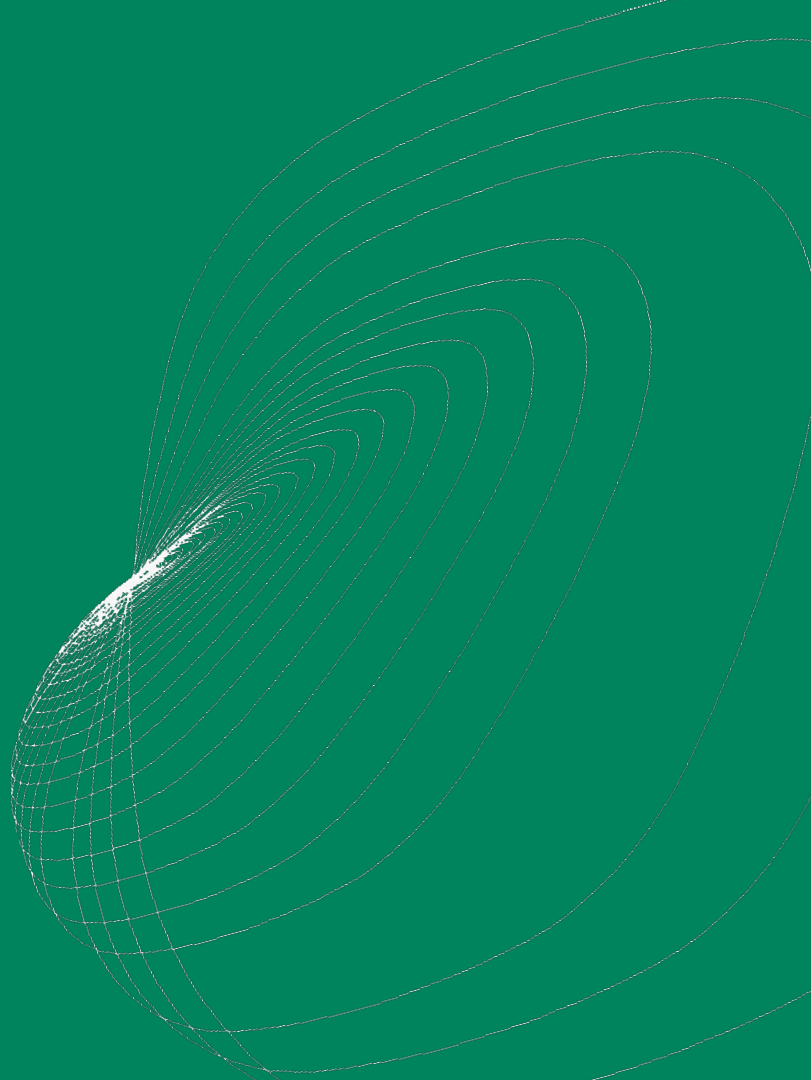
PAXOS

Bitcoin Vs. Ethereum

Ethereum built on top of Bitcoin's innovations

	Bitcoin	Ethereum
Decentralized Blockchain	✔	✔
Non-native Tokens Enables Stablecoins Etc.	✔	✔
Turing Complete Fully Programmable	x	✔

Smart Contracts



PAXOS

What is a smart contract?

Smart Contracts are collections of functions with a distributed DB

- Can store **state** such as address balances and roles
- Has **public** and **private** methods
- Can **call** other smart contracts
- Contract calls are executed by all Ethereum nodes deterministically updating contract state

Smart contracts enable

lending protocols, token exchanges, NFTs etc

Smart Contract Features

USDK is built on top of tested and audited standards created by OpenZeppelin

	1	2	3	4
Description	ERC20.sol	Ownable.sol	SafeMath.sol	USDK.sol
Feature 1	Adheres to ERC20 standard	Adds administrative features	Wraps solidity arithmetic operations	is ERC20 & Ownable
Feature 2	Enables Transfers	Can set owner of the contract	Protects against overflows/underflows	Uses SafeMath
Feature 3	Stores Balances	Can change the owner	Protects against division by zero	Exposes mint/burn to owner

Exercise 1

Finish the smart contract and deploy it

* If you have extra time you can research how you would deploy this to testnet

Exercise 1 Code

Hint: Smart Contracts can call functions on contracts they inherit

```
17      /** @dev Creates `amount` tokens and assigns them to `account`, increasing
18       * the total supply. This is done after dollars are deposited into the reserve.
19       *
20       * Emits a {Transfer} event with `from` set to the zero address.
21       *
22       * Requirements:
23       *
24       * - `account` cannot be the zero address.
25       */
26      function mint(address account, uint256 amount) public onlyOwner {
27          // TODO: fill out this function for exercise 1!
28          // HINT: Smart Contracts can call functions on contracts they inherit. This contract imports
29          // [ERC20.sol](https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/ERC20.sol),
30          // [Ownable.sol](https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable.sol), and
31          // [SafeMath.sol](https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/math/SafeMath.sol).
32      }
```

Compile USDK

Run `make contract-bindings` to compile and write go bindings

```
13  contract-bindings: SHELL:=/usr/bin/env bash
14  contract-bindings: contracts/USDK.sol
15          npm install
16          npm run compile
17          cat build/contracts/UsdToken.json | ./node_modules/node-jq/bin/jq -c .abi > build/USDK.abi
18          cat build/contracts/UsdToken.json | ./node_modules/node-jq/bin/jq -r .bytecode > build/USDK.bin
19          docker run -v $(shell pwd):/sources ethereum/client-go:alltools-v1.10.6 abigen --type USDK \
20                  --bin="/sources/build/USDK.bin" \
21                  --abi="/sources/build/USDK.abi" \
22                  --pkg=contracts --out="/sources/build/USDK.go"
```

Exercise 1 Code

Fill in the main function in cmd/deploy/main.go

Hint: make sure `make start-local` is running in another terminal

```
41  func main() {
42          ethClient, err := ethclient.Dial(GanacheNetworkAddr)
43          if err != nil {
44                  panic(err)
45          }
46          // TODO: fill out this function for exercise 1!
47  }
```

Exercise 1 SOLUTION

Smart Contracts can call functions on contracts they inherit

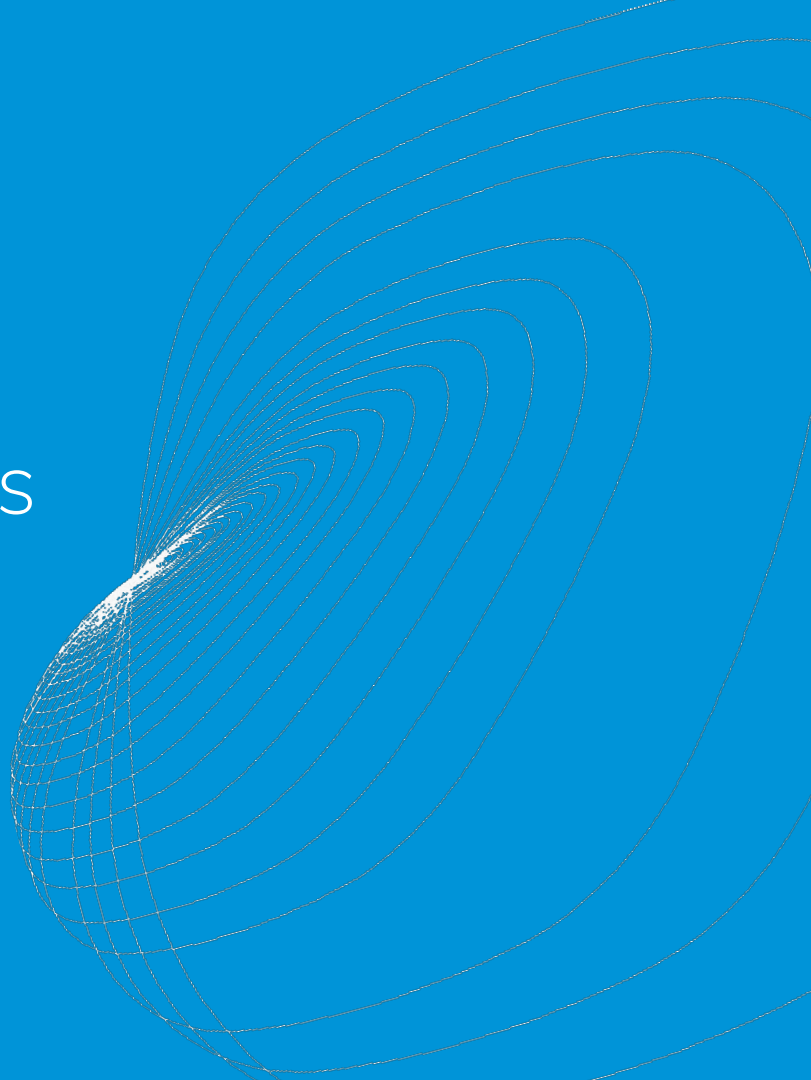
```
16      /** @dev Creates `amount` tokens and assigns them to `account`, increasing
17       * the total supply. This is done after dollars are deposited into the reserve.
18       *
19       * Emits a {Transfer} event with `from` set to the zero address.
20       *
21       * Requirements:
22       *
23       * - `account` cannot be the zero address.
24       */
25      function mint(address account, uint256 amount) public onlyOwner {
26          return _mint(account, amount);
27      }
```

Exercise 1 SOLUTION

Make sure `make start-local` is running in another terminal

```
10  func main() {
11          ethClient, err := ethclient.Dial(server.GanacheNetworkAddr)
12          if err != nil {
13                  panic(err)
14          }
15          addr, _, _, err := contracts.DeployUSDK(server.OwnerTransactor, ethClient)
16          if err != nil {
17                  panic(err)
18          }
19          log.Print("contract address: ", addr) // 0xc4680463046E64b10Da390d9049D24b8EC43AaAB
20  }
```

Smart Contract Interactions



PAXOS

What's in an Ethereum Transaction?

Nonce:	Incrementing counter - number of txs previously sent
To:	Destination address (note: often a contract address)
Gas:	A number of fields that define gas limitations for the tx
Amount:	Amount of ETH transferred in this transaction
Sig Data:	V, R, & S fields that make up the signature for the transaction
Data:	Additional data including data to send to the smart contract

Decoding Transaction Data

Example Data:
0xa9059cbb000000000000000000000000b85d233efaa52d928e4aae460610365ab462b8da0
0000000000000000000000000000000000000000000006f05b59d3b2000000

0xa9059cbb -> This hex value is derived from taking the method name and its argument types, removing any whitespace, taking the keccak hash of the result, and then taking the first 4 bytes of it and displaying it in hex

000000000000000000000000b85d233efaa52d928e4aae460610365ab462b8da ->
32 byte hex of the “to” address

0000000000000000000000000000000000000000000006f05b59d3b2000000 ->
32 byte hex of the amount to send

Special Report:

Good News

Using contract bindings in Go/JS abstracts this away

Contract Methods	Bindings expose methods for Contract functionality
Arguments	Go's typing system is integrated
Helper Functions	Bindings include helpers like "WatchTransfer"

```
tx, err := USDK.Transfer(to, amount)
```

PAXOS

Exercise 2

Create a mint transaction

Exercise 2 Code

Fill in the `mintWithBindings` function in pkg/server/mint.go

Hint: use the helper functions within mint.go

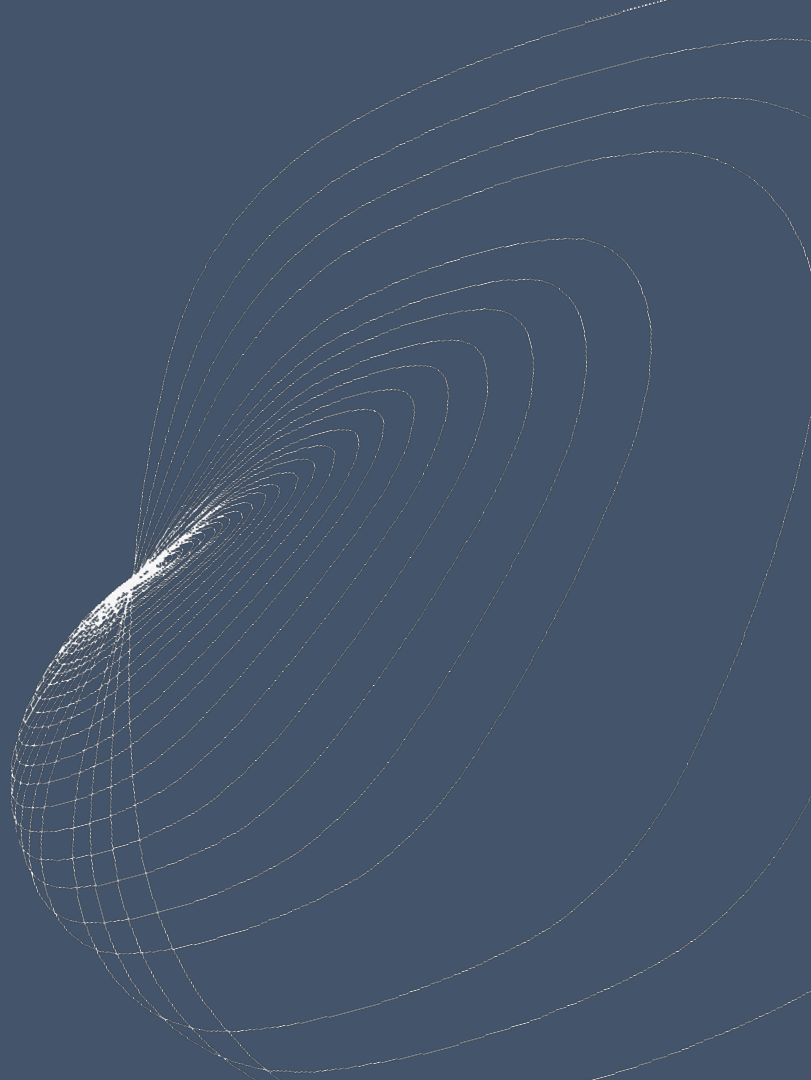
```
53  func (s *Server) mint(destination string, amount decimal.Decimal) error {
54          return s.mintWithBindings(destination, amount)
55  }
56
57  func (s *Server) mintWithBindings(destination string, amount decimal.Decimal) error {
58          // TODO: fill out this function for exercise 2!
59          // test it with `go test ./pkg/server -run TestMint`
60          return nil
61  }
```

Exercise 2 SOLUTION

To support “Hot” signing when you have active access to a private key, you can use the Mint function to create, sign and broadcast the transaction in one method call.

```
60  func (s *Server) mintWithBindings(destination string, amount decimal.Decimal) error {
61          usdkBindings, err := s.getUSDKBindings()
62          if err != nil {
63                  return err
64          }
65          _, err = usdkBindings.Mint(OwnerTransactor, addrToGethAddr(destination), decimalToBigInt(amount))
66          return err
67  }
```

Signing Transactions



PAXOS

Cryptographic Signing Basics

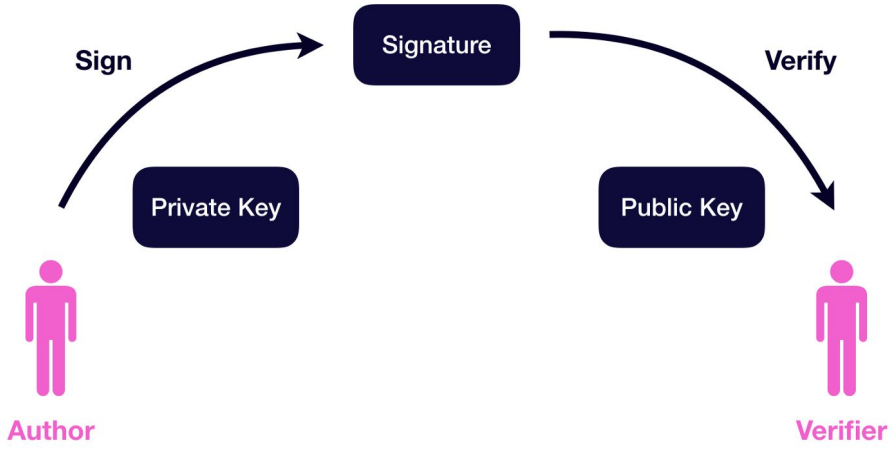
Ethereum signatures uses ECDSA and secp256k1 constants to define the elliptic curve

Asymmetric Key Features

Author can sign a message with their private key

Verifier can use the signature to recover the public key (and the Author's address)

Any change to the message or signature will result in a different public key



Broadcasting A Transaction

You just send the transaction to any Ethereum node and it eventually gets mined!

1. Send the transaction to an Ethereum Node
2. The node communicates to peer nodes who send it throughout the network
3. A miner picks up the transaction and mines it into a block
4. The new block is published back to the network

Blockchain Mining

Blockchain mining is what creates new transactions

- **Fees**. Miners choose transactions that pay them the highest fees. Choosing a higher gas fee leads to faster mining.
- **Hashpower**. Miners try to find the right random hash by creating many hashes until they randomly find one that fits the criteria of the chosen mining problem of the blockchain. The first one to solve it gets the fees and creates the next block

Blockchain mining is brute force protection against changing history

To change history one would have to mine alternative blocks to create a blockchain that makes sense, which would be very expensive in terms of hashpower.

Exercise 3

Explicitly Sign A Mint Transaction

Exercise 3 Code

```
68
69  func (s *Server) mintWithExplicitSigning(destination string, amount decimal.Decimal) error {
70      // ctx := context.Background()
71      // TODO: fill out this function for exercise 3!
72      // test it with `go test ./pkg/server -run TestMint`
73      return nil
74  }
75
```

Exercise 3 SOLUTION

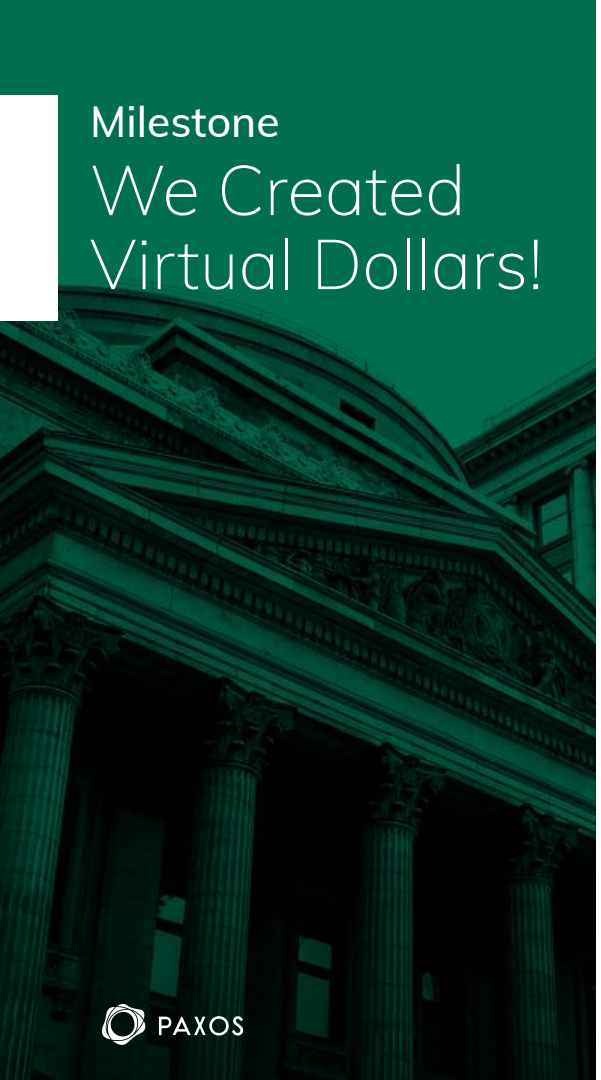
Create a Transaction, Sign it, and Broadcast it

To support separate signing

This pattern allows you to take the unsigned transaction and send it over the wire or offline for safer signing.

Cold signing is where you sign in an environment not connected to the internet, using a key that isn't available from the internet.

```
68
69  func (s *Server) mintWithExplicitSigning(destination string, amount decimal.Decimal) error {
70      ctx := context.Background()
71      x, err := s.createMintTransaction(ctx, destination, amount)
72      if err != nil {
73          return err
74      }
75      signedTx, err := signTransaction(OwnerTransactor, x)
76      if err != nil {
77          return err
78      }
79      err = s.Broadcast(ctx, signedTx)
80      if err != nil {
81          return err
82      }
83      return nil
84  }
85
```



Milestone

We Created Virtual Dollars!

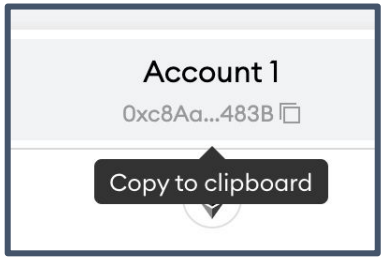
We have all the pieces to get dollars on the blockchain. Now let's see them in a user wallet!

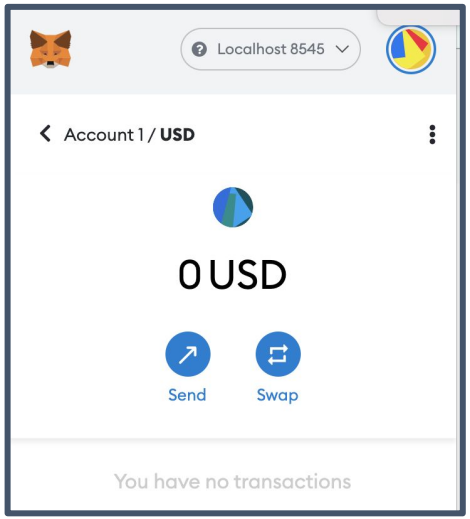
Smart Contract	We created a token contract and deployed it.
Transactions	We created token balances by signing and broadcasting a "mint" transaction.
External Wallets	Now we can try looking at the dollars in a metamask wallet and move them around.

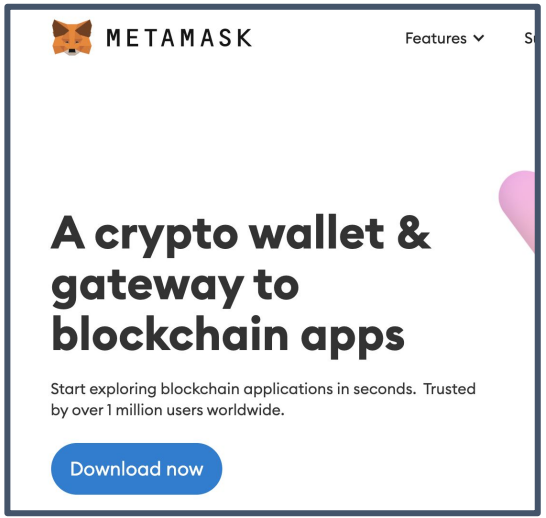
PAXOS

Exercise 4: Use MetaMask To Receive Tokens!

- **Install** for chrome at https://metamask.io/
- **Open the extension** and click “Get Started”
- **Create a Wallet** with a password
- **Switch Network** to Localhost
- **Add the token** in Assets
- **Find Your Address** in Accounts







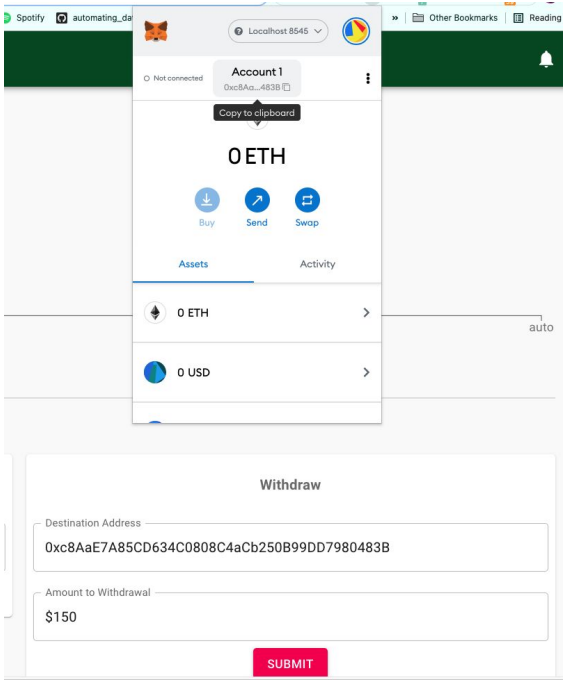
Exercise 4: Full Flow

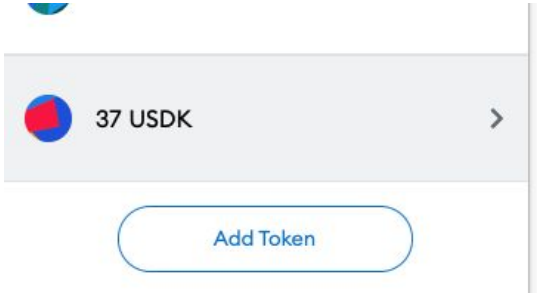
Three Terminals

```
r) make start-local
```

```
r) x make run-backend
```

```
er) make run-frontend
```





Other Topics We didn't Cover

Handling Token Deposits

Monitoring blocks for transactions that involve designated deposit addresses

Reserve Banking

Integrating and automating interactions with reserve banks

Node Infrastructure

Running multiple node implementations for reliability and resilience

Fiat Network Integrations

Handling fiat deposits and withdrawals and enabling autoconversion between fiat and crypto rails

PAXOS

Thank you

GET IN TOUCH - We're Hiring!

Engineering Manager
bperreault@paxos.com

Engineering Manager
igitter@paxos.com