DLT

DISTRIBUTED LEDGER TECHNOLOGY

**BLOCKCHAIN**

Guy R. Derenoncourt

University of San Diego

MS Cyber Security Engineering  
CYBR 504 – Applied Cryptography

Dr.

Dr. Gordon W. Romney

April 20, 2020

Table of Contents

1. Executive Summary 2

2. Introduction 2

2.1 Project Description 2

2.2 Technology Overview 2

2.3 Deployment Strategy 4

2.4 Learning Objectives 5

**3. Implementation**

3.1 Solidity Token

4. Demonstration

4.1 Blockchain "TEST NET"

5. Pros Vs. Cons

5.1 Consensus Protocols

5.2 Digital Signatures

6. Conclusions and Recommendations

6.1

References

Appendix A – Infrastructure

A.1 Baseline Hardware and Software

A.2 Sandbox Network

Appendix B - Project Plan

**Appendix C - Screenshots**

# Executive Summary

Blockchain is considered a sweeping change because it triggered the re-think and engineering of modern financial instruments. As a result, Blockchain emerged to become the most successful case use implementation of Distributed Ledger Technology (DLT).

The core concept of Blockchain is rooted in a distributed ledger where records are stored and secured with cryptography, and each block, is chained together to form an immutable ledger. It is important to denote that all blockchains are DLT’s, but not all DLT’s are blockchains.

Payments and remittances are the first-born and most mature application of blockchain technology; nevertheless the technology has evolved to support other enterprise use cases. The focus of this project is to however experiment with workflows and applications that will form a core component of Blockchain and the IT infrastructure.

This project will show you how Azure services such as virtual machine scale sets, Virtual Network, Key Vault, Storage, Load Balancer, and Monitor can be quickly provisioned for the deployment of an efficient private Ethereum PoA blockchain where member banks can establish their own nodes. Blockchain key characteristics propose it is secure, transparent, fraud resistant, and cost-efficient.

# Introduction

The Blockchain distributed ledger must be commissioned across a network which can either be permissioned, meaning the identity of each participant is known, or permission-less whereas participants are anonymous. Participants of the network are referred to as nodes and each node contains a copy of the ledger hence the decentralized characteristic of Blockchain.

The mechanism by which transactions are submitted to the network is engineered on asymmetric cryptography. A private key or secret, is safeguarded by the network participant, then used to sign a submitted transaction. The public key is then used to validate the identity of the individual or organization who submitted the transaction. Furthermore, given the ledger is universal, another mechanism is to ensure that transactions are added in the same order and are universal across all ledgers; this is achieved through a consensus.

The prevalent advantages offered by blockchain renders it very suitable for application in financial services because it addresses expensive and time-consuming workflows. High fees, delays, complex workflows, and layers of middleman have been implied characteristics of the conventional financial system. Nonetheless, the global banking system serves more than a billion people each day. It enables trade, enterprises, start-ups, and is unquestionably an integral and indispensable service to our modern lives.

## Project Description

This example scenario is useful for banks or any other institutions that want to establish a trusted environment for information sharing without resorting to a centralized database. The architecture can be applied to any scenario where a consortium of organizations wants to share validated information with one another without resorting to the use of a central system ran by one single party.

Relevant use cases include:

* Movement of allocated budgets between different business units of a multinational corporation
* Cross-border payments
* Trade finance scenarios
* Loyalty systems involving different companies
* Supply chain ecosystems.

The following is used to develop the project:

* Ganache Private Blockchain Server—https://trufflesuite.com/ganache/
* Trufflesuite—https://github.com/trufflesuite/truffle
* MetaMask plugin for Chrome/Firefox/Safari—https://metamask.io/

## Technology Overview

Based on the results of an initial investigation, the following requirements were established:

1. Application must run virtually on a platform provided by a Cloud Service Provider

2. Database must employ a minimum of five tables, using some form of database management system. Data must be in 3rd Normal Form (3NF) and employ foreign keys.

3. Application must be user-friendly and accessible via an internet connection.

This scenario covers the back-end components that are necessary to create a scalable, secure, and monitored private, enterprise blockchain network within a consortium of two or more members. Details of how these components are provisioned (that is, within different subscriptions and resource groups) as well as the connectivity requirements (that is, VPN or ExpressRoute) are left for your consideration based on your organization's policy requirements. Here's how data flows:

1. Bank A creates/updates an individual's credit record by sending a transaction to the blockchain network via JSON-RPC.
2. Data flows from Bank A's private application server to the Azure load balancer and subsequently to a validating node VM on the virtual machine scale set.
3. The Ethereum PoA network creates a block at a preset time (2 seconds for this scenario).
4. The transaction is bundled into the created block and validated across the blockchain network.
5. Bank B can read the credit record created by bank A by communicating with its own node similarly via JSON-RPC.

### Components

* Virtual machines within virtual machine scale sets provides the on-demand compute facility to host the validator processes for the blockchain
* Key Vault is used as the secure storage facility for the private keys of each validator
* Load Balancer spreads the RPC, peering, and Governance DApp requests
* Storage hosting persistent network information and coordinating leasing
* Operations Management Suite (a bundling of a few Azure services) provides insight into available nodes, transactions per minute and consortium members

### Cloud Services

Microsoft Azure is a complete cloud platform able to host existing applications and streamline new application development. Azure can also enhance on-premises applications. The Azure portal allows for the management of Azure services. Services are also made available programmatically by using service-specific APIs and templates.

### Operating System

Linux (ubuntu 18.04) LTS release Server Edition provides a common, minimalist base for a variety of server applications, such as file/print services, web hosting, email hosting, etc. This edition supports five (5) major architectures: AMD64, ARM, POWER9, LinuxONE and z Systems, and introduces initial support for RISC-V.

### Operational Framework

Ganache is a personal blockchain for Ethereum development used to deploy contracts, develop applications, and run tests. It is available as both a desktop application as well as a command-line tool (formerly known as the TestRPC). Ganache is available for Windows, Mac, and Linux.

Ganache is considered a world class development environment, testing framework and asset pipeline for blockchains using the Ethereum Virtual Machine. The Ethereum Virtual Machine or EVM is the runtime environment for smart contracts in Ethereum. Code running inside the EVM has no access to network, filesystem, or other processes, it is entirely isolated.

There are two kinds of accounts in Ethereum which share the same address space: **External accounts** that are controlled by public-private key pairs and **contract accounts** which are controlled by the code stored together with the account. Every account has a persistent key-value store mapping 256-bit words called **storage**.

Furthermore, every account has a **balance** in Ether (in “Wei” to be exact, 1 ether is 10\*\*18 wei.) which can be modified by sending transactions that include Ether. The basic example of a simple smart contract sets the value of a variable and exposes it for other contracts to access.

Storage example:

**pragma solidity** >=0.4.16 <0.7.0;

**contract** SimpleStorage {

uint storedData;

**function** set(uint x) **public** {

storedData = x;

}

**function** get() **public** **view** **returns** (uint) {

**return** storedData;

}

}

### The Solidity Compiler

A contract in the sense of Solidity is a collection of code (its *functions*) and data (its *state*) that resides at a specific address on the Ethereum blockchain. The line uint storedData; declares a state variable called storedData of type uint (*u*nsigned *int*eger of *256* bits).

### Subcurrency Example

The example contract code below implements the simplest form of a cryptocurrency. The contract allows only its creator to create new coins (different issuance schemes are possible). Anyone can send coins to each other without a need for registering with a username and password, all you need is an Ethereum keypair.

**pragma solidity** >=0.5.0 <0.7.0;

**contract** Coin {

*// The keyword "public" makes variables*

*// accessible from other contracts*

address **public** minter;

**mapping** (address => uint) **public** balances;

*// Events allow clients to react to specific*

*// contract changes you declare*

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

*// Constructor code is only run when the contract*

*// is created*

**constructor**() **public** {

minter = msg.sender;

}

*// Sends an amount of newly created coins to an address*

*// Can only be called by the contract creator*

**function** mint(address receiver, uint amount) **public** {

require(msg.sender == minter);

require(amount < 1e60);

balances[receiver] += amount;

}

*// Sends an amount of existing coins*

*// from any caller to an address*

**function** send(address receiver, uint amount) **public** {

require(amount <= balances[msg.sender], "Insufficient balance.");

balances[msg.sender] -= amount;

balances[receiver] += amount;

**emit** Sent(msg.sender, receiver, amount);

}

}

### Front-end Application Development Environment.

MetaMask is an Ethereum browser wallet and which can easily be installed as an extension for Google Chrome or Firefox. The applet displays notifications, communicate with cooperating websites, and can also modify data. The application can be accessed via new browser tabs, a **window.ethereum** object available in the console. MetaMask enables users to manage accounts and their keys in a variety of ways, including hardware wallets, while isolating them from the site context. This is considered a security enhancement given storing the user keys on a single central server, or even in local storage can inadvertently result in mass account thefts.

MetaMask Documentation—https://docs.metamask.io/guide/

## Deployment Strategy

To work and interact with smart contracts, they must be migrated to a secure test blockchain.

Migrations carry out the following tasks:

1. Deploy the compiled contract code to the blockchain.
2. Establish interlinking between dependent contracts.
3. Initialize the initial values through the constructor.
4. Manage the different versions of the contracts deployed.

In the traditional model, every time a contract is deployed, a new Ethereum address is generated that then needs to be updated to the code of the blockchain application.

Truffle allows to abstract this concept and invoke the contract directly through a contract object instead of its address.

Once online, the Ganache blockchain can be used to deploy smart contracts. The Ganache test server commonly runs on **localhost:8545**.

## Learning Objectives

To understand and interact with ERC20 and ERC721 contract standards.

ERC20 tokens have been implemented in multiple use cases such as payment tokens, loyalty coins, gift cards, and so on but their most popular implementation by far is as Initial Crypto Offering (ICO) tokens.

ERC 721 is the token standard used to build smart contracts that issue Non-Fungible Tokens (NFTs). Popular implementations include government documents, land titles, digital identities, and real estate.

The OpenZeppelin Solidity library provides smart contract templates for developing Decentralized Applications (DApps). The OpenZeppelin framework consists of reusable contract code for Ethereum and other Ethereum Virtual Machine (EVM).

# Implementation

There are different implementation strategies to implement a blockchain network:

* All stakeholders own and maintain a node on the blockchain network.
* A cloud-hosted blockchain operated by a trusted service provider which is also known as Blockchain as a Services (BaaS).
* Accessing a public permission less blockchain data so that it cannot be tampered with. A good example is storing the hash signature of a document to the public Bitcoin or Ethereum network. The transaction ID and the document is then shared with other stakeholders who need to validate the authenticity of the document in the future.

The Ethereum blockchain (often called "mainnet", for "main network") requires a hard currency to use it, in the form of Ether (its native currency).

To solve this, several "testnets" (for "test networks") exist: these include the Ropsten, Rinkeby, and Kovan blockchains. Testnets work very similarly to mainnet, with one difference: Ether can be harnessed from testnets without incurring a financial burden. However, you will still need to deal with private key management, blocktimes in the range of 5 to 20 seconds, and getting this free Ether.

During development, it is a better idea to instead use a *local* blockchain. It runs on your machine, requires no Internet access, provides you with all the Ether that you need, and mines blocks instantly. These reasons also make local blockchains a great fit for [automated tests](https://docs.openzeppelin.com/learn/writing-automated-tests#setting-up-a-testing-environment).

## Create a token

Truffle Boxes are helpful boilerplates that contain helpful modules, solidity contracts and libraries. The MetaCoin box is an example of box (project template) used to create a token that can be used between accounts.

1. Create new directory for truffle project i.e. mkdir MetaCoin and execute the command truffle unbox metacoin
2. To create a bare Truffle project with no smart contracts included in the directory use the command truffle init .

Once the above operation completes, a project structure with the following hierarchy emerges:

contracts/: Directory for Solidity contracts

migrations/: Directory for scriptable deployment files

test/: Directory for test files for testing your application and contracts

truffle.js: Truffle configuration file

## Run the Solidity test

Truffle has a built-in personal blockchain that can be used for testing. This blockchain is local to the system and does not interact with the main Ethereum network.

Once the blockchain is created, interaction occurs by using Truffle Develop.

## Compile Smart Contracts.

To compile a Truffle project, change to the root of the directory where the project is located and then type the following into a terminal **truffle init**.

Upon first run, all contracts will be compiled. Upon subsequent runs, Truffle will compile only the contracts that have been changed since the last compile.

Artifacts of compilation are automatically placed in the **build/contracts/** directory, relative to the project root.

## Reading and Writing Data

The Ethereum network makes a distinction between writing data to the network and reading data from it. writing data is called a **transaction** whereas reading data is called a **call**.

A fundamental characteristic of a transaction is that it writes (or changes) data. When a contract execution is triggered via a transaction or a contract's function, the function's return value of the transaction is not processed instantly. In general, functions executed via a transaction will not return a value; but will however return a transaction ID.

In summary, transactions:

* Cost gas (Ether)
* Change the state of the network
* Are NOT processed immediately
* Will not expose a return value (only a transaction ID).

## The Difference between calls and transactions

Calls can be used to execute code on the network, though no data will be permanently changed. Calls are free to run, and their defining characteristic is that they read data.

In summary, transactions:

* Are free (do not cost gas)
* Do not change the state of the network
* Are processed immediately
* Will expose a return value

## ERC 20 Token Contract.

This standard delivers basic functionality to transfer tokens, as well as permit tokens approval for the purpose of being spent by another on-chain third party. A standard interface allows any tokens on Ethereum to be re-used by other applications: from wallets to decentralized exchanges.

An ERC-20 token contract keeps track of fungible tokens: any one token is exactly equal to any other token; no tokens have special rights or behavior associated with them. This makes ERC20 tokens useful for things like a medium of exchange currency, voting rights, staking, and more.

pragma solidity ^0.4.24;

import "./IERC20.sol";

import "../../math/SafeMath.sol";

//\* @title Standard ERC20 token

contract ERC20 is IERC20 {

using SafeMath for uint256;

mapping (address => uint256) private \_balances;

mapping (address => mapping (address => uint256)) private \_allowed;

uint256 private \_totalSupply;

// \* @dev Total number of tokens in existence

function totalSupply() public view returns (uint256) {

return \_totalSupply;

function balanceOf(address owner) public view returns (uint256) {

return \_balances[owner];

}

function allowance(

address owner,

address spender

)

public spender

)

public

view

returns

## ERC 721 Token Contract

The ERC-721 Non-Fungible Token Standard is a standard interface for non-fungible tokens, also called deeds. The standard allows for the implementation of a standard API for NFT’s within smart contracts. The standard provides basic functionality to track and transfer NFTs.

This is the “ERC721 Metadata JSON Schema”

{

"title": "Asset Metadata",

"type": "object",

"properties": {

"name": {

"type": "string",

"description": "Identifies the asset to which this NFT represents"

},

"description": {

"type": "string",

"description": "Describes the asset to which this NFT represents"

},

"image": {

"type": "string",

"description": "A URI pointing to a resource with mime type image/\* representing the asset to which this NFT represents. Consider making any images at a width between 320 and 1080 pixels and aspect ratio between 1.91:1 and 4:5 inclusive."

}

}

}

## Cryptography

Elliptic Curve Digital Signature Algorithm (ECDSA) functions can be used to verify that a message was signed by the holder of the private key of a given address.

FUNCTIONS

**recover (hash, signature)**

**toEthSignedMessageHash(hash)**

# Demonstration

4.1 The local USD Blockchain **“TEST NET”** can be accessed via Remote Desktop. Please note the IP address may change as the server gets allocated and deallocated in Azure Portal. This server is not suitable for a Blockchain production environment.

A screenshot of a cell phone

Description automatically generated

4.2 Live demo request can be addressed via [gderenoncourt@sandiego.edu](mailto:gderenoncourt@sandiego.edu) .

# Pros Vs. Cons

Smart contracts have pros and cons and are unsuitable to solve every single security application challenge.

The pros of smart contracts are as follow:

* Accuracy
* Transparency
* Time efficiency
* Security
* Intermediaries

The cons of smart contract can be listed as:

* Legality
* Enforceability
* Government Regulation
* Taxation
* Standardization
* Errors in code

## Consensus Protocols

Consensus algorithms enable network participants to agree on the contents of a ledger in a distributed trust-less manner.

Proof of Work

Cryptocurrency such as Bitcoin and Litecoin use the concept of proof of work. The model necessitates miners competing among themselves to solve a complex cryptographic puzzle. During this process, the first miner to have solved the challenge broadcasts results to the rest of the network to achieve verification.

Once the proof of work is verified, the miner then receives a reward, a fraction of the coin.

Proof of Stake

Proof of Stake is another concept used. The model differs from Proof of Work in the sense that minors are not a requirement for verification and validation. The Proof of Stake concept replaces miners with forgers. They are selected in accordance with the amount of coins held which serves as a security deposit.

It is important to denote that forgers do not receive a reward or fee; they simply collect pooled network fees.

A screenshot of a cell phone

Description automatically generated

## Digital Signatures

Digital signatures are used to ensure:

* Authenticity
* Data integrity
* Non-repudiation

In the instance of Blockchain digital signatures provide the authenticity of source, guarantees the integrity of the data, and provides proof that a data exchange took place.

Digital signatures are based on combination of two cryptographic keys.

* A “private” key is used to sign the message.
* A “public” key is used to verify the signature.

A picture containing screenshot

Description automatically generated

# Conclusions and Recommendations

The distributed ledger framework has been a staple of computer networks for generations; however, it is the release of the Bitcoin Whitepaper (Nakomoto, 2008) that commanded intense interest and development in these systems. Currently, the market capitalization of digital assets stands at $274 billion (Coinmarketcap.com, July. 2019). It is expected for crypto assets to become more mainstream and a target of interest for cybercriminals.

Beebe and Rao (2005) laid out their revised situational crime prevention theory to explain information security in the digital age; the current paper expands on their ideas to accommodate the security issues found in the blockchain era. Some of these measures include encryption, public key identification, mainchain validation, usable chain with responsive community, and acceptable use policies.

The realm of systems engineering analyzes systems from the vantage point of “plausibility of a cyber-attack means it will occur.” The most discussed criminal act in the blockchain ecosystem is the 51% attack, it entails a group of miners who control over 50% percent of a blockchain hash rate. This majority control affords the ability to reverse transactions and permit double spending.

Several assumptions must be made to fully assess attack vectors in connected blockchain; nevertheless, the main factor to be taken in consideration is simply whether the risk, effort, and cost to break the network is worth the reward.

# References

Blasco, N. J., & Fett, N. A. (2019). Blockchain security: Situational crime prevention theory and distributed cyber systems. International Journal of Cybersecurity Intelligence & Cybercrime.

R.C. Merkle, "Protocols for public key cryptosystems," In *Proc. 1980 Symposium on Security and*

*Privacy*, IEEE Computer Society, pages 122-133, April 1980.

# Appendix A – Infrastructure

A picture containing sitting, small, large, holding

Description automatically generated

# Appendix B – Project Plan

A screenshot of a cell phone

Description automatically generated

# Appendix C – Screenshots

## C.1 Cloud Service Provider (CSP) – Microsoft Azure

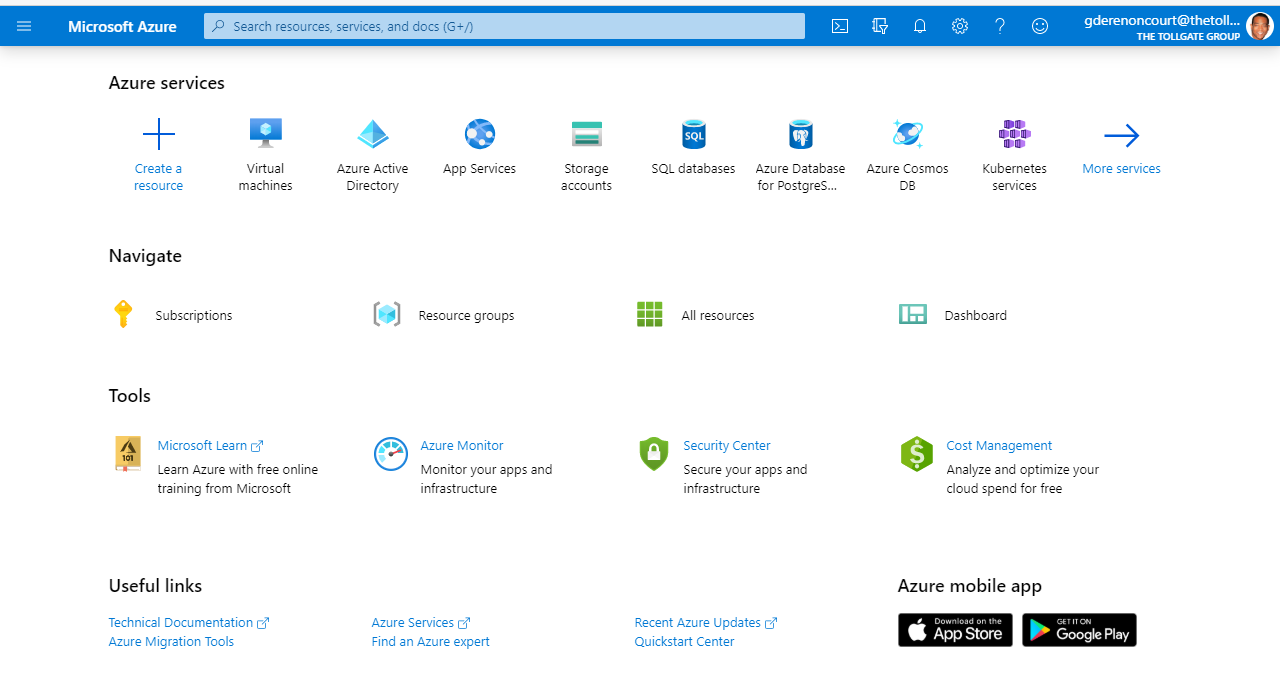


Figure C.1.1 - Creating the virtual machine and selecting the operating system to virtualize.

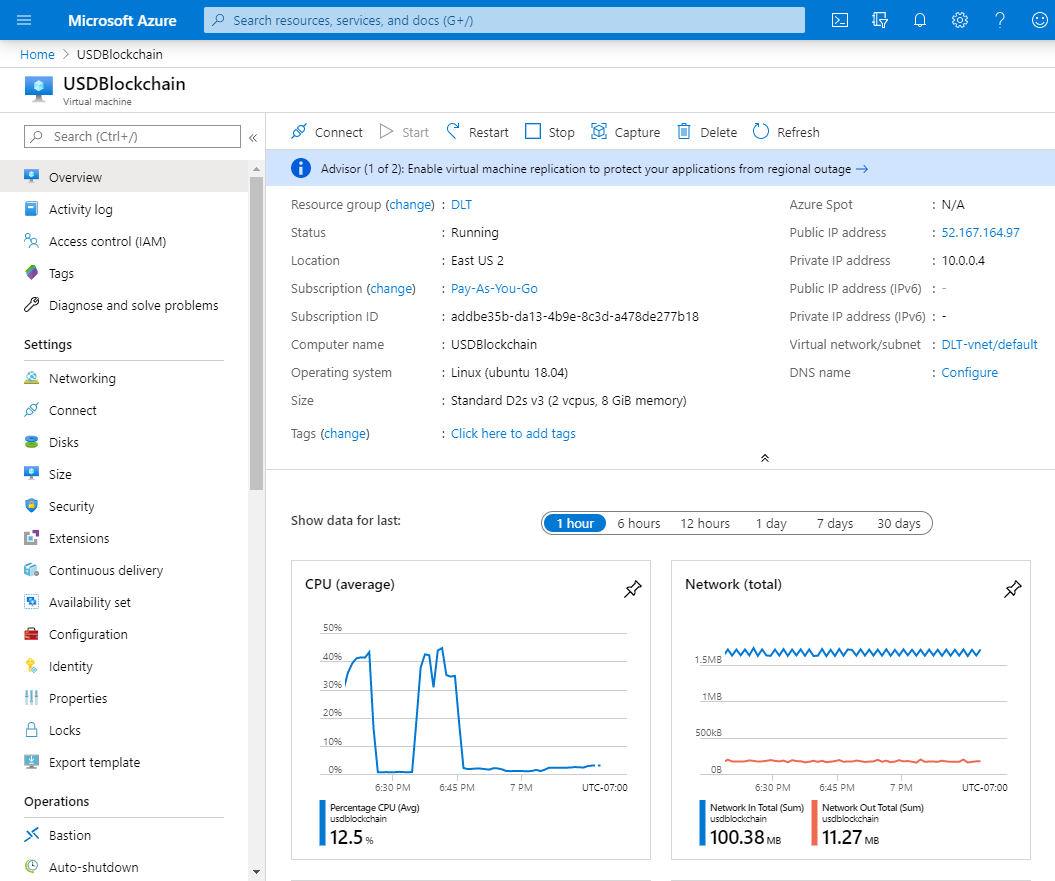


Figure C.1.2 – Controlling the virtual machine using Azure Portal.

## C.2 Application Support Framework – MetaMask

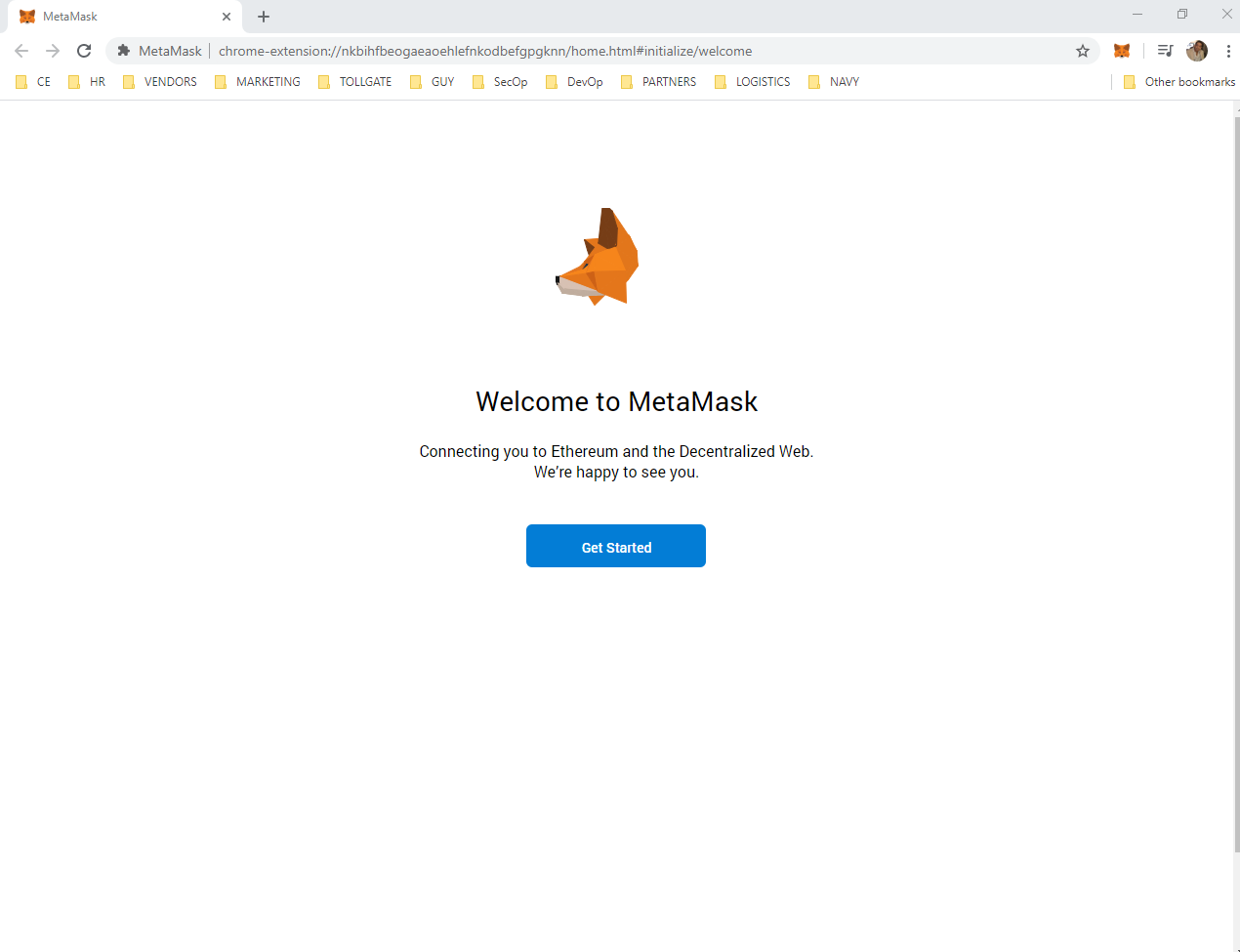


Figure 14.2.1 – MetaMask extension available for Chrome & Firefox

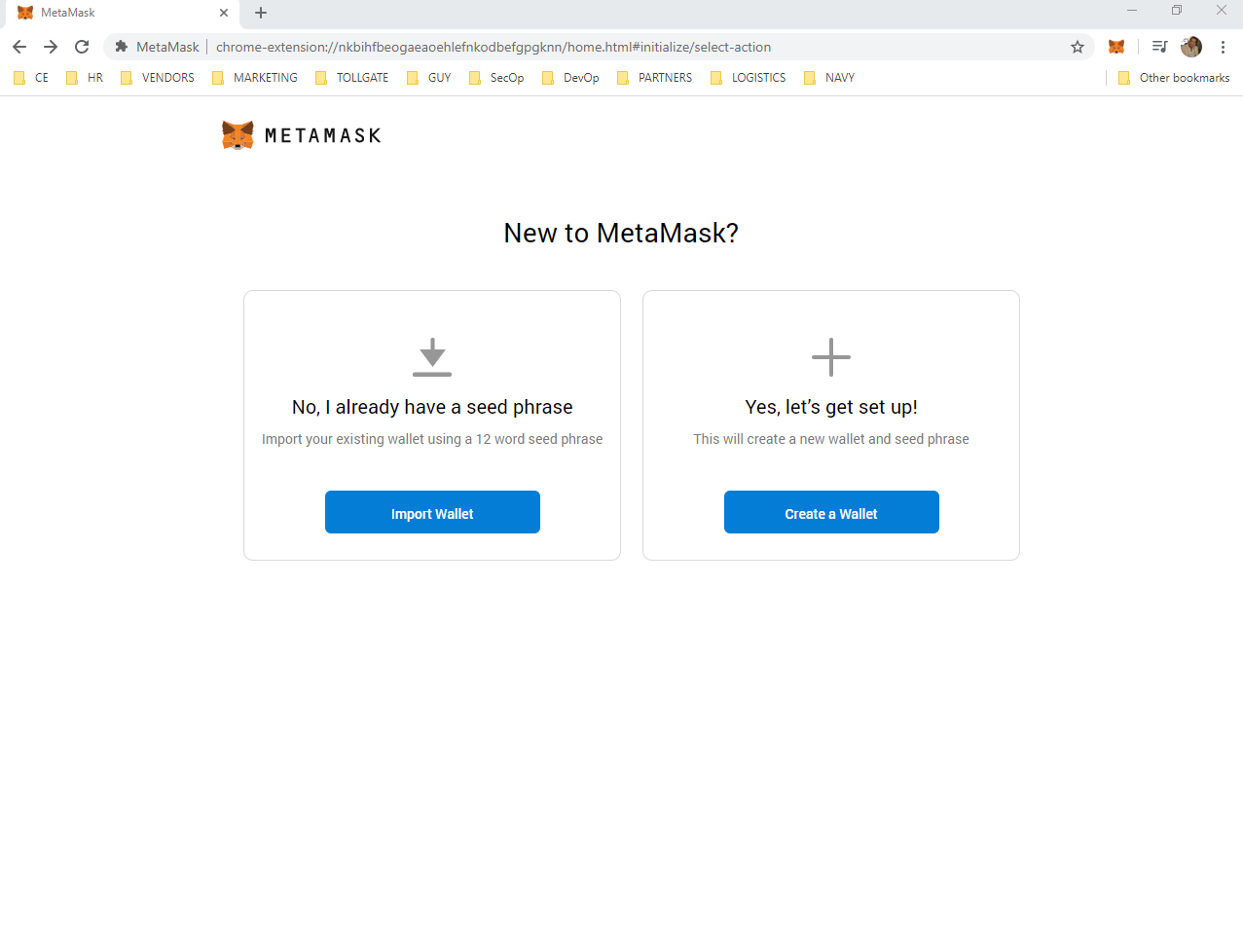


Figure 14.2.2 – Create a secure wallet or import one.

## C.3 Database Management.

The following screenshots are how to secure a crypto wallet to interact with Blockchain accounts using the MetaMask interface.

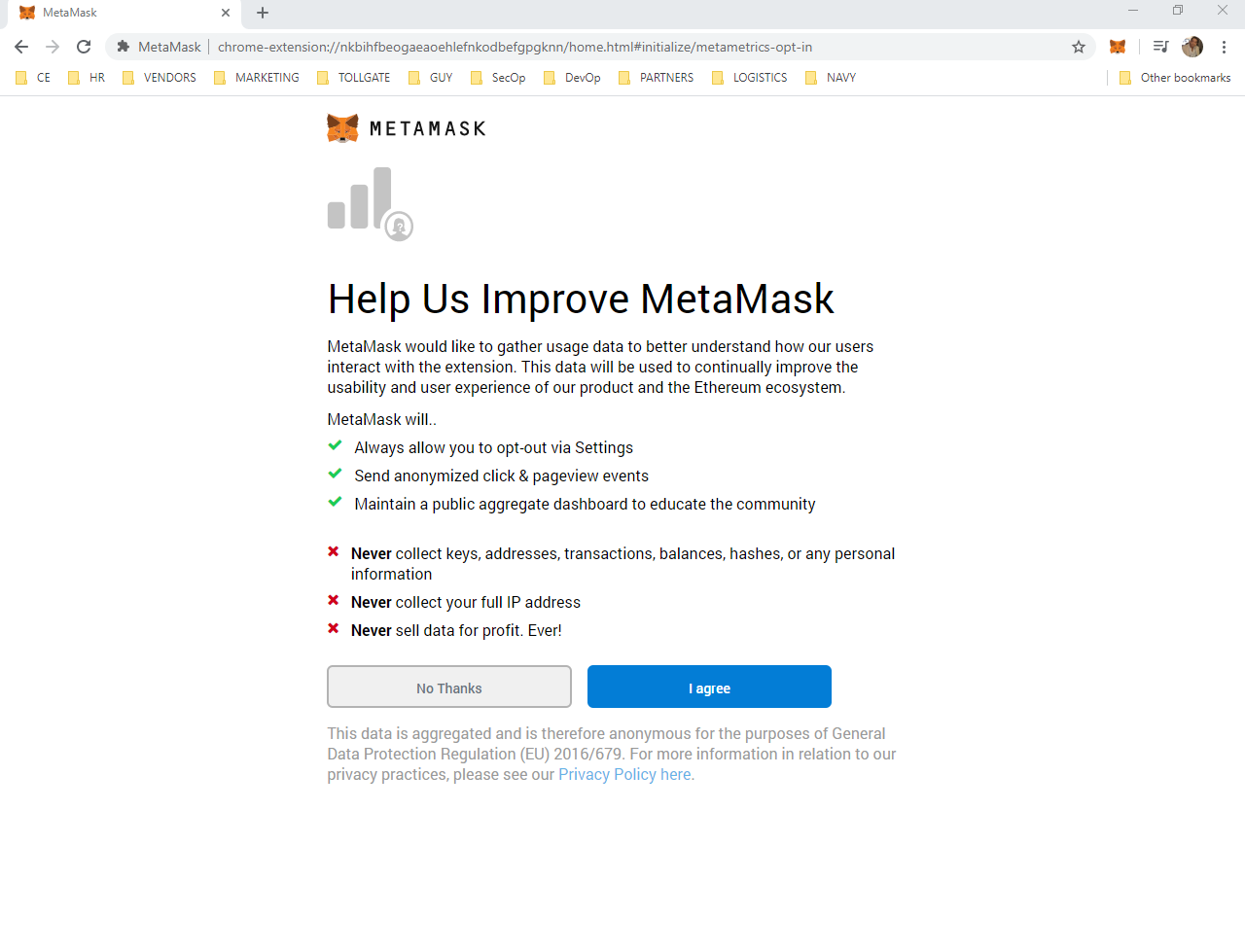


Figure 14.3.1 –Creating a password. This password does not provide access to the private key.

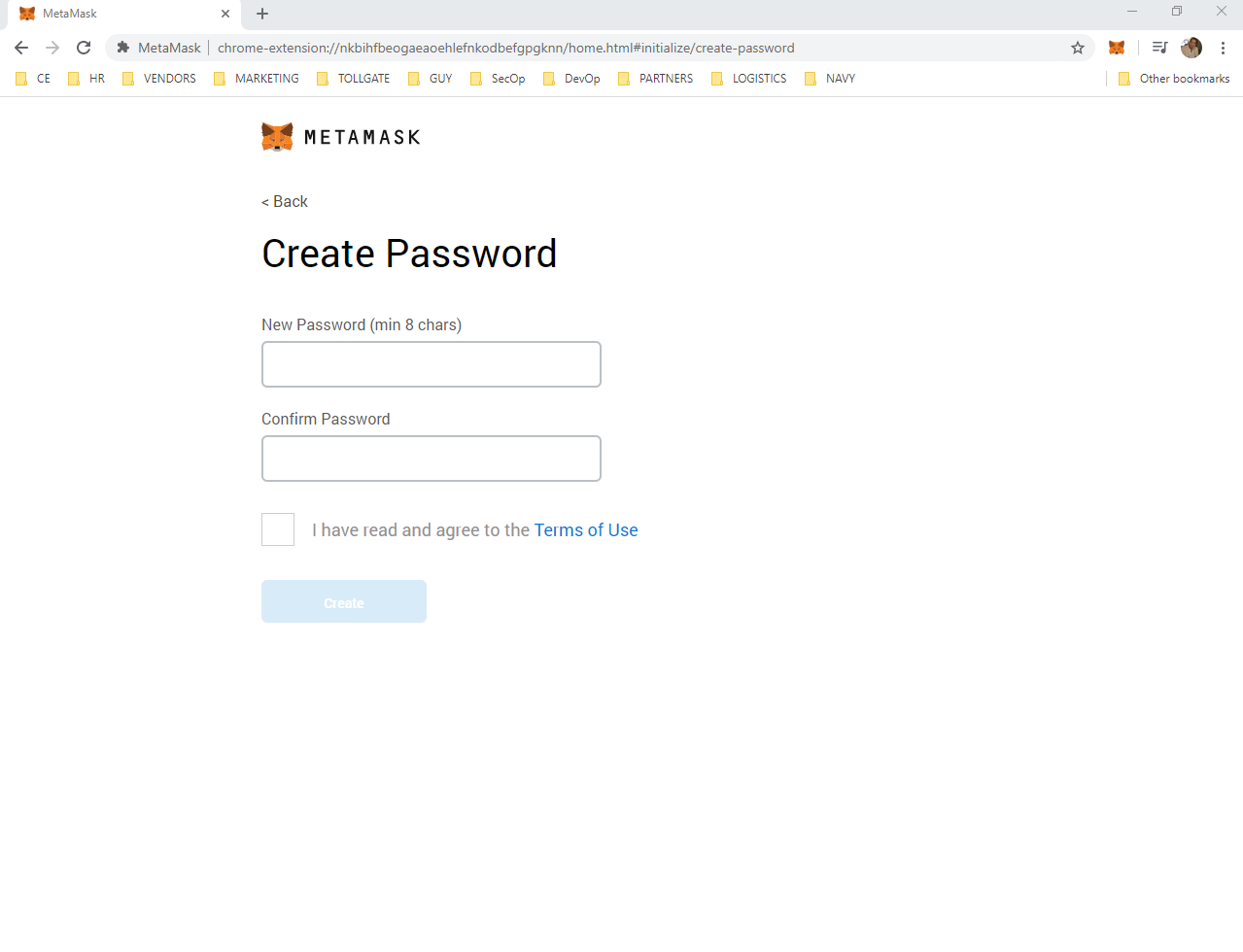


Figure 14.3.2 It is recommended to save the Secret Backup Phase on a different machine.

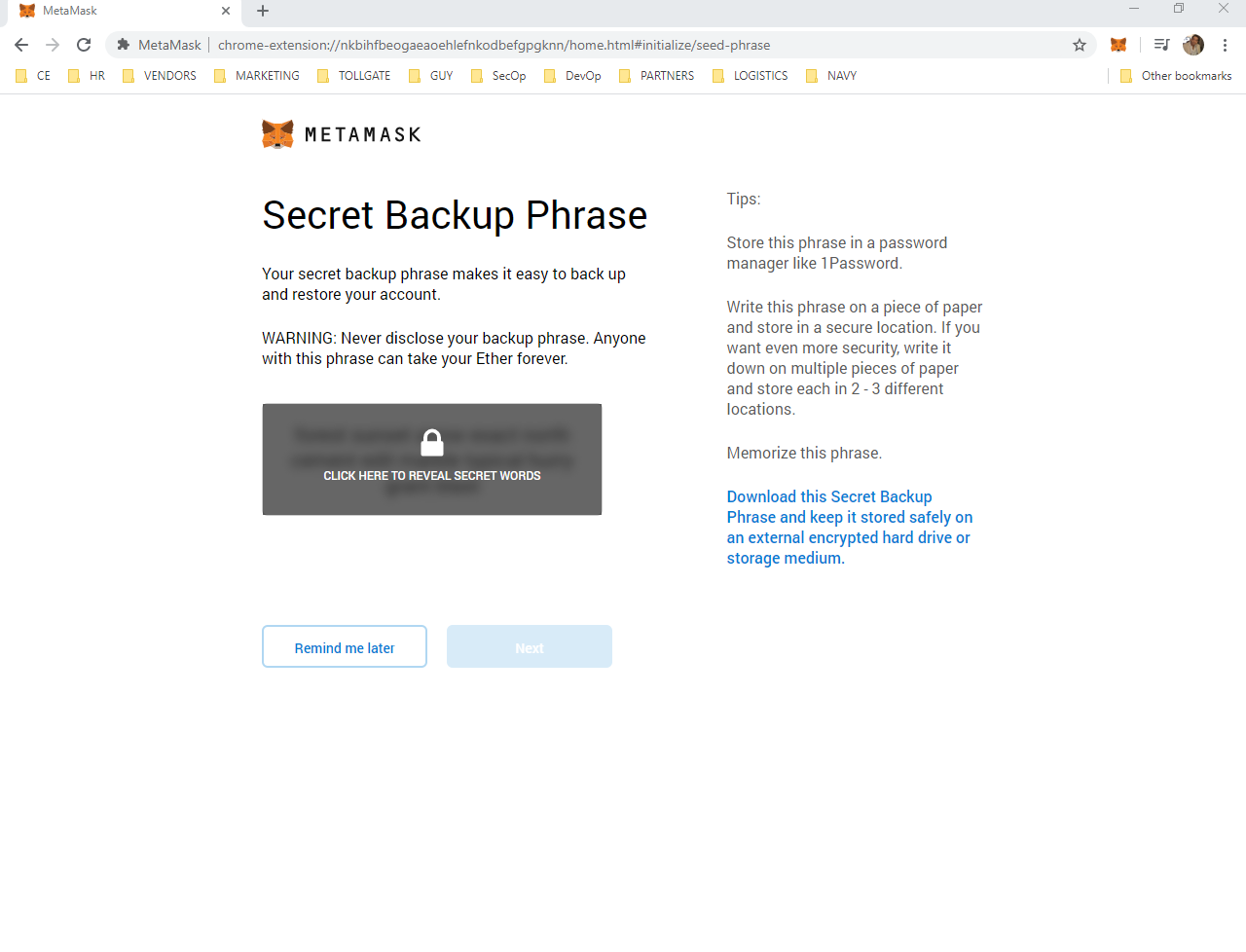


Figure 14.3.3 Keeping the seed phrase safe is the user’s responsibility

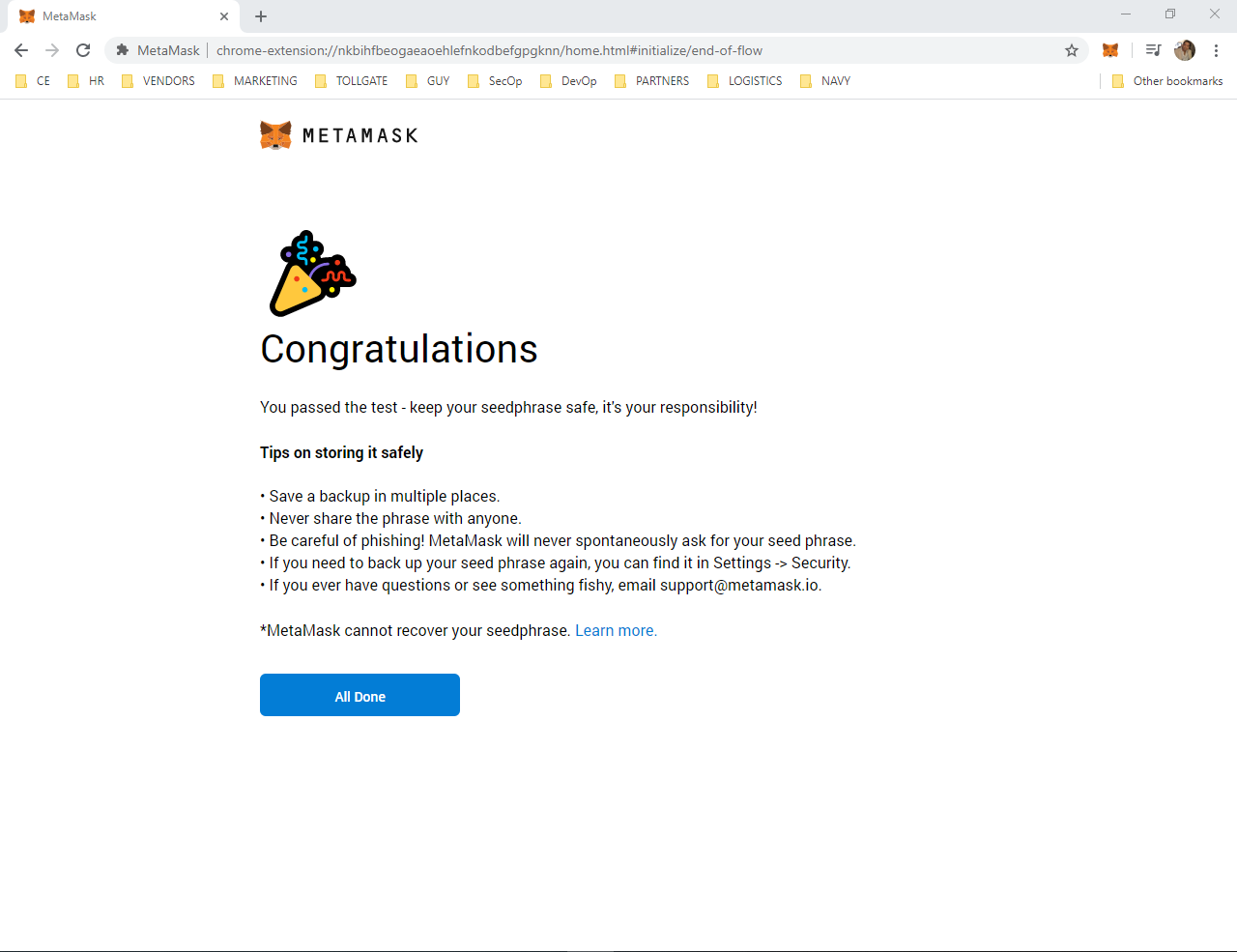


Figure 14.3.4 –This result is a proof-of-concept for the project assignment, the MetaMask interface.

## C.4 Front-end user interface using MetaMask wallet

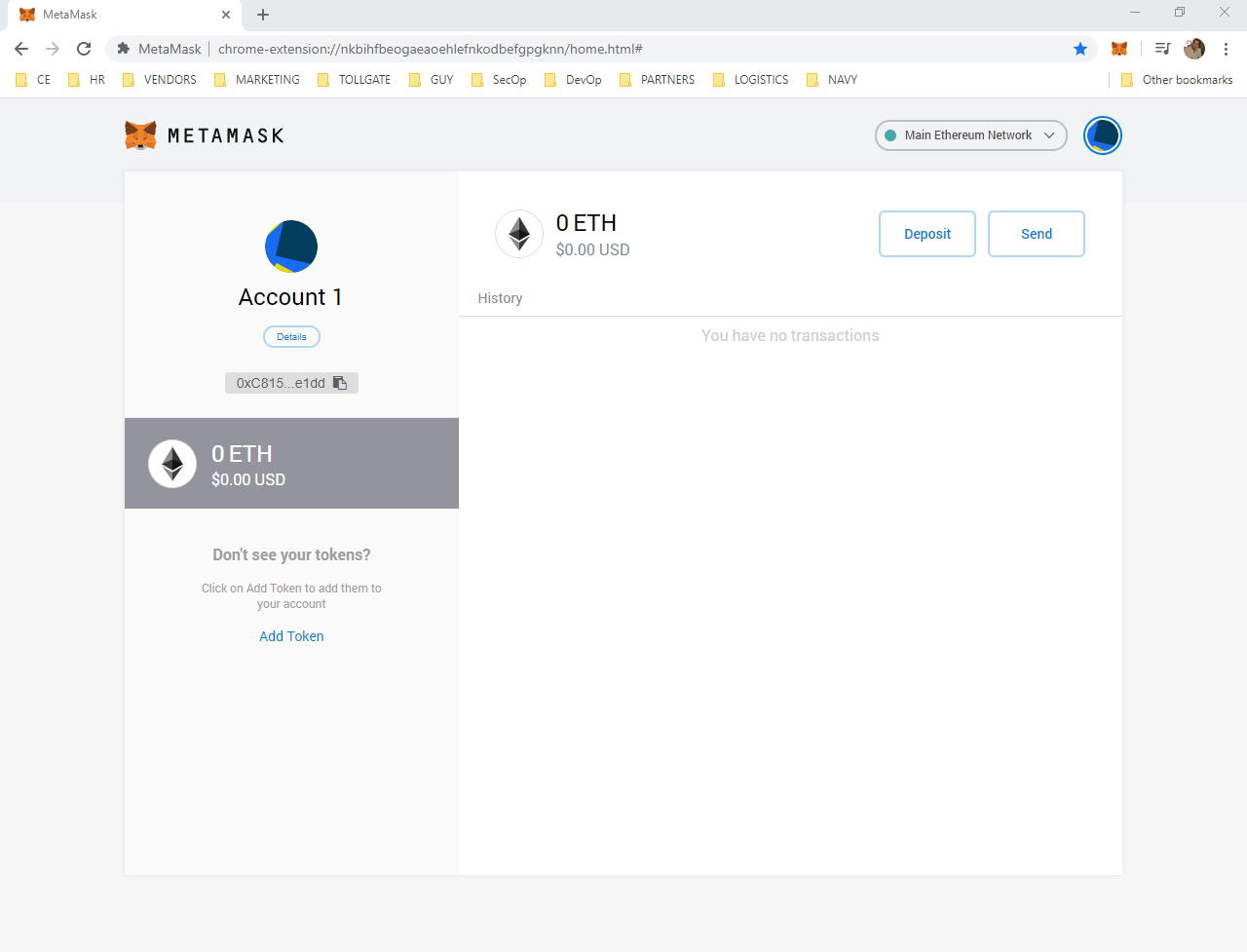


Figure 14.4.1 – ETH Account to hold smart contracts

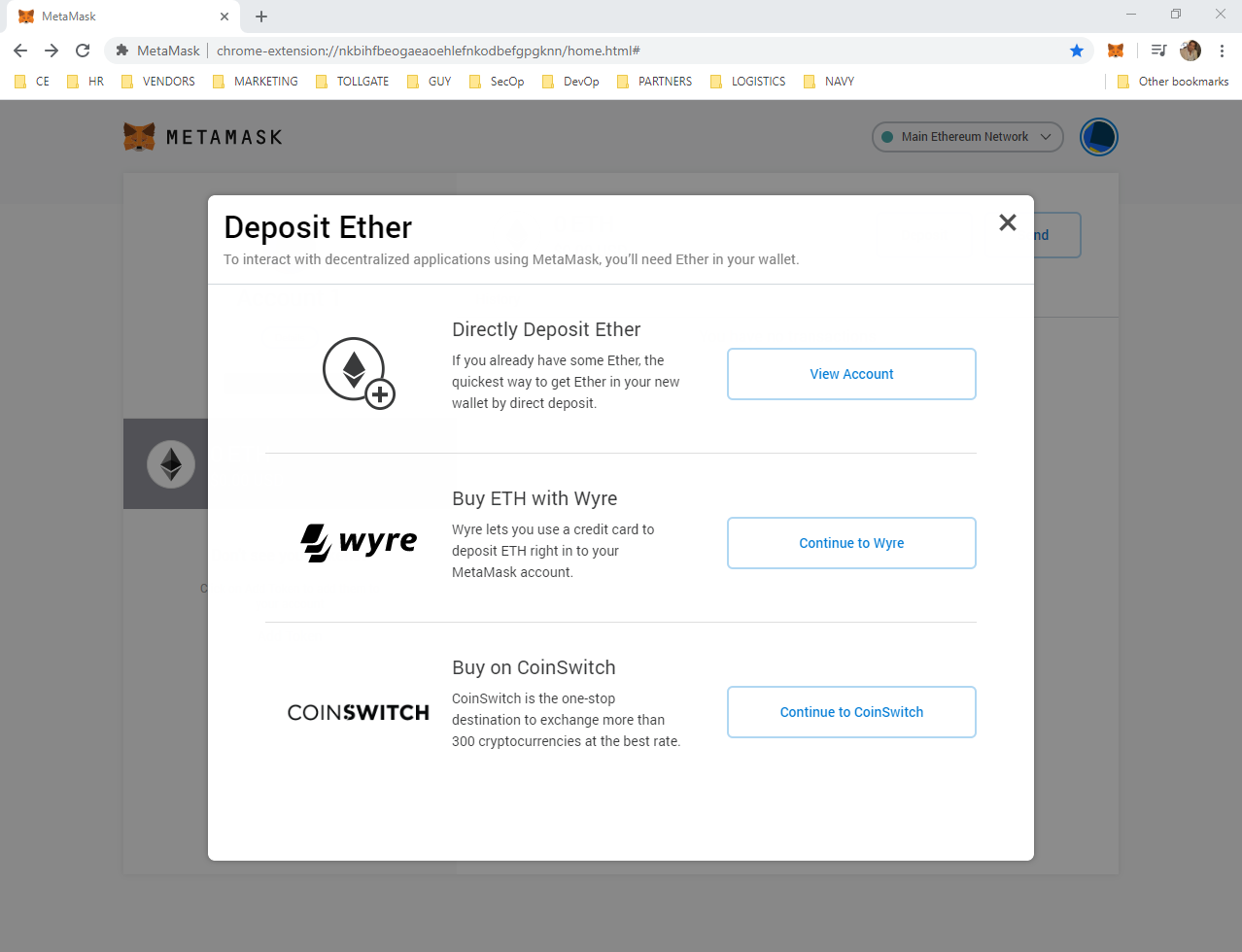


Figure 14.4.2 – ETH Deposit options.

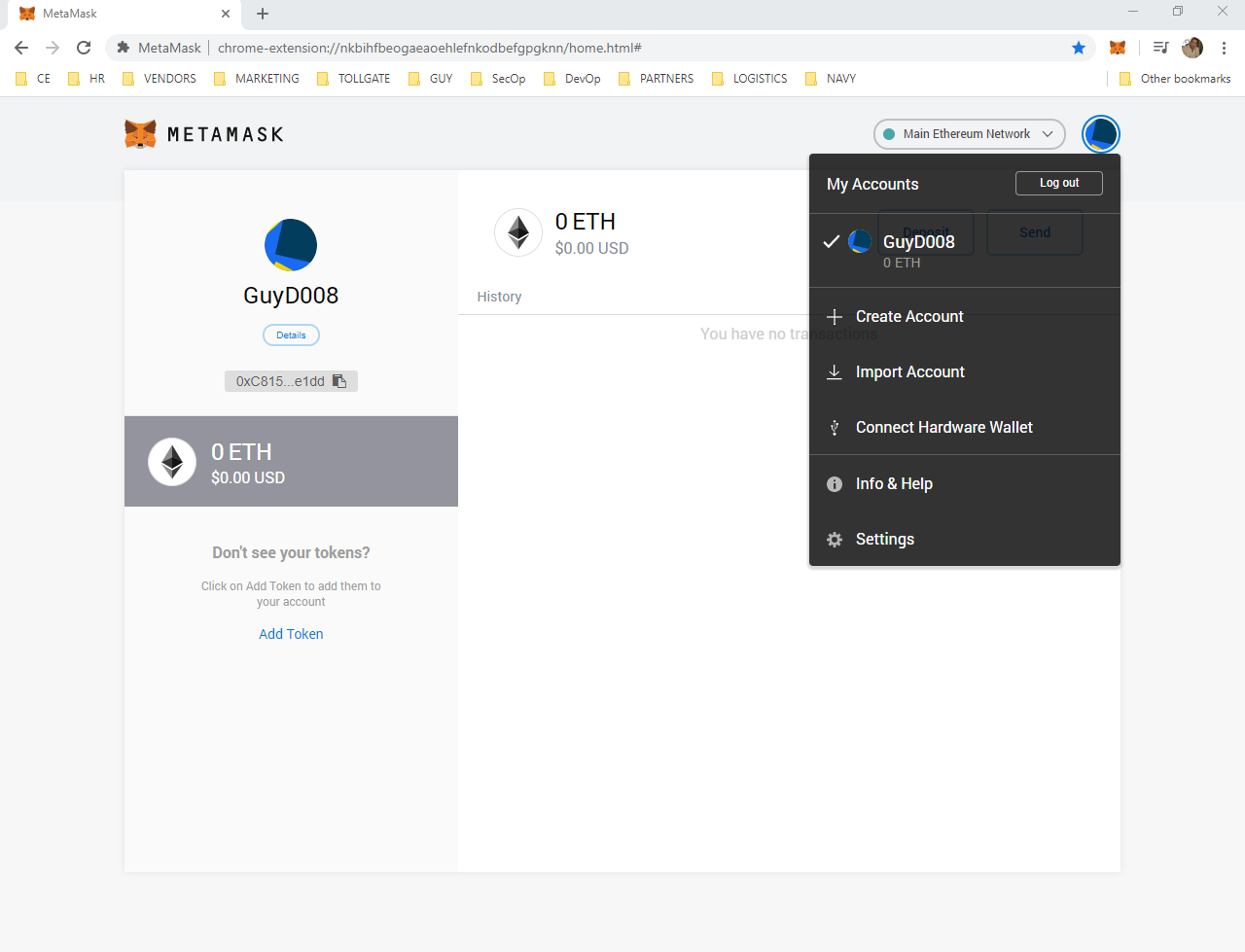


Figure 14.4.3 – Account Management Interface.

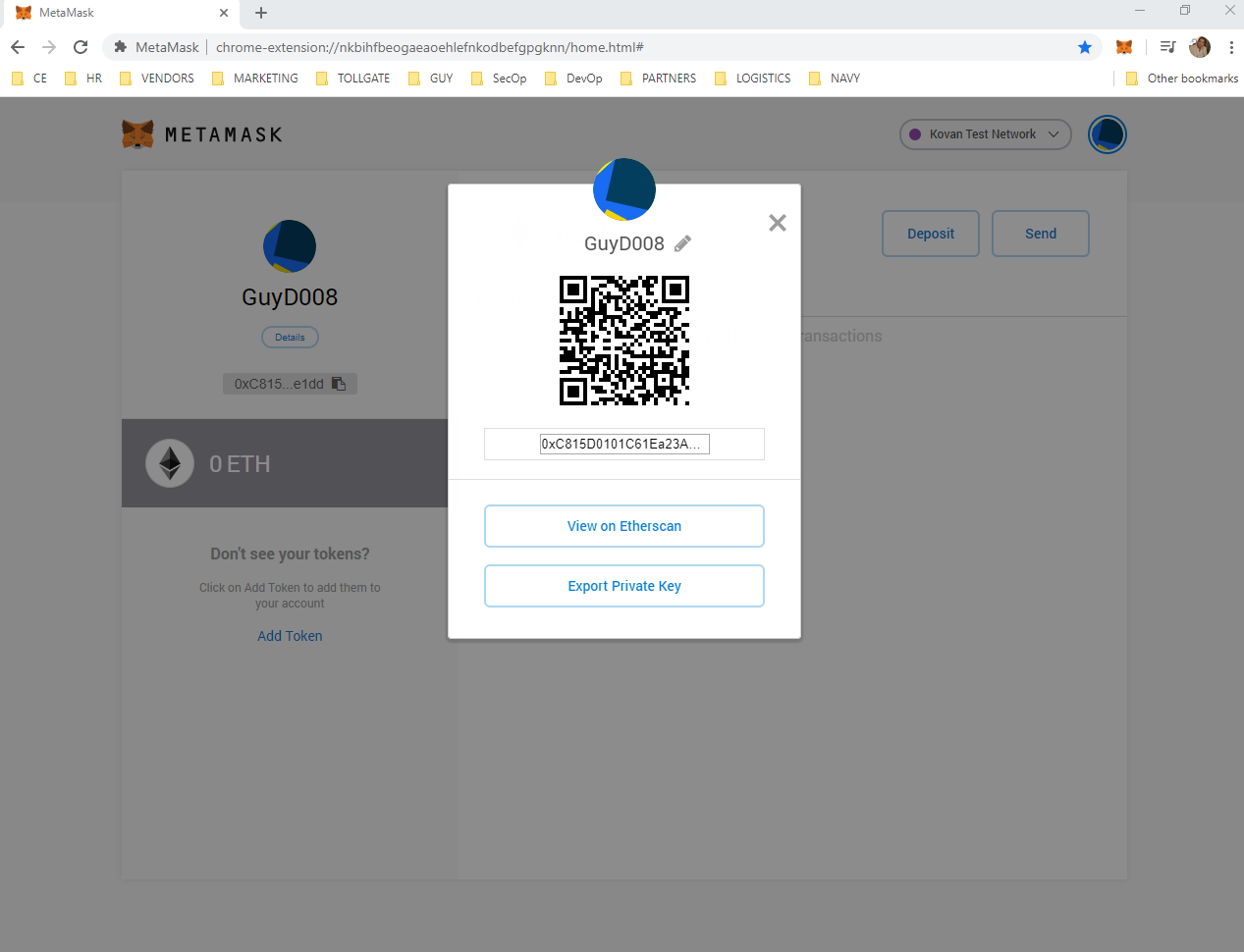


Figure 14.4.4 – Account QR code.

## C.5 Application Execution

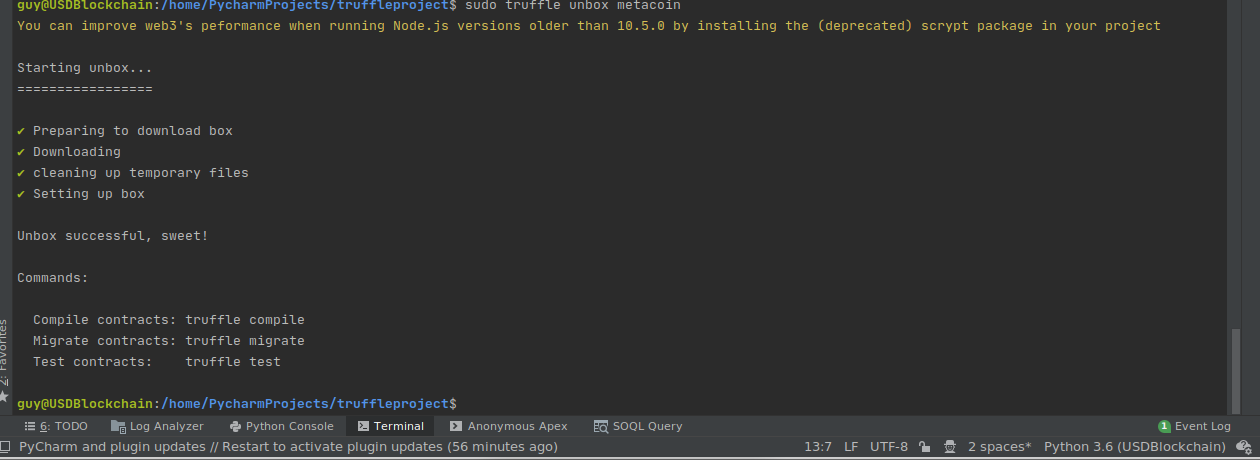


Figure 14.5.1 – Connected to the local Blockchain application from within the virtual server



Figure 14.5.2 – Smart contract tests are in the test folder

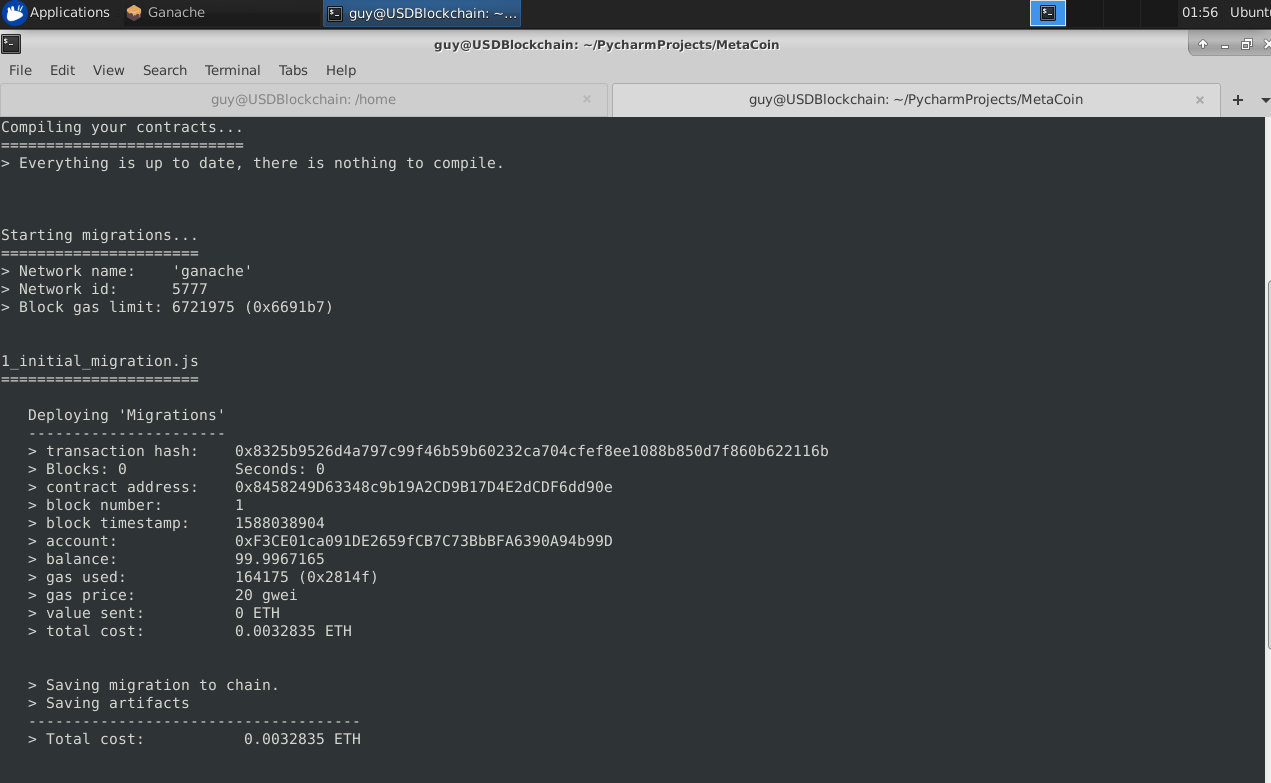


Figure 14.5.3 – Deploy smart contract migrations with **truffle migrate**

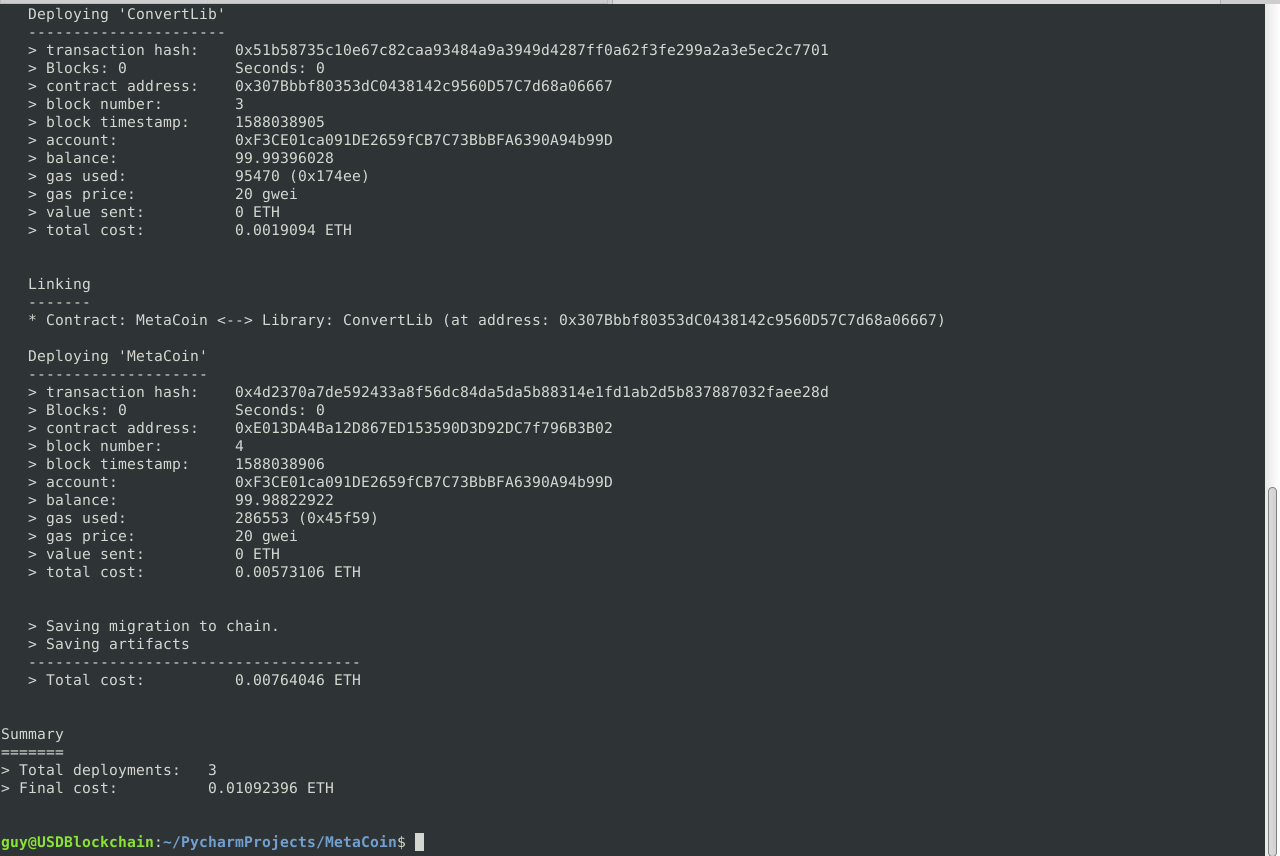


Figure 14.5.4 – Deployment Summary

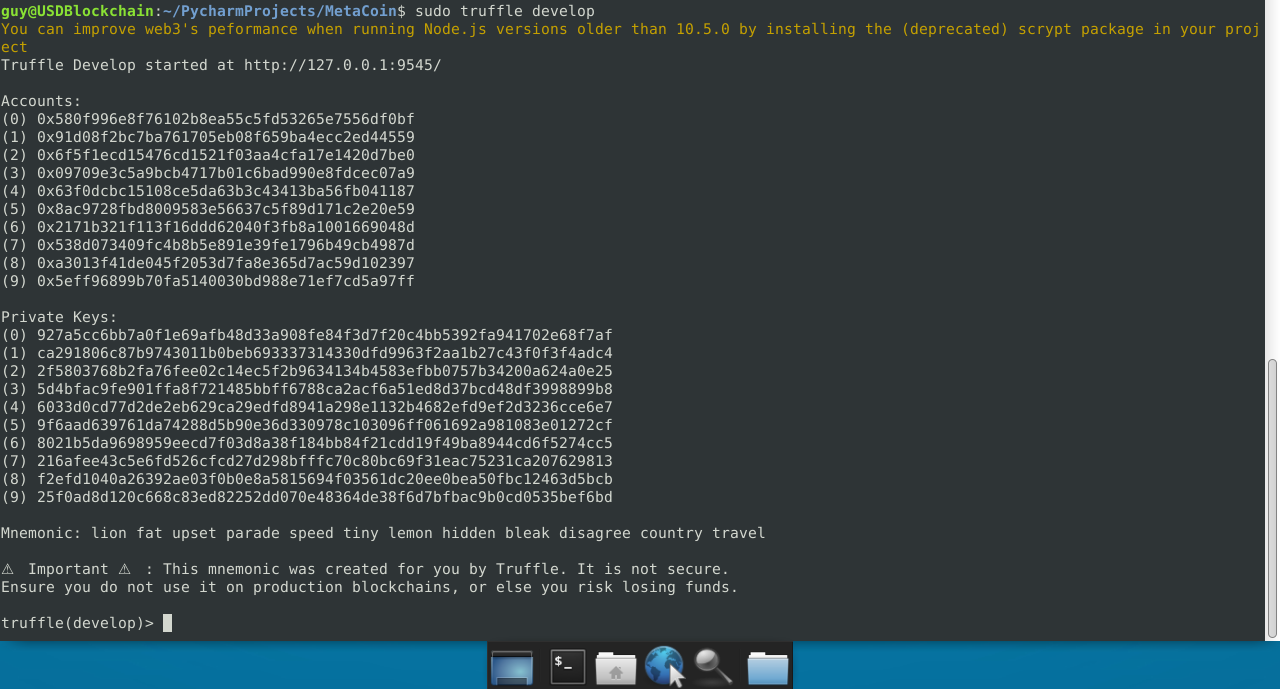


Figure 14.5.5 – Local Blockchain server running at http://127.0.0.1:9545

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

Figure 14.5.6 – Connect the local Blockchain to MetaMask and name the network **dev**.

A screenshot of a cell phone

Description automatically generated

Figure 14.5.7 – Default MetaMask Account connected to USD Blockchain Network.

A screenshot of a cell phone

Description automatically generated

Figure 14.5.8 – Fund Transfer – This output shows the transfer from one Blockchain account to another.

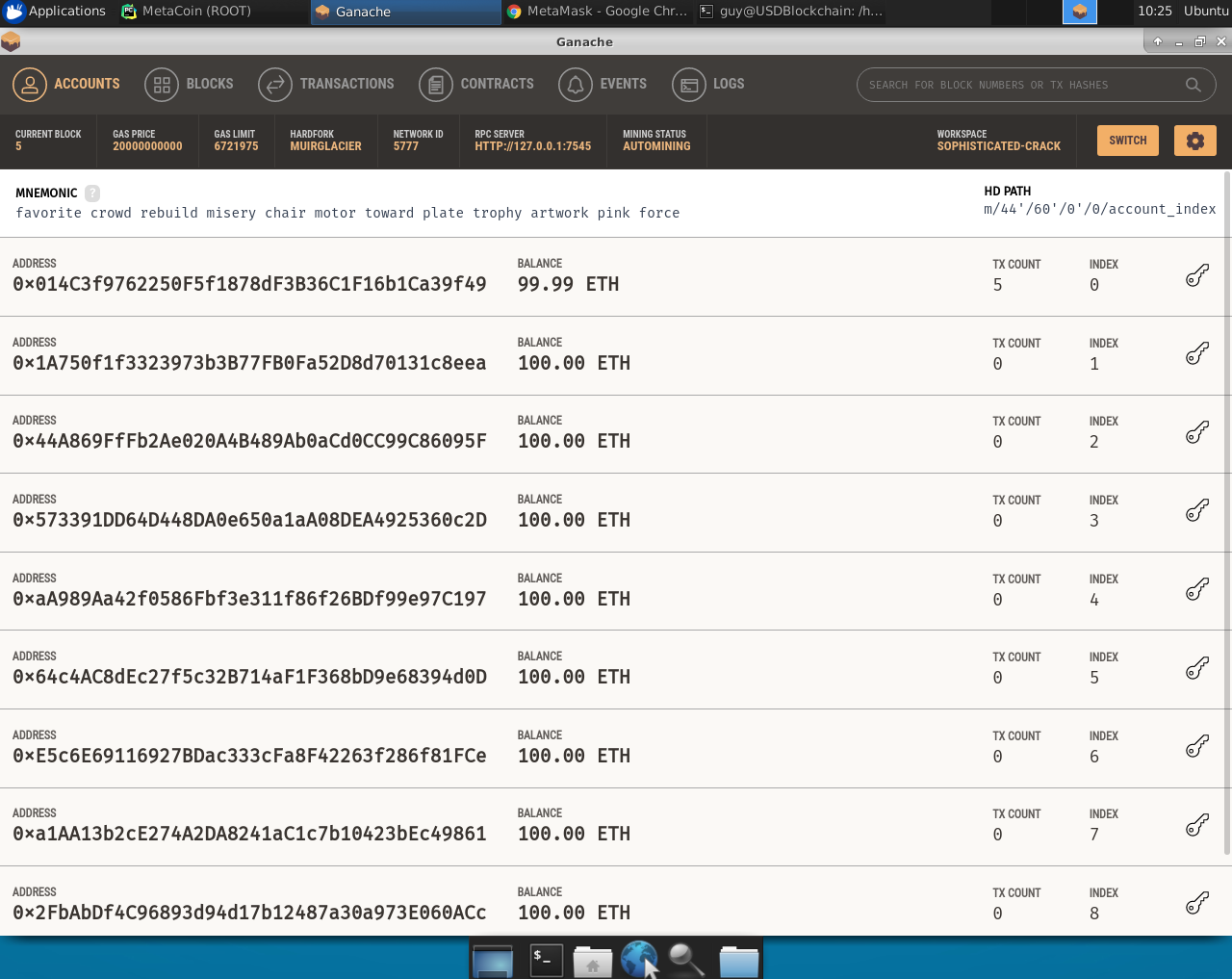


Figure 14.5.9 – Ganache RPC Client