FUTURE OF PAYMENT PLATFORMS

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Electrical Engineering

By

Salim Youssefzadeh

June 2014

© 2014

Salim Youssefzadeh

ALL RIGHTS RESERVED

COMMITTEE MEMBERSHIP

TITLE: Future of Payment Platforms

AUTHOR: Salim Youssefzadeh

DATE SUBMITTED: June 2014

COMMITTEE CHAIR: Franz Kurfess, PhD
Professor of Computer Science

COMMITTEE MEMBER: David Janzen, PhD
Professor of Computer Science

COMMITTEE MEMBER: Dennis Derickson, PhD
Professor of Electrical Engineering

ABSTRACT

Future of Payment Platforms

Salim Youssefzadeh

With the vast increase in smartphones, there have been an increasing number of opportunities growing in the app industry. One in particular is the way we deal with money. There are huge overheads in the current payment systems around the world particularly in the United States, many of which include large transaction fees. Many new businesses have grown to solve these inefficiencies and create a new platform that provides a new user experience, security, and convenience among many other things. However, many of these platforms are still centralized, making them more susceptible to attacks. This thesis goes over the various methods of payments, starting from their origins and discusses their flaws and ways they are being improved. This study explains where payment platforms are going and how they line up against other platforms in terms of security and usability.

We look at the origins of credit cards and why the US is lagging behind other countries in credit card security. Digital wallets like PayPal, Venmo, Square, etc. have done a remarkable job, but still have room for improvement in terms of security and usage. I try to solve these problems with the mobile application AnyCoin by bringing one platform that houses different types of digital wallets. The goal of this application was to grow a large user base and collect data off the transaction for future analysis and advertising. This study goes through an in depth analysis on the application from the

perspective of merchants and consumers to understand what users are looking for in digital wallets.

Decentralized platforms and crypto-currencies like Bitcoin have also created different ways to send money by creating a trustless system that does not depend on any central authority. I discuss what Bitcoin is and exactly how it works and the flaws in the current system. Mining is the process that puts Bitcoin into circulation and secures the network. However, as more customized hardware is released, Bitcoin will fall subject to becoming more centralized, and unfortunately become heavily regulated if it is to be used as a currency.

Ethereum is a new technology that takes the concepts of Bitcoin and creates a platform for a developer to create a decentralized application. I create a few contracts that show how we can create a decentralized version of PayPal that works using other crypto-currencies. Ethereum is still in its alpha stage and has yet to be released to the public, but has already improved on the problems that Bitcoin and other crypto-currencies hold.

ACKNOWLEDGMENTS

This thesis is dedicated to my family. Without their support, I could never be where I am today. They have helped and guided me throughout my life and have made me who I am today.

I would also like to thank my advisor, Dr. Franz Kurfess, for supporting me throughout this project and giving me valuable advice. I also wanted to thank Dr. Derickson for allowing me to pursue a thesis and take classes in the computer science department. I also wanted to thank Professor Janzen for giving me tips on how to make a better Android application.

TABLE OF CONTENTS

LIST OF FIGURES	ix
CHAPTER 1 INTRODUCTION	1
1.1 Motivation	1
1.2 Background	2
1.3 Credit Card Fees	3
1.4 Credit Card Security	4
1.5 Digital Wallets	6
CHAPTER 2 ANYCOIN	8
2.1 AnyCoin Motivation	8
2.2 Related Work	9
2.3 About AnyCoin	10
2.4 AnyCoin Experimental Study	12
Merchants	13
Starbucks	14
Sally Loo's	14
Motion Maintenance	15
Merchant Analysis	16
Students	16
Systems Engineer	17
Student	18
Student	18
Peer-to-Peer Analysis	18
Expert Evaluation	19
Reed Morse	19
Michael Youssefmir	20
Nat Welch	20
Expert Analysis	21
2.5 AnyCoin Conclusion	21
CHAPTER 3 BITCOIN	23
3.1 Bitcoin Mining	24
3.2 Under the Hood	27
3.3 ECDSA	30
3.4 Bitcoin as a Currency	32
3.5 Legal issues in Bitcoin	34
3.6 Future of Bitcoin	36
CHAPTER 4 ETHEREUM	39
4.1 What is Ethereum	39
4.2 Accounts, Messages, and Transactions	41

4.3 State Transition Function	44
4.4 Ethereum Blockchain	47
4.5 GHOST Protocol.....	54
4.6 Stable-Value Currency	55
4.7 Contracts	56
4.8 Future of Ethereum	62
CHAPTER 5 CONCLUSION	63
BIBLIOGRAPHY	66
APPENDICES	
ECDSA	68
Escrow Contract Python	71
Contract Stack Operations	75
GLOSSARY	77

LIST OF FIGURES

Figure 1: Global Payment per Volume [7]	3
Figure 2: Percentage of Respondents who have Experienced Card Fraud [22]	5
Figure 3: Transaction Fee Breakdown per Digital Wallet	9
Figure 4: AnyCoin Account Layout (Left) and Account Balance (Right)	11
Figure 5: AnyCoin Transaction List (Left) and Creating a Transaction (Right)	11
Figure 6: AnyCoin Credit Card Details (Left) and card QR (Right)	12
Figure 7: Bitcoin Key Block Diagram	27
Figure 8: Bitcoin price over lifetime	34
Figure 9: Ethereum Client	41
Figure 10: Ethereum State Transition Function	44
Figure 11: Ethereum Blockchain	49
Figure 12: PayPal Contract GUI	62

CHAPTER 1 INTRODUCTION

1.1 Motivation

The current advancements in mobile devices have opened up a new range of possibilities particularly in payment platforms. According to the Federal Reserve, a mobile payment is defined as "purchases, bill payments, charitable donations, payments to another person, or any other payments made using a mobile phone. Mobile payments can be used by accessing a web page through the web browser on your mobile device, by sending a text message (SMS), or by using a downloadable application on your mobile device. The amount of the payment may be applied to your phone bill (for example, Red Cross text message donation), charged to your credit card, or withdrawn directly from your bank account [23]." Currently our interactions with payment systems have mostly been through credit cards and cash. Most of us have been in a situation where we scan our credit card on an ancient card reader and try to sign the screen with something not even close to what we expected. We are now beginning to see a shift towards mobile payment platforms and digital wallets where we no longer need to carry around a stack of credit cards and anyone can become their own merchant. However these digital wallets still have vast room for improvement in terms of security and usability. When coming up with how to tackle this project, I wanted to apply the MBA skills I have acquired along with my Computer Science/ Electrical Engineering knowledge to explore the problem of how to create a new payment platform to replace or improve current platforms.

Throughout this project, I developed the mobile application AnyCoin for both iOS and Android, which brings many of the current digital wallets to one platform in a consistent and clean interface. In chapter 2, I give an in depth analysis on the application

from a merchant's stand point as well as from a customer's stand point. I also go over how Bitcoin works and the current flaws in crypto-currencies. The concepts of Bitcoin have lead to the development of Ethereum, which creates a platform for developers to create a variety of decentralized application through the use of contracts. In Chapter 4.7 I explain how to create contracts on Ethereum and how to implement a decentralized version of PayPal that allows customers to rate the merchants. We are seeing a transition into digital wallets and decentralized services and Ethereum is taking the right approach by creating the foundation.

1.2 Background

In recent years, we have seen a huge increase in the different types of digital wallets or payment methods available. There is PayPal, Venmo, Square, Stripe, and crypto-currencies like Bitcoin, and the list goes on and on. The one thing that has remained the same throughout this process is the credit card fee. The fee typically ranges from 2% to 4% and increases when the card is used abroad. The introduction of crypto-currencies has significantly reduced these fees and has caused quite a bit of noise in the industry. Many centralized platforms are subject to distributed denial of service attacks (DDoS), creating a single point for attackers to infiltrate. This can be mitigated through decentralized platforms like Bitcoin and other alt-coins.

Payment platforms have come a long way; we first begin by discussing the origins of the credit card. The story first begins in 1949, when Frank McNarma was at a diner and was unable to pay for his meal because he forgot his wallet. He wanted there to be another way to pay for items so he and his partner created a cardboard card dubbed the Diners Club Card which allowed a customer to pay the bill at the end of each month.

Within two years over 20,000 Club Cards were in circulation and were later replaced with plastic. By 1959 American Express introduced their credit card and is now in multiple currencies across the world. By 1966 MasterCard was formed as an open-loop system, requiring interbank cooperation and fund transfers [25].

1.3 Credit Card Fees

Credit cards are now a large part of peoples' lives and contribute to nearly \$6 trillion of the global payment volume. The breakdown per network can be seen in the image below.

Figure 1: Global Payment per Volume [7]

From the figure above, we can see that Visa accounts for nearly half of all the global payment volume in 2010, followed by one-third for MasterCard [7]. The debit card volume is growing more rapidly compared to credit cards nation wide. International debit cards in particular have become far more popular over the recent years.

Most credit card users do not consider the various fees associated with the transaction. Every time a transaction is made, there are two types of fees that occur, the transaction fee and the discount rate fee. The transaction fee is associated with the flat fee for using the credit card machinery and software. The transaction fee includes the authorization fee, return fee, Address Verification Service (AVS) fee and gateway fee[9]. The authorization fee is charged each time the business authorizes a credit card transaction and accounts for a majority of the overall fees.

Lets take for example a sale of \$40 with a \$0.20 authorization fee, and a 0.25% interchange. This will lead to a discount rate expense of \$0.10 (0.0025×40), which is still less than the \$0.20 authorization fee. These fees increase dramatically under certain platforms like digital wallets on mobile devices.

1.4 Credit Card Security

With such a large number of users, it is unlikely that we will see cards disappear anytime soon. This leaves open a huge issue of security in credit cards. Target is one of the most recent examples where as many as 110 million customer accounts were compromised, exposing credit card details among other personal information. The bottom-line however is that the US has one of the worst security measures in place. While many other countries have already switched to the chip and pin method, much of the US is still dependent on traditional swipe and sign cards. The image below displays the percentage of respondents who have experienced credit card fraud from a number of different countries.

Percentage of Respondents Who Have Experienced Card Fraud
(N=5,114)

Source: Aite Group, ACI Worldwide study of 5,223 consumers in 17 countries, Q3 2012

Figure 2: Percentage of Respondents who have Experienced Card Fraud [22]

The problem with swipe and sign cards is that if a card is stolen it can still be used to make transactions before the card has been cancelled. This problem is reduced with the chip and pin method since even if a card is stolen, the user must enter in a pin to authorize the transaction. Chip and pin cards have an embedded microchip and are authenticated through the use of a pin number that is set by the owner. When the customer wishes to make a purchase, they insert their card into the Point of Sale terminal and enter their pin number to authorize the transaction. The pin can also easily be changed by the card owner through an online portal, thus making it harder for a compromised card to be used. Over \$20 billion is still handled using traditional swipe and sign cards and this is all scheduled to change by next year. So why is it taking the US so long to switch over to chip and pin while most other countries have already made the transition? For one, some countries have had a higher fraud rate, and the move helped

them combat fraud. Second, the chip and pin method can operate in offline mode, which allows a card and terminal to make a transaction independent of communication from the bank. These factors were not highly attractive in North America, and have now caused the US to fall far behind in card security. Both MasterCard and Visa have agreed to make the transition to chip and pin and have agreed to make the change by October 2015 [12]. This in result will force many retail outlets to also make the change in their current machines to allow for the old swipe and sign method as well as the chip and pin method. The incentive for merchants to also make the change comes at the risk that merchants using the old system to make a transaction will be held liable for fraudulent charges. Likewise, merchants using the newer system to make a transaction will cause the banks to be held liable for any fraudulent charges.

1.5 Digital Wallets

We are now starting to see a vast shift towards digital wallets. Many of the well-known wallets currently in use include: PayPal, Square, Venmo, Stripe, Splitwise and others. Venmo uses a very simple business model that allows users to send money to friends in a fun and social manner, while the company collects interest on the amount in the user's digital wallet. The transaction fee is free for debit card or bank transactions and 3% for credit card transactions. However, few people know that the most successful digital wallet today is centered in Kenya and known as M-PESA. Safaricom, Kenya's largest mobile operator, launched the service in 2007 in order to make microfinance loan repayments. After beta testing, the company decided to extend the service to a more general money transfer business. M-PESA is now used by over 17 million Kenyans as their main form of making transactions [24]. To deposit money into their account, the

user goes to one of the 40,000 Safaricom agents and hands them cash. This amount gets added to the user's M-PESA account, allowing them to send money from their "dumb" phone. To withdraw money, the user simply goes to one of the agents and debits the amount from the account and hands over cash to the user. This service has reduced the overhead, making it especially useful for Kenyans sending money to their families. BitPessa, a newly funded Bitcoin project, is trying to use the same concept in Kenya and the neighboring countries using Bitcoin. They have currently launched their pilot program and are in their beta phase. Bitcoin will be discussed in more detail in Chapter Three.

CHAPTER 2 ANYCOIN

As part of some previous interest, I spent the past couple of months developing a mobile application for both iPhone and Android that acts as a platform that brings together a majority of the digital wallets to on location. This chapter will discuss the application in detail and the key findings after conducting interviews with merchants, customers, and experts.

2.1 AnyCoin Motivation

There are currently dozens of digital wallet application available, each having a different user interface. For people that use multiple types of digital wallets, their interactions have to be specifically tailored to each app. This can often lead to a large cluster of apps on their phone and specific steps to make a transaction on each app. AnyCoin brings together various digital wallets on to a single platform as well as allowing users to store credit cards. The app contains two Bitcoin wallets (Coinbase and Blockchain) as well as PayPal, Venmo, and Stripe. The hope of this app is to bring all the various wallets into one location and giving it an extremely user-friendly and simple interface for users to make a transaction across all the wallets. The digital wallets that currently exist have various transaction fees associated to them. The table below illustrates the transaction fee for some of the different digital wallets, as well as their user base.

	Transaction Fee	Key In Transaction Fee	Revenue Per Year	User Base	Founded
Intuit	2.75%	3.75%	\$3 Billion	-	1983
PayPal	2.70%	3.50%	\$6.6 Billion	110 million	1998

Venmo	3%	-	-	-	2007
Square	2.75%	3.50%	\$165 million	2 million	2009
Stripe	2.90%	-	-	-	2010

Figure 3: Transaction Fee Breakdown per Digital Wallet

The Key In transaction Fee refers to cards that are manually entered in, rather than “swiped” using the card reader. PayPal has the largest user base and revenue mainly because it was an early player in the digital wallet industry. These digital wallets each have their own app for mobile devices, but can lead to a cluster of different applications and user interfaces for those users that have many apps.

The idea of having a merchant set up an account and having to carry around a credit card reader seems somewhat ridiculous. Through the interface I have created in the application, merchants no longer need to carry around additional hardware, but can simply scan the buyers credit card QR, which is generated within the app or can be entered manually by the merchant. The app allows a user to set up a password at launch or every time they wish to make a transaction. After a merchant makes the transaction, the buyer receives an emailed receipt to keep for his or her own record. The data on each transaction is pushed to Amazon’s DynamoDB for future analysis.

2.2 Related Work

There are many apps that can be found on the Google Play Store as well as the iPhone App Store that offer digital wallets. Some apps include: Square, PayPal, Venmo, Stripe, Google Wallet, as well as some Bitcoin wallets. Most of these apps (aside from Square) offer an API that allows third party developers to integrate these wallets into their own application. However, there is currently no app that integrates these payment

platforms along with a user's credit/debit cards. Square, PayPal, and Intuit have merchant apps that require a credit card reader to be inserted in the audio jack of the phone, which adds an extra level of inconvenience for merchants since they now have to carry additional hardware and cannot begin making transaction until they receive the card reader. This also adds a layer of inconvenience to the users since they are required to carry around their credit cards at all times rather than having it readily available on their smartphones. Apple's current fingerprint sensor on the iPhone 5s opens the door to new ways we can make payments through our phone with an added layer of security, and have just released an API for iOS 8 developers. There are also many devices that are now equipped with NFC chips that allow them to communicate to one another, but this technology has yet to be adopted by many large phone manufacturers and will most likely fair out over time.

2.3 About AnyCoin

AnyCoin is an iOS and Android application that brings PayPal, Stripe, Venmo, Coinbase, Blockchain, and credit cards to one platform. Using the API's provided from each of the individual platforms, I was able to create one platform where users are given a list of all their accounts and can select the specific account to view transaction history and create new transactions. Users are also able to generate a QR of the credit cards they have stored within the app and are able to make a card transaction with a Stripe merchant that scans the QR within the app on their device. The layout can be seen in the screen shots below:

Figure 4: AnyCoin Account Layout (Left) and Account Balance (Right)

Figure 5: AnyCoin Transaction List (Left) and Creating a Transaction (Right)

Figure 6: AnyCoin Credit Card Details (Left) and card QR (Right)

2.4 AnyCoin Experimental Study

The objective of this section is to evaluate and get feedback on current mobile payment platforms as well as on my application AnyCoin. The feedback will help identify the key points that both merchants and consumers are looking for in a digital wallet in terms of functionality and usability. For instance, Sally Loo's, a local café in San Luis Obispo, uses the Square credit card scanner with an iPad as their only credit card reader and rotates the screen when the buyer needs to sign and enter in the gratuity. The layout when the user needs to sign consists of a dialog box where they swipe their signature along with a few buttons for gratuity options or for entering in your own. Out of convenience however, most users would press on one of the existing gratuity options

rather than typing in their own. This is a very simple design and yet gets people to knowingly give more to the seller. There are still instances where the buyer does not understand the technology, as was the case for an older woman who tried to write her signature with the tip of her fingernails. This section goes through various interviews and experiments conducted on the app to test the demand and get a sense of what users are looking for in a digital wallet.

Merchants

The first step involved interviewing merchants to get a sense of what the features and functionality in a point of sale device and what they like or dislike about the current systems. The interview questions can be found below:

1. *How long have you been in business?*
2. *Tell me about your experience with Square*
3. *How do you currently take payments? Besides cash, what is your preferred method (credit, debit, check, PayPal, etc.)*
4. *What percentage of your sales are through Square?*
5. *Tell me about making sales on the go.*
 - a. *If no on the go sales: would this be something you think your business can benefit from? (farmers market, festivals, mobile merchants, sales)*
6. *Why did you choose Square over alternative payment types?*
7. *Are you happy with Square?*
 - a. *Yes-Why?*
 - b. *No- What kind of problems with Square?*
 - i. *Did your problem result in loss of sale?*
8. *How do you currently give receipts for sales?*
9. *Is there anything that you would like to see Square have that it doesn't?*
10. *Do you use the card reader or QR code?*
11. *How do you feel about Square's learning curve? Do you feel proficient*

Below are the results for several interviews conducted on merchants.

Starbucks

The Starbucks on Chorro St uses the Square QR code reader as a form of making transactions.

The Starbucks originally started out with Square's credit card reader that plugs into an iPad as their method of making transactions. However, the card readers were not nearly as robust as their traditional system, and got sold off as merchandise. Currently they have the Square QR reader, as well as traditional card readers as their Point of Sale system. Over half of their customers use the QR reader along with the Starbucks app to make transaction and they find it very easy and simple to use. The Starbucks app allows Starbucks to gather data on users to monitor the types of drinks they are purchasing and the frequency at which they go to Starbucks. This can then be used for many types of applications, including being able to send advertisements to the users. Instead of using the Starbucks app to generate the QR, the Square Wallet app can also be used to generate the QR and make the transaction. The Square Wallet app also allows for auto check-in, which allows a user to walk in the store and make a transaction only by giving their name. However, this feature was not available in any of the stores in San Luis Obispo.

Sally Loo's

Sally Loo's is a café located near the San Luis Obispo train station and has been in operation since 2009. They started using Square last year because they use an iPad for their register so it "made sense." They are happy, but would like some more custom features for the interface. About 50% of their customers use cards. They can print a receipt in the store or through email. They have not had many problems with the Square system other than when an update in the Square app caused the employees to re-learn the

interface. Not much training is required to get the employees to use the system and the layout is fairly straightforward.

As I waited to speak with Jen, the manager, I watched some transactions take place at the register and one in particular was an older woman who had to sign on the iPad with her finger. She very carefully started scribing her name with her fingernail, but nothing was appearing on the screen. The cashier tried to tell her to use her fingertip but she could not hear him. After nothing was written on the signature line, the cashier was able to explain to the customer that she needed to use her fingertip. Then she carefully and slowly signed her name again. Another interesting feature on the Square app is the ability to give gratuity. The app shows three options for gratuity by default as well as an option to enter in a custom amount. As I watched the transactions being made, it was interesting to note that nearly everyone would select one of the three preset gratuities rather than entering in a custom amount. Customers would rather select a preset option simply out of convenience and time rather than going through the hassle of pushing a few more button sequences to set a custom amount.

Motion Maintenance

Steve is a mobile mechanic who travels to cars for repair instead of them coming to a physical shop. Currently Steve only accepts cash and check as forms of payment. As car repairs can be very expensive, this can limit the customers he is able to retain (people can't charge the expensive repair on their CC). Steve does not currently have a smart phone and is reluctant to obtain one. This is for multiple reasons including expensive data plans and the self-training he will have to do. Motion Maintenance has been in business since 1985. The company currently does not have any experience with Square or any

other digital forms of payment. It currently accepts only cash or check. Steve thinks this sometimes restricts his business because most people cannot afford to pay up front for more expensive repairs. When Motion Maintenance makes a sale on the go, an invoice is generated separating the labor from the parts (like any mechanic/repair shop). Sometimes individuals pay immediately upon service, and other times an invoice is left and payment is remitted at a later date. Steve thinks digital payments would only help him for the immediate payments because, as a mobile mechanic, not all clients are present when a job is finished. Steve has not picked Square as a payment type and the service charges are a big reason why. His business fluctuates in how busy it is, so he is reluctant to give up his earnings to a service he can live without.

Merchant Analysis

Of the companies that I interviewed, I got the impression that they were reluctant to change and happy with their current system. The only time merchants would consider a new payment system is during the initial process where they are a growing business and are looking for ways to make transaction on the go or through a storefront.

Students

The next step involved interviewing students and observing their interactions with the app and what they look for in a digital wallet. The interview questions can be found below:

1. *How do you split bills with your roommates or friends (ex. out at meals)*
 - a. *Can you think of a time that the way you split a bill was unfairly split? Tell us about it.*
2. *Do you buy or sell on websites such as Craigslist? If so, what form of transactions currently do you utilize for craigslist transactions?*
 - a. *Do you have any safety security concerns with this process?*

3. *What are your concerns regarding security and digital transactions?*
4. *What are your feelings regarding “digital wallets”?*
 - a. *Have you utilized Square, PayPal, Intuit, Venmo? If so, how many times a month would you say you used them?*
 - b. *Do you find them easy to use? If no, what can be done to make the process easier*
5. *Have you ever wanted to make a transaction but did not have your wallet on you?*

Systems Engineer

Paul is an IT professional who is often up to date in with various IT trends. As a user of PayPal and Venmo, he is familiar with the strengths and weaknesses of their digital platforms.

Paul is currently renting two rooms in his house and receives his rent via check. He says he is not proactive about depositing his checks at the bank because it is inconvenient. Being paid with a digital app would be ideal for him because it would link directly to his bank account. His roommates are less tech savvy, an app with simple user interfaces is something he thinks can be explained to them easily. Paul states that he buys periodically on Craigslist (3-5/yr). However, these can sometimes be large price items. He would prefer to not carry cash on him and have an alternate means of paying them. He expects a comparable amount of security from an app such as AnyCoin as he would with any other that has his personal information. He is not willing to sacrifice security for price (i.e., free app). He still expects free apps to be secure. Paul has used PayPal and Venmo and uses them a “handful” of times a month, mostly as a way to split costs of events (vacations, concert tickets, etc). He finds them easier to use, but he is extremely interested in an app to bring them all together. He is confident AnyCoin is a type of app for his needs. Paul has never not had a wallet when he needed, but would prefer to not carry a wallet unless necessary.

Student

Reeves uses Venmo to make transactions between his roommates and finds the app very easy and simple to use. He was unaware about how the money is stored on a digital wallet rather than going straight to his bank account. He was able to incentivize his roommates to get the app by sharing a link with his roommates to get the \$5 referral reward from Venmo. They like the concept of using one app to house different types of digital wallets but were hesitant about security of the app and encryption of data. After demoing the app he commented on the ease of use within the app and the simple interface. He noted that he was already using Venmo on his smartphone and saw no reason why to use AnyCoin instead. Most users do not care about how the back end works, and only care about the ease of use and functionality.

Student

Daniel is a student with roommates and currently uses Splitwise for splitting bills. He finds Splitwise very easy to use and is able to track all his past transactions and delegate a schedule of payments to his roommates. He likes the idea of having one app that brings together all the digital wallets but sees no incentive to switch over when his current system works for him. He also is hesitant about the security features and the risks associated with bringing all account and card credentials to one location.

Peer-to-Peer Analysis

Of the students who already use a service like Venmo or Splitwise, I have found that there is no incentive for them to migrate over to my app. They see no value added by switching over and in the end; user convenience is not enough for them to change platforms. There needs to be a greater incentive. Of the students that still use cash or

check, they may start off using the app, but there needs to be a greater incentive for them to go through AnyCoin rather than the provider's native application. The students using cash or check love the idea of a digital wallet because it will give them the ability to have a history of transactions without the hassle of always writing out checks or carrying cash.

Expert Evaluation

Lastly, a few interviews were conducted with "experts" to gain insight on where they see payment systems going in the future and what they think about current technologies and the app.

Reed Morse

Reed is a Cal Poly alumni and the creator of Punchd, an app designed to help small business owners and reward customers through coupons. Reed now works in Google's YouTube division and has some expertise in the commerce and Google Wallet. We discussed Google Wallet and how they are having difficulties because major credit card companies feel threatened and do not want their model to be disrupted. Reed sees a big change coming to the mobile payment industry and thinks a lot of companies are doing a great job at getting there. One of the features he likes on Square Wallet is allowing users the ability to automatically check into a location based on their phone's GPS location and their name in the database. When discussing AnyCoin, he likes the idea of having one app, but sees a problem in incentivizing users to use the app versus the native applications. He likes the user interface and the experience, but at the end of the day, convenience is not enough to get a user to switch over. Venmo and PayPal started out basically buying users by giving them money for every successful referral. We also talked about the future of payment systems and how it can be used to buy anything from

any location using any type of currency. One instance would be to allow European residents to buy a product from the US and having a middleman to get the local delivery and execute the overseas shipping. He sees huge growth in the mobile payment sector and thinks that companies like Square have done a remarkable job at marketing and getting a large user base.

Michael Youssefmir

Michael works at Google and is a Bitcoin expert. We discussed the future of mobile payment and Google Wallet's plans to create a platform that brings their own credit card, digital wallet, and vast database together. This essentially allows a user to use any method of payment that he or she finds most convenient, and the merchant can use their existing setup to complete a transaction. Michael thinks Square has done an amazing job to grow and make it easy for merchants to get started quickly. He likes the Square Wallet app and how certain shops allow users to make a transaction simply by knowing the user's GPS location and looking the user up in the database.

Nat Welch

Nat is one of the founders of Punchd and worked in Google's commerce division over the past year before moving to London to work on a different project. He was very helpful in explaining the goal of Google Wallet and how Google originally wanted to compete against credit card companies but were bullied out by the major companies. Google then backed out and limited the number of capabilities of their wallet and has now introduced their debit card again. The debit card is tied directly to a user's digital wallet making Google effectively act as a bank. He also mentioned that most European countries are using chip and pin cards rather than magnetic strips because they are far

more secure. This change reduced credit card fraud by up to 80% and has greatly reduced the number of swipe machines in Europe. The US credit card companies are now being forced to do the same thing and will be transitioning over to the new system by 2015/2016. This change will impact many of the point of sale systems that we currently have set up and will increase competition in the mobile payment sector. Regarding my app specifically, Nate saw it as a good idea but had trouble recognizing the true value proposition of the app. He noted that although convenient, it is unlikely to get many users because there is not enough of an incentive to transfer over.

Expert Analysis

Talking with the experts, I have come to the conclusion that AnyCoin, although convenient, will not grow a large user base since there is not enough of an incentive to get users to switch over. Although there are a lot of features in the app, users are looking at trying to save something, whether that is time or money. And they are much more likely to start using a product if they feel like they are gaining something from using it.

2.5 AnyCoin Conclusion

Going over all the analysis, I have found that there is huge potential in the mobile payment industry, however my app will not take the lead. After discussing with merchants who use Square, I have found that they are very happy with the system and are reluctant to transfer over to a new system any time soon. They like having the option of viewing the analytics and are comfortable with the layout of the app. The users of Square Wallet also enjoy the app and like the ability to walk into a store and having their QR scanned to make a transaction. The students that already are using a digital wallet are also reluctant to transfer over because they too have a system that they are comfortable with

using and see no incentive to transfer over. The studies conducted have shown that the app is convenient, but in the end users need more of an incentive. They want to see that they are saving/gaining something and convenience is just not enough. The change in credit cards in the US opens up the opportunity to take advantage of this new change that is coming and will increase the competition for THE mobile wallet. Since a lot of the companies that are already in this sector already have a strong foundation, it may be very tough to compete in the US. I see a huge opportunity in other countries and believe that soon every country will have their own version of Venmo or Square. Ideally, a combination of Google Wallet, Venmo, and Square should be implemented in other countries however the user experiments will have to be re-implemented to find specifically what users are looking for in other countries.

CHAPTER 3 BITCOIN

The previous chapters have gone over centralized payment platforms and some of their defects. To re-iterate, many of the centralized platforms can be susceptible to attacks. One example in particular, occurred during Square's early stages of launching their service. Mid-2011, Verifone (one of Square's competitors) revealed that it was possible to skim card data using Square's card reader [14]. Square has updated their card reader and now encrypts the card data as it is read in, and then decrypts the card within the Square app. Target is another company that suffered from a massive security breach, resulting in 70 million accounts to be compromised. The attack occurred through malware sent to employees' email accounts and later made its way to the point of sale machines [15]. This attack could have easily been prevented through proper virus protection software or by limiting employees' access to email on point of sale machines. The following chapters will discuss decentralized platforms, particularly Bitcoin and Ethereum, and some of the advantages and disadvantages that they present.

It is important to understand the concept of Bitcoin and other alternate crypto currencies for they have certain properties that can be used with traditional payment platforms. "Bitcoin is a consensus network that enables a new payment system and a completely digital money. It is the first decentralized peer-to-peer payment network that is powered by its users with no central authority or middlemen [21]." Satoshi Nakamoto released the first Bitcoin specification and concept in a mailing list in 2009. His identity still remains a mystery despite recent allegations and articles on a Satoshi Nakamoto living outside Los Angeles. One of the main components of Bitcoin is the "blockchain." This ledger is distributed over all the computers directly connected to the Bitcoin

network. Transactions are pooled together in “blocks” and are added to the blockchain [20].

Key pairs are another important aspect of Bitcoin that consists of two mathematically related large numbers. A user who knows one of these numbers can perform an action that can be verified, but not recreated, by the user who knows the second number. This, in a sense, is similar to a lock that requires two keys in order to be unlocked. If both users are not present, the lock cannot be opened. This key pair is often broken up into a “private key” and a “public key.” The public key, also known as the user’s Bitcoin address, is used to send or receive Bitcoin, which is verified and signed using the private key to verify the legitimacy of the origin [20]. Bitcoin are created through a process called mining, which will be described in more depth in the next section.

Bitcoin are put into circulation at a decreasing and predictable rate, so that every four years the number of Bitcoin produced will be halved. Ultimately, there will be a total of 21 million Bitcoin put into circulation; nearly half of these have already been put out in the public. Each Bitcoin can be divided up to 8 decimal places to create what is called a satoshi. Bitcoin are produced through a process called mining and is discussed in the next section.

3.1 Bitcoin Mining

Mining is the process of consuming computer power to process transactions, secure the network, and keep everyone in the system synchronized together. Anyone can become a miner by running the Bitcoin client on special hardware. During the early stages of Bitcoin, users were able to mine Bitcoin using their computer CPU. This later

shifted over to mining on the GPU, which allowed for faster processing. There are now mining farms with special machines dedicated to processing transactions and running the client. The mining software listens for a transaction broadcast through the network and performs certain tasks to authorize these transactions. Even after the total 21 million Bitcoin are in circulation, there will still be a need for mining in order to approve transactions [21]. For this reason, even though the transaction fee is considered to be negligible, there is still a small fee associated for making transactions in order to give miners an incentive to approve transactions. This will be discussed in more detail in the next section.

Each transaction is included in a block along with a mathematical proof of work. The generation of such proofs is extremely difficult to produce, forcing miners to perform the calculations before the network accepts the blocks and then rewarding the miner for generating that block. As the amount of competition is increased, the difficulty of producing a valid block is also increased in order to ensure that a block is only produced once every 10 minutes. Each block also depends on the block before it in the blockchain, which stops people from trying to create artificial blocks or reversing transactions. If two blocks are found at the same time, miners work on the first block they receive and switch to the longest chain as soon as the next block is found. Currently the reward for mining a block is 25 Bitcoin, which halves after every 210,000 blocks created (around 4 years) [21]. This reward halves nearly every 4 years, until all 21 million Bitcoin are in circulation.

The example below explains what makes mining more difficult and how exactly it works. Let's say we start with the base string "Hello, world!" and our target is to find a

variation of this string such that the SHA-256 hash value begins with 0000. We vary this string by adding an integer to that end, also called a nonce, and incrementing it each time until we achieve the desired hash. Thus finding the match for “Hello, world!” takes 4251 tries.

```
"Hello, world!0" =>
1312af178c253f84028d480a6adc1e25e81caa44c749ec81976192e2ec9
34c64
"Hello, world!1" =>
e9afc424b79e4f6ab42d99c81156d3a17228d6e1eef4139be78e948a933
2a7d8
"Hello, world!2" =>
ae37343a357a8297591625e7134cbea22f5928be8ca2a32aa475cf05fd4
266b7
...
"Hello, world!4248" =>
6e110d98b388e77e9c6f042ac6b497cec46660deef75a55ebc7cfdcf65cc
0b965
"Hello, world!4249" =>
c004190b822f1669cac8dc37e761cb73652e7832fb814565702245cf26e
bb9e6

"Hello, world!4250"=>

0000c3af42fc31103f1fdc0151fa747ff87349a4714df7cc52ea464e12d
cd4e9
```

Running 4251 hashes on a modern computer will not be very much work, since most machines are capable of achieving 4 million hashes per second. However, Bitcoin automatically varies the difficulty in order to achieve a fairly constant growth in the block generation [16]. To make things even more difficult, the header contains the Merkle Tree, which links the node back to the genesis block and adds an additional layer of security that ensures blocks cannot be faked.

3.2 Under the Hood

At the core level, Bitcoin is nothing more but a digital ledger that is maintained on every computer on the Bitcoin network. When sending money, Alice broadcasts to the network that the amount on her account should go down and the amount on Bob's account should increase. Nodes on the network apply that transaction to their copy of the ledger and pass on that transaction to other nodes. Everyone knows everything about other transactions. So how has this system been designed so that no trust is needed? How can a group of strangers manage each other's transactions? How can nodes be sure that the request is authentic and that only the rightful owner has sent the message? This is all done by the digital signature, which proves the authenticity of a message and is different for every transaction.

Figure 7: Bitcoin Key Block Diagram

The private key is used to create a signature and the public key is used to verify the transaction. The public key is actually the “send to” address that one sees linked to a Bitcoin wallet. To spend money, users must prove that they are the owners of the public key address where money was sent. In order to do this, the user creates a signature through a function of the private key and the transaction message, and other nodes can

use that signature in a different function to verify that it corresponds to the user's public key. Any changes to the message after being passed around the network will cause the signature to become invalid, thus invalidating the transaction. The math behind this is done using the elliptic curve digital signature algorithm (ECDSA), which will be discussed in more detail in the next section. The account balance in the ledger is found by summing up all the transactions sent to and from that address. So in order to verify that an owner has enough funds, all the transactions linked to that address are referenced to make sure that there are sufficient funds. Other nodes will also check these inputs to verify the recipient and that there are indeed sufficient funds. Therefore, each transaction is dependent on previous transactions, which can be linked all the way to the first transaction made. When first installing the Bitcoin client on a machine, all transactions in the network are downloaded and verified. This process can take over 24 hours, but only needs to be done once. After a transaction has been used once, it is considered spent and cannot be used again, which prevents the issue of double spending. Complex transactions can also be created for the use of escrow transactions for which more than one signature is required to gain access to the funds. Lost private keys can lead to Bitcoin taken out of circulation and can lead to deflation over the years. Users can create an unlimited amount of private and public keys to use. The probability of two people obtaining the same private and public key is nearly zero as there are 2^{160} different possibilities.

How can the system tell which transaction comes before another? Because of propagation effects, some nodes may receive a double spending transaction before the legitimate one comes through, causing the legitimate transaction to be invalid. This is

solved by grouping transactions into blocks, which allows the network to agree on the order of transactions. This process takes on average 10 minutes. Blocks are then linked in the blockchain, and referenced to the previous block. Blocks cannot be pre-computed since each block depends on the cryptographic hash of the previous block. Many nodes can work on computing a block and there may be collisions where multiple nodes create blocks at the same time. In the case of multiple blocks being created at the same time, the network simply builds on top of the first block it received. Others may have received the block in a different order and build on that block. The tie breaks when the next block is solved and users switch to the longest branch available. The blockchain quickly stabilizes as more blocks are added [8]. This means that blocks that have been in the network longer are more secure than blocks that were just added to the network. For this reason, it is better to wait for several blocks to be computed before confirming a transaction as final, thus avoiding the issue of a double spending attack.

Having a block rate of ten minutes means that it can take up to ten minutes before a user's transaction is executed and propagates to other nodes on the network. In addition, it is better to wait for several blocks to be mined before confirming that a large transaction has made it through the network and is securely in the blockchain. However, ten minutes can be a large inconvenience when compared to current payment systems. For instance, a transaction made on PayPal, Venmo, or other point of sale machines goes through almost immediately. So comparing this to the ten minutes that it takes for a Bitcoin transaction to go through can seem a bit inconvenient. However, there is a case to be made for transactions that occur from country to country. Money transferred via Western Union can take up to 24 hours to process. Comparing this to Bitcoin, a user can

send money to another user overseas and can be certain that the transaction is in the blockchain and secure after multiple blocks have been added on top of the block containing their transaction. The more blocks that have been added after the block containing the users transaction, the more certain that user can be that their transaction went through and has propagated through the network. From the examples stated above, it may be safe to conclude that Bitcoin may never make it to retail outlets, since the current point of sale systems authorizing a card transaction early instantaneously.

3.3 ECDSA

The elliptic curve digital signature algorithm (ECDSA) is used in Bitcoin to verify that a transaction is made by the rightful owner and creates a public key (E, G, Q, p) from a prime number q , an elliptic curve $E \bmod q$, a random point G of some prime order p , a point $Q = dG$, where d is the user's private key, and ensures that funds can only be spent by their rightful owners [19]. The equations below give the specific numbers used in Bitcoin:

$$q = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$$

$$E: y^2 = x^3 + 7$$

The cardinality of the elliptic curve results in the prime factorization of a single large prime number given as

$p = 115792089237316195423570985008687907852837564279074904382605163141518161494337$

This means that any point lying on the elliptic curve will be of some order of this prime number. The base point G used is given in compressed hexadecimal form as

$G = 02\ 79BE667E\ F9DCBBAC\ 55A06295\ CE870B07\ 029BFCDB\ 2DCE28D9\ 59F2815B\ 16F81798$

We also define a function $\Phi(x)$ that takes a number mod q and lifts it to be an integer between 0 and $q-1$ that reduces to it. Then returns this value mod p , this basically is a mapping from q to p . Next we generate a random secret key d which is a random number modulo p , and now we have everything we need to sign a message [19].

We initialize Q as $G * d$ and our public key is given as Q, p, E , and G [19]. To sign a message, we take the message and hash it in SHA256 modulo p , which results in a 64-bit integer that we will define as z . We now generate a random number k between 0 and $p-1$. It is very important that this random value k is regenerated for every message we wish to sign, or it will be very easy to derive the secret key, as was this case with the random number generator in Android. The early versions of the Android operating system generated a random number that was hard coded and never changed [8]. This caused many of the applications and Bitcoin wallets to be vulnerable to attacks, and resulted in the loss of Bitcoin. Now that we have the random value k , we compute kG and sign the message using the equations below to give the pair (r,s) [19]:

$$r = \phi(x(kG)) \in F_p^* \quad \text{and} \quad s = \frac{z + rd}{k} \in F_p$$

Using our values above, $r=kG \bmod p$ and $s=(z+rd)/k \bmod p$. We can now check the signature is valid without any prior knowledge of the private key. Which is very important since exposing or hardcoding a private key can lead to vulnerabilities. To check that this signature is valid we compute C defined below [19]:

$$C = \frac{z}{s}G + \frac{r}{s}Q \in E(F_q)$$

The signature is valid if $\phi(x(C)) = r$. The proof behind why this is a sensible

calculation to verifying that a signature is valid is shown below [19]:

Proof: since (r,s) is valid, we have $s=(z+rd)/k$, so $k=(z+rd)/s$. Thus

$$kG = \frac{z + rd}{s} G = \frac{z}{s} G + \frac{rd}{s} G = \frac{z}{s} G + \frac{r}{s} Q = C$$

Again it is very important that we recalculate a random k for every message we sign in order to avoid exposing our private key. For instance, if we use the same k to sign two messages, we will be given z_1, z_2 and s_1, s_2 . Now solving $(z_1 - z_2)/(s_1 - s_2)$ will give us our random value k . Now, $(s_1 * k - z_1)/r$ will give us our private key d [19]. The code for ECDSA can be found in the appendix and was written using SAGE (an open source math software). Ethereum uses the same elliptic curve for their platform.

3.4 Bitcoin as a Currency

A common misconception about Bitcoin is that it is completely anonymous. This however is not true. All transactions are shown in the blockchain and can be followed. The blockchain can be used to show the balance of a Bitcoin address, as well as the transactions that occurred on that address. The part that is somewhat anonymous is that there is no name or identity tied to the address. This however can be overcome by using what is called a “colored” Bitcoin. The “colored” coin can be used to add an additional layer of information, and can be used to store information such as a username or location. In a recent case involving the online marketplace SilkRoad, which is known for selling illegal drugs using Bitcoin, the FBI used colored coins to track the origin of the marketplace to Ross Ulbricht, a San Francisco resident [13]. There are also a lot of misconceptions on the security of Bitcoin. The biggest vulnerabilities arise from user error [21]. A user may accidentally publish his or her private key or the key may

accidentally be deleted or stolen. This can lead to the user's wallet being compromised or lost forever. Once a private key is lost, the Bitcoin in that wallet are put out of circulation because they can never be retrieved. The Bitcoin protocol itself is all open-source and although some vulnerabilities or bugs have been found, these were quickly patched through the open-source community.

Bitcoin have value for the properties that they hold. A Bitcoin has all the same characteristics as money in terms of durability, profitability, fungibility, scarcity, divisibility, and recognizability, but is based on the principles of mathematics rather than a central authority [21]. Early this year, the IRS declared Bitcoin as a commodity, meaning that it will be treated just like stock [1]. Like any stock, we can see how it is possible for Bitcoin to be profitable or scarce, especially since there is a fixed amount of 21 million Bitcoin that will be put into circulation. Like all commodities, a Bitcoin's price is mainly determined by the laws of supply and demand. When demand increases, so does the price and when demand falls, the price follows. The adoption rate is also a key factor in Bitcoin's success. The adoption rate has been steadily increasing over recent years, with many companies offering users the ability to purchase goods using Bitcoin. Bitcoin is still highly volatile since it is still under heavy speculation and has a fairly low adoption rate. The table below illustrates the price fluctuation over recent years. In order for Bitcoin to stabilize, the crypto-currency needs to be adopted by a large economy. However, a large economy would not adopt the currency until the price stabilizes. This leads to the classic chicken and the egg problem, since one must come first.

Figure 8: Bitcoin price over lifetime

3.5 Legal issues in Bitcoin

There have been many recent events with Bitcoin that have made the price even more volatile. China has been the source of many hot topics in the Bitcoin community over the past several months. In mid-2013, Chinese users fueled the Bitcoin craze by trading nearly 100,000 Bitcoin daily and pushing the value of a Bitcoin to over \$1000. However, now the number of transactions in China has dropped to under 2,000 Bitcoin traded daily. So what happened? Mainly, the country passed several new regulations that highly limited the use of Bitcoin. In early December 2013, China declared that it would not recognize Bitcoin as a currency. This caused a slight dip, but investors were still present in the market and there was a reasonable amount of Bitcoin being traded. In response, China passed another regulation early this year limiting all the banks in China

from making any Bitcoin transactions. This nearly killed the incentive for any investors to pursue Bitcoin in China.

The same sorts of rulings are occurring in the US, but they are not as extreme. The Financial Crimes Enforcement Unit (FinCEN) stated that any Bitcoin exchange or company must declare the company as a money transfer unit and must comply with the necessary documentation process. However, in March the IRS stated that it will not treat Bitcoin as a currency but rather as an investment and any gain must be declared and taxed as capital gain [1]. This has now lead to all sorts of cases in which companies that were seen as participating in money laundering are now stating that the claim cannot be valid since Bitcoin is not officially recognized as a currency.

So should Bitcoin be seen as a currency? Even though Satoshi Nakamoto conceived of Bitcoin as an alternative currency in the white paper, I do not believe Bitcoin should be used as a currency. The fact is that if Bitcoin is ever treated as a currency, it will become heavily regulated until it is under the full control of the government. However, it is best to use Bitcoin more as a token or intermediary to reduce the friction in a transaction. For example, a user who wishes to transfer money from the US to Africa can first exchange the amount to Bitcoin and send it. This makes the transaction fee nearly free and time to de nearly instantaneous, thus reducing the friction. Then in Africa, the user can convert the Bitcoin back to their native currency. This scenario rides on the Bitcoin network only to make the process easier and reduce the friction. However, it may still lead to problems with governments since large transactions are usually monitored for authenticity.

3.6 Future of Bitcoin

One of the biggest fears of Bitcoin is its scalability and volatility. If Bitcoin were to ever go mainstream as a currency, it must be able to compete against major credit card companies, which process nearly 2000 transactions per second. As of now, the Bitcoin network has been hard coded to limit the number of transactions to seven transactions per second. This limit was set in place to prevent users from inflating the size of the blockchain before the network and community was ready for it. Theoretically, if Bitcoin does catch on, all that is needed to increase the number of transactions is to release an update where a few lines of codes containing this threshold have been removed. Next it is important to realize how much bandwidth is required to handle a system with around 2000 transaction per second (tps). Each transaction can vary from 0.2 to 1 kilobyte, with an average of 0.5 kilobytes as of now. This means that 2000 tps multiplied by 0.5 kilobytes is 1000 kilobytes per second, or roughly 0.97 megabytes per second. This results in 7.8 megabits per second bandwidth, which is already common in many residential connections today. However, with a very large number of transactions, this will lead to an increasing size per block [12]. As of now, the current blockchain is about 15GB and grows at nearly 1 MB per hour. So if Bitcoin were to handle 2000 tps, the blockchain would grow nearly 1MB every 3 seconds, or 8 TB per year. Keep in mind that every miner has a record of the entire blockchain. This means that every user on the network will have to constantly grow their storage as the overall size of the blockchain grows at a more rapid rate. This could lead to a huge centralization risk, where a small number of large businesses could run the full nodes, while most users would run a light version that does not depend on downloading the entire blockchain. The risk in this is that

the businesses can work together to cheat and change the block reward to give themselves more Bitcoin [3]. One way the Bitcoin network is trying to deal with this ever-increasing blockchain is to apply a method of pruning. In the white paper, Satoshi explains a method to delete unnecessary data on transactions that are fully spent. Even though this will reduce the size of blocks and decrease the growth rate of the blockchain, the long-term effect will still result in an extremely large blockchain.

Since Bitcoin is open-source, it is very easy to copy the code and create a new alt-coin. Many alt-coins try to approach the problem of scalability differently using a different structure in the blockchain. One such paper discusses the concept of a new crypto-currency that uses a finite mini-blockchain and a rolling blockchain [2]. First, we need to identify the three basic functions of a blockchain: 1) To coordinate how the network coordinates transactions 2) Encapsulate the proof of work which secures the networks 3) To manage account balances and record the ownership of coins. This proposal relies on the concept of only storing the updated balance on an account rather than keeping track of all the transactions that have occurred. This vastly simplifies the blockchain and reduces much of the overhead, but will most likely fail due to the simple fact that people may want to keep track of all the transactions made. The mini-blockchain is the next factor in the proposal, which keeps a finite amount of the blockchain rather than constantly growing. The mini-blockchain in which we shift a window of fixed size would not work for Bitcoin, since it depends on the full ledger to sum up all the transactions occurred on an address. Each block in the mini-blockchain must also embed the master hash in the header in order to ensure that the blocks were not artificially created and remain secure [2].

There are many other methods and ideas discussed in the Bitcoin forums for reducing the size of the blockchain. Chapter 5 will discuss how Ethereum plans to tackle this problem.

CHAPTER 4 ETHEREUM

4.1 What is Ethereum

Bitcoin presented two very new concepts to the world. One of which was the idea of the blockchain, which represents a very transparent public ledger that all can see and cannot be removed. Second was a transaction system that allows a user to move positions in the ledger from one party to another, much like a currency. There is a third concept about a robust scripting language, but was never presented to reduce contamination on the Bitcoin platform before the world was ready for it. Five years later, we see that it is now possible to implement a Turing-complete scripting language and merging it with the idea of a trustless blockchain. The idea behind Ethereum is to do just that. It takes a very simple and complete Turing-complete language, and combines it with a blockchain, allowing anyone to implement his or her own application on top of that platform. The Turing-complete language allows contracts to run as AI clients and to pass and retrieve data from other contracts. Contracts can also change their current state and modify their own code. We will go into more detail about how contracts work in a later section. If we look at all the alt-coins, they all have a long list of features that have been added. Ethereum, rather than trying to have a ton of features and be the Swiss army knife of alt-coins, is a universal language and platform that allows developers to build any features they want on top of it. A Distributed Autonomous Organization (DAO) is the idea of an autonomous agent, much like an AI. The DAO takes the agents and embeds them on top of the blockchain to fulfill the facility of an organization. This will be discussed in further detail later on. A contract is one of the most important aspects of Ethereum and is the fundamental building block for developers. The contract is like a DAO in that it lives

inside the Ethereum network and is executed by the nodes [11]. The code in the contract can be created to do anything from being another alt-coin to an escrow service or decentralized Dropbox. The contracts can be used to create a wide variety of decentralized applications.

“The intent of Ethereum is to merge together and improve upon the concepts of scripting, alt-coins and on-chain meta-protocols, and allow developers to create arbitrary consensus-based applications that have the scalability, standardization, feature-completeness, ease of development and interoperability offered by these different paradigms all at the same time [3].” The foundational layer behind Ethereum lies in a blockchain with a Turing-complete programming language, allowing anyone to write their own smart contract and decentralized applications with specified rules of ownership, transaction formats, and state transition functions.

The Ethereum client is currently in its alpha stage and can run on Windows, Mac, and Linux. Currently, I have the client running on Linux through a Virtual Machine. The layout can be found in the image below:

Figure 9: Ethereum Client

The current version allows users to connect to the network and mine Ethereum.

The software allows users to create contracts and upload them to the Ethereum blockchain. Users can also view other contracts as well as user accounts on the network. The software also displays a log console and network status. Users can test contracts on the network by creating multiple addresses and executing the contract. The amount mined however will be reset when the actual client goes live in the fourth quarter of this year. With the current version, we can see other people on the network as well as the contracts that are stored in the blockchain. This is especially useful in testing the contracts that other users created.

4.2 Accounts, Messages, and Transactions

State transitions in Ethereum are made up of accounts, each having their own 20-byte address, where each state transition is an exchange of value and information between accounts. The account is composed of the following four fields:

- The nonce, a counter used to ensure transactions are only processed once
- The account's current ether balance
- The account's contract code (if present)
- The account's storage (empty by default)

“Ether” is the main internal fuel for Ethereum and is used to pay for transaction fees within a contract. There are two types of accounts in Ethereum: externally owned accounts and contract account. The externally owned accounts are controlled by private keys and have no code. A user can send messages from an externally owned account by creating and signing a transaction. The contract account is controlled by the contracts code and activates every time every time the account receives a message. Once the account is activated, the contract can read and write to internal storage and send other messages or run other contracts [3]. The main reason for having a currency in the network is to incentivize and award miners for authorizing transactions and securing the network, and secondly it serves as a means for paying transaction fees for anti-spam purposes. Ether will have a limit of 2^{128} units, compared to Bitcoin's $2^{50.9}$ units, but only 2^{100} will be released in the future. The breakdown of ether and its divisions is shown in the table below:

- 1: wei
- 10^3 : ___
- 10^6 : ___
- 10^9 : Koblitz
- 10^{12} : Szabo
- 10^{15} : Finney
- 10^{18} : Ether

The name for 10^3 and 10^6 will be auctioned off during the release of the network to raise money for future development. Ether will be sold in a fundraiser for 0.0001BTC per ether. However, to avoid all the ether to go to a select few, the money raised will be distributed in such a way as to give a fair chance to individuals entering at a future time. Suppose that X ether is collected in the fundraiser, $0.25X$ ether will go to the founders, $0.25X$ ether will be given to the Ethereum organization as a reserve pool to pay expenses in ETH such as ETH salaries or bounties for those developers who want part or all of their compensation to be in this form, and $0.5X$ ether will be mined per year forever after that point (ie. permanent linear inflation).

The so-called “messages” are much like the transactions in Bitcoin, with several key differences. A message can be created by an external account or a contract, where in Bitcoin, a transaction can only be created from an external account. Next messages have an explicit option to contain data, much like the data that can be put into a Bitcoin transaction [3]. Finally, if the recipient of a message is a contract, it has the option to run another function and return a response.

Each transaction in Ethereum is a signed data package that stores a message to be sent from an external account. The transaction contains the recipient of the message, the sender’s signature, the amount of ether and data, as well as the “startgas” and “gasprice.” Unlike Bitcoin, Ethereum can contain loops, which can lead to infinite loops or exponential blowups. To prevent this, each transaction is required to set a limit, STARTGAS, on the number of computational steps of code execution it can spawn, including the initial message as well as any messages produced during execution. The GASPRICE is the fee to pay to the miner for each computational step [3]. This way, in

the case of an infinite loop, the transaction will eventually run out of gas and revert back to the original state. If a transaction finishes execution with left over gas, the remainder is then refunded back to the sender.

A transaction is stored as:

[nonce, receiving_address, value, [data item 0, data item 1 ... data item n], v, r, s] [3]

Where nonce is the number of transactions already sent by that address, and (v, r, s) is the raw Electrum-style signature.

4.3 State Transition Function

Figure 10: Ethereum State Transition Function

The state transition function that applies the State and Transaction to State' (as shown above) can be described as follows [3]:

1. Check that the transaction is well formed, the signature is valid, and nonce matches that of the sender's account. Else return error

2. Set transaction fee to $\text{STARTGAS} * \text{GASPRICE}$ and get the sending address from the signature. Deduct the fee from the sender's address and increment the nonce. If balance is too low, return error.
3. Initialize the GAS as the STARTGAS and deduct a certain amount of GAS per byte in the transaction.
4. Transfer the transaction value from the sender to the receiver. If the receiver's account does not yet exist, create it. If the receiver is a contract, run the contract's code.
5. If the transfer failed due to insufficient funds or code execution running out of gas, revert all states back to their original state and add the fees to the miner's account.
6. Otherwise, the code execution has completed and the remaining fees are refunded back to the sender, and the gas consumed is given to the miner.

Below we will explain a very simple sub-currency contract to gain a better understanding of how contracts work.

```
1  if tx.value < 100 * block.basefee:
2      stop
3  elif contract.storage[1000]:
4      from = tx.sender
5      to = tx.data[0]
6      value = tx.data[1]
7      if to <= 1000:
8          stop
9      if contract.storage[from] < value:
10         stop
11     contract.storage[from]=contract.storage[from]-value
12     contract.storage[to]=contract.storage[to]+value
13 else:
14     contract.storage[MYCREATOR]=10^18
15     contract.storage[1000]=1
```

In the first line, we are checking that the value sent to the contract is enough. We get sixteen computation steps for free, but if we run out, the contract will hold somewhere in the middle. To trigger the execution of the contract we have to send a transaction with a certain amount of ether attached to it. The “basefee” is a variable or constant that is set in the Ethereum server code and may change over time based on the demand and number of miners. The first time we run this contract, if we pass line one, we then go to lines fourteen through fifteen, since the storage locations are always initialized to zero. We are using the arbitrary storage location 1000 to check the contract state, and to store extra information. So on lines fourteen and fifteen, we are assigning the balance in the storage location, where MYCREATOR is a placeholder for the hash of an account’s public address and we set the state of the contract at location 1000 to one. This specific contract is immutable, but contracts can also be created to change themselves. The contracts code lies in the contracts storage, which is why we choose a large location like 1000 so that we do not overwrite some of the contracts code. So the next time someone sends a transaction to this contract, we will go to line four. Here we get the sender of the transaction and store it in a local variable. The additional data contains the recipient and the value of the transaction. If the recipient’s address is not valid, we stop the contract. If the value to be sent is less than the balance of the sender, we quit execution. Otherwise, we deduct the value from the sender’s balance, and increment the receiver’s balance. It is important to note that some parts of the code are more expensive to run than others. For instance, it costs more to read `contract.storage[from]`, so if we wish to optimize this code further, we can create a local variable to store this value.

The contract code uses its own language referred to as Ethereum virtual machine code, which is a low-level, stack-based byte code language known as LLL (low-level lisp-like language) [3]. Each operation has access to the stack in which to push and pop value, an infinitely expandable byte array of memory, and the contract's long-term storage that works as a key/value store. There are also applications developed that allow for the code to be written in multiple languages and later compiled down to the Ethereum virtual machine code.

4.4 Ethereum Blockchain

Data in Ethereum is stored using Recursive Length Prefix (RLP) encoding, which serializes arrays of strings of arbitrary lengths into strings. As example, ['dog', 'cat'] is serialized in byte array format as [130, 67, 100, 111, 103, 67, 99, 97, 116]; the general idea is to encode the data type and length in a single byte followed by the actual data (eg. converted into a byte array, 'dog' becomes [100, 111, 103], so its serialization is [67, 100, 111, 103]. Thus, a full block is stored as [3]:

[block_header, transaction_list , uncle_list]

The block header is [3]:

[parent hash, sha3(rlp_encode(uncle_list)), coinbase address, state_root, sha3(rlp_encode(transaction_list)), difficulty, timestamp, nonce]

Where the data for the proof of work is the RLP encoding of the block without the nonce. Uncle_list and transaction_list are the lists of the uncle block headers and transactions in the block, respectively. The state_root is the root of a Merkle tree containing key/value pairs for all addresses. At each address, the value stored in the

Merkle tree is a string, which is the RLP-serialized form of an object of one of the following two forms [3]:

[balance, nonce]

[balance, nonce, contract_root]

The nonce is the number of transactions made from the address, and is incremented every time a transaction is made. This means that each transaction is only valid once and prevents replay attacks. In addition, this makes it nearly impossible to construct a contract with the same hash as a pre-existing contract. Balance refers to the contract or address's balance (in wei). Contract_root is the root of another Patricia tree, containing the contract's memory, if that address is controlled by a contract.

Ethereum uses a very similar structure to the blockchain as Bitcoin with some subtle differences. The main difference being that Ethereum contains a copy of the transaction list and the most recent state. Each block also contains the block number as well as the difficulty. Block validation is broken down in the following steps:

1. Check that the previous block exists and is valid
2. Check that the timestamp of the block is greater than that of the previous block and less than 15 minutes into the future
3. Check that the block number, difficulty, transaction root, uncle root, and gas limit are all valid.
4. Check that the proof of work of the block is valid
 - a. Define $s[0]$ as the state_root of the previous block

- b. For all n transactions in the block, set $s[i+1]$ equal to $\text{apply}(s[i], \text{transaction}[i])$. If process returns an error or the block has run out of gas, the block is not valid and return error
- c. Define s_{final} as $s[n] + \text{block reward paid to miner}$
- d. If s_{final} is the same as the state_root , then the block is valid, otherwise we return an error

Figure 11: Ethereum Blockchain

This approach may seem inefficient at first since we are carrying all the information from the last block with us, but helps in the long run since the growth of the block chain is vastly reduced. The difference between two blocks is very little so we can store the data once and reference twice using pointers and a Patricia Tree. This means that we do not need to store the entire blockchain history and results in 10 to 20 times reduction in space usage compared to Bitcoin [3].

The blockchain mining time is also vastly shorter than that of Bitcoin. Currently blocks in Bitcoin are mined roughly every 10 minutes, in contrast, Ethereum blocks are mined roughly every 60 seconds. This has its advantages and disadvantages. Starting with the disadvantages, this can lead to more network insecurity by increasing the ability to allow double spending. This process has been resolved through the “Greedy Heaviest Observed Subtree” (GHOST) protocol [3]. Earlier we learned that when two blocks are

found at the same time, only one of the miners gets the 25 BTC reward and the other block is disregarded after the longest chain is made. In Ethereum however, even the second place miner gets rewarded for their work through the increased security used in the GHOST protocol. Blocks with fast confirmation times lead to increased stale rate, meaning that two miners solving a block at the same time will lead to one of the blocks being wasted and not contributing to network security. There is also a centralization issue where if mining pool A owns 30% of the hashing power and B owns 10%, A will be at risk of producing a stale block 70% of the time and B will produce a stale block roughly 90% of the time. Thus having a shorter block period would give an advantage to A over B. Ethereum uses a modified version of the GHOST protocol, which will be discussed in detail in the next section. GHOST solves the issue of network security by including stale blocks in the calculation of the longest chain. In addition there is also a small reward associated with stale blocks where the stale block receives 87.5% of its base reward, and the nephew of the block receives a 12.5% reward [3]. The modified version of the protocol is broken down below [6]:

1. Hash the data, and if it has not yet been received, pass it along to the data parser.
2. If the data parser verifies the data item is a valid block, go to step 3. If the data item is a transaction, check that the funds in the sending address is sufficient for the transaction to go through, if so add it to the local transaction list and publish it to the network. If the data item is a message, send the message to the message responder and return the response.
3. Check if the "parent" parameter in the block is already stored in the database, and exists. Otherwise:

- a. Check if every block header in the "uncles" parameter in the block has the block's parent as its own parent, otherwise exit.
- b. Call the state updater with the parent of the block, the transaction list of the block, the timestamp of the block, and the coinbase of the block and see if the block header outputted by the state updater is exactly the same. If yes, add the block to the database and publish it to the network, otherwise exit.
- c. Determine $TD(\text{block})$ ("total difficulty") for the new block. TD is defined recursively by $TD(\text{genesis_block}) = 0$ and $TD(B) = TD(B.\text{parent}) + \sum(u.\text{difficulty for } u \text{ in } B.\text{uncles}) + B.\text{difficulty}$.
If the new block has higher TD than the current block, set the current block to the new block.

If the node is mining, the node performs the following additional steps upon receiving a block [6]:

1. Determine if the block's parent is in the database. If not, discard it.
2. Determine $TD(\text{block})$ ("total difficulty"). If $TD(\text{block})$ is higher than any existing block in the database, start mining on that block and clear all uncles. Also, remove all transactions in the new block from the transaction pool if present, and immediately apply the remaining transactions to the new block.
3. If the block's parent is the parent of the highest block, add the block header to the set of uncles and restart the proof of work.

This approach increases the network security by using stale blocks to help verify the

longest chain. It also incentivizes miners with a low percentage of the overall mining pool to continue mining because they still receive a reward for stale blocks.

Ethereum also prevents miners from using dedicated application-specific integrated circuits (ASIC's) in order to prevent several detrimental effects [6]:

1. ASIC's negate the democratic distribution aspect of mining. The fact that everyone has a computer guarantees him or her at least some of the money supply. With specialized hardware such as ASIC's, this equality no longer exists and is dominated by only the larger players.
2. ASIC's increase resource waste. As seen in efficient markets, marginal revenue approaches marginal cost. In hardware, total revenue approaches total cost. So in a specialized hardware dominated market, the wasted resources is close to the security level of the network.
3. ASIC centralizes mining in the hands of the more powerful players. This causes 51% attacks to become more likely and increases regulatory pressure. A 51% attack occurs when over 50% of the network is controlled by a single party, allowing them to fake blocks.

In Ethereum, increasing the memory hardness can mitigate the problems that arise from using ASIC's. This analogy can be compared to the CPU and GPU. If we think of the CPU as a few very intelligent brains, and the GPU as many dumb brains, then the GPU is good for doing very repetitive tasks like taking the hash of a string and incrementing a nonce. Ethereum reduces this problem by increasing the memory intensity, making it more difficult for the dumb brains to solve the

problem. Of course hardware manufacturers can begin packing more memory into their chips, but hobbyists can also achieve these same results. In addition, memory is far more expensive to produce compared to SHA256 hashing chips. The way Ethereum proposes to solve the problem is through what is known as the DAGGER Algorithm [5]. The algorithm works by creating a directed acyclic graph (nodes of a tree can have multiple parents) with a total of $2^{23}-1$ nodes in sequence. Each node depends on 3-15 randomly selected nodes before it. If the miner finds a node between index 2^{22} and 2^{23} such that this resulting hash is below 2^{256} divided by the difficulty parameter, the result is a valid proof of work. For data D and nonce N, the code is as follows [5]:

```
D(data,xn,0) = sha3(data)
D(data,xn,n) =
  with v = sha3(data + xn + n)
    L = 2 if n < 2^21 else 11 if n < 2^22 else 3
    a[k] = floor(v/n^k) mod n for 0 <= k < 2
    a[k] = floor(v/n^k) mod 2^22 for 2 <= k < L
    sha3(v ++ D(data,xn,a[0]) ++ D(data,xn,a[1]) ++ ... ++
D(data,xn,a[L-1]))
```

Because the 2^{21} to 2^{22} phase requires 11 parents, the time-memory tradeoff attack is severely weakened - attempting to store 2^{21} nodes instead of 2^{23} reduces memory usage by a factor of 4 but slows down computation by a factor about 20. Thus, no practical time-memory tradeoff attack exists; close to the full 256 MB is required for any reasonable level of efficiency. Of course this algorithm is more memory intensive, but it regulates ASIC's from dominating mining. The algorithm takes 512 MB to evaluate, 112 KB memory and 4078 hashes to verify, and even the slightest time-memory tradeoff is not worthwhile to implement because of the bottom-level branching adjustment [5].

These parameters require 512 MB of RAM for a single thread. Because the primary determinant of hardness is memory, and not computation, specialized hardware has only a tiny advantage.

4.5 GHOST Protocol

The Greedy Heaviest Observed Sub-Tree was developed by Yonatan Sompolinsky and Aviv Zohar to increase the network stability and avoid 51% attacks [18]. Their proposal took advantage of blocks that were not on the main chain because they still contribute to the chain's irreversibility. For instance for a block B with children blocks C1 and C2 that were mined at the same time, will only use one of the children in the blockchain and drop the other. However, both children have accepted block B and its entire history as correct, so we can then add more weight to block B to ensure that it gets added to the blockchain. The overall algorithm behind the GHOST protocol is explained below [18]:

1. Set B as the genesis block
2. If B has no children, then return B
3. Otherwise, set B to the largest subtree
4. Repeat from step 2

This algorithm makes every block that was added as a subtree of B harder to omit B from the blockchain and assigns a higher priority to it since the block has been verified multiple times.

4.6 Stable-Value Currency

As we have seen from most crypto-currencies, the price is very volatile and is one of the main reasons that it could not survive or catch on as a replacement to fiat currencies. The government has regulation over the amount put into or taken out of circulation, mainly to adjust for inflation or deflation. With a volatile currency, it is very difficult to regulate the price to avoid extreme swings in inflation and deflation. One idea behind stabilizing the price of a crypto-currency is to create a contract that hedge against the volatility of ether or other crypto-currencies with respect to the US dollar. This can be done through a data-feed contract maintained by a special party that can update the exchange between the crypto-currency and fiat currency and allows other contracts to send a message to this contract to return the exchange rate. This contract could be implemented as follows [3]:

1. Wait for Alice to input X ether
2. Wait for Bob to input X ether
3. Record the USD value of X ether by sending a message to the data-feed contract, say this is \$Y.
4. After 30 days, allow Alice and Bob to reactivate the contract in order to send \$Y worth of ether, calculated from the data-feed contract, to Alice and the remainder to Bob.

This approach reduces the risks involved with investing in crypto-currencies and maps the crypto-currency to the value of a fiat currency. But how can we trust the data-feed to give us an accurate, unbiased exchange rate? This can be solved using financial derivatives. Instead of polling one user, or contract for the exchange rate, we can poll

decentralized market of speculators, betting on whether the price will either increase or decrease. Speculators have no option to default on their side of the bargain since their funds are also held in the hedge fund in escrow. This is still not fully decentralized, since we are still polling a single location for the exchange rate, but provides a vast improvement in reliability and fraud detection.

4.7 Contracts

Ethereum alone is just a programming language and any features that you want would have to be implemented in a contract. A contract is like an automated agent that lives inside the blockchain with some built in script. The contract can store ether or currency units and is activated by sending a small transaction to it. Users can send both money and messages to contracts in order to make them carry out certain computations.

A smart contract is “a mechanism involving digital assets and two or more parties, where some or all of the parties put assets in and assets are automatically redistributed among those parties according to a formula based on certain data that is not known at the time the contract is initiated [5].” One of the key properties of a smart contract is that there is a fixed number of parties. The number of parties does not need to be known at initialization time, and can run forever. An example would be an escrow or hedging contract. However, an entire decentralized exchange would not qualify as a smart contract since the number of known parties is not fixed.

A decentralized application is similar to a smart contract with two major differences. First off, a decentralized application has an unbound number of participants on both sides of the market. The second difference is that the application is not limited to

just financial applications and applies to a wide range of different programs. An example of a decentralized application includes torrent applications as well as Internet browsers.

The Decentralized Autonomous Organization was explained in an earlier section, but is redefined below for further clarification. The DAO is an entity that lives in the Internet and works autonomously, but also relies on individuals to conduct tasks that it cannot do alone [5]. The DAO has certain internal properties that can be used as a mechanism for rewarding certain activities. Torrent applications do not have this property, but we see it in Bitcoin and many of the other alt-coins.

Sending a transaction to the contract activates the contract and executes its scripting code. The contract creation and validation occurs as follows:

1. De-serialize the transaction and extract the sending address from the signature.
2. Calculate the transaction fee and make sure the creator is sending at least the item price plus the fee. Recall from earlier that each contract has a certain amount of gas to run and if the contract has gas left over, the amount is refunded back to the sender.
3. Check that the last 20 bytes of the hash of the transaction making the contract does not exist and create a contract at that address.
4. Copy data item k to memory slot k for all k in $[0 \dots n-1]$ in the contract.

When the contract receives a transaction, it performs the following steps [6]:

1. The contract's ether balance increases by the amount sent
2. The index pointer is set to zero, and $\text{STEP_COUNT} = 0$
3. Repeat forever:
 - a. if the command at the index pointer is `STOP`, invalid or greater than 255, exit from the loop
 - b. set $\text{MINER_FEE} = 0$, $\text{VOID_FEE} = 0$

- c. set `STEPCOUNT` <- `STEPCOUNT` + 1
- d. if `STEPCOUNT` > 16, set `MINERFEE` <- `MINERFEE` + `STEPFEE`
- e. see if the command is `LOAD` or `STORE`. If so, set `MINERFEE` <- `MINERFEE` + `DATAFEE`
- f. see if the command will fill up a previously zero memory field. If so, set `VOIDFEE` <- `VOIDFEE` + `MEMORYFEE`
- g. see if the command will zero a previously used memory field. If so, set `VOIDFEE` <- `VOIDFEE` – `MEMORYFEE`
- h. see if the command is `EXTRO` or `BALANCE`. If so, set `MINERFEE` <- `MINERFEE` + `EXTROFEE`
- i. see if the command is a crypto operation. If so, set `MINERFEE` <- `MINERFEE` + `CRYPTOFEE`
- j. if `MINERFEE` + `VOIDFEE` > `CONTRACT.BALANCE`, `HALT` and exit from the loop
- k. else, subtract `MINERFEE` + `VOIDFEE` from the contract's balance and add `MINERFEE` to a running counter that will be added to the miner's balance once all transactions are parsed. Note that `MINERFEE` may be negative in some cases, in which case the contract balance would actually increase
- l. run the command
- m. if the command did not exit with an error, update the index pointer and return to the start of the loop. If the contract did exit with an error, break out of the loop.

The language is maintained in a stack, which is initialized to zero. All the operations either manipulate the stack or perform special operations. The individual operations can be found in the appendix.

There are seven primary fees in Ethereum, six of which apply to contracts.

1. `TXFEE` (100x) - fee for sending a transaction
2. `NEWCONTRACTFEE` (100x) - fee for creating a new contract, not including the memory fee for each item in script code
3. `STEPFEE` (x) - fee for every computational step after than first sixteen in contract execution
4. `MEMORYFEE` (100x) - fee for adding a new item to a contract's memory, including when first creating a contract. The memory fee is the only fee that is not paid to a miner, and is refunded when memory from a contract is removed.
5. `DATAFEE` (20x) - fee for accessing or setting a contract's memory from inside that contract
6. `EXTROFEE` (40x) - fee for accessing memory from another contract inside a contract
7. `CRYPTOFEE` (20x) - fee for using any of the cryptographic operations

The STEPFEE stops the problem of an infinite loop occurring in a contract, since the contract will eventually run out of gas and exit.

The contract I decided to investigate was a decentralized version of PayPal, this means that the contract consists of three parties, the sender, the receiver, and the intermediary. On top of these three, the receiver should be able to associate a rating to the sender, which builds up over time and establishes a score on how trustworthy they are. This concept is fairly simple to implement in Ethereum using a few dozen lines of code. The next step is to allow this contract to work for any currency. Although Ethereum cannot be used for fiat currencies, it can still be used for other digital crypto-currencies. The contract for creating an escrow like service can be done in a dozen lines of code and is explained below:

```
1. if tx.value < 100 * block.basefee:
2.     stop

3. state = contract.storage[1000]
4. if state == 0 and tx.value >= PRICE:
5.     contract.storage[1000] = 1
6.     contract.storage[1001] = tx.sender
7.     contract.storage[1002] = tx.value
8.     contract.storage[1003] = block.timestamp
9. else if state == 1:
10.    if tx.sender == VERIFIER:
11.        mktx(MERCHANT, contract.storage[1002], 0, 0)
12.        contract.storage[1000] = 2
13.    else if block.timestamp > 30 * 86400 +
contract.storage[1003]:
14.        mktx(contract.storage[1001],
contract.storage[1002], 0, 0)
15.        contract.storage[1000] = 3
```

The first two lines checks to make sure we have enough ether to run the contract. We then store the current state of the contract in a local variable. The first time the contract is called, state will be initialized to zero. On line four, we check to see if the

contract is in the initial state and the sender is sending an amount greater than or equal to the PRICE of the item. If so, we want to update the state of the contract to one. We then store the sender's address, the sender's balance, and the timestamp of the execution. The timestamp will be used later to offer a refund if not paid within 30 days. The next time the contract is run, the state will be set to 1 and we will go to line nine. If the state is one, we check if the sender is the VERIFIER (the person that will be shipping the item). If the sender is the verifier, then we want to make a transaction to the merchant for the amount that the customer paid initially. We then set the status of the contract to 2 (meaning shipped/paid). The last thing we want to do is issue a refund to the customer if nothing happened after 30 days. If so, we make a transaction back to the customer for his original payment and set the contract state to three (meaning refunded). Note that PRICE, VERIFIER, and MERCHANT are constants that need to be initialized when starting the contract.

We now want to modify the contract to add the option for the customer to assign a rating to the merchant, much like we see on PayPal. There are two ways we can do this. The first would be to create a new transaction when the contract is in state '2' where the amount will be a micro transaction between 0-100 as the rating. This will save on CALLDATALOAD, but may also disincentivize customers from giving a good rating. Although, most users probably won't cry over 100 wei (the smallest breakdown of ether). The second method for doing this is to send a transaction with the rating data attached. Since data is currently cheap, this is the preferable method. In order to do this, the customer sends a new transaction to the contract after the item has been shipped. This

occurs the next time the contract is run and the current state is at 2. The below contract includes this modification.

```
1. if tx.value < 100 * block.basefee:
2.     stop

3. state = contract.storage[1000]
4. if state == 0 and tx.value >= PRICE:
5.     contract.storage[1000] = 1
6.     contract.storage[1001] = tx.sender
7.     contract.storage[1002] = tx.value
8.     contract.storage[1003] = block.timestamp
9. else if state == 1:
10.    if tx.sender == VERIFIER:
11.        mktx(MERCHANT, contract.storage[1002], 0, 0)
12.        contract.storage[1000] = 2
13.    else if block.timestamp > 30 * 86400 +
contract.storage[1003]:
14.        mktx(contract.storage[1001],
contract.storage[1002], 0, 0)
15.        contract.storage[1000] = 3
16. else if state == 2:
17.    if tx.sender == CUSTOMER:
18.        mktx(MERCHANT, 0, DATA, 0)
19.        contract.storage[1000] = 4
```

The last four lines of code include this modification and check to see that the contract state has been shipped and that the transaction sender is the customer. We then make a transaction to the MERCHANT with a rating in the DATA field. Lastly, we update the state of the contract to four. We have now made a contract that users can rate merchants after an item has been shipped.

To test the contract, we start with the escrow test script written in Python (found in appendix) and add in the rating feature and rating test cases.

The next step of this contract would be to create a GUI that allows the customer to view detailed ratings of the merchant. A mockup of this can be seen in the image below:

Figure 12: PayPal Contract GUI

4.8 Future of Ethereum

The future of Ethereum holds huge potential. Currently the platform is still in the alpha stages and is scheduled to go live in the fourth quarter of this year. Ethereum is just a platform that runs off a similar blockchain to that of Bitcoin, with its main potential being in the smart contracts that develops can make. There are a great variety of types of contracts that can be made, and each will be completely decentralized. A decentralized version of PayPal can come in handy, as it provides a method for customers to assign a rating to merchants and provides a decentralized third party to act as the trustee for making the transaction legitimate. Ethereum runs off its own version of currency, which acts mainly as an incentive for miners and to run the contracts. Contracts can also be implemented to create an entirely new type of digital currency, which simply runs on the Ethereum network. Ethereum has taken the concept of the blockchain to an entirely new level and has created a platform that allows a developer to come in and create a decentralized application.

CHAPTER 5 CONCLUSION

This thesis has gone from the origins of payment platforms to where I see the industry going next. My interest in payment systems grew throughout the creation of AnyCoin and helped stem into Bitcoin and Ethereum. AnyCoin was a great project that helped me learn a great deal about how current digital wallets work and what users are looking for in a mobile wallet. Although my analysis of AnyCoin concludes that convenience is not enough of an incentive to get a wide user base, I still learned a great deal from this project. It allowed me to see what goes into the digital wallets like PayPal and Venmo, and what users are looking for in a mobile application.

Comparing cash or card systems to digital wallets, like PayPal and Venmo, gives a good analysis of the pros and cons in each system. The digital wallets like PayPal, Venmo, Square, and Stripe have a slightly higher transaction fee compared to using a card on one of the credit card company's point of sale machines. When comparing the time for a transaction to go through on a digital wallet compared to using credit card company's machines, the two are nearly identical, since the process of authorizing a transaction is the same. Many of these companies have gone through denial of service attacks or have had users wallets compromised. The Denial of Service attacks can be eliminated in the case with Bitcoin and Ethereum, however still exists on the Bitcoin exchanges and Bitcoin online wallet providers like Coinbase.

Bitcoin changed the way we think about money by introducing the concept of a trust less blockchain that allows people to move money around without any central party in a way that does not depend on trust, but is bounded by algorithms. This study explains

how Bitcoin works and the flaws in the current system. We learned that Bitcoin can fall subject to becoming too centralized or becoming too regulated. Comparing the speed of which a transaction is authorized on Bitcoin versus digital wallets depends on whether we are making a local transaction or sending money overseas. For instance, the case where a user wishes to buy merchandise from a retail store will be able to make this transaction faster with credit cards and digital wallets compared to Bitcoin. This is because a transaction can take up to ten minutes before being authorized and showing up on the blockchain, whereas using a card is nearly instantaneous. However, when it comes to sending money abroad Bitcoin is a far better option because it greatly reduces the transaction fee and can be far faster to approve. Bitcoin is still fairly new and still has great room for growth, however I believe it is best to use Bitcoin only as an intermediary to reduce the friction that occurs in making transactions. For instance, a user living in the US can convert their amount to Bitcoin and send the Bitcoin to a user living in Chad. That user can then convert the Bitcoin to CFA to use locally. The conversion process occurs at various Bitcoin exchanges that are setup around the world. Mt Gox was one of these exchanges, which filed for bankruptcy after being hacked.

What we learned from Bitcoin though is that it is possible to create a decentralized service, much like we see with torrents, that relates to currencies. Ethereum takes this concept a step further by introducing a platform that allows a developer to create decentralized applications. This thesis talks about how Ethereum works and discusses the concepts and flaws of Bitcoin and how Ethereum improves on them to create a decentralized platform for developers to build on top of. The contracts I created do show that it is possible to create an alt-coin and decentralized version of PayPal with a

few dozen lines of code. Ethereum is still in its very early stages and it will be very interesting to see where this technology goes.

Throughout this project, I have created and tested a mobile application on both the iPhone and Android devices. Even though AnyCoin was a great learning experience, this study shows that convenience is not enough to create a widely used application.

However, centralized applications have many drawbacks and provide a single point of attack for hackers. This led to the study of decentralized crypto-currencies like Bitcoin, and how they are different from services like PayPal, Venmo, and Square. This study shows that it is not likely that crypto-currencies will ever replace credit cards or cash, at least for some time. However, Bitcoin shows that it is possible to create a decentralized system, which led to the study of Ethereum and how they take the concepts of Bitcoin and improve on them to create a platform for developers to create decentralized applications. The contracts I created on Ethereum show that it is possible to create a decentralized version of PayPal with a few dozen lines of code. I plan to continue working on Ethereum and look forward to its release later in the year. The trend shows that there is a big change coming to the way we use money and how it is transitioning into a decentralized era.

BIBLIOGRAPHY

- [1] "Bitcoin in China: A Dream Dispelled." *Economist*. 12 04 2014: n. page. Web. 12 Apr. 2014.
- [2] Bruce, J.D. "Purely P2P Crypto-Currency With Finite Mini-Blockchain."
- [3] Buterin, Vitalik. "A Next-Generation Smart Contract and Decentralized Application Platform." . Github, 20 Apr. 2014. Web.
<<https://github.com/ethereum/wiki/wiki/%5BEnglish%5D-White-Paper>>.
- [4] Buterin, Vitalik. "Dagger" Github, 16 Feb. 2014. Web.
<<https://github.com/ethereum/wiki/wiki/%5BEnglish%5D-Dagger>>.
- [5] Buterin, Vitalik. " DAOs, DACs, DAs and More: An Incomplete Terminology Guide" Ethereum Blog, 6 May. 2014. Web. < <http://blog.ethereum.org>>.
- [6] Buterin, Vitalik. "Ethereum: A Next-Generation Generalized Smart Contract and Decentralized Application Platform." . V. Buterin, 1 Feb. 2014. Web. .
<<http://vbuterin.com/ethereum.html>>.
- [7] Chen, Tim. "Credit Card and Debit Card Transaction Volume Statistics." *NerdWallet*. N.p., 8 June 2010. Web. 24 Apr. 2014.
- [8] CuriousInventor. "How Bitcoin Works Under the Hood." YouTube. July 14, 2013
- [9] Ducklin, Paul . "Android random number flaw implicated in Bitcoin thefts." *Naked Security*. N.p., 12 Aug. 2013. Web. 5 June 2014.<<http://nakedsecurity.sophos.com/2013/08/12/android-random-number-flaw-implicated-in-bitcoin-thefts/>>.
- [10] Dwyer, Ben. "Credit Card Transaction Fees." *CardFellow*. N.p., n.d. Web. 24 Apr. 2014.
- [11] Ethereum. "Introducing Ethereum." Vimeo. February 19, 2014
- [12] Gara, Tom. "October 2015: The End of the Swipe-and-Sign Credit Card." *The Wall Street Journal* 6 Feb. 2014, sec. Business: n. pag. Print.
- [13] Greenberg, Andy. "End Of The Silk Road: FBI Says It's Busted The Web's Biggest Anonymous Drug Black Market." *Forbes*. Forbes Magazine, 2 Oct. 2013. Web. 2 June 2014. <<http://www.forbes.com/sites/andygreenberg/2013/10/02/end-of-the-silk-road-fbi-busts-the-webs-biggest-anonymous-drug-black-market/>>.
- [14] Mathews, Lee. "Square's mobile payment system has been hacked. Twice. | Chips | Geek.com." *Geekcom*. N.p., 5 Aug. 2011. Web. 12 June 2014.

- <<http://www.geek.com/chips/mobile-payment-processor-square-hacked-twice-1410009/>>.
- [15] Osborne, Charlie. "How hackers stole millions of credit card records from Target | ZDNet." *ZDNet*. N.p., 13 Feb. 2014. Web. 12 June 2014.
<<http://www.zdnet.com/how-hackers-stole-millions-of-credit-card-records-from-target-7000026299/>>.
- [16] "Proof of Work." Bitcoin Wiki, 20 Mar. 2014. Web. <https://en.bitcoin.it/wiki/Proof_of_work>.
- [17] "Scalability." Bitcoin Wiki, 20 Mar. 2014. Web.
<<https://en.bitcoin.it/wiki/Scalability>>.
- [18] Sompolinsky, Yonatan, and Aviv Zohar. "Accelerating Bitcoin's Transaction Processing Fast Money Grows on Trees, Not Chains." . N.p., 1 Jan. 2014. Web. .
<<https://eprint.iacr.org/2013/881.pdf>>.
- [19] Stein, William. "Elliptic Curve Digital Signature Algorithm in the SAGEMathCloud." Youtube. Feb. 14, 2104.
- [20] Surda, Peter. *Economics of Bitcoin: is Bitcoin an alternative to fiat currencies and gold?* MS Thesis. WU Vienna University of Economics and Business, Vienna, 2012
- [21] The Bitcoin Foundation. "Frequently Asked Questions." Bitcoin Project 2009-2014. Bitcoin.org
- [22] Touryalai, Halah. "Countries With The Most Card Fraud: U.S. And Mexico." *Forbes*. Forbes Magazine, 22 Oct. 2012. Web. 12 June 2014.
<<http://www.forbes.com/sites/halahtouryalai/2012/10/22/countries-with-the-most-card-fraud-u-s-and-mexico/>>.
- [23] United States. Federal Reserve. *Consumers and Mobile Financial Services*. Washington DC: Board of Governors of the Federal Reserve System, 2012. Web.
- [24] "Why does Kenya lead the world in mobile money?." *The Economist*. The Economist Newspaper, 27 May 2013. Web. 11 June 2014.
<<http://www.economist.com/blogs/economist-explains/2013/05/economist-explains-18>>.
- [25] Woolsey, Ben, and Emily Gerson. "The History of Credit Cards." *Credit Cards*. N.p., 11 May 2009. Web. 10 Apr. 2014.

APPENDICES

ECDSA

```
q=2^256-2^32-2^9-2^8-2^7-2^6-2^4-1; q
```

```
115792089237316195423570985008687907853269984665640564039457584007908834  
671663
```

```
Fq=GF(q)  
E=EllipticCurve(Fq,[0,7])  
E.cardinality().factor()
```

```
115792089237316195423570985008687907852837564279074904382605163141518161  
494337
```

```
#P=E.random_point();P  
P=E\  
([3038837052154010581161995997553971832967030536650494460811447733954780  
4011474 ,  
793147652980049752756300015768870806732579661042384502581669776115932881  
90689])
```

```
factor(P.order())
```

```
115792089237316195423570985008687907852837564279074904382605163141518161  
494337
```

```
G=2*P;p=G.order();p  
Fp=GF(p)
```

```
115792089237316195423570985008687907852837564279074904382605163141518161  
494337
```

G

```
(47225893577912507754096881373894066927272912118471919967200095964523107  
501366 :  
180650317818684422139681188944030582466436361860710970453284426416591437  
59683 : 1)
```

```
def phi(x): a=Fp(x.lift())
```

if a==0: a=Fp(1)
return a

```
#d=Fp.random_element(); d
d=Fp\
(75528383185237032611138024943254965195788052782038119567386761828023843
516149)\
```

```
Q=lift(d)*G
public_key= {'E':E, 'G':G, 'Q':Q, 'p':p}
```

```
public_key
```

```
{'Q':
(14221495674962211512784713580541677716641522232187924643924130386851409
21165 :
772659649007677119182899123862815683603287434180277224628891623209697895
12499 : 1), 'p':
115792089237316195423570985008687907852837564279074904382605163141518161
494337, 'E': Elliptic Curve defined by  $y^2 = x^3 + 7$  over Finite Field of size
115792089237316195423570985008687907853269984665640564039457584007908834
671663, 'G':
(47225893577912507754096881373894066927272912118471919967200095964523107
501366 :
180650317818684422139681188944030582466436361860710970453284426416591437
59683 : 1)}
'b94d27b9934d3e08a52e52d7da7dabfac484efe37a5380ee9088f7ace2efcde9'
```

message="hello world" import hashlib
h=hashlib.sha256(message).hexdigest();h

```
'b94d27b9934d3e08a52e52d7da7dabfac484efe37a5380ee9088f7ace2efcde9'
```

```
z=hash(h)%p;z
```

```
3236743691350926847
```

```
#time to sign
k=Fp.random_element(); k
kG=lift(k)*G; kG
```

943717338011868809635757389462166840786538102065333131430829680266561119
89002
(43874632519964408218965908648316107164621493103607155216642220549694662
336667 :
248636063307748443636240322041614255141489577518963359202471039916112977
16204 : 1)

$r = \text{phi}(kG[0]); s = (z + r*d)/k$
 $\text{sig} = (r, s); \text{sig}$

(43874632519964408218965908648316107164621493103607155216642220549694662
336667,
974801156821070048950562231903681497914306701873821813975645890227202838
44373)

$C = \text{lift}(z/s)*G + \text{lift}(r/s)*Q; C$

(43874632519964408218965908648316107164621493103607155216642220549694662
336667 :
248636063307748443636240322041614255141489577518963359202471039916112977
16204 : 1)

$\text{phi}(C[0])$

438746325199644082189659086483161071646214931036071552166422205496946623
36667

Escrow Contract Python

```
from sim import Block, Contract, Simulation, Tx, mktx, stop

# Constants to modify before contract creation
MERCHANT = "mike"
SHIPPER = "sam"
PRICE_ETHER = 3995
CONFIRMATION_TIMEOUT = 30 * 86400

# Status enumeration
S_START = 0
S_CUSTOMER_PAID = 1
S_SHIPPED = 2
S_REFUNDED = 3
S_RATED = 4

# Contract Storage indexes
I_STATUS = 1000
I_CUSTOMER_ADDRESS = 1001
I_CUSTOMER_PAID_AMOUNT = 1002
I_CUSTOMER_PAID_TS = 1003

MIN_FEE = 1000

class Escrow(Contract):
    """
    Escrow example to demonstrate contract basics.

    Customer pays specified amount. Shipper confirms
    shipping and releases
    funds to Merchant. Otherwise when not confirmed within
    a certain timeframe the
    Customer is refunded.

    Constants:
        MERCHANT - Address of the Merchant
        CUSTOMER - Address of the Customer
        DATA - Data containing the rating
        SHIPPER - Address of the Shipper
        PRICE_ETHER - Price of the order in Ether
        CONFIRMATION_TIMEOUT - Amount of seconds after
    which the customer can be refunded

    Storage:
        1000: Status (I_STATUS)
        0 = start (S_START)
```



```

        1 = customer paid (S_CUSTOMER_PAID)
        2 = shipped (S_SHIPPED)
        3 = customer refunded (S_REFUNDED)
        4 = customer rated (S_RATED)
1001: Customer address (I_CUSTOMER_ADDRESS)
1002: Customer paid amount (I_CUSTOMER_PAID_AMOUNT)
1003: Customer paid timestamp (I_CUSTOMER_PAID_TS)

Tx Triggers:
    Customer:
        sufficient amount => pay
    Shipper:
        mark as shipped
    Anyone:
        trigger expiration
"""

def run(self, tx, contract, block):
    if tx.value < MIN_FEE * block.basefee:
        stop("Insufficient fee")

    state = contract.storage[I_STATUS]
    if state == S_START and tx.value >= PRICE_ETHER:
        contract.storage[I_STATUS] = S_CUSTOMER_PAID
        contract.storage[I_CUSTOMER_ADDRESS] =
tx.sender
        contract.storage[I_CUSTOMER_PAID_AMOUNT] =
tx.value
        contract.storage[I_CUSTOMER_PAID_TS] =
block.timestamp
    elif state == S_CUSTOMER_PAID:
        if tx.sender == SHIPPER:
            contract.storage[I_STATUS] = S_SHIPPED
            mktx(MERCHANT,
contract.storage[I_CUSTOMER_PAID_AMOUNT], 0, 0)
            elif block.timestamp >=
contract.storage[I_CUSTOMER_PAID_TS] +
CONFIRMATION_TIMEOUT:
                contract.storage[I_STATUS] = S_REFUNDED
                mktx(contract.storage[I_CUSTOMER_ADDRESS],
contract.storage[I_CUSTOMER_PAID_AMOUNT], 0, 0)
            elif state == S_SHIPPED
                if tx.sender == CUSTOMER:
                    contract.storage[I_STATUS] = S_RATED
                    mktx(MERCHANT, 0, DATA, 0)
            else:
                stop("Invalid state transition")

```

```

# Constants for test purposes
CUSTOMER = "carol"
TS = 1392000000

class EscrowRun(Simulation):

    def test_insufficient_fee(self):
        contract = Escrow()

        tx = Tx(sender=CUSTOMER, value=10)
        self.run(tx, contract)

        assert self.stopped == 'Insufficient fee'

    def test_customer_paid(self):
        contract = Escrow()

        tx = Tx(sender=CUSTOMER, value=PRICE_ETHER)
        block = Block(timestamp=TS)
        self.run(tx, contract, block)

        assert contract.storage[I_STATUS] ==
S_CUSTOMER_PAID
        assert contract.storage[I_CUSTOMER_ADDRESS] ==
CUSTOMER
        assert contract.storage[I_CUSTOMER_PAID_AMOUNT] ==
PRICE_ETHER
        assert contract.storage[I_CUSTOMER_PAID_TS] == TS

    def test_shipped(self):
        contract = Escrow()

        tx = Tx(sender=CUSTOMER, value=PRICE_ETHER)
        block = Block(timestamp=TS)
        self.run(tx, contract, block)

        tx = Tx(sender=SHIPPER, value=MIN_FEE)
        block = Block(timestamp=TS + 1)
        self.run(tx, contract, block)

        assert contract.storage[I_STATUS] == S_SHIPPED
        assert len(contract.txs) == 1
        assert contract.txs == [(MERCHANT, PRICE_ETHER, 0,
0)]

```

```

def test_confirmation_timeout(self):
    contract = Escrow()

    tx = Tx(sender=CUSTOMER, value=PRICE_ETHER)
    block = Block(timestamp=TS)
    self.run(tx, contract, block)

    tx = Tx(sender=CUSTOMER, value=MIN_FEE)
    block = Block(timestamp=TS + CONFIRMATION_TIMEOUT +
1)
    self.run(tx, contract, block)

    assert contract.storage[I_STATUS] == S_REFUNDED
    assert len(contract.txs) == 1
    assert contract.txs == [(CUSTOMER, PRICE_ETHER, 0,
0)]

def test_rating(self):
    contract = Escrow()

    tx = Tx(sender=CUSTOMER, value=PRICE_ETHER)
    block = Block(timestamp=TS)
    self.run(tx, contract, block)

    tx = Tx(sender=CUSTOMER, value=MIN_FEE)
    block = Block(timestamp=TS + 1)
    self.run(tx, contract, block)

    assert contract.storage[I_STATUS] == S_RATED
    assert len(contract.txs) == 2
    assert contract.txs == [(MERCHANT, 0, DATA, 0)]

```

Contract Stack Operations

- (00) STOP - halts execution
- (01) ADD - pops two items and pushes $S[-2] + S[-1] \bmod 2^{256}$.
- (02) MUL - pops two items and pushes $S[-2] * S[-1] \bmod 2^{256}$
- (03) SUB - pops two items and pushes $S[-2] - S[-1] \bmod 2^{256}$
- (04) DIV - pops two items and pushes $\text{floor}(S[-2] / S[-1])$. If $S[-1] = 0$, halts execution.
- (05) SDIV - pops two items and pushes $\text{floor}(S[-2] / S[-1])$, but treating values above 2^{255} as negative (ie. $x \rightarrow 2^{256} - x$). If $S[-1] = 0$, halts execution.
- (06) MOD - pops two items and pushes $S[-2] \bmod S[-1]$. If $S[-1] = 0$, halts execution.
- (07) SMOD - pops two items and pushes $S[-2] \bmod S[-1]$, but treating values above 2^{255} as negative (ie. $x \rightarrow 2^{256} - x$). If $S[-1] = 0$, halts execution.
- (08) EXP - pops two items and pushes $S[-2] ^ S[-1] \bmod 2^{256}$
- (09) NEG - pops one item and pushes $2^{256} - S[-1]$
- (0a) LT - pops two items and pushes 1 if $S[-2] < S[-1]$ else 0
- (0b) LE - pops two items and pushes 1 if $S[-2] \leq S[-1]$ else 0
- (0c) GT - pops two items and pushes 1 if $S[-2] > S[-1]$ else 0
- (0d) GE - pops two items and pushes 1 if $S[-2] \geq S[-1]$ else 0
- (0e) EQ - pops two items and pushes 1 if $S[-2] == S[-1]$ else 0
- (0f) NOT - pops one item and pushes 1 if $S[-1] = 0$ else 0
- (10) MYADDRESS - pushes the contract's address as a number
- (11) TXSENDER - pushes the transaction sender's address as a number
- (12) TXVALUE - pushes the transaction value
- (13) TXDATAN - pushes the number of data items
- (14) TXDATA - pops one item and pushes data item $S[-1]$, or zero if index out of range
- (15) BLK_PREVHASH - pushes the hash of the previous block (NOT the current one since that's impossible!)
- (16) BLK_COINBASE - pushes the coinbase of the current block
- (17) BLK_TIMESTAMP - pushes the timestamp of the current block
- (18) BLK_NUMBER - pushes the current block number
- (19) BLK_DIFFICULTY - pushes the difficulty of the current block
- (1a) BASEFEE - pushes the base fee (x as defined in the fee section below)
- (20) SHA256 - pops one item and then $\text{ceil}(S[-1] / 32)$ further items. Then takes those items as 32-byte strings, concatenates them in top-to-bottom order, taking out the low-order bytes of the bottommost if necessary, and pushes the SHA256 of the resulting string.
- (21) RIPEMD160 - works just like SHA256 but with the RIPEMD-160 hash
- (22) ECMUL - pops three items. If $(S[-2], S[-1])$ are a valid point in secp256k1 , including both coordinates being less than P , pushes $(S[-2], S[-1]) * S[-3]$, using $(0,0)$ as the point at infinity. Otherwise, pushes $(2^{256} - 1, 2^{256} - 1)$. Note that there are no restrictions on $S[-3]$.

- (23) ECADD - pops four items and pushes $(S[-4], S[-3]) + (S[-2], S[-1])$ if both points are valid, otherwise $(2^{256} - 1, 2^{256} - 1)$.
- (24) ECSIGN - pops two items and pushes (v, r, s) as the Electrum-style RFC6979 deterministic signature of message hash $S[-1]$ with private key $S[-2] \bmod N$.
- (25) ECRECOVER - pops four items and pushes (x, y) as the public key from the signature $(S[-3], S[-2], S[-1])$ of message hash $S[-4]$. If the signature has invalid v, r, s values (ie. v not in $[27, 28]$, r not in $[0, P]$, s not in $[0, N]$), return $(2^{256} - 1, 2^{256} - 1)$.
- (26) ECVALID - pops two items and pushes 1 if $(S[-2], S[-1])$ is a valid secp256k1 point (including $(0, 0)$) else 0
- (27) SHA3 - works just like SHA256 but with the SHA3 hash, 256 bit version.
- (30) PUSH - pushes the item in memory at the index pointer + 1, and advances the index pointer by 2.
- (31) POP - pops one item.
- (32) DUP - pushes $S[-1]$ to the stack.
- (33) DUPN - reads item in memory at the index pointer + 1, say N , and pushes $S[-N]$ to the stack. Advances the index pointer by 2. If $N = 0$, exits with an error instead.
- (34) SWAP - pops two items and pushes $S[-1]$ then $S[-2]$
- (35) SWAPN - reads item in memory at the index pointer + 1, say N , and swaps $S[-1]$ and $S[-N]$ to the stack. Advances the index pointer by 2. If $N = 0$, exits with an error instead.
- (36) LOAD - pops one item and pushes the item in memory at index $S[-1]$
- (37) STORE - pops two items and sets the item in memory at index $S[-1]$ to $S[-2]$
- (40) JMP - pops one item and sets the index pointer to $S[-1]$
- (41) JMPI - pops two items and sets the index pointer to $S[-2]$ only if $S[-1]$ is nonzero
- (42) IND - pushes the index pointer
- (50) EXTRO - pops two items and pushes memory index $S[-2]$ of contract $S[-1]$
- (51) BALANCE - pops one item and pushes balance of that address, or zero if the address is invalid
- (60) MKTX - pops three items and initializes a transaction to send $S[-2]$ ether to $S[-1]$ with $S[-3]$ data items. Pops that many data items and adds them in order popped as data items to the transaction. Then sends the transaction. Note that if $S[-1] = 0$ then this creates a new contract.
- (ff) SUICIDE - pops one item, destroys the contract and clears all memory, sending the entire balance plus the negative fee from clearing memory to the address at $S[-1]$

GLOSSARY

1. private key: A secret number, known only to the person that generated it. A private key is essentially a randomly generated number. In Bitcoin, someone with the private key that corresponds to funds on the public ledger can spend the funds. In Bitcoin, a private key is a single unsigned 256 bit integer (32 bytes).
2. public key: A number that corresponds to a private key, but does not need to be kept secret. A public key can be calculated from a private key, but not vice versa. A public key can be used to determine if a signature is genuine (in other words, produced with the proper key) without requiring the private key to be divulged. In Bitcoin, public keys are either compressed or uncompressed. Compressed public keys are 33 bytes, consisting of a prefix either 0x02 or 0x03, and a 256-bit integer called x . The older uncompressed keys are 65 bytes, consisting of constant prefix (0x04), followed by two 256-bit integers called x and y ($2 * 32$ bytes). The prefix of a compressed key allows for the y value to be derived from the x value.
3. signature: A number that proves that a signing operation took place. A signature is mathematically generated from a hash of something to be signed, plus a private key. The signature itself is two numbers known as r and s . With the public key, a mathematical algorithm can be used on the signature to determine that it was originally produced from the hash and the private key, without needing to know the private key. Signatures are either 73, 72, or 71 bytes long, with probabilities approximately 25%, 50% and 25% respectively, although sizes even smaller than that are possible with exponentially decreasing probability.