NAVAL
POSTGRADUATE
SCHOOL

MONTEREY, CALIFORNIA

THESIS

BLOCKGRID: A BLOCKCHAIN-MEDIATED CYBER-PHYSICAL INSTRUCTIONAL PLATFORM

by

Gregory M. Moore

December 2020

Thesis Advisor:	Cynthia E. Irvine
Second Reader:	Peter R. Ateshian

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** December 2020	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE BLOCKGRID: A BLOCKCHAIN-MEDIATED CYBER-PHYSICAL INSTRUCTIONAL PLATFORM	**5. FUNDING NUMBERS**
6. AUTHOR(S) Gregory M. Moore	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.	**12b. DISTRIBUTION CODE** A

13. ABSTRACT (maximum 200 words)

Blockchain technology has garnered significant attention for its disruptive potential in several domains of national security interest. For the United States government to meet the challenge of incorporating blockchain technology into its IT infrastructure and cyber warfare strategy, personnel must be educated about blockchain technology and its applications.

This thesis presents both the design and prototype implementation for a blockchain-mediated cyber-physical system called a BlockGrid. The system consists of a cluster of microcomputers that form a simple smart grid controlled by smart contracts on a private blockchain. The microcomputers act as private blockchain nodes and are programmed to activate microcomputer-attached circuits in response to smart-contract transactions. LEDs are used as visible circuit elements that serve as indicators of the blockchain's activity and allow demonstration of the technology to observers. Innovations in networking configuration and physical layout allow the prototype to be highly portable and pre-configured for use upon assembly. Implementation options allow the use of BlockGrid in a variety of instructional settings, thus increasing its potential benefit to educators.

14. SUBJECT TERMS blockchain, education, smartgrid, BlockGrid, blockchain education, blockchain demonstration, blockchain demo, Ethereum, smart contract, cyberphysical system, cyber-physical system, internet of things, IOT	**15. NUMBER OF PAGES** 225
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

BLOCKGRID: A BLOCKCHAIN-MEDIATED CYBER-PHYSICAL INSTRUCTIONAL PLATFORM

Gregory M. Moore
Civilian, CyberCorps: Scholarship for Service
BA, University of California, San Diego, 2005

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 2020

Approved by: Cynthia E. Irvine
Advisor

Peter R. Ateshian
Second Reader

Gurminder Singh
Chair, Department of Computer Science

ABSTRACT

Blockchain technology has garnered significant attention for its disruptive potential in several domains of national security interest. For the United States government to meet the challenge of incorporating blockchain technology into its IT infrastructure and cyber warfare strategy, personnel must be educated about blockchain technology and its applications.

This thesis presents both the design and prototype implementation for a blockchain-mediated cyber-physical system called a BlockGrid. The system consists of a cluster of microcomputers that form a simple smart grid controlled by smart contracts on a private blockchain. The microcomputers act as private blockchain nodes and are programmed to activate microcomputer-attached circuits in response to smart-contract transactions. LEDs are used as visible circuit elements that serve as indicators of the blockchain's activity and allow demonstration of the technology to observers. Innovations in networking configuration and physical layout allow the prototype to be highly portable and pre-configured for use upon assembly. Implementation options allow the use of BlockGrid in a variety of instructional settings, thus increasing its potential benefit to educators.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

API	Application Programming Interface
ARM	Advanced RISC (Reduced Instruction Set Computing) Machine
CLI	Command Line Interface
CONOPS	Concept of Operations
COTS	Commercial off-the-shelf
CVE	Common Vulnerabilities and Exposures
DAG	Directed Acyclic Graph
DApp	Decentralized Application
DARPA	Department of Advanced Research Projects Agency
DB	Database
DoS	Denial of Service
DDoS	Distributed Denial of Service
DDR	Double Data Rate
DLT	Distributed Ledger Technology
DNS	Domain Name System
DOD	Department of Defense
DPOS	Delegated Proof of Stake
EVM	Ethereum Virtual Machine
GB	Gigabyte (10^9 bytes)
GPIO	General Purpose Input/Output
GPU	Graphical Processing Unit
GUI	Graphical User Interface
HTML	Hypertext Markup Language
ICS	Industrial Control Systems
IDE	Integrated Development Environment
IoT	Internet of Things
IP	Internet Protocol
IT	Information Technology
JRE	Java Runtime Environment
KB	Kilobyte (10^3 bytes)

LAN	Local Area Network
LED	Light-Emitting Diode
MITM	Man-in-the-Middle
MB	Megabyte (10^6 bytes)
NPM	Node Package Manager
NPS	Naval Postgraduate School
OS	Operating System
PBFT	Practical Byzantine Fault Tolerance
RAM	Random Access Memory
RPI	Raspberry Pi
RPOW	Reusable Proofs of Work
RIMPAC	Rim of the Pacific, a U.S. and Pacific allies annual naval exercise
RLP	Recursive Length Prefix
SCU	System Control Unit
SDK	Software Development Kit
USG	United States Government
WAP	Wireless Access Point
WLAN	Wireless Local Area Network
VM	Virtual Machine

ACKNOWLEDGMENTS

First and foremost, I would like to thank my thesis advisor, Dr. Cynthia Irvine, for her guidance, encouragement, and patience with me as I brought this project to a point of completion, and for giving me the opportunity to showcase the prototype during Discover NPS Day 2019. Additionally, without Dr. Irvine's leadership of the SFS program, none of my experience at NPS would have been possible. For that I am deeply grateful.

I'm also grateful to my second reader, Peter Ateshian, who conceived the original concept of the project, coordinated use of the prototype as a research tool for other students, and showcased the prototype at RIMPAC 2018. Major thanks are also due to Vincent Lopez, a 2018 CS Department intern and all-around blockchain wiz, who was incredibly helpful in troubleshooting numerous technical challenges as they arose during the initial deployment of the blockchain and smart contract on the prototype.

To the many officers I had the pleasure of studying alongside at NPS, thank you for your enthusiasm, camaraderie, and for being so generous with your expertise. To my SFS cohort, I could not have asked for better fellow travelers along this journey. Thank you to all my instructors and professors for their encouragement, patience, and inspiration. Additionally, a special thanks to Mark Gondree, Megan Rock, and Cecelia Davis for all the administrative work they did to make the SFS program a success.

Finally, I'd like to thank my family for their support. Thanks to my father, Captain George M. Moore, and my mother, Stella J. Kennedy, who encouraged me to apply for the SFS program and supported me all the way through. Thanks to my sister, Sarah, for always keeping things upbeat. And thanks to my late grandfather, Commander Robert A. Moore, who earned a BS in Meteorology from NPS in 1953 and later returned to campus to teach, making NPS a special place for my family for the last 70 years.

This material is based upon activities supported by the National Science Foundation under Agreement No 1565443. Any opinions, findings, and conclusions or recommendations expressed are those of the author and do not necessarily reflect the views of the National Science Foundation.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

Blockchains are an emerging approach to decentralized trustless database implementation. A key feature driving their adoption is that they allow databases to support applications without requiring a trusted centralized authority to maintain them. This distributed quality of blockchains allows records to be updated simultaneously throughout a network, eliminating the need for intermediaries as well as lowering tracking and processing times for some applications [1]–[4]. Blockchain technology has been highlighted for its disruptive potential in multiple domains of national security interest including finance, trade, voting, health data management, and Internet of Things (IoT) network computing [5]–[11].

A. BLOCKCHAIN TECHNOLOGY USE CASES FOR THE UNITED STATES GOVERNMENT

Within the national security sphere, blockchain technology has been envisioned for several use cases. One suggested application is to utilize blockchains to track inventory for military supply chain management and logistics [12], [13]. Another proposed concept is to utilize blockchains for Keyless Signature Infrastructure (KSI) to secure communications for critical weapons systems [14]. Blockchain technology has also been proposed for use within IoT networks, which are themselves a subject of active Department of Defense (DOD) research for applications such as deployed sensors [15]–[17].

Across the United States Government (USG), agencies are currently using blockchain technology in a variety of ways. At the level of research, the Department of Homeland Security (DHS) Science and Technology Directorate and the Defense Advanced Research Projects Agency (DARPA) have sponsored multiple blockchain technology research grants for topics like forgery prevention, tracking of cross-border imports, and secure messaging [18]–[21]. The Food and Drug Administration (FDA), the Centers for Disease Control and Prevention (CDC), and Department of the Army also have blockchain proof-of-concept research projects underway [22].

Several agencies have blockchain projects in the pilot stage of development. The DOD, for instance, is piloting blockchain technology in its 3D Print File Security program [23]. DARPA uses pilot blockchain technology as part of its Supply Chain Hardware Integrity of Electronics Defense (SHIELD) initiative [24]. Additionally, the U.S. Treasury has piloted blockchain technology to track government-issued physical assets such as laptops and mobile phones [25].

Two agencies are currently using blockchain technology as an integrated part of their production technology stack. The General Services Administration's (GSA) FASt Lane program employs a prototype blockchain to track acquisition contracts [26]. The Department of Health and Human Services (HHS) uses blockchain technology to optimize grant and contract processing efficiency within both its GrantSolutions and HHS Accelerate programs [27], [28].

B. NATIONAL SECURITY IMPLICATIONS OF BLOCKCHAIN TECHNOLOGY ADOPTION

As the adoption of blockchain technology has increased, so too has the USG's interest in the national security implications of both domestic and adversarial deployment of the technology. For example, in the 2018 National Defense Authorization Act, the House Armed Services Committee called for a blockchain technology briefing from the Secretary of Defense. Requested in this briefing was, "a description of potential offensive and defensive cyber applications of blockchain technology," "an assessment of efforts by foreign powers, extremist organizations, and criminal networks to utilize such technologies," and "an assessment of the use or planned use of such technologies by the Federal Government and critical infrastructure networks [29]." While the contents of this briefing were not published, these congressional briefing subject requests underscore the perceived need for government personnel to understand blockchain technology and the consequences of its use for the United States.

1. Cybersecurity Implications of Domestic Blockchain Technology Adoption

Domestic adoption of blockchain technology has been noted for both positive and negative cybersecurity implications. On the positive side, the 2019–2023 DOD Information Resource Management Strategic Plan's DOD Modernization Strategy report noted that as a result of their fault tolerance, "blockchain networks not only reduce the probability of compromise, but also impose significantly greater costs on an adversary to achieve it." [30] Authors from Pacific Northwest National Laboratory (PNNL) have proposed that blockchain technology may enable improved automated patching and auditing of critical infrastructure [31]. Additionally, the application of blockchains to supply chain management has been identified as a way to reduce supply chain risk in an environment where manufacturing supply chains cross national boundaries and include fabrication and assembly facilities within adversarial nation states [32].

Domestic adoption of blockchain technology may also pose some cybersecurity risks. One area of risk is the potential influence of adversarial nation states on domestically deployed global blockchain networks for which adversarial nation states possess a disproportionate share of network computing power. For example, presently over 50% of the mining nodes for both Bitcoin and Ethereum are located within China [33], [34]. Disproportionate control of a blockchain network's computing power can enable attacks such as deanonymizing and censuring users as well as selectively dropping transactions. It can also allow a blockchain's performance to be undermined through destabilization of network consensus [35].

Another potential risk associated with domestic blockchain technology adoption is the increased challenge of securing private keys [36]. In systems secured by conventional public key cryptography, a trusted third party often generates and maintains private keys for users. However, with blockchain technologies the burden of securing private keys tends to fall to the end user. This creates a vulnerable point of failure which when compromised can lead to severe consequences for data integrity and availability [37]. Additionally, with blockchain technologies the compromise of an end user private key often not only affects the individual user but also users with downstream dependencies on the compromised account [38].

Vulnerabilities in the code of public blockchain technologies are yet another area of concern associated with blockchain adoption. While the integrity of a blockchain's data is secured through cryptography and network consensus algorithms, blockchain software vulnerabilities can enable exploitation of end user accounts and the computing devices that comprise a blockchain's network [39], [40]. Unlike conventional technologies that can be placed behind network security infrastructure to defend proprietary data against vulnerability exploitation attempts, both the blockchain and the data stored in its ledger are often beyond the scope of an individual user or organization to secure due to the blockchain's distributed nature [41]. In the case of smart contracts, source code is written to the blockchain and can be easily inspected, making vulnerability discovery easier for exploit developers [42]. Additionally, as access to many blockchain technologies is not purchased, patching incentives for developers of freely accessed blockchain technology may not be as security-aligned as they are for developers of licensed software [43].

All of these cybersecurity risks not only have the potential to directly impact blockchain users, but may also have downstream risk effects through multi-party dependencies. For example, if a third-party service provider relies on blockchain technology and experiences an attack, users of its service could be affected as well. Downstream dependencies on blockchain technology for the USG and domestic users can be expected to grow as telecom, network, and cloud services providers move to adopt blockchain technology and offer more blockchain-based products [44]–[47].

2. Cybersecurity Implications of Foreign Threat Actor Blockchain Technology Adoption

The adoption of blockchain technology by foreign threat actors has implications for national security as well. As an emerging technology front, blockchain technology offers adversaries an opportunity to compete with the United States for technological supremacy from a more even starting position. A 2018 whitepaper coauthored by representatives of the DOD, DHS, Director of National Intelligence (DNI), Federal Bureau of Investigation

(FBI), and industry partners titled “Blockchain and its Suitability for Government Applications” noted that Russia and China are investing heavily in blockchain technology [48]. In addition to providing greater opportunities to potentially influence and compromise blockchain technologies, these investments may position adversarial nation states to use blockchain technology to more effectively avoid sanctions and dependency on the U.S. financial system [49]. Additionally, the decentralized nature of blockchains may enable smaller adversaries like terrorist organizations to access and transfer financial assets in ways that have not been previously possible with state-controlled currencies [50], [51].

C. THE NEED FOR BLOCKCHAIN TECHNOLOGY EDUCATION

While popular enthusiasm for blockchain technology is growing, it is not yet a mature technology and the merits of its application to many use cases is not yet proven. Under such conditions, there is an urgent need for relevant USG personnel to be educated about blockchain technology so that it can be appropriately incorporated into IT infrastructure and cyberwarfare strategy. With a proper approach to blockchain technology education, the DOD can take full advantage of blockchain-based tools as they become available, anticipate and account for the challenges posed by adversary adoption of blockchain technology, and avoid costly acquisition mistakes that could result from applying blockchain technology to inappropriate use cases.

D. THESIS PROBLEM STATEMENT

Despite the urgent need for blockchain technology to be incorporated into curricula for USG personnel, there is presently no convenient way to demonstrate the salient features of blockchain technology in an educational environment for beginning students. The purpose of this thesis, then, is to develop a cyber-physical system that enables concrete demonstration of blockchain technology in the classroom.

The intended audience for this cyber-physical system includes both non-technical and technical students. Demonstrations can be used with a non-technical audience to develop awareness and appreciation of blockchain technology. For more technical groups, demonstrations can be used as a gateway to deeper technical discussions and the cyber-physical system can be used to support hands-on exploration of the technology.

E. THESIS OBJECTIVES

The principal objectives of this thesis are to (1) determine how a cyber-physical system could be constructed to enable demonstration of blockchain technology, (2) to construct a blockchain-controlled cyber-physical system as a demonstration platform prototype, (3) to generate the guidelines necessary for educators to assemble and use the blockchain technology demonstration platform.

This thesis focuses on the creation of a cyber-physical blockchain demonstration prototype as well as supporting materials that enable users to reproduce and operate the system. The thesis introduces a model for a suitable cyber-physical demonstration platform, termed a "BlockGrid." A BlockGrid is defined as a blockchain-controlled cyber-physical system in which smart contracts on the blockchain control the behavior of a distributed set of isolated or cooperating circuits.

The thesis also describes a BlockGrid prototype that was designed to optimize cost, portability, and ease-of-use. The prototype, which was inspired by the Chainskills tutorial series from Eloudrhiri, uses an internally networked cluster of Raspberry Pi microcomputers to run a private Ethereum blockchain [52]. The prototype uses a smart contract, SmartSwitch, to control light-emitting diodes (LEDs) attached to the Raspberry Pis in order to provide a visual demonstration of blockchain technology for observers. The prototype is available within the NPS Computer Science Department for demonstration, curriculum development, and research.

F. ORGANIZATION OF THESIS

This thesis is organized into six chapters. Beyond the present introduction chapter, Chapter II provides background that explains the foundations of blockchain technology required to understand a BlockGrid's operation. Chapter III presents the concept of operations (CONOPS), which describes the general demonstration platform requirements for a BlockGrid and the envisioned operation environment for the platform. Chapter IV details the available options for BlockGrid system design and configuration as well as the rationale for the choices that were made in the BlockGrid prototype's implementation.

Chapter V explains the prototype’s design, function, and operation. Chapter VI covers the conclusions of the project and explores possible topics for future work.

This work also provides three appendices and supplemental materials. Appendix A lists all of the hardware and software technical specifications for the prototype, including code for the smart contract and script that run on the BlockGrid prototype. Appendix B provides links to the Eloudrhiri Chainskills tutorials and explains how they were adapted to create the prototype, creating a prototype reproduction guide. Appendix C provides a walk-through of how to set up and operate the prototype. The supplemental materials include mp4 file video walkthroughs of the BlockGrid prototype’s setup and operation that were prepared by the author. The supplemental materials also include PDF copies of the installation tutorials used to deploy the private blockchain and smart contract on the prototype’s hardware, and a cited conceptual diagram of the Ethereum blockchain’s structure that was too large to include as a figure. This thesis and the supplemental materials will be available upon publication through the Naval Postgraduate School’s “Calhoun” digital research and institutional materials repository [53].

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND

This chapter provides a review of blockchain technology. The chapter begins by defining a blockchain and by providing a history of how blockchains were pioneered. It goes on to detail some of the key properties of blockchains that are central to their operation. A description of the salient and unique features of the Ethereum blockchain protocol that runs on the BlockGrid prototype is also provided. The chapter concludes with a review of several common cyberattacks against blockchains that may be relevant to a security-oriented blockchain curriculum making use of a BlockGrid as a demonstration, laboratory, or research platform.

A. BLOCKCHAIN DEFINITION

A blockchain can be understood as a decentralized peer-to-peer ledger data structure that is distributed across a network of nodes. Within a conventional blockchain network, each node maintains an identical copy of the distributed ledger. The ledger is cryptographically secured and can only be appended, making it an immutable and chronological record. The blockchain properties of decentralization and immutability are principal factors driving the adoption of blockchain technology. These properties allow blockchain-based applications to require neither a centralized governing authority nor trust between system users.

B. BLOCKCHAIN HISTORY

To understand the features of blockchain technology and its most high-profile implementations, it is helpful to understand how blockchains emerged as a technology. Blockchain technology was brought to popular attention with a 2008 paper written under the pseudonym Satoshi Nakamoto which described the Bitcoin cryptocurrency protocol [54]. In the decade since, numerous blockchain protocols have been authored and today nearly 7,000 blockchain-based cryptocurrencies are actively traded in addition to an ever-growing number of non-currency blockchain applications [55]. However, approaches to decentralized and trustless control of computer systems by both computer scientists and cryptographers have a rich history that predates Bitcoin’s invention by decades, and which created the foundation for blockchain technology as it is implemented today.

A foundational theoretical description of how to approach decentralized trust in computer systems was given by Lamport, Shostak, and Pease in their 1982 paper describing the Byzantine General’s Problem and including several solutions [56]. In the Byzantine General’s Problem a scenario is presented in which multiple generals on different sides of a city must achieve consensus on how to coordinate an attack while accounting for the possibility that their communications or messengers have been compromised. The paper analogizes this scenario to a computer systems network in which accurate consensus must be achieved in order to correctly execute a task. Lamport, Shostak, and Pease showed in their paper as well as subsequent work that given a sufficient number of trusted generals, achieving consensus is possible given known bounds on the number of compromised messages or messengers [57]–[59]. One notable drawback of the solutions presented to the Byzantine General’s problem, however, was that they all required at least some number of the Generals to be trusted and could therefore not be extrapolated to trustless applications.

In the same year, cryptographer David Chaum described a blockchain-like system in his doctoral dissertation “Computer systems established, maintained and trusted by mutually suspicious groups [60].” Chaum’s system made use of cryptographically secured communication records and distributed public ledgers to facilitate trust between untrusting parties within a network. The approach presaged several recognizable features of blockchains as we know them today, but it also relied upon both the physical security of the network nodes and the trustworthiness of communications to achieve consensus [61].

Without a viable approach to trustless distributed computing, early applications of distributed computing were predominantly focused on environments in which a high degree of trust could be assumed amongst network nodes such as isolated industrial systems. Within the civilian sector, for example, distributed computing systems were explored for several applications in factory settings [62], [63]. In the military and aerospace domains, distributed computing approaches were adopted for applications within isolated systems like submarines and aircraft for purposes such as weapons system control and avionics [64]–[66]. In both these early industrial and military technology implementations, distributed computing was primarily integrated to achieve improvements in fault tolerance.

Parallel to these improvements and pioneering applications of distributed computing in the 1980s, a community of cryptography enthusiasts began to emerge that would later come to be known as cypherpunks in the following decade [67]. This community was ideologically rooted in opposition to classified sequestration of cryptographic technologies for military and intelligence purposes as well as opposition to cryptography patent restrictions and export controls [68], [69]. Seminal cypherpunk publications such as Chaum's "Security without Identification: Transaction Systems to Make Big Brother Obsolete" and Eric Hughes' "A Cypherpunk's Manifesto" articulated a cypherpunk philosophy that would lead the community to advocate for globally decentralized electronic approaches to currency [70], [71].

Throughout the 1990s cypherpunks explored and collaborated on approaches to decentralized electronic currencies. One such example came from the computer engineer and cypherpunk Wei Dai in 1998 with his paper "B-Money" that described many fundamental properties of a cryptocurrency but failed to articulate a viable consensus mechanism for real world implementation [72]. Indeed, the chief obstacle to creating a functional electronic currency during this period was the challenge of creating a consensus mechanism capable of preventing double-spending by bad-faith actors in the system [73].

The answer to the consensus dilemma would ultimately come from efforts by cryptographers and computer scientists to combat email spam and denial-of-service (DoS) attacks. In 1992, Dwork and Naor proposed combatting email spam and DoS attacks by making system access conditional upon providing proof that a computationally expensive math problem had been successfully solved [74]. This approach, which was later described as Proof of Work by Jakobsson and Juels in their seminal 1999 paper "Proofs of Work and Bread Pudding Protocols," uses the computational expense of achieving access to disincentivize participation by bad actors [75]. The Proof of Work approach laid out by Dwork and Naor was successfully implemented as an anti-spam and DoS consensus mechanism by prominent cypherpunk Adam Back in his Hashcash protocol in 1997 [76]. Hashcash derives its name from the concept of using proof of successfully calculating a hash collision as the ‘cash’ required to gain system access. In 2004 cypherpunk Hal Finney created a “Reusable Proofs of Work (RPOW)” software protocol that tokenized Hashcash and allowed it to be transferred as an asset between system users [77]. RPOW was later incorporated into a Proof of Work based cryptocurrency concept called “Bit Gold” that was outlined by cypherpunk Nick Szabo in 2005 [78].

Several data structure innovations were also important in creating the foundations for blockchain technology. During the 1990s Haber and Stornetta pioneered mechanisms for linked timestamping of digital resources with several papers and patents [79]–[83]. Their approaches initially utilized linear hash chains to associate resources in chronological order and create a secure authenticated data structure that functioned as an unalterable digital timestamp ledger.

A key innovation that allowed these timestamp data structures to be verified more efficiently and to be smaller was the introduction of Merkle trees into the linked timestamp data structures [84]. Merkle trees, which were introduced by Ralph Merkle in his 1979 doctoral dissertation and later patented in 1982, are a hash tree structure in which every terminal leaf contains a label that is the hash of a data block, and every parent leaf contains a hash of all of its child leaf labels [85], [86]. Merkle trees allow for rapid verification of which tree branch contains a given data block. A conceptual diagram of a Merkle tree is given in Figure 1.

Incorporation of Merkle trees for timestamping rather than relying solely on linear hash chains allowed for considerably faster traversing of the linked timestamp data structure to verify the presence of a given timestamp. It also reduced the amount of memory necessary to store timestamp ledgers. The Merkle tree data structure would later become central to the digital transaction ledger data structures that underpin blockchain technology.

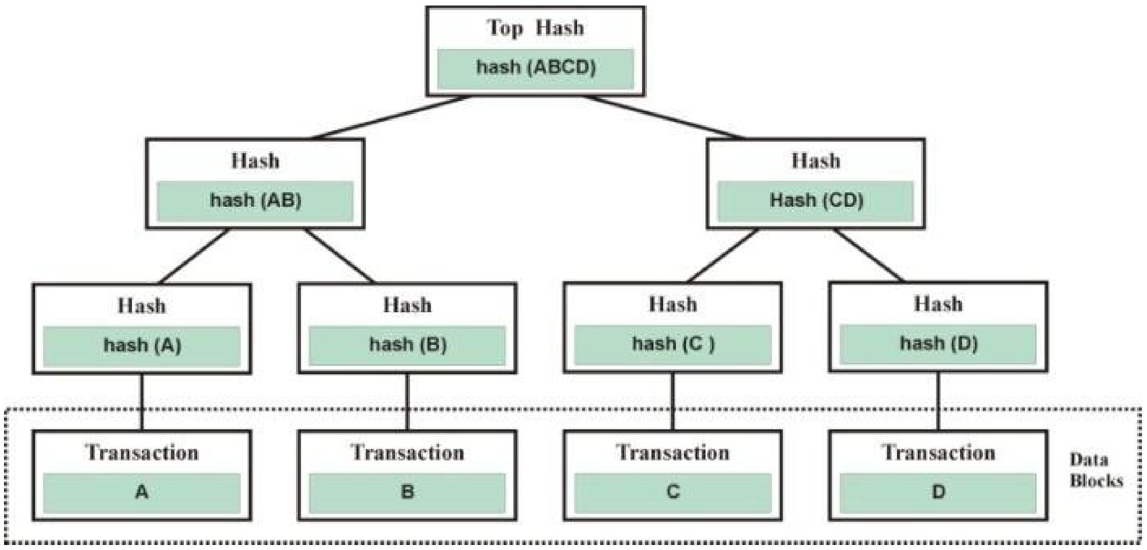


Figure 1. A Conceptual Diagram of a Merkle Tree. Source: [87].

1. Bitcoin

In 2008 these previously parallel and occasionally overlapping trajectories of cryptography innovation were combined by Satoshi Nakamoto into the Bitcoin cryptocurrency protocol. The features of the protocol were laid out in Nakamoto's whitepaper [88]. Within the protocol the currency unit, or coin, is defined as a chain of digital signatures. Public Key Cryptography and the SHA-256 hashing algorithm are used to cryptographically secure transactions between parties.

In order to prevent double-spending, transactions are timestamped in Bitcoin's protocol using a linked timestamping distributed ledger approach inspired by Haber and Stornetta. Consensus on additions of timestamps to the distributed ledger is achieved using Hashcash-like protocol that incorporates a Proof of Work in which network participants find the correct nonce needed to match a provided SHA-256 hash. The transaction histories are stored in a Merkle tree data structure in each block's header. This approach effectively uses economic resources to limit the number of user identities and prevent both takeover of the system and double spending activity. The implementation of the protocol also supports a non-Turing-complete scripting language called Script [89].

Although Bitcoin was architected to support a digital currency, soon after its release a number projects began exploring use of Bitcoin's blockchain as a basis for other digital applications. For example, Namecoin, a Bitcoin-based distributed Domain Name System (DNS), was launched in 2010 [90]. Efforts like Namecoin to utilize Bitcoin as the basis for new protocols centered on specific asset types were followed by attempts to use Bitcoin as the basis for protocols that supported multiple asset types.

One such example was the MasterCoin project that launched in 2013. MasterCoin sought to create a Swiss Army Knife protocol that offered support for multiple different digital asset transactions, each with their own unique asset specifications [91]. While offering a much broader support for digital asset transactions than previous protocols, this approach had the drawback of not being generalizable and not allowing users to define their own assets [92]. Additionally, scaling difficulties were a persistent issue encountered by developers of Bitcoin-based blockchain applications [93]. For example, running applications on top of the Bitcoin blockchain can require computationally expensive back-tracing of Merkle Trees to determine transaction histories [94].

2. Ethereum

To escape the challenges of using the Bitcoin blockchain as a foundational layer for higher layer software, Vitalik Buterin created the Ethereum protocol in 2014 and launched it publicly in 2015 [95]. Ethereum supports Turing-complete scripting languages that generalize the possible interactions between digital assets defined by the user [96]. The code written to define assets and to govern the interaction between them is called a *smart contract*. These can be financial contracts but can more broadly include any blockchain-based interaction between assets that is mediated by code [97]. Ethereum's support for smart contracts has made possible the application of blockchain technology to the broad range of applications being explored today. The BlockGrid prototype runs an Ethereum blockchain, and the features of the Ethereum protocol necessary to understand its function are described in greater detail in Section H.

C. TYPES OF BLOCKCHAINS

There are three fundamental types of blockchain with regard to restricting user access to the blockchain and applying different privileges for users. These three blockchain models are public, private, and permissioned. Some blockchain protocols support all three access models, while others support only one or two options.

1. Public

Public blockchains have no restrictions on user access and assume no trust between network participants. Any user with sufficient resources can participate in the consensus mechanism for adding blocks to the chain and any user can execute transactions [98]. Public blockchains can be accessed from the open internet as well. Perhaps the most recognizable example of a public blockchain is Bitcoin.

2. Private

A private blockchain is only accessible to a defined set of members. A centralized authority is required to maintain the access list [99]. Private blockchains are often adopted for internal applications specific to an organization. The network infrastructure for a private blockchain may therefore be isolated from the public internet, although it need not necessarily be.

The access restriction making a blockchain private can be achieved in several ways. One way is through physical isolation within network hardware infrastructure. Another manner of achieving restricting blockchain access is through logical isolation within virtual network infrastructure, as is common in the cloud. Access restriction can also be implemented in the blockchain protocol’s software. Many blockchain protocols known primarily for their widely used public chains also support creation of private blockchains for development purposes. Ethereum is one such protocol, and the BlockGrid prototype runs a private Ethereum blockchain on its network.

3. Permissioned

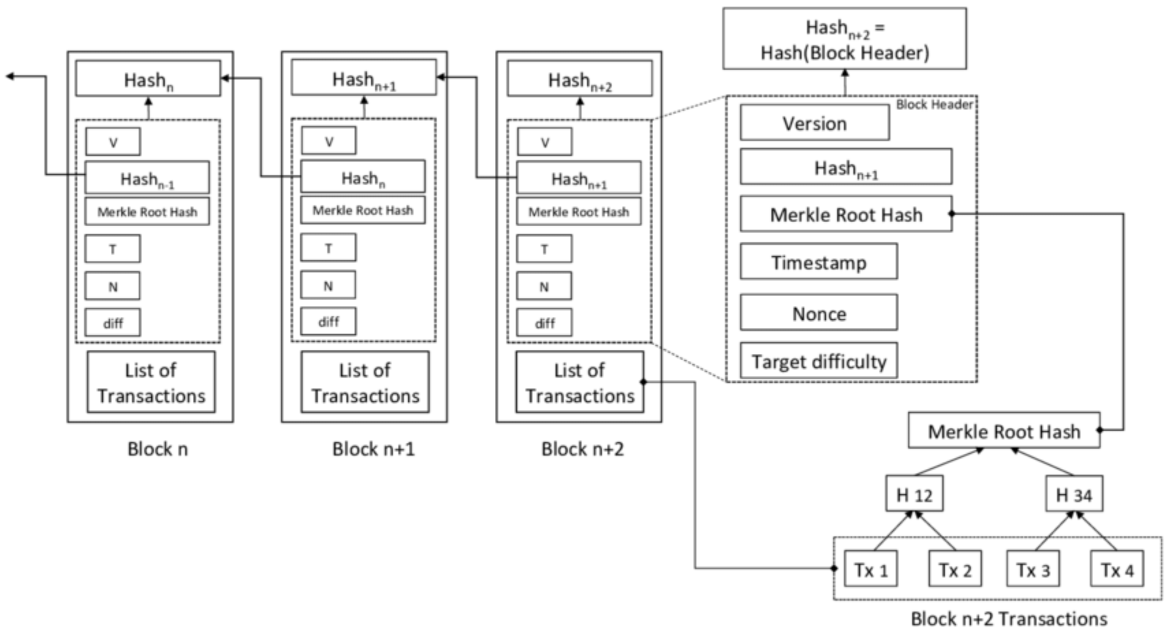
Permissioned blockchains offer an intermediate level of access control. Typically, permissioned blockchains will allow any user to join. However, a permissioned blockchain supports the ability to engineer different user roles and assign them different permission levels [100]. The permission features can be applied to both public and private blockchains. One of the most well-known permissioned blockchain protocols is Hyperledger Fabric, an open source blockchain framework that allows for considerable customization in its blockchain implementations [101].

D. BLOCKCHAIN DATA STRUCTURE BASICS

To appreciate how blockchains operate, an understanding of the basic features of a blockchain's data structure is required. The Bitcoin data structure will be used as an example. Entries to a blockchain's distributed ledger are cryptographically secured and added in cotemporal groupings called blocks. A block data structure typically consists of several header fields and a list of the encrypted ledger entries. Within a blockchain individual blocks are identifiable by a hash of the block's header that is included as a header field. The linking of blocks into a chain, or blockchain, is accomplished by inclusion of another header field that contains the identifying hash of the previous block.

In a conventional blockchain implementation such as Bitcoin, individual transactions within the block are also referenced by a Merkle tree that contains the hashes of all transactions in the block. As explained in Section B earlier in the chapter, a Merkle tree is a version of a tree data structure in which every node stores the hash of the hashes of its children, which enables searching and content verification. The hash of the Merkle tree's root, or root hash, is usually included as a block header field. Block headers typically contain a timestamp field as well.

A conceptual diagram of Bitcoin's blockchain data structure is given in Figure 2. As Figure 2 shows, in addition to the previously mentioned header fields that are common to most blockchains, Bitcoin blocks also have header fields for a nonce and target difficulty value that are used for in Bitcoin's Proof of Work consensus algorithm. The target difficulty is a four-byte value that sets a maximum length of the block's header hash. This in turn leaves a number of leading zeroes in the header field that sets the difficulty of the Proof of Work challenge for the protocol's consensus mechanism. The nonce is an arbitrary number used to make a unique hash value for the Proof of Work challenge. The winner of Bitcoin's Proof of Work challenge determines a nonce that yields a resultant hash less than or equal to the length of the target difficulty value; i.e., a hash with at least as many leading zeros as are present in the target difficulty value header field.



Note: In the Merkle tree portion of the diagram (in the lower right-hand corner), H stands for Hash and Tx stands for transaction. For example, H12 stands for the hash of child node transaction values Tx1 and Tx2. The arrows indicate the direction of dependency, with parent hash values depending on the values of child nodes.

Figure 2. A Conceptual Diagram of the Bitcoin Blockchain Data Structure. Adapted from [102].

E. APPROACHES TO BLOCKCHAIN CONSENSUS

Updates to a blockchain require a consensus algorithm to ensure that all of the blockchain network's nodes maintain identical copies of the distributed ledger. A consensus algorithm allows broadcasted ledger updates to be trusted as valid without the nodes of the blockchain network trusting one another. The three most common consensus algorithms are:

1. Proof of Work

Proof of work was the initial basis for consensus used in blockchains and remains the most widely used type of consensus algorithm today. In a Proof of Work consensus algorithm, users are required to provide a solution to a computationally difficult math problem in order to have the right to commit new blocks to the blockchain [103]. Users that engage in this Proof of Work computation to gain block-committing privileges are called *miners*.

Blockchain protocols that use Proof of Work typically have miners compete against one another for the right to append each new block. The competitions involve having the miners race to solve the same math problem, with the miner that can demonstrate Proof of Work the fastest winning the right to commit a new block. A financial reward in the form of the blockchain protocol's cryptocurrency is typically given to the competition winner to incentivize participation. The newly appended block is then broadcast to the other nodes in the system that verify the addition and add it to their copy of the distributed ledger.

a. Power Consumption Concerns

As Proof of Work-based blockchain protocols like Bitcoin and Ethereum have grown in scale, the electrical power required to support the computation needed for Proof of Work challenges has grown to the point that it is now the subject of environmental concern [104]. Recently, for example, Bitcoin power usage reached the level of small industrialized nation states such as Switzerland [105]. With Ethereum usage growing as well, power consumption challenges are expected to continue to escalate [106]. This issue has inspired research into alternative consensus mechanisms that have a lower computational burden and environmental impact [107].

2. Proof of Stake

Proof of stake is a type of consensus algorithm that has significantly lower computational costs than Proof of Work [108]. In Proof of Stake, proof of access to the cryptocurrency of the system is what confers power in the consensus-making process [109]. In a typical Proof of Stake approach, an account willing to dedicate a portion of its cryptocurrency to acquiring mining capacity will be conferred a proportional opportunity to commit new blocks to the blockchain. The proportion of mining capacity granted usually translates to the frequency level at which the account is enabled to commit blocks to the chain [110]. For example, an account with twice the amount of cryptocurrency allocated for mining will be conferred the opportunity to commit blocks to the chain at double the frequency.

While Proof of Stake is not yet widely adopted, some believe that it could become the predominant consensus mechanism for blockchains in the near future. Ethereum, for example, has announced plans to switch to Proof of Stake as its primary consensus mechanism [111]. In November 2020, Ethereum began allowing its mainchain users to purchase stakes in a new Ethereum Proof of Stake blockchain that is expected to initiate in December 2020 and run parallel to the Proof of Work main chain until a full switchover is made. The date of the full switchover that has yet to be determined [112].

3. Practical Byzantine Fault Tolerance

Practical Byzantine Fault Tolerance (PBFT) is another type of consensus algorithm that avoids the computational costs of Proof of Work. First introduced by Liskov and Castro in 1999, PBFT optimizes for speed with an approach that has four basic phases of execution given a defined upper bound on the fraction of malicious nodes in the system [113]. In most blockchain implementations, this malicious node threshold is set at one third the number of nodes in the blockchain network. Consequently, consensus is considered achieved when there is agreement between greater than two thirds of the system's nodes [114].

In phase 1 of PBFT, a new block is submitted for validation to a node that has been assigned 'primary node' status within the network. In phase 2, the primary node broadcasts the validation request to the rest of the nodes in the system. In phase 3, all nodes perform the validation computation and return their output. In phase 4, the returned outputs are evaluated and a block is accepted as valid once the number of identical returned values is high enough to ensure the malicious node threshold has not been surpassed. Since for most PBFT protocols the malicious node threshold is one third of the total number of nodes in the network, the number of identical replies required for validation is two thirds of the total number of nodes plus one [115].

While PBFT requires less computation than Proof of Work, it also has a large amount of communication overhead that can cause scaling issues [116]. Consequently, PBFT is most commonly used for custom blockchain implementations where the number of nodes is small, rather than global public blockchains such as Bitcoin or Ethereum. Examples of blockchain protocols that support PBFT are HydraChain, Hyberledger Fabric, and Corda [117]–[119].

F. ADDITIONAL BLOCKCHAIN FEATURES

1. Smart Contracts

Smart contracts are computer programs that govern interactions between digital assets on a blockchain. Digital assets can be anything described in code and assigned to a user [120]. Digital assets can range from purely digital constructs, like video game scores, to records of ownership for real-world items, like insurance policies and property deeds [121]–[123].

Smart contracts can be written to execute automatically in response to conditions prescribed in the smart contract's code logic. For example, a smart contract could be written to execute a stock purchase every time the amount of currency in a blockchain account rises above a certain threshold. Smart contracts can also be written to have functions that blockchain users call via blockchain transactions to execute. The smart contract deployed on the BlockGrid prototype, which allows a user to increment and decrement token values for accounts, is a simple example of this style of smart contract.

In most implementations, smart contract code resides as its own addressable entry on the blockchain's distributed ledger and executes within a decentralized runtime environment supported by the blockchain protocol [124]. However, each blockchain protocol has its own method of implementing smart contracts and their associated execution environment. Ethereum's approach is introduced in Section G.

2. Decentralized Applications

Decentralized applications, or DApps, are software applications with a web-facing frontend and blockchain backend that leverages smart contracts for computation [125]. DApps allow users to interact with blockchain-based software without having a user identity within the blockchain network. Users merely have to interact with the web application frontend, with no obligatory participation in the blockchain that supports the application on the backend.

While DApps can enable users to interact with blockchain-based digital assets, they can also utilize a blockchain backend for storage of conventional data. In such DApps, a blockchain's distributed ledger replaces the data storage role that databases have in supporting conventional applications [126]. The blockchain backend of DApps also can confer resiliency and latency benefits to the application if deployed on a large public blockchain such as Ethereum [127]. Despite their potential benefits, DApps have not yet been widely adopted and are not presently a serious rival to conventional web application implementations.

3. Forking

Forking is a procedure in which the linear continuity of a blockchain is broken and two versions of the blockchain proceed from their departure point with different transaction histories and operational procedures. There are two types of forks that can be made: soft forks and hard forks. In a soft fork the changes that are made to a new branch of the blockchain are backwards compatible and historical continuity is maintained with the pre-fork chain [128]. In a hard fork, however, a new branch has no backwards compatibility with the chain prior to the forking event. A new branch of a hard fork essentially starts the blockchain anew, with the forked chain's first block serving as a new starting block [129].

To fork a blockchain a majority of the miners must consent to the forking procedure. A decision to fork will typically be made if a majority of the miners believe it is in their best interest to do so. For example, following a large-scale cyber attack in 2016 the Ethereum blockchain issued a hard fork [130]. Forking can also be executed maliciously as a type of cyberattack, but this requires the attacker to gain a majority of the system's mining power first [131].

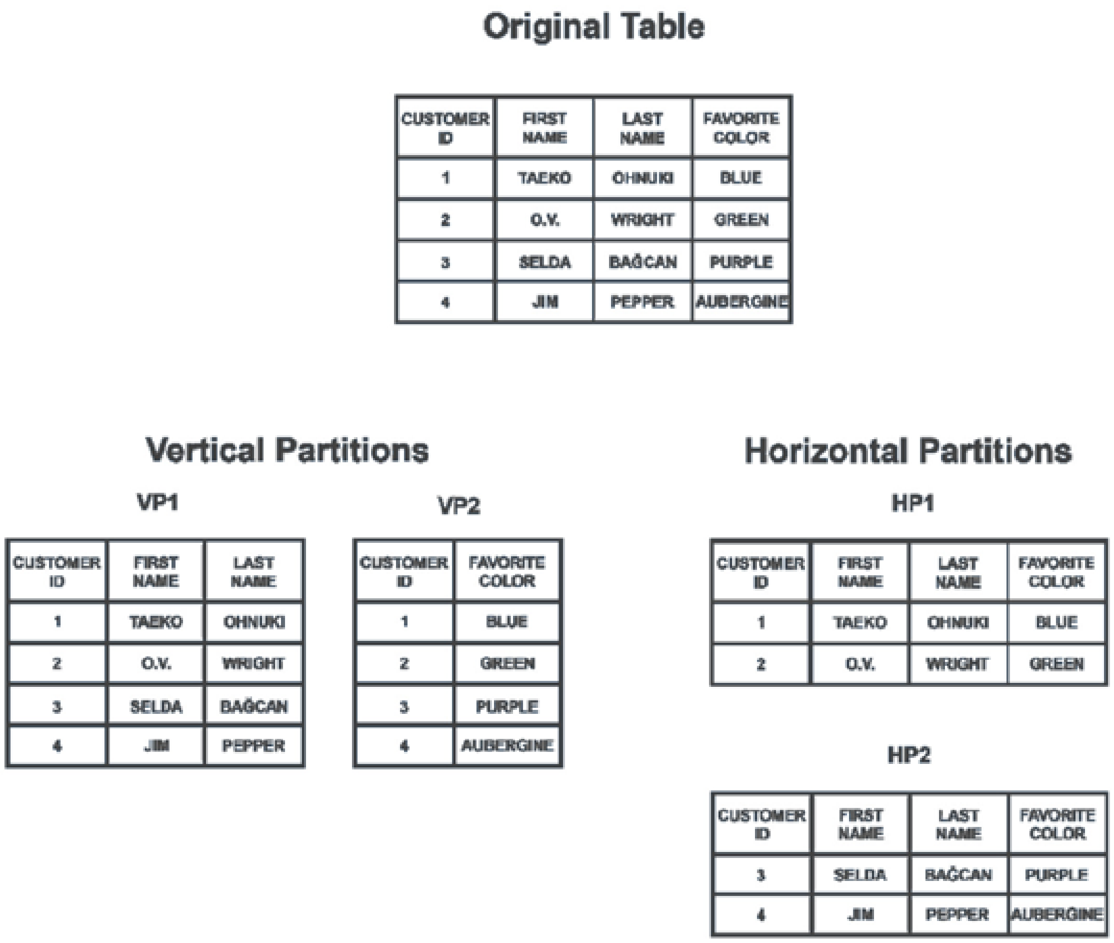
4. Scalability Issues and the Sharding Solution

A persistent issue that threatens to limit the applicability of blockchains to real world use is scalability. A potential solution to the scalability issue is sharding. Sharding is a style of database architecture that takes inspiration from horizontal partitioning, in which either the rows or columns of a database are separated into different tables called partitions or shards [136]. In this approach, each partition contains the same scheme of row or column types, but contains a different and unique set of rows or columns. A conceptual diagram of partitions is given in Figure 3.

The primary consequence of adopting a sharding approach to a blockchain's distributed ledger is that not every node in the blockchain network is required to download, verify, and retain a full copy of the system's ledger. Rather, each node gets a shard of the ledger and is only responsible for downloading and verifying a small subset of transactions [137].

Sharding reduces both the memory usage and the computational load of updating a distributed ledger. This allows a blockchain to continue to operate efficiently at large scales and creates a lower energy consumption footprint for the network. Additionally, with sharding the workload is randomly distributed which supports system resilience in the face of node failures or compromises.

However, sharding requires a much more complicated networking approach to support the consensus process. With a sharded distributed ledger, consensus algorithms require subnetworks and the ability to communicate and quickly switch between them [138]. Where conventional Proof of Work consensus approaches utilize broadcasting across a flat network of nodes to communicate updates everywhere, a sharding network approach requires broadcasting updates only to those subnets maintaining the portion of the ledger being appended.



Original Table

CUSTOMER ID	FIRST NAME	LAST NAME	FAVORITE COLOR
1	TAEKO	OHNUKI	BLUE
2	O.V.	WRIGHT	GREEN
3	SELDA	BAĞCAN	PURPLE
4	JIM	PEPPER	AUBERGINE

Vertical Partitions

VP1

CUSTOMER ID	FIRST NAME	LAST NAME
1	TAEKO	OHNUKI
2	O.V.	WRIGHT
3	SELDA	BAĞCAN
4	JIM	PEPPER

VP2

CUSTOMER ID	FAVORITE COLOR
1	BLUE
2	GREEN
3	PURPLE
4	AUBERGINE

Horizontal Partitions

HP1

CUSTOMER ID	FIRST NAME	LAST NAME	FAVORITE COLOR
1	TAEKO	OHNUKI	BLUE
2	O.V.	WRIGHT	GREEN

HP2

CUSTOMER ID	FIRST NAME	LAST NAME	FAVORITE COLOR
3	SELDA	BAĞCAN	PURPLE
4	JIM	PEPPER	AUBERGINE

Figure 3. Conceptual Diagram of Database Partitions Used in Sharding [139].

5. Directed Acyclic Graph (DAG)

A DAG is a graph structure consisting of vertices connected by directional paths with no possible path combination that would create a cyclical route. DAG’s can be traversed without risk of falling into a perpetual looping behavior. This makes them convenient as a model for processes that require the property of guaranteed halting. Figure 4 shows a conceptual diagram of a DAG.

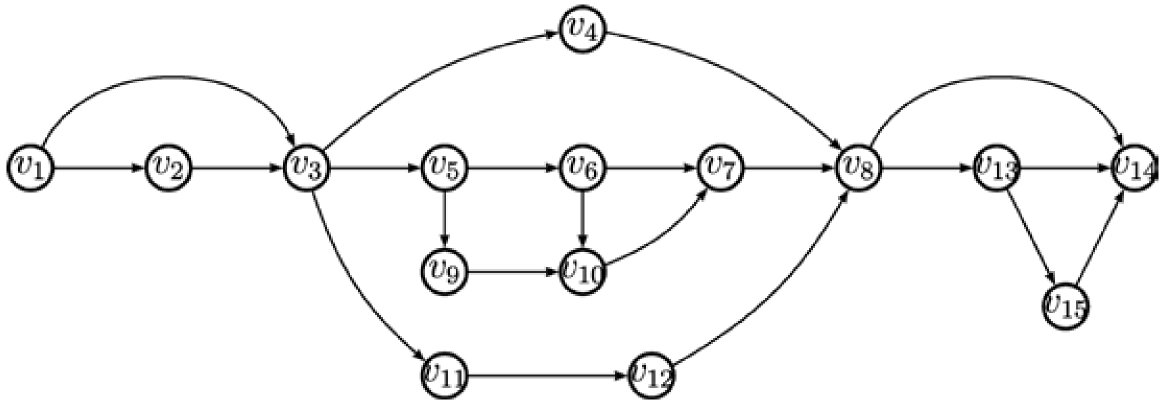


Figure 4. A Conceptual Diagram of a DAG. Source: [140].

DAG's have multiple applications in the context of blockchain technology. One, made use of by Ethereum, is as a fixed resource to facilitate Proof of Work challenges within a blockchain protocol's consensus mechanism. In Ethereum's Ethash consensus mechanism, miners are required to calculate a nonce-dependent subset of a DAG in order to show proof of work [141]. This Proof of Work mechanism was chosen to make Ethash more resistant to the advantages that miners can gain from using specialized mining hardware.

Another increasingly common application of DAGs is as an alternative distributed ledger data structure. In contrast to the conventional linearly linked data structure of blockchains, in a DAG distributed ledger multiple network nodes can be connected to one another [142]. This can allow connected nodes to verify one another, increasing transaction speeds and eliminating the need for Proof of Work and its attendant transaction fees [143]. However, DAG distributed ledgers require implementation of some centralized authority features to achieve security and are therefore more susceptible to some types of cyberattacks [144].

G. ETHEREUM PROTOCOL OVERVIEW

In order to understand how the BlockGrid prototype functions, it is necessary to understand the unique features of the Ethereum protocol. Ethereum is a blockchain protocol centered around smart contracts. In Ethereum, blockchain accounts can hold assets and memory as well as code. An account is considered to be a smart contract when it contains code that specifies the conditions of interaction between assets [145].

In Ethereum, smart contracts adhere to a composability property that allows other applications to reference their outputs without requiring approval or consent. This allows the output of one smart contract to be a term within another smart contract [146]. For example, if one smart contract defines a property title asset, another smart contract could be written that confers an insurance policy asset pending acquisition of a property title defined by the original contract. Decentralized financial applications on Ethereum such as digital currency exchanges make heavy use of this property.

1. Ethereum Consensus

The Ethereum protocol uses the Ethash consensus algorithm, which was initially called Dagger Hashimoto [147]. In addition to the DAG-based Proof of Work consensus mechanism described in Section F, Ethash requires that a tax, called *Gas*, be paid from a user's account into the system whenever any transaction, whether an Ether currency transfer or a smart contract function, is carried out. This feature of the protocol disincentivizes users from running computationally expensive or non-halting programs that overwhelm the computational capacity of the devices in the network [148].

The quantity of Ether currency that must be paid in the Gas tax by a user for a given transaction depends on the amount of computation required to complete the transaction. The Gas tax is then transferred to the mining node accounts as part of the compensation to the miners for performing the Proof of Work required to add new blocks to the chain. When the blockchain is instantiated, the miners are able to select an upper limit for the amount of Gas that can be charged per transaction [149]. Due to the manner in which the Ethereum protocol incentivizes miners, this declared upper Gas limit effectively determines the blockchain's maximum block size [150]. Initiators of blockchain transactions are also able to specify the maximum amount of Gas that can be paid for their transaction to be computed [151]. This means that if the maximum amount to be paid is less than the minimum Gas a miner is willing to accept, the transaction will not go through. In other words, the higher the Gas paid by the transaction initiator, the greater the miner's reward. This in turn means that miners are incentivized to compute and add those transactions for which the transaction initiators have set the highest Gas limits. Transaction initiators that offer a low maximum Gas limit risk having miners refuse to add their transactions to the blockchain.

a. Uncle Blocks

Uncle blocks are created when two blocks are successfully mined and committed to the blockchain's ledger at the same time [132]. This scenario is resolved in Ethereum's protocol by adding the block submitted by the miner that won the harder Proof of Work competition for their block, known as POW share [133]. The losing block then becomes an 'uncle block' that is retained by the next seven blocks of the growing chain [134]. Ethereum confers an Ether currency reward on the miners of uncle blocks along with those that are added to the blockchain in order to incentivize miners to maximize their output and reduce block times for the system [135].

b. Miner Incentives and Downstream Consequences

Another noteworthy feature of Ethash is that the protocol incentivizes miners to constantly compete with one another to add new blocks to chain irrespective of any transaction activity between users. As Ethash rewards winners of the Proof of Work competition with new Ether currency regardless of whether the blocks that winners add to the chain contain transaction data, the system's mining nodes continually race to add to new blocks and grow the blockchain independent of transaction frequency [152]. In practice, this means that empty blocks are continually added approximately every twelve seconds to the system's blockchain between transactions and that the size of the blockchain grows longer as the system continues to run.

The mining of empty blocks and corresponding growth of the blockchain does not have implications for the system's frequency of block mining, or block time. Ethash has a default automatic Proof of Work difficulty-adjustment feature that attempts to modulate algorithm difficulty so that new blocks are added every twelve seconds [153]. Additionally, when the blockchain is initiated, parameters for the blockchain's block-size (the Gas level) and algorithm difficulty can be set during the initial deployment to affect block time as desired [154].

c. State Trie Updates and Downstream Consequences

It should be noted that the increasing size of the blockchain and the unremittent addition of empty blocks do have significant implications for the node synchronization process time, which in turn influences the system-wide effective transaction time. In addition to updating their copy of the ledger's new blocks with each addition, nodes must also perform a "state trie download," which requires each node to update a state trie data structure that cryptographically links all data for the blockchain's nodes [155].

A trie is a form of tree data structure that is optimized for data retrieval in which connections between parent and child nodes form an associative data structure of their contents [156]. In the Ethereum protocol the state trie data structure is a combined form of a Merkle tree and a Patricia trie. A Merkle tree is a hash tree data structure in which each parent node contains a hash of its child nodes' hashes. A Patricia trie (also known as a prefix or radix tree), is a trie structure in which the paths of the trie connect nodes with a common string prefix. An example Patricia tree is depicted in Figure 5.

The Ethereum state trie is referred to as a Merkle Patricia trie because its nodes contain a sequence of child node hashes that are linked in the trie by a common hash prefix [157]. The hash component is reminiscent of a Merkle tree, but the hashes are linked in a Patria trie style rather than each node containing hash of the child node hashes.

In Ethereum the state trie is used to validate that the blockchain's account values have not been tampered with since the last update [158]. The longer the blockchain grows, and every time a blockchain update is required due to the addition of new empty blocks, the more computation and time is required for each node to synchronize with the most up to date copy of the ledger and the state trie. This leads to increases in the time required to complete a transaction on the blockchain, as the results of the new block additions are not visible to the nodes in the system until the node has fully verified the blockchain ledger's updated version as part of consensus mechanism.

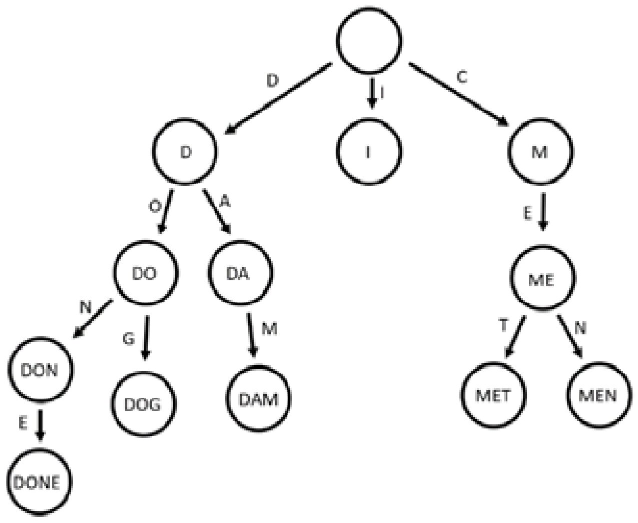


Figure 5. An Example Patricia Trie in Which the Paths through the Trie Connect Nodes with a Common Prefix.

2. The Ethereum Virtual Machine

To understand how smart contracts function within an Ethereum blockchain network, it is essential to understand some features of the Ethereum Virtual Machine (EVM). The EVM is a software-defined runtime environment conceptually similar to Java’s Java Runtime Environment (JRE) that allows nodes to execute the Ethereum bytecode of smart contracts. Both mining nodes and passive nodes process the smart contract code in their EVMs. The EVM allows the node to execute smart contract code independent of the node’s host OS and hardware [159]. A conceptual diagram of the EVM and its functionality is shown in Figure 6. More details regarding the EVM can be found in the book “Mastering Ethereum: Building Smart Contracts and Dapps” by Andreas Antonopoulos and Gavin Wood [160].

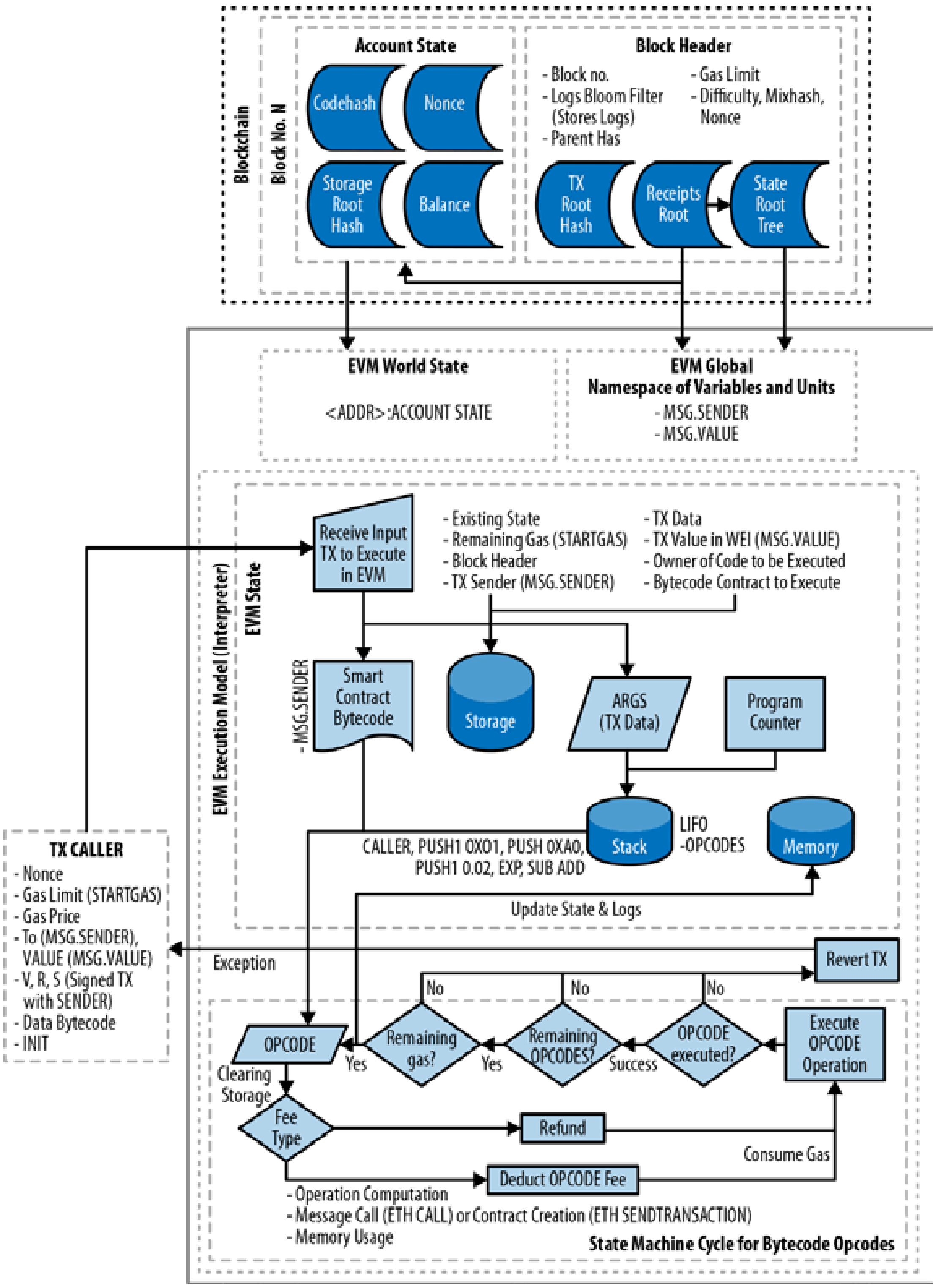
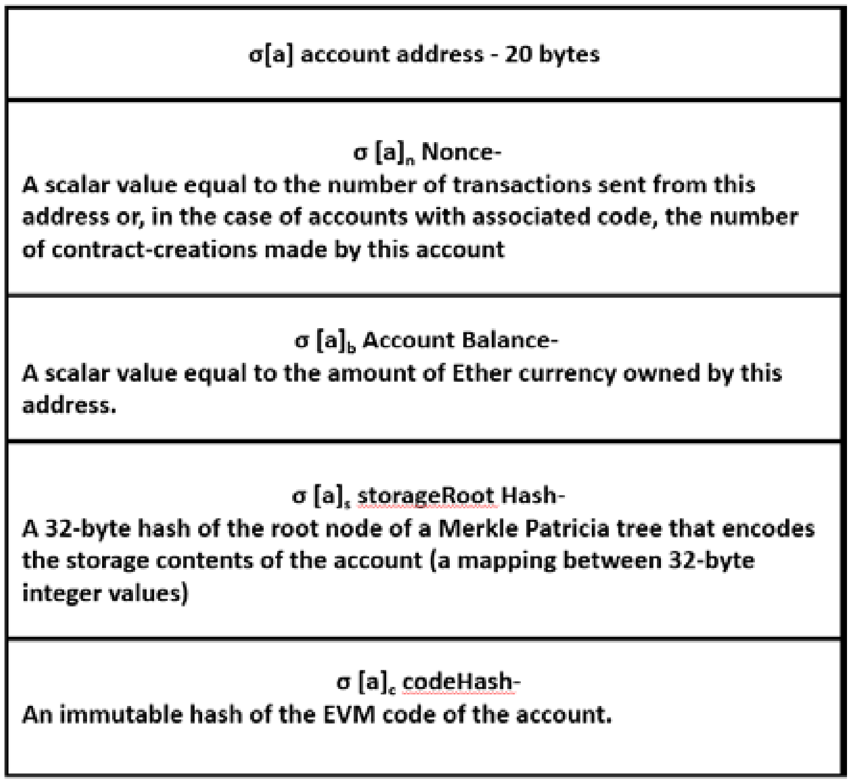


Figure 6. The Ethereum Virtual Machine (EVM) Architecture and Execution Context. Source: [160].

3. Ethereum Node Accounts

Within the Ethereum protocol an account is fundamentally an allocation of memory within the state trie of the blockchain that is assigned an address and which links users with Ether currency balances and blockchain assets. Accounts are stored as a Recursive Length Prefix (RLP) data structure with four elements that represent an account state σ for a given address, *a*, or σ[*a*] as shown in Figure 7 [161]. As Figure 7 illustrates, an account is assigned a 20-byte address. Within the account data structure, the address is followed by a nonce, which is a scalar value equal to the number of Ether currency or smart contract transactions initiated by the account. The nonce is followed by the account's Ether currency balance. The account balance is followed by a storageRoot Hash, which links to its associated assets.



σ[a] account address - 20 bytes
σ [a]$_n$ Nonce- A scalar value equal to the number of transactions sent from this address or, in the case of accounts with associated code, the number of contract-creations made by this account
σ [a]$_b$ Account Balance- A scalar value equal to the amount of Ether currency owned by this address.
σ [a]$_s$ storageRoot Hash- A 32-byte hash of the root node of a Merkle Patricia tree that encodes the storage contents of the account (a mapping between 32-byte integer values)
σ [a]$_c$ codeHash- An immutable hash of the EVM code of the account.

Figure 7. Ethereum Account Data Structure. Adapted from Wood [162].

Associated assets can be completely digital, such as identification records, or a digital record linked to ownership of an asset in the real word, such as a property deed

[163], [164]. These assets are encoded in a Merkle Patricia trie that is hashed to generate the storageRoot Hash [165]. The last element of an Ethereum account is the codeHash, which is a hash of the account's EVM code. As the Ethereum Yellow Paper by Wood explains, the EVM code for an account is the code that is executed to return and adjust account information if the account's address is referenced as part of a blockchain transaction [162].

Unlike the other elements of an account, the account's EVM code cannot be changed so the codeHash for a given account is immutable. Within the complex structure of the Ethereum blockchain, accounts are linked by the state trie for the blockchain, which is downloaded by every node and hashed as part of each block's header. A full diagram of the Ethereum blockchain structure attributed to Lee Thomas, which is too large to include in this document as a figure, is included within the CoinMonks' online article on Ethereum primitives [166].[1]

4. Sequence of Events in a Smart Contract's Execution

To execute a smart contract, the user must submit a blockchain transaction that includes the following elements:

(1) the blockchain address of the smart contract

(2) the requested smart contract function call and associated arguments

(3) the blockchain address of the account making the function call (which also pays the associated Gas tax)

(4) the blockchain address for the recipient account whose digital asset or currency amount is to be queried or modified.

Once the smart contract transaction has been signed with the user's private key, this transaction data is submitted to the blockchain's transaction pool where it can be

accessed by miners assembling the next block.

[1] A file copy of this diagram has been provided for reference in the supplemental materials for this thesis, which can be found along with the thesis in the Naval Postgraduate School's "Calhoun" digital research and institutional materials repository [53].

The mining nodes competing to add the next block to system's blockchain execute the smart contract functions within their node's Ethereum Virtual Machine (EVM) and update their copy of the blockchain ledger to reflect the outcomes of the function call that was made by the user in the smart contract transaction. When the mining nodes finish running all of the transactions in the transaction pool needed for the next block to be added to the system's blockchain, all of the other nodes in the system execute the smart contract code within their respective EVMs as part of the block verification process in the Ethereum protocol's consensus algorithm. The mining node that wins the Proof of Work competition then commits the new block to the blockchain, the Gas tax for the computation is assessed against the account that submitted the transaction, the Gas tax is used to compensate the miner's account, and the new state of the smart contract is reflected on every copy of the blockchain's ledger, which is duplicated on each node throughout the blockchain network.

5. Future of Ethereum

Ethereum is currently the second most widely used blockchain protocol behind Bitcoin. It is presently undergoing development for several major upgrades with the gradual deployment of Ethereum 2.0, which will implement Proof of Stake and sharding [167]. The deployment of Ethereum 2.0 is set to take place over three phases [168]. While the plan for Ethereum 2.0's rollout has been in place since shortly after Ethereum's initial launch in 2015, it has taken years to implement the desired protocol changes [169]. As of October 2020, the first phase of Ethereum 2.0, phase 0, is still in testing and yet to be fully deployed [170]. Once Ethereum 2.0 is fully available, it may be preferable to deploy on future BlockGrids as its Proof of Stake consensus algorithm and sharding implementation will likely lead to faster transactions on a BlockGrid's private Ethereum chain. Using Ethereum 2.0 may also have greater educational value and relevance for students once the public global Ethereum blockchain is running Ethereum 2.0.

In phase 0 of the Ethereum 2.0 rollout, a Proof of Stake network is established and users are provided with the option to utilize Proof of Stake. This Proof of Stake network makes use of the Casper FFG consensus algorithm. Casper FFG combines the two most common approaches to consensus algorithm design, 50% fault tolerance but dependent on network synchrony and 33% fault tolerance but capable of handling asynchrony [171]. Casper FFG utilizes a combined approach in which if conditions are such that there is greater than 50% node fidelity and adequate network synchrony, a chain consensus approach is taken. However, if network synchrony is compromised, the last few blocks in the chain prior to the compromise are replaced, but anything finalized by the asynchronous parallel protocol is added [172]. This creates a best-of-both-worlds scenario in which the consensus algorithm is able to handle a broader range of network conditions and optimize its performance.

In Ethereum 2.0 phase 1, sharding is added to data storage within the Proof of Stake network. Phase 1 does not implement sharding of computation, so during this period smart contracts will continue to operate on the Proof of Work Ethereum 1.0 blockchain. During phase 1 users will be able to be Proof of Stake validators and to store data needed for other blockchain applications within the Proof of Stake network [173].

In Ethereum 2.0 phase 2, a switchover to the Proof of Stake network is made and the Proof of Work blockchain is deprecated. As of 2020, phase 0 is nearly fully implemented and is actively undergoing security auditing and testing [174].

H. COMMON BLOCKCHAIN CYBER ATTACK SCENARIOS

Despite their cybersecurity features, there are many ways that a blockchain can be compromised. Saad et al. provide an extremely thorough review of the full spectrum of currently known attacks against blockchains [175]. Three of the most well-known attacks against blockchains are described below. An understanding of these attacks is foundational to understanding the national security implications of blockchain technology. A review of these attacks may be usefully incorporated into a course curriculum on blockchains.

1. 51% Takeover Attack (n/2 attack)

Perhaps the most feared and consequential attack scenario for blockchains is one in which an adversary gains control of greater than 50% of the mining power in a blockchain network [176]. As described in Chapter 1, the asymmetrical global distribution of mining power for the most widely used public blockchains has elevated concerns about the potential for these types of attacks. Successful n/2 attacks have already occurred several times within less widely used public blockchains such as GHash.IO and Bitcoin Gold [177], [178]. Once the 50% threshold of mining power is exceeded, there are a number of destructive things that the adversary is empowered to do [179].

First, the attacker may arbitrarily block undesired transactions from being verified and added to the blockchain [180]. Second, majority control allows transactions to be reversed. This, in turn, can enable double spending. Third, and perhaps most critically, the attacker can execute a fork of the blockchain and only add new blocks that the attacker approves of [181].

2. Selfish Mining Attack

Selfish Mining is an attack approach in which miners try to increase their mining power rewards by refusing to publish the blocks that they mine [182]. Taking this approach allows a group of selfish miners to create a parallel branch of the blockchain containing only the transactions they wish to include [183]. If the selfish miners are able to achieve a chain length that exceeds the chain forged by the network's honest miners, they can attempt a hostile fork of the blockchain. If a group of bad actors have already successfully executed an n/2 attack, it is trivial for selfish mining and a subsequent hard fork to be carried out. Even if a pool of selfish miners is unable to achieve a hard fork, they can still disrupt network performance by attempting to invalidate block additions from honest miners [184].

3. Eclipse Attacks

An eclipse attack allows an attacker to abuse the peer-to-peer features of the blockchain network to exclude or mislead targeted victim nodes that accept incoming connections [185]. The attack is carried out by an adversary that controls a sufficient number of IP addresses within the network. The adversary monopolizes all connections to and from the targeted node, in effect 'eclipsing' it [186]. It is effectively a Man-in-the-Middle (MITM) attack within a blockchain network. While the level of control imparted to a successful attacker also allows for DoS attacks, typically an attacker will maximize the benefit of an eclipse attack with MITM manipulations of data within the blockchain network.

Eclipsing a victim node is most commonly achieved by maliciously filling the victim node's peer tables with IP addresses controlled by the attacker through repetitive connection attempts to the victim node [187]. Once the peer tables are full of attacker IPs, if the victim node restarts, it will connect to only the attacker's IPs. Attackers can try to force restarts with other attacks or simply wait for a node to restart of its own volition. Once the eclipse is achieved, the attacker can then either exclude the victim node from participation in the network or feed the victim node misleading information. This can be carried out discretely so that the victim node is unaware it has been eclipsed. Eclipse attacks can also be carried out on multiple victim nodes simultaneously if the attackers control a sufficient number of IP addresses within the network.

A successful eclipse attack can enable further attacker capabilities. For example, if enough eclipsed nodes are effectively taken out of the mining pool, an eclipse attack can be leveraged to carry out an n/2 attack. This can occur even if the attacker possesses less than half of the mining power initially because, once a sufficient number of eclipsed nodes are unable to mine, the attacker's percentage of remaining mining power can be above 50%, thus providing effective majority control. An eclipse attack can also allow the attacker to effectively partition the blockchain network [188]. This, in turn, can allow the attacker to engage in double-spending, as some parties are blocked by the effective partitioning from seeing the double spending activity.

I. CHAPTER SUMMARY

This chapter provided a background on blockchain technology. The definition of blockchain, as well as the history of blockchains were reviewed. Key blockchain components and concepts were detailed, including features of the Ethereum protocol used for the BlockGrid prototype. The chapter also described several types of known blockchain cyberattacks. The next chapter provides the system requirements, CONOPS, and an overview of the BlockGrid demonstration.

THIS PAGE INTENTIONALLY LEFT BLANK

III. SYSTEM REQUIREMENTS, CONOPS, AND USE CASES

This chapter defines a cyber-physical blockchain technology demonstration platform called a BlockGrid and provides an overview of its intended functionality. The chapter describes high-level system requirements, fundamental system components, and the system's intended use.

A. BLOCKGRID—A DEFINITION

A BlockGrid is defined here as a blockchain-controlled cyber-physical system in which smart contracts on the blockchain control the behavior of a distributed set of isolated or cooperating circuits. This concept takes inspiration from the large number of available tutorials describing how to implement blockchain nodes on microcomputers [189]–[192]. To meet the need for a cyber-physical blockchain demonstration platform, a BlockGrid creates a private blockchain system out of these microcomputer-blockchain implementations and leverages attached electrical circuits with visual elements to visualize the steps involved in the blockchain's operation for observers.

B. SYSTEM REQUIREMENTS

The purpose of a BlockGrid is to enable demonstration of blockchain technology as applied to physical systems in educational environments. A BlockGrid achieves this purpose by using smart contracts to manipulate the platform's circuits in a manner that creates a visual demonstration of blockchain technology for observers. The smart contracts on the BlockGrid's blockchain control the flow of power from an external power source, such as a wall outlet, to this "grid" of circuits, which consume the electrical power required to perform a physical task (i.e. "smartgrid function") such as lighting, mechanical movement, or multi-media playback. The distributed nature of a BlockGrid's blockchain allows a system operator to activate smartgrid functions from anywhere in the network, creating a blockchain-mediated smartgrid or "BlockGrid." General system requirements are described here.

(1) Small Physical Footprint

The system should be small enough in scale that it can be used in a classroom and transported between different teaching locations. Ideally, the entire system should be run locally so that students can observe all system components during a demonstration. The equipment required for the system should be small enough to fit on a desk.

(2) Cyber-Physicality

To facilitate concrete demonstration of blockchain technology, the BlockGrid should be associated with a cyber-physical system, in contrast to a purely digital one. Transactions that take place on a BlockGrid should have physical consequences that are visible to observers during demonstrations.

(3) Network Functionality

A BlockGrid must be compatible with the local network infrastructure at the demonstration site. Depending on facility conditions such as WLAN access, broadband access, and facility IT-security policies, different approaches can be taken to networking a BlockGrid. However, one way to avoid any possible incompatibility with local network infrastructure and to and maximize system portability between different demonstration environments is to run a BlockGrid on a totally isolated network.

(4) Physical Durability

As a system intended for use in classroom and laboratory environments, a BlockGrid should be resilient to common operational hazards as well as wear-and-tear. The system should be configured to be physically stable and its components minimally fragile. Durability should also inform choices of electronics, casing, and wiring.

(5) Adaptability

For optimal use as a demonstration platform, BlockGrid should be adaptable. It should be possible to modify a BlockGrid beyond its default configuration so the system can be used for alternative applications and demonstrations. To enable this, it is important that the system permit incorporation of software and hardware elements suited to these new configurations.

(6) Low Cost

As the goal of a BlockGrid is to enable educators to demonstrate blockchain technology, the cost of the system should be kept as low as possible. Costs can be minimized through the use of inexpensive microcomputers as system nodes and by constructing the smartgrid from low-cost circuits.

(7) User-Friendliness

As some educators and students may be unfamiliar with one or more BlockGrid system components, the system should be as easy as possible for a novice to assemble and operate. This can be achieved through appropriate system design choices, as well as easy-to-follow user guides. Constructing the system with widely used commercial off-the-shelf (COTS) hardware and software can also help in this regard.

C. SYSTEM COMPONENTS

The fundamental BlockGrid system components can be broken down into three main categories: hardware, software, and users. There are lots of potential choices to suit different environments and applications, many of which are detailed in Chapter IV. However, there are several key types of components common to all implementations.

1. Hardware

The first key hardware component for a BlockGrid is the system control unit (SCU). This is the device through which the user interfaces with the blockchain and executes smart contract transactions. It is also likely to be the device that performs the blockchain's consensus algorithm, depending on the node devices chosen.

The next most significant hardware components of a BlockGrid are the microcomputer network nodes. These computational devices act as nodes for the blockchain network. In order to create a cyber-physical system, these nodes must also run the scripts that control the circuits of the smartgrid. Controlling the circuits of the smartgrid may also require that the nodes generate or gate the current that powers the circuits.

Another hardware element of the system is its networking equipment. Depending on the network configuration, the type of equipment required can vary. For example, if the devices in the system can associate directly with local facility network, the system may not require any additional hardware to set up the blockchain network. Alternatively, if local hotspots are permitted at the demonstration facility, a network could be set up with only a laptop, a router, or cell phone. A fully isolated network, by contrast, might require Ethernet cables, a router, and a switch to connect all the system nodes.

Finally, the system has microcomputer-attached circuits. These connect to the blockchain nodes and are controlled by smart contract transactions on the system. The microcomputer-attached circuits are vital for effective demonstrations as they contain visual elements that observers can witness changing in response to transactions on the blockchain during a demonstration. A conceptual depiction of all these hardware elements is provided in Figure 8.

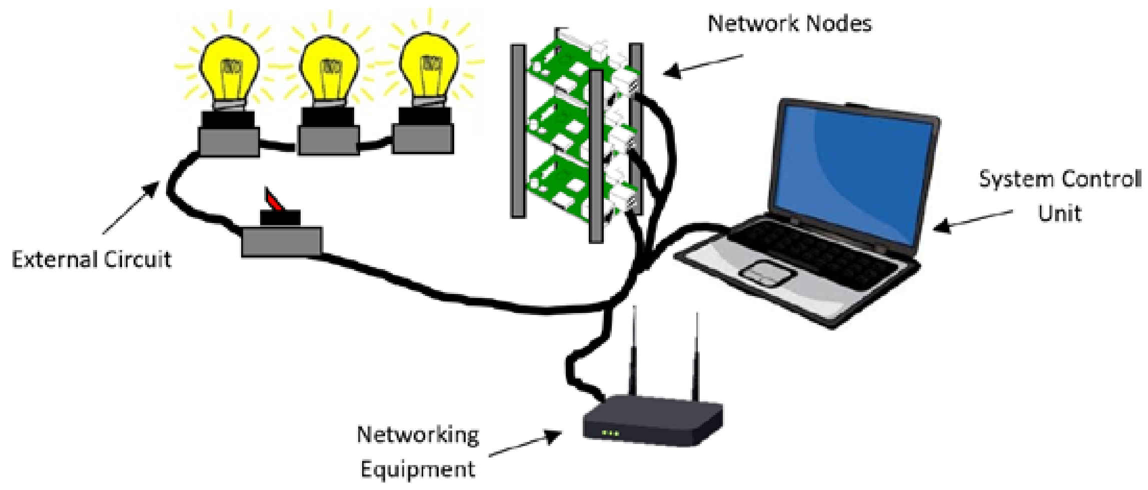


Figure 8. Illustration of BlockGrid Hardware Components.

2. Software

There are several important software components of a BlockGrid. The most significant is its blockchain software. This software is installed on both the SCU and the system microcomputer nodes along with relevant application programming interfaces (APIs) to create the blockchain network.

The SCU also relies on several other software components such as the SCU's operating system (OS) and the smart contract development software. The latter includes the compiler for the programing language in which the smart contracts are written. Software is also installed within the SCU OS that interfaces with the blockchain and its smart contracts through APIs to monitor and execute transactions on the network.

The system microcomputer nodes have software components as well. As with the SCU, each node has an OS. The nodes also run software that interfaces with the blockchain and its smart contracts through APIs. This software allows the nodes to respond to smart contract transactions on the blockchain network when they occur. Scripts that run locally on the node are also required to control the attached circuits.

3. Users

In a BlockGrid system, three user roles are envisioned. These are the administrator, instructor, and student.

a. Administrator

The system administrator configures and maintains a BlockGrid. Administrators assemble the system for use by instructors as well as students and provide technical support for system operation.

b. Instructor

Instructors teaching about blockchain technology may illustrate its application to cyber-physical systems by incorporating simple BlockGrid demonstrations or introductory lab exercises that use a BlockGrid. Instructors may also modify a BlockGrid for more advanced demonstrations or guide students to use the BlockGrid as part of more complex laboratory exercises.

c. Student

Students use a BlockGrid to learn about blockchain technology. This may consist of observing the demonstrations provided by the instructor. Students may also set up their own BlockGrid, modify it, or use a BlockGrid for smart contract development as part of a course laboratory sessions or research.

4. Definition of Active Entities within the BlockGrid Prototype

As the BlockGrid Prototype is a simple implementation meant to be a starting point for the discussion of blockchain technology, it is important to clarify some features of the system.

There are three fundamentally active entities within the BlockGrid prototype system. These are the user, the smart contract data structure, and the Raspberry Pi node LEDs. Their roles are clarified here.

a. Operator

The operator is the human user of the BlockGrid prototype. This can be an instructor giving a demonstration, a student using the prototype for lab work or research, or an administrator servicing the device. The BlockGrid prototype has been designed so that all blockchain and node activity is controlled and initiated by the operator. For example, the operator initiates all smart contract and blockchain transactions between node accounts. While the prototype contains two mining nodes and six non-mining nodes, all of the prototype’s nodes are controlled by the operator via the SCU VM.

b. SmartSwitch Smart Contract Data Structure

The SmartSwitch smart contract data structure is created by the source code of the the SmartSwitch smart contract. It is essentially a two-dimensional array of blockchain node account addresses and an associated integer representing the account’s SmartToken balance. This data structure resides within the smart contract's entry on the blockchain's ledger.

c. Raspberry Pi Node LEDs

The BlockGrid Prototype's Raspberry Pi node LEDs are pairs of red and green LEDs attached to each the system's six Raspberry Pi nodes as part of a breadboard circuit powered and controlled by the Raspberry Pi's GPIO (General Purpose Input Output) unit.

D. CONOPS

This section describes the conditions of operation for the BlockGrid prototype.

1. Cyber-Physical Node States

Within the BlockGrid Prototype there are two node types, mining nodes and passive nodes. The mining nodes reside within the SCU's VM and perform the Proof of Work for the blockchain. The passive nodes reside on the prototype's Raspberry Pi microcomputers. They update and maintain a copy of the blockchain's ledger. The accounts for all nodes in the system, however, have equivalent ability to engage in blockchain transactions. While only the mining nodes accrue Ether currency independent of currency transfers, all node accounts can send and receive currency as well as participate in smart contract transactions.

a. System Control Unit (SCU)

The SCU runs two virtual mining nodes that compete with one another to perform the blockchain's Proof of Work consensus algorithm. As the Ethereum blockchain protocol permits transaction-less empty block generation to incentivize miners on the main chain, these virtual miners are always working and adding new blocks to the blockchain whether or not the system operator initiates any transactions on the BlockGrid Prototype's private chain. The SCU node accounts, therefore, experience a regular accrual of Ether currency with which they can engage in transfers to other node accounts.

However, aside from mining role of the nodes it hosts, the SCU only facilitates user control of the system and does not have any smart contract-mediated cyber-physical state changes. The SCU allows the user to perform setup, perform administrative functions, and to execute Ether currency and smart contract transactions from one interface. It does not, however, power LEDs as the Raspberry Pi's do. Consequently, the SCU has no other defined cyber-physical states within the BlockGrid Prototype network as the default configuration prototype does not give the SCU any defined smartgrid functionality.

b. Microcomputer Nodes

The Raspberry Pi nodes in the BlockGrid Prototype have only two possible cyber-physical states in the default configuration. These are the "red LED" state and the "green LED" state. Within the prototype each node functions independently; i.e., the cyber-physical state of one Raspberry Pi node has no inherent consequence for the status of any other Raspberry Pi node within the system. However, the Raspberry Pi nodes can affect one another via blockchain transactions at direction of the SCU system operator. For example, the operator can execute Ether currency transfers between node accounts. The operator can also initiate SmartSwitch contract function calls from any Raspberry Pi node account that modify the SmartToken balance for other accounts, changing the recipient node's red LED or green LED status. Aside from these blockchain transactions, the prototype's default configuration does not establish any physical node interdependency.

2. Node Events

The only node events possible with the BlockGrid Prototype's default configuration are the switching of power between the nodes' red and green LEDs. These node events result from changes in the nodes' SmartToken balance within the SmartSwitch contract residing on the system's blockchain. These changes in SmartToken balance are created by the SCU operator via depositToken() and withdrawToken() SmartSwitch function calls. In future work more smartgrid functionality could be engineered into the nodes through the creation of more complex smart contracts or the construction of more advanced circuits attached to the Raspberry Pis.

3. System Control

The BlockGrid Prototype is controlled completely by the operator from the SCU interface. Changing node states can only be accomplished via transactions initiated manually by the operator. The timing of these state changes is governed by the timing of system operator’s function calls.

The timing is also affected by the size of the system’s blockchain. Due to the manner in which Ethereum protocol’s Proof of Work consensus algorithm incentivizes constant mining irrespective of transaction frequency, the size of the chain grows quickly. As this growth in the chain occurs, so too does the computational demand of node syncing. Since the Raspberry Pi’s have limited computational power, this can introduce fairly long delays once the number of blocks in the chain grows to order of thousands. With the Ethereum protocol automatically adjusting its difficulty to allow a block to be added approximately every 15 seconds, the blockchain length can cross the thousand block threshold at which the transaction delays become cumbersome in about 4 hours. From observation, a BlockGrid that has been running for over four cumulative hours can experience transaction delays of up to 2–3 minutes. Consequently, the blockchain has to be reset after every few uses to maintain demonstration-friendly transaction times.

4. Power Flow within the BlockGrid Prototype

The BlockGrid Prototype is fundamentally a blockchain-mediated grid of LEDs connected to Raspberry Pis. A blockchain-mediated system is one in which blockchain technology plays a controlling role in the system’s function. In the BlockGrid prototype, a blockchain is used to control the system’s LEDs through smart contract functions executed by the user.

Given the role of decentralized computing in controlling the electricity flow to the prototype’s LEDs, the prototype can be considered an extremely simple form of blockchain-controlled smart grid (hence the name ‘BlockGrid’). However, unlike a real-world smart grid, the microcomputer nodes of the system do not generate power or consume power from one another. Rather, all the microcomputer nodes of the prototype are powered by a conventional wall outlet. A smart contract on the prototype’s blockchain, called SmartSwitch, allows the user to control power flow to the prototype’s LEDs.

This control results from SmartSwitch defining a SmartToken that can be deposited to or withdrawn from a node’s account. The number of SmartTokens in a node’s account determines whether a red or green LED is powered on the node. If the number of SmartTokens for a node's account is 0, the node's red LED is powered on. If the number of SmartTokens for a node's account is greater than 0, the node's green LED is powered on.

A flow chart explaining the control sequence governing power flow to LEDs of the prototype is provided in Figure 9. As shown in Figure 9, to start the power flow control sequence the user executes the SmartSwitch contract functions depositToken() or withdrawToken() to adjust the number of SmartTokens in a Raspberry Pi node's account. A script runs that runs locally on the Raspberry Pi watches for any changes in the SmartSwitch contract and calls the SmartSwitch getToken() function when a change occurs to determine the updated number of SmartTokens in the node's account. This process is event driven. If the number of SmartTokens is 0, the script directs power flow out through the Raspberry Pi's GPIO pin that is connected to the red LED on the attached circuit board. If the number of SmartTokens is greater than 0, the script directs power flow out through the Raspberry Pi's GPIO pin that is connected to the green LED on the attached circuit board.

Within the LED control function of the prototype, the system's blockchain functions to create a distributed ledger that reflects the SmartToken balance of each of the system's nodes. This allows for the node balances to be adjusted in a decentralized way via the SmartSwitch contract's functions. The BlockGrid Prototype therefore demonstrates merely how smart contracts can "gate" access to resources within a cyber-physical system rather than directly managing the generation or consumption of power. It does not facilitate any smart grid functionality beyond the blockchain-mediated control of power flow to its LEDs.

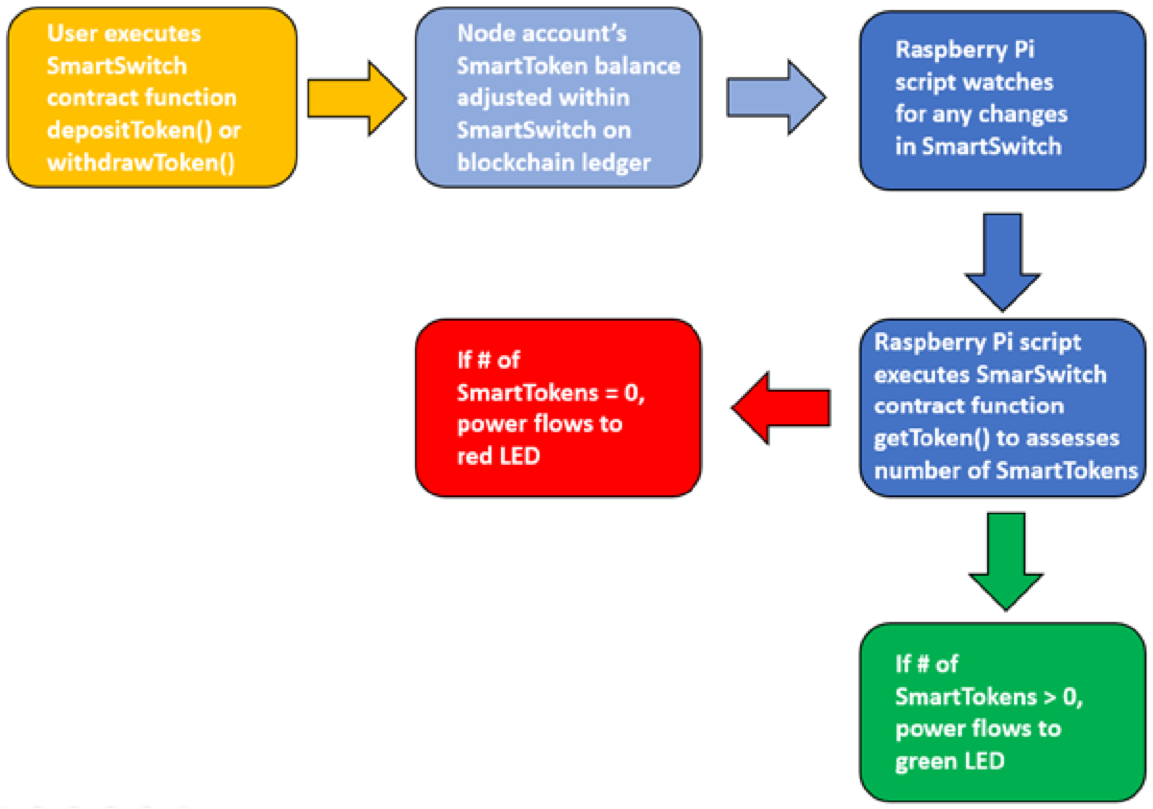

User executes SmartSwitch contract function depositToken() or withdrawToken()
Node account's SmartToken balance adjusted within SmartSwitch on blockchain ledger
Raspberry Pi script watches for any changes in SmartSwitch
Raspberry Pi script executes SmarSwitch contract function getToken() to assesses number of SmartTokens
If # of SmartTokens = 0, power flows to red LED
If # of SmartTokens > 0, power flows to green LED

Figure 9. Smart Contract-Mediated Power Flow Control in the BlockGrid Prototype.

5. Role of SmartTokens in the BlockGrid Prototype

The SmartTokens defined by the SmartSwitch contract are not used in the BlockGrid Prototype for the direct purchase of resources within the system. They only function as an arbitrary unit that can be counted and analyzed by scripts that run on each of the nodes. Within the SmartSwitch contract they are just integer values associated with a node account's address via a two-dimensional array data structure. Essentially, SmartTokens allow the system operator to assign zero; i.e. "red light," and nonzero; i.e. "green light," values to each node's account. The SmartSwitch contract is coded not to allow a SmartToken balance to go below zero.

E. OVERVIEW OF SYSTEM OPERATION

Three use cases are envisioned for a BlockGrid. The first is in classroom demonstrations. Another is to explore alternative system configurations, possibly for new classroom demonstrations or research. Finally, the system may be used for smart contract development.

1. Classroom Demonstrations

To carry out a BlockGrid demonstration, the system can be set up on a table in the demonstration environment. The SCU can be connected to a projector so that students can observe the actions taken by the instructor to manage the system in addition to the observable physical effects of the blockchain’s activity on the system’s microcomputer-attached circuits. A photo from a BlockGrid demonstration given at Discover NPS Day 2019 using such a setup is shown in Figure 10.

Figure 10. A BlockGrid Demonstration at Discover NPS Day 2019.

2. Modifications to Default Configurations

Once the default configuration and initial demonstration have been fully utilized, it is anticipated that BlockGrid users may explore modifying the system. This could involve, for example, attaching different nodes and circuits to create a more complex smartgrid. Alternative blockchain demonstrations could be created for class assignments, or for blockchain-based physical computing research.

3. Smart Contract Testing

The third envisioned application of a BlockGrid is as a test bed for smart contract development. Use of a BlockGrid for this purpose could take place as part of class-related projects or research. The cyber-physical nature of the system permits convenient experiments on smart contract functionality and the system contains all of the software required for smart contract generation.

F. CHAPTER SUMMARY

This chapter has provided a description of the requirements, components, and applications of the BlockGrid. The next chapter will review currently available hardware and software options for BlockGrid system design. The rationale behind the specific implementation choices for the BlockGrid prototype is also explored.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. CONFIGURATION RATIONALE

A blockchain-mediated smartgrid can be constructed in a variety of ways to meet the BlockGrid system requirements outlined in the previous chapter. This chapter is intended to provide readers with sense of the possible options for each of the system's components should they wish to design their own BlockGrid. The chapter also details the rationale for the choices that were made in BlockGrid prototype's implementation.

This chapter begins with a description of the system hardware implementation options. These include the SCU, the system node microcomputers, the networking configuration, and the smartgrid circuits. Each section reviews the system subunit requirements, options available for users, and the rationale for the choice made in the BlockGrid prototype's implementation.

The hardware section is followed by a description of the software implementation options. These include the SCU OS, the node microcomputer operating system, and the blockchain software. Each of these sections also details the software requirements, choices available for users, and the rationale for the selection made in the BlockGrid prototype's implementation.

A. METHODS OF EVALUATION

When evaluating the available options for each BlockGrid system requirement, the options were ranked against one another on several relevant evaluation criteria with the ranking of 1 being the best. Each option's ranks were summed across the evaluation criteria to create an aggregate score. In this system requirement option with the lowest aggregate score was considered the best-performing choice. Generally, the best-performing option was selected for the BlockGrid prototype's implementation. However, in some cases a lower-performing option was chosen because it was easier to obtain, more user-friendly, or had more extensive supporting documentation.

B. SYSTEM HARDWARE

The BlockGrid’s fundamental hardware components are the SCU, the microcomputer nodes, the networking equipment, and the smartgrid circuits. For each of the system hardware components there are several options available, and their respective desirability may vary depending on the overall desired system configuration. This section outlines the most common options that are currently available and explains the choices that were made for the BlockGrid’s default configuration.

1. System Control Unit (SCU)

The most important hardware component for the BlockGrid is the SCU. Depending on the desired system configuration, there are several different suitable hardware platforms. For instance, the SCU could be a desktop or a powerful microcomputer. However, the most easy-to-use and versatile platform for the SCU is a laptop. This section evaluates the available SCU laptop configuration options.

a. Criteria

There are several important criteria for a laptop-based SCU. First, it must have sufficient computational power to perform the consensus algorithm operations for the blockchain software. It should also be easy to operate for the user. As the intended use environment is an educational setting, cost should be minimized. Additionally, portability is an important consideration as the BlockGrid may need to be transferred to different demonstration locations. Finally, the SCU configuration should have robust user support materials that can be referenced for troubleshooting.

b. Available Choices

Among possible laptop configurations there are several options. These include using a designated laptop with its native OS, having the system operator use their own laptop with its native OS, using the operator’s laptop to run a cloud-based virtual machine instance, and having the user run a provided virtual machine (VM) on their own laptop. Each option has benefits and disadvantages that might make it an ideal choice depending on the what the BlockGrid architect is optimizing for. For the BlockGrid prototype, user-friendliness and low cost were the highest-priority SCU traits.

(1) Designated laptop computer with native OS

In this approach, the BlockGrid is controlled by an administrator-owned laptop that is designated for the system. The biggest benefit of this approach is that the user can simply log on and follow operation instructions to run the system with no installations or setup required. This option also allows the administrator to select a laptop with hardware specifications ideally suited to run the BlockGrid.

This approach, however, has several downsides. First, it negatively impacts the system's portability. If the system needs to be moved to different locations or reproduced at an alternative site, the designated laptop has to be transported or acquired. This approach may also constrain potential modification of the system, with installs on the designated system requiring administrator privileges and close oversight. System resiliency in the event of a hardware error is also a concern. If a problem occurs with the designated laptop, it has to be replaced rather than merely switched out for a different laptop onsite.

(2) User laptop with native OS

An alternative but similar approach is to utilize the user's laptop with its native OS. With this approach, the user must install the required SCU software on their own system to operate the BlockGrid. The user must also configure their laptop to interface with the rest of the BlockGrid system hardware.

This option maximizes portability and flexibility for the user. It may be especially useful in educational settings where students create their own BlockGrid system or develop smart contracts as part of class assignments. This approach may also offer a better user experience because it allows BlockGrid operation with an OS familiar to the user.

However, there is a significant downside for both the user and the system administrator with this option. Since users may differ substantially from one another in their host operating system environments, it is infeasible for system administrators to develop user support materials that cover all of the possible user laptop environments.

Consequently, users may face greater troubleshooting difficulties and are likely to have disparate user experiences with the system.

(3) User laptop with cloud instance

Another approach is to have the user's laptop launch a cloud VM instance that acts as the SCU. A cloud VM instance is a virtual machine hosted by a cloud services provider (CSP). Administrators using this approach could allow users to install the SCU software on their own instance or provide a customized cloud-hosted image with the SCU software pre-installed.

The biggest upside to this cloud-based approach is that it offers maximal computational resource flexibility. With a cloud VM instance, users and administrators can choose the SCU processing power most appropriate for their BlockGrid system. In an educational setting this option also allows students to gain experience with cloud computing, allowing the BlockGrid to be a curricular vehicle for exploring possible integration of blockchain, IoT, and cloud computing technology. The pay-as-you-go model of cloud computing can also be less expensive than buying dedicated SCU hardware with the desired capabilities.

However, requiring users to access and use cloud resources has several drawbacks. In order to access a cloud VM instance, users must first set up a billable account with a CSP. This process takes time requires users to provide personally identifiable information (PII) which creates an additional logistical burden in an educational setting. This approach also requires users to monitor the charges they generate once they set up a billable account with a CSP and to take appropriate action to curtail excessive charges.

Running a cloud VM instance also requires that users have reliable internet access and that they are operating the BlockGrid on a network that allows blockchain applications to run from the cloud and communicate with devices that may be behind local firewalls. This approach could therefore be challenging to implement an environment with heightened security infrastructure such as a DOD facility.

(4) User laptop with provided VM

The final option is to control the BlockGrid from a VM installed on the user's laptop. In this approach the system administrator preinstalls all SCU software and configuration settings on a disc image that is made available for users to install on their laptop.

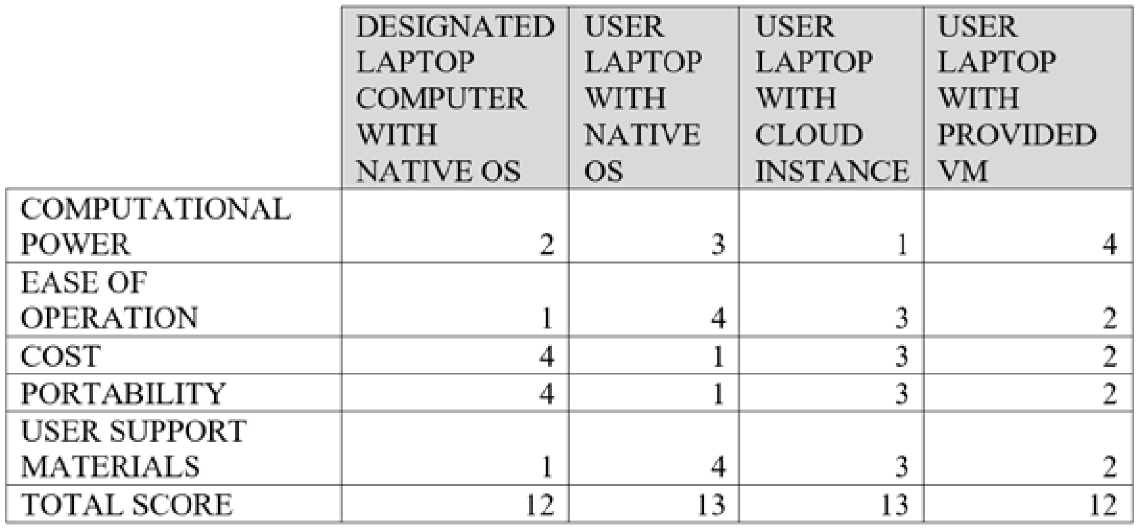
This approach has several advantages. Using an administrator-provided VM allows many of the factors that affect user experience to be held constant. This SCU configuration also makes troubleshooting problems that arise in an educational setting with many users much easier to resolve because all users operate the system within the same SCU environment. Furthermore, this approach also removes the burden of multiple software installs on the user. With this option only the installation of VM management software and the provided VM is required to make their laptop an operational SCU.

The chief disadvantage of the administrator-provided VM is that it requires the user to be proficient in setting up and troubleshooting the network connections from SCU to the rest of the prototype through a VM on their laptop. While students are likely proficient at installing and utilizing VMs, depending on the pre-existing configuration of the SCU laptop some adjustments to the laptops network settings may be required to connect the VM to the prototype's network. Particularly if the administrator-provided VM uses an operating system that is unfamiliar to the user, this approach may introduce an unavoidably steep learning curve for the user to become proficient setting up the SCU and using it to control the BlockGrid. This option also requires that the user possess a laptop powerful enough to allocate sufficient RAM (Random Access Memory) and hard drive space for the VM to perform efficiently as a BlockGrid SCU. A preconfigured VM may require multiple GB (Gigabytes) of hard drive space to install, and multiple GB of RAM may be required to run the desired blockchain protocol effectively.

(5) BlockGrid Prototype implementation choice

For the BlockGrid prototype's default SCU configuration, the "user laptop with a provided VM" option was selected. As shown in Table 1, each of the four options were ranked against one another with 1 being the best and 4 being the worst on the criteria of computational power, ease of operation, cost, portability, and user support materials. The rankings for each criterion were added for each option to create a composite score. In this scoring system, the SCU option with the lowest composite score is considered the best choice. Using this scoring method, the designated laptop and the administrator-provided VM are tied for the lowest composite score with 12. Despite the tie score, the administrator-provided VM option was chosen for the BlockGrid due to its substantial advantages for low cost and high portability.

Table 1. Ranking of System Control Laptop Options



	DESIGNATED LAPTOP COMPUTER WITH NATIVE OS	USER LAPTOP WITH NATIVE OS	USER LAPTOP WITH CLOUD INSTANCE	USER LAPTOP WITH PROVIDED VM
COMPUTATIONAL POWER	2	3	1	4
EASE OF OPERATION	1	4	3	2
COST	4	1	3	2
PORTABILITY	4	1	3	2
USER SUPPORT MATERIALS	1	4	3	2
TOTAL SCORE	12	13	13	12

2. System Node Microcomputers

The next hardware elements to implement for a BlockGrid are the system's microcomputer nodes. These devices can vary in their roles within the system depending on the system's smartgrid design, but at minimum they function as nodes for the blockchain and hosts of scripts that control the smartgrid's circuits. They likely also generate or gate the current that powers the smartgrid's circuits.

a. Criteria

For the BlockGrid to function effectively as both a blockchain network and a smartgrid, the system node microcomputers need to have several important traits. First, the microcomputers must have sufficient computational power to support both the system's blockchain protocol and local smartgrid applications. To support smartgrid computing, the devices should have adequate General Purpose Input/Output (GPIO) interfaces with accompanying programming libraries. Additionally, the devices should be easy for a beginner to use. User support resources for the microcomputers should also be well-developed. As the BlockGrid is intended for use in an educational setting, cost should also be minimized. For the BlockGrid prototype implementation, minimizing microcomputer cost was an especially high priority.

b. Available Choices

Recent years have seen a dramatic increase in the number of microcomputers on the market as chip sizes and manufacturing costs decrease [193], [194]. Increasingly, microcomputers are being designed for specific applications with a range of different specifications and capabilities [195]. The following are the most common general-purpose microcomputer options available at the time of this writing.

(1) Arduino Due

Arduino is one of the most popular commercial makers of microcomputers for physical computing [196]–[198]. They offer several different models, with the Due model having the most computational power by a significant margin. The Due runs on a 32-bit ARM processor with the custom Arduino RIOT OS, making it serviceable as a low-end microcomputer that can run full-scale software rather than merely physical computing scripts. It also has a 54-pin GPIO and a 12-pin analogue input unit. Presently, the device costs approximately $40 [199].

The Arduino Due is a compelling option for the BlockGrid because it has unparalleled physical computing attributes and support resources. The 54-pin GPIO offers the most surface area for smartgrid connections of any microcomputer available. Extensive script libraries exist for a wide range of physical computing applications. There are also extensive tutorials and user support forums available [200]–[202].

The main drawback to using the Arduino Due is that while it the most computational power of any Arduino product, it has limited computational power compared to other microcomputer alternatives. It also runs an unusual OS, which would make it difficult to run blockchain software on the devices. The Arduino-specific OS also creates a steep learning curve for users who have not used an Arduino before.

(2) Orange Pi PC2

The Orange Pi PC2 is one of the least expensive general-purpose microcomputers available. It features a quad-core 64-bit ARM processor, a hexa-core Mali GPU, a 40-pin GPIO, and 1GB DDR3 (Double Data Rate 3) RAM. The device can run an Android, Ubuntu, Debian, or Raspbian OS. Presently, the device costs approximately $20 [203].

The most significant upside to using the Orange Pi PC2 is its combination of relatively high computational power and low price. Though its 1GB of RAM likely does not allow the device to handle Proof of Work consensus algorithm computations, its CPU and GPU make relatively sophisticated smartgrid applications possible. It also has a variety of memory storage interfaces that could make it well suited to smartgrid applications with higher data storage requirements.

The most significant drawback to the Orange Pi PC2 its dearth of user support materials. The board is primarily sold in China, and English is not the default language for many of the available support materials and documentation [204]. For users at USG facilities that block web access to sites hosted at Chinese IP addresses, it may be challenging to access software and user support resources.

(3) NanoPi A64

The NanoPi A64 from FriendlyARM is another inexpensive microcomputer. It has a quad-core 64-bit ARM processor, a 40-pin GPIO, and 1GB DDR3 RAM. There are Ubuntu Core and Ubuntu MATE OS images available for the device. Presently, the device costs approximately $20 [205].

The biggest upside to the NanoPi A64 is its low cost. It has computational power that exceeds that of the more common Raspberry Pi 3B for an even lower price [206]. The device also has one of the smallest microcomputer boards available and is therefore well suited for smartgrid applications in which system nodes need to be attached to larger IoT devices.

The biggest downside to using this device is the minimal user support resources available. The manufacturer provides a Wiki page for the device and a general English language forum for their product line as a whole, but at the time of this writing there are no user support forums specific to the A64 model. There are also only two Linux images currently available for the NanoPi A64, which may constrain the types of applications that could be developed for a BlockGrid using these devices.

(4) Raspberry Pi 3B

The Raspberry Pi 3B is one of the most popular microcomputers available [207]–[209]. It features a 32-bit ARM v7 chip, 1GB DDR2 RAM, and a 40-pin GPIO. The manufacturer offers two Linux OS distributions, and the device is compatible with numerous third-party OS options. Presently the device costs approximately $35 [210].

The biggest upside of the Raspberry Pi 3B is that it has robust user support materials [211]–[213]. Troubleshooting forums and tutorials for software installations, physical computing, and IoT computing are widely available [214]–[216]. There are a large number of tutorials available for implementing blockchain protocols on Raspberry Pis [217]–[219]. There are also precedents for Raspberry Pis being explored for smart grid applications [220].

The main drawback to using the Raspberry Pi 3B is its low computational power. The device cannot handle rigorous computing applications like Proof of Work blockchain consensus algorithms. It can therefore only act as a passive node on the blockchain network that retains copy of the distributed ledger but does no mining. Even in a passive node role, it may also struggle to perform the computation required to sync the large data trie structures common to some protocols [221]. Consequently, the transaction speed for a BlockGrid running blockchain protocol with a Proof of Work consensus algorithm with

Raspberry Pi 3B nodes will be dependent on the computational power of the SCU for mining and the Raspberry Pi 3B nodes for updating their copies of the ledger. This may cause a BlockGrid to execute its software functions slowly.

(5) Banana Pi M64

The Banana Pi M64 is a microcomputer with intermediate computational power and an intermediate price. It has a quad-core 64-bit ARM processor, 2GB DDR3 RAM, and a 40-pin GPIO. It can run a wide variety of OS options. Presently its price is approximately $50 [222].

The advantage of using the Banana Pi M64 is that it offers slightly higher computational power than the Orange Pi PC2, Nano Pi A64, and Raspberry Pi 3B. Although the major hardware difference between the device and its cheaper alternatives is only an extra GB of RAM, this can enable the device to run more demanding software. It also has a battery power interface that allows it to be used with portable IoT devices.

There are two main downsides to the Banana Pi M64. First, it is notably more expensive than its cheaper alternatives while providing only marginally more computational power [223]. The manufacturer also offers limited user support for the device. The manufacturer hosts a GitHub page and a user support forum for the device, but these have far fewer resources than exist for more widely used devices such as the Raspberry Pi 3B. Consequently, troubleshooting problems with the Banana Pi M64 may be challenging for users.

(6) ROCKPro64

The ROCKPro64 from Pine64 is a fairly expensive but very powerful microcomputer. It features a hexa-core ARM processor, a quad-core Mali GPU, 4GB of DDR4 RAM, and a 40-pin GPIO. It also supports a wide variety of Linux operating systems. Presently, the device costs approximately $80 [224].

The biggest advantage of using the ROCKPro64 is its combination of outstanding computational power and relatively modest price. In a BlockGrid implementation these devices could likely function as mining nodes on the system rather than just passive updater nodes in the blockchain network. If a blockchain protocol with a Proof of Work consensus algorithm is run on the BlockGrid, using these devices could improve block commit times and help the system function more smoothly.

The impressive computational resources of the device are offset, however, by comparatively limited user support resources. Pine64 offers a Wiki and community forums for its line of products as a whole, but little content is specific to the ROCKPro64 model. Additionally, at his time there is not a complete GPIO programming library currently available for the platform, which complicates programming smartgrid applications [225].

(7) Odroid XU4

The Odroid XU4 is one of the most powerful ARM chip microcomputers available. It has an octa-core 64-bit ARM CPU, a hexa-core Mali GPU, 2GB DDR3 RAM, and separate 12-pin and 30-pin GPIO units. The manufacturer offers a variety of Linux and Android OS options. Presently, the device costs approximately $60 [226].

The Odroid XU4 is an appealing choice because of its impressive computational power and relatively low cost. The device is powerful enough to run laptop applications. If a blockchain protocol with a Proof of Work consensus algorithm is chosen for the BlockGrid, using these devices could reduce block commit times and improve system performance.

However, there are several disadvantages to the Odroid XU4. It lacks onboard WIFI and Bluetooth capabilities, which could complicate BlockGrid implementations that are wirelessly networked. It also lacks an audio codec, so sound output is only possible through HDMI or a USB adapter [227]. And while the manufacturer offers a Wiki and user forum site for their line of products, the user support materials for the XU4 are sparse compared to more widely used options like the Raspberry Pi 3B [228].

(8) UP Board

The UP Board is a powerful 64-bit microcomputer with a quad-core x86 intel atom processor. It has a 40-pin programmable GPIO, 4GB DDR3 RAM, a 64 GB flash drive, and a battery interface. It can run a desktop version of Windows 10 and the manufacturer also offers other OS options. Presently, the device costs approximately $170 [229].

The biggest advantage of the UP Board is that with an x86 chip it can run a conventional OS. This makes a much broader range of software available for the device. With its chip and RAM specifications, the UP board has the computational capacity of a small laptop. It is therefore likely capable of performing Proof of Work consensus algorithms for the blockchain and advanced IoT applications. The manufacturer also provides robust and user-friendly support materials [230].

There are only two significant drawbacks to the UP Board. Despite its advanced computational capacity, it lacks onboard WIFI and Bluetooth capabilities. As with the Odroid XU4, this could complicate wireless BlockGrid implementations and IoT smartgrid applications. The other drawback is the high price of the device. This high device cost reduces the feasibility of acquiring enough Up Board devices for multi-device BlockGrid implementations and creates challenges with replacing devices should they break.

(9) LattePanda 4G/64GB

The LattePanda is another powerful Windows microcomputer. It has a quad-core 64-bit intel processor, 4GB DDR3 RAM, and an onboard 64GB flash drive. It has 32 GPIO pins in total, comprised of a 20-pin Arduino Leonardo GPIO unit, 6 gravity sensor connectors, and 6 processor chip pins. The device comes preinstalled with 64-bit Windows 10 operating system as well as Arduino’s physical computing software. The manufacturer provides an Ubuntu image for download. It is also among the few high-power microcomputers with both onboard WIFI and Bluetooth. The device presently costs approximately $150 [231].

The LattePanda is a compelling choice for the BlockGrid for several reasons. First, it is powerful enough to do Proof of Work algorithms for the blockchain. As a native Windows computer, it can also run a wide variety of software [232]. Additionally, its Arduino GPIO unit allows the device to make full use of the vast existing script libraries and tutorials available for Arduino devices [233].

However, there are two significant downsides to the LattePanda. The first is there are not many user-support materials for the device. The manufacturer provides a user community forum on its website along with fairly good device documentation, but the user base for the device is small and there are few preexisting guides for physical computing applications or troubleshooting [234]. The second disadvantage of the LattePanda is that the high price of the device. As noted with other expensive microcomputers, the high device cost reduces the feasibility of acquiring enough devices for multi-device BlockGrid implementations and creates challenges with replacing devices should they break.

(10) UDOO x86 II ULTRA

The UDOO x86 II ULTRA is the most powerful microcomputer currently available. It has a 64-bit 2.56 GHz Intel Pentium CPU, 8GB DDR3 RAM, a 32GB flash drive, and an Intel HD 405 graphics card. It also runs a separate onboard 32MHZ dual-core SE/ARC microcontroller and has a full Arduino 101-compatible GPIO with programmable 5V output pins and the Arduino 101 IDE. Presently, the device costs approximately $250 [235].

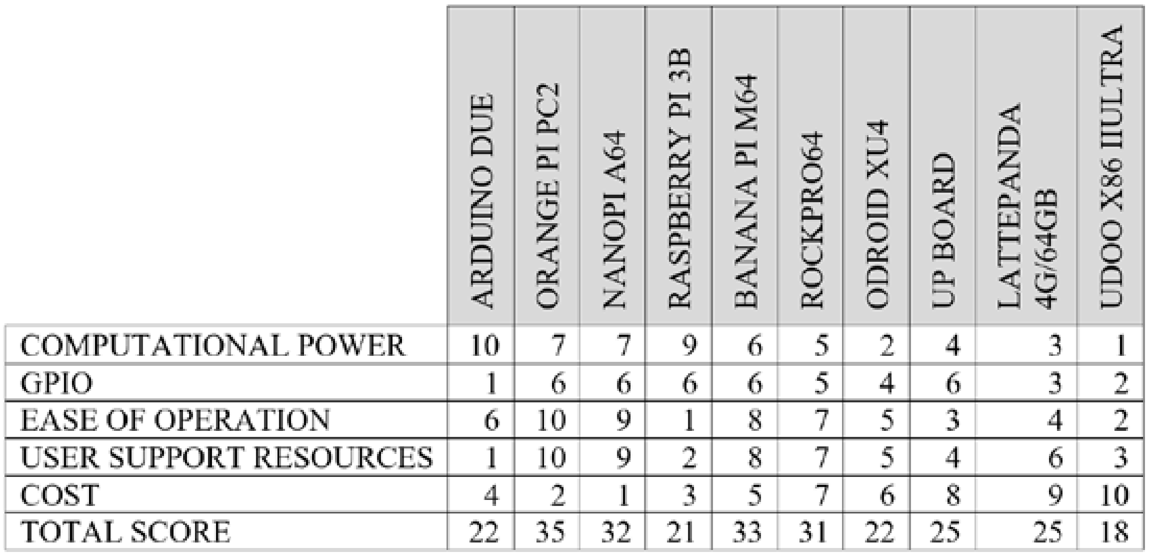
The UDOO x86 II Ultra is an excellent choice for a BlockGrid. Its powerful CPU likely allows it to perform blockchain protocol Proof of Work algorithms. As a fully-fledged Windows x/86 computer, it has a wide variety of available software along with the hardware capacity to run full-scale computer applications [236]. The Arduino GPIO unit and IDE provide access to the full range of existing Arduino physical computing software [237]. Additionally, the separate onboard microcontroller unit minimizes the computational load of GPIO programming on the device [238].

Despite its impressive features, there are two significant drawbacks to the UDOO x86 II Ultra. The first potential issue is the device's size. It is roughly twice the size of the other microcomputer options, which could limit its utility for some smartgrid applications [239]. The other significant drawback to this board is its high cost. As noted with other expensive microcomputers, the high device cost reduces the feasibility of acquiring enough devices for multi-device BlockGrid implementations and creates challenges with replacing devices should they break.

c. *Selection For Default Configuration*

The Raspberry Pi 3B was chosen to be the microcomputer model for the BlockGrid prototype. As shown in Table 2, all ten microcomputer options were ranked against one another one with 1 being the best and 10 being the worst on the criteria of computational power, GPIO functionality, ease of operation, user support resources, and cost. In this scoring system the lowest aggregate score across all criteria corresponds to the best option.

Table 2. Ranking of Microcomputer Node Options



	ARDUINO DUE	ORANGE PI PC2	NANOPI A64	RASPBERRY PI 3B	BANANA PI M64	ROCKPRO64	ODROID XU4	UP BOARD	LATTEPANDA 4G/64GB	UDOO X86 IIULTRA
COMPUTATIONAL POWER	10	7	7	9	6	5	2	4	3	1
GPIO	1	6	6	6	6	5	4	6	3	2
EASE OF OPERATION	6	10	9	1	8	7	5	3	4	2
USER SUPPORT RESOURCES	1	10	9	2	8	7	5	4	6	3
COST	4	2	1	3	5	7	6	8	9	10
TOTAL SCORE	22	35	32	21	33	31	22	25	25	18

The UDOO X86 ULTRA had the lowest aggregate score of all the microcomputer options at 18. However, it was not selected for the BlockGrid prototype implementation due to its high cost. Although the Raspberry Pi 3B had a higher composite score at 21 and considerably less computational power, its combination of robust user support materials and low cost made it the most feasible option to implement for the prototype.

Indeed, Raspberry Pi user support materials proved to be extremely useful in the construction of the BlockGrid prototype. The Chainskills tutorial series by Eloudrhiri referenced in Appendix B explains how to implement Ethereum on a Raspberry Pi and use a smart contract to control an attached circuit. As described in Appendix B, these tutorials were used as a guide for the prototype's node design and can be used with a few modifications to reproduce the BlockGrid prototype. The low device cost of the Raspberry Pi 3B also allowed for the creation of a BlockGrid prototype with many nodes, which enhances its utility as a demonstration platform.

3. Networking Configuration

Another necessary hardware implementation decision for a BlockGrid is the system's network configuration. This decision has a significant impact on the portability of the system. For a networking configuration to be viable in a demonstration setting, there are several criteria that it must meet.

a. Criteria

First, the network setup has to be amenable to the BlockGrid's blockchain software. This requires the networking configuration to support blockchain network communications between nodes and the SCU. It also requires that the networking configuration be compatible with local facility requirements at the demonstration site. For example, if the local facility WLAN does not permit blockchain software to run on its network, the BlockGrid's networking configuration must support an isolated network.

Second, the network setup must be amenable to the BlockGrid's intended smartgrid functionality. For example, a smartgrid implementation that uses smart contracts to power mobile IoT devices like model trains would require a wireless network. Additionally, for management of both the blockchain and smartgrid system elements, the system nodes need to be identifiable by an IP address. This is most easily accomplished if the networking approach allows IP addresses to be statically defined, such as with a programmable COTS router.

The networking configuration should also be as user-friendly as possible. For a BlockGrid to be a user-friendly demonstration platform, the system's network must also be reliable, easy to set up, and possible to troubleshoot if something goes wrong. As the BlockGrid is intended for an educational setting, minimizing cost should also be a high priority.

To evaluate the best networking implementation for the BlockGrid prototype, the options were ranked against one another on the criteria of blockchain compatibility, smartgrid compatibility, setup difficulty, network reliability, troubleshooting difficulty, and cost with 1 being the best. The rankings for each option were summed to create a composite score. In this scoring system the lowest composite score corresponds to the best option.

b. Available Choices

There are five main network configuration options that can be implemented on a BlockGrid. Each option has its own advantages and disadvantages. The best option depends on the intended demonstration environment and envisioned smartgrid functionality. As the BlockGrid prototype is intended for use at the Naval Postgraduate School and other DOD environments, minimizing the need for network connectivity was a high priority in the design choice.

(1) Facility WLAN

Perhaps the most straight-forward networking arrangement for the BlockGrid is to have each system node connect to the local facility WLAN. This option has several advantages. First, it eliminates the need for the user to go through an extensive networking setup. Each node merely needs to authenticate to the local facility WLAN. Once each node's assigned IP has been noted, any further intra-node transfers or connections can be established as necessary. This option also allows the microcomputer nodes to have maximum freedom in their physical positioning as well as their computational role. This setup could, for example, allow nodes to independently access cloud or internet-based resources to carry out more elaborate smartgrid functionalities. A conceptual diagram of this networking approach is shown in Figure 11.

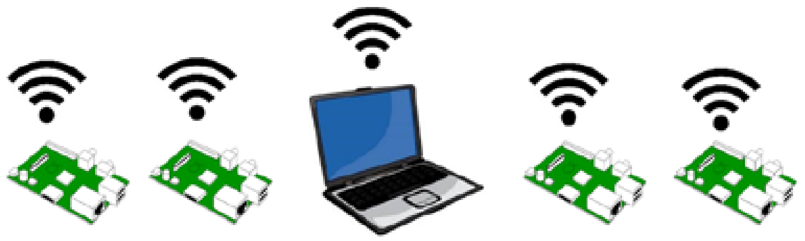


Figure 11. Networking Configuration with All Nodes Connected on Local WLAN.

The chief drawback to this networking approach is that its success depends crucially on the system having access to an environment in which blockchain applications and microcomputers are permitted to run on the local facility WLAN. Some USG facilities prohibit microcomputers from authenticating to networks because they typically have poor cybersecurity features and have led to high profile compromises in the past [240]. It is also increasingly feasible for facility networks to detect and block execution of blockchain applications [241]. This is commonly done as the execution of blockchain software, particularly cryptocurrency mining, can be an indicator of a compromised device [242]. This wireless networking approach also requires node IP addresses to be updated in the blockchain software configuration tables after each new network authentication by a microcomputer node. The frequent configuration table updates are necessary because many facility wireless networks utilize the dynamic host configuration protocol (DHCP) in which nodes can potentially be assigned a new IP address every time they authenticate.

(2) WLAN with Software Router

Another BlockGrid networking approach that requires no extra hardware is to run a wireless access point (WAP) using a software defined router on the SCU. In this configuration all the nodes are connected to the blockchain wirelessly. This arrangement can also be configured to provide the system nodes internet access through the local facility network if the WAP is set up as a new node for the local facility WLAN. Optionally, users may wish to attach an external antenna to their SCU laptop to boost the WAP signal.

This option makes the system maximally portable and lightweight. It also frees up the system nodes to be positioned in whatever physical orientation is optimal for the smartgrid design. The use of a software defined router also permits the assignment of static IP addresses for the microcomputer nodes. Additionally, it enables implementation of firewall features, if desired, to control traffic to and from the blockchain network. A conceptual diagram of this networking approach is shown in Figure 12.

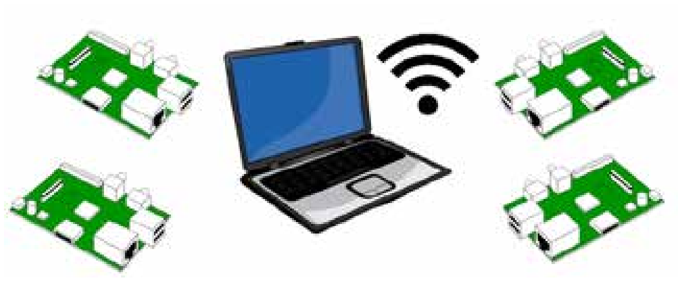


Figure 12. Networking Configuration with WLAN through a Software Router.

There are several downsides, however, to this SCU software router approach. First, of all of the possible options this network requires the most expertise to set up. Instructions can be provided but there are numerous points along the software router setup process where differences in hardware, network cards, and operating systems can result in different responses to the administrator-provided setup commands. This can result in troubleshooting difficulties and disparate user experiences.

Another issue with this option is that the software router must be established in an environment that permits unofficial WAPs at the facility or on the facility network [243]. It is increasingly common for local IT-Security defense tools to scan for unauthorized access points and to shut them down upon discovery, as rogue WAPs can be an effective tool for cyberattacks [244], [245]. Additionally, if internet connectivity is desired on the WLAN, this setup requires that the facility permit running of blockchain applications on the facility network which, as previously noted, is often prohibited.

Finally, there is the problem of network reliability that is common to all WLAN options. Wireless networks are vulnerable to interference from materials in the environment and antennas can drop connections. WLAN operations also involve an interdependence of authentications, credentials, protocols, and software interactions that can fail and which can be difficult to troubleshoot when errors occur.

(3) WLAN with COTS Router

A third BlockGrid network configuration option is to use a COTS wireless router to set up a WAP for the blockchain. As with the other WLAN options, this allows the system nodes to connect to the blockchain wirelessly. It also allows maximal freedom for the physical orientation of the nodes in the smartgrid design.

There are also some COTS router features that can be an advantage. First, the COTS router setup is much less complicated than the software router setup. Many COTS routers also have an HTML (Hyper Text Markup Language) configuration GUI accessible at the default router IP address of 192.168.1.1. This can enable straightforward assignment of static IPs for the system's microcomputer nodes. Additionally, many COTS routers have comprehensive troubleshooting guides and technical support/customer service options for troubleshooting. A conceptual diagram of this networking approach is shown in Figure 13.

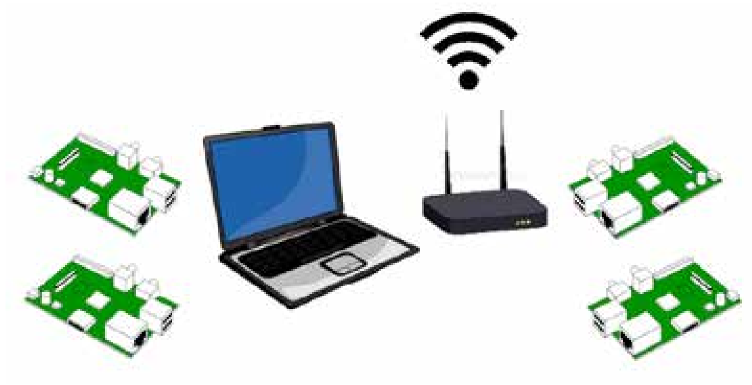


Figure 13. Networking Configuration with WLAN Through a COTS Router.

The disadvantages for this approach are similar those of the software router setup mentioned previously. For this configuration to work, the BlockGrid must be set up in an environment which permits the operation of non-facility WAPs. Further, if internet access is desired on the WLAN the local facility must permit operation of blockchain software. Additionally, there are a number of reasons why the WLAN may not be reliable such as signal interference, antenna issues, and operating system complications. All of these can be challenging for users to troubleshoot should they arise.

(4) WLAN via Broadband Hotspot

One networking option that can allow circumvention of local facility network prohibitions on blockchain software is to set up a WLAN through a broadband-connected wireless hotspot. This could be accomplished with a cellular phone, a broadband-connected computer, or a COTS broadband hotspot router. By securing internet access from a broadband provider, the system can avoid direct interaction with the facility's network infrastructure and the facility's IT-security controls. A conceptual diagram of this networking approach is shown in Figure 14.

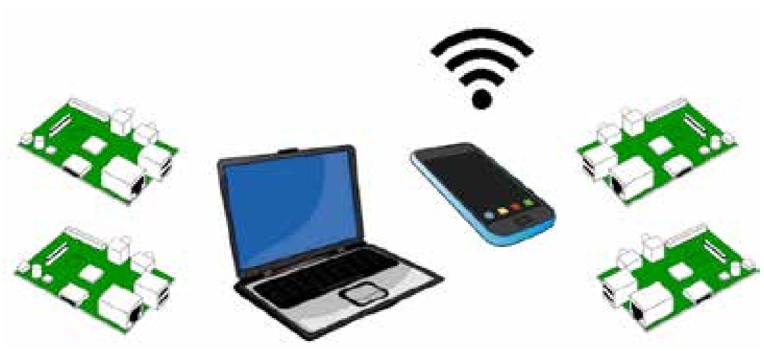


Figure 14. Networking Configuration with WLAN Through a Broadband Hotspot.

Though this approach may solve some issues, there are also several potential drawbacks. One issue is that the WAP may still be detected by network security tools as a rogue WAP and flagged as unauthorized. Some DOD facilities may prohibit any cell phone or broadband access technology altogether for security reasons. Network viability with a broadband hotspot is also dependent on the strength of the broadband signal at the demonstration site. Finally, the router configuration required with this approach may be technically challenging for users, particularly if the hotspot is being generated by a phone or broadband-enabled computer.

(5) Offline Internal Ethernet Network

The fifth BlockGrid networking configuration option is to create an isolated network by connecting all of the microcomputer nodes to a COTS router with Ethernet cables. If the BlockGrid system has more microcomputer nodes than the router has ports, an additional COTS Ethernet switch can be used. This configuration establishes an Ethernet LAN with no external internet access dependencies and no wireless networking features.

This networking approach has several advantages. Most notably, once assembled the Ethernet LAN is extremely reliable and fast. Despite requiring the most networking hardware to implement, this configuration is highly portable due to its lack of dependence on a pre-existing facility or ISP network. With a private Ethernet LAN, the BlockGrid can be deployed anywhere that the microcomputer nodes, router, and switch have access to a sufficient power source. With this isolated network there is no interaction required with local facility IT infrastructure and therefore no likely conflict with local facility security controls.

Another considerable advantage of this option is that the network only needs to be configured once since it is totally isolated. Once the router is configured with all of the appropriate static IP addresses for the microcomputer and SCU devices in the BlockGrid's network, the corresponding network settings for the blockchain can be pre-installed on the VM. This requires users to simply connect the SCU device and microcomputer nodes to the router vis Ethernet cables in order to run the BlockGrid. A conceptual diagram of this networking approach is shown in Figure 15.

Despite its numerous advantages, there are a few drawbacks to the isolated network approach. As previously mentioned, this approach introduces additional equipment requirements. A wired Ethernet network also physically constrains the position of the microcomputer nodes. This creates some significant constraints on possible smartgrid physical layouts. This type of network may also be more challenging for users to set up and troubleshoot as Ethernet networks are less commonly used today and could be unfamiliar to some users.

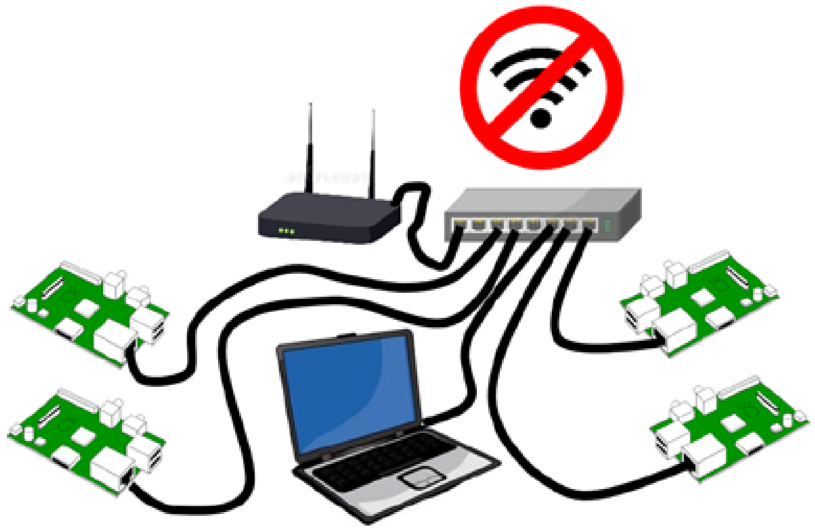


Figure 15. Networking Configuration with an Internal Ethernet Connection.

c. *Selection for Default Configuration*

The offline Ethernet network configuration illustrated in Figure 15 was selected for the BlockGrid prototype. As shown in Table 3, all five network design options were ranked against one another with 1 being the best and 5 being the worst on the following criteria; blockchain compatibility, smartgrid compatibility, setup difficulty, network reliability, troubleshooting difficulty, and cost. In this scoring system, the option with the lowest composite score is the best option. The offline internal Ethernet network had the lowest aggregate score of 14 and was chosen for the BlockGrid prototype’s implementation.

Although the Ethernet network option placed the most constraint on possible smartgrid designs and had the highest cost due to its additional network hardware requirements, it was by far the most user-friendly and compatible with blockchain software. Additionally, while the Ethernet network configuration is less portable in one sense due to its larger footprint, the fact that it allows the BlockGrid to be run on an isolated network means that it is arguably the most portable option because it can be set up in a wider variety of environments.

Table 3. Ranking of Networking Configuration Options

	FACILITY WLAN	WLAN WITH SOFTWARE ROUTER	WLAN WITH COTS ROUTER	WLAN VIA BROADBAND HOTSPOT	OFFLINE INTERNAL ETHERNET NETWORK
BLOCKCHAIN COMPATIBILITY	5	4	3	2	1
SMARTGRID COMPATIBILITY	1	4	3	2	5
SETUP DIFFICULTY	2	5	3	4	1
NETWORK RELIABILITY	3	4	2	5	1
TROUBLESHOOTING DIFFICULTY	3	5	2	4	1
COST	1	2	3	4	5
TOTAL SCORE	15	24	16	21	14

4. Smartgrid Circuits

The final hardware elements to be implemented for a BlockGrid are the smartgrid circuits that are powered by the system’s microcomputer nodes. These circuits create visible consequences to the smart contract transactions taking place on the blockchain. When chosen appropriately, these can clarify to observers what is happening on blockchain and how the system works. In order for the circuits to function in the BlockGrid, they must have several attributes.

a. Criteria

First, the circuits must physically interface well with the microcomputer nodes. This likely requires a physical connection to and a compatibility with the microcomputer’s power output. It is possible in some BlockGrid configurations that the circuits could be manipulated over wireless connections such as Bluetooth or WIFI networks, but a standard implementation such as that used in the BlockGrid prototype requires a hardwired connection. For example, the Raspberry Pi 3B, the microcomputer chosen for the BlockGrid prototype's implementation, only has 3.3V programmable output pins. The low voltage programmable output does not permit the Raspberry Pi to power circuits that require a high amount of current. However, if the 3.3V programmable output is used to control a current gate rather than directly power the external circuit, a power source with voltage to drive current through the desired circuit can be used.

Second, the circuit should have interesting visible elements such as lights or moving parts that clearly demonstrate to an observer that they have been activated. The circuits should also be user-friendly to operate. This requires that they be easy to set up, modify, and troubleshoot. The circuits also should also be as durable as possible as the BlockGrid is intended to be demonstration platform requiring frequent setup and disassembly as well as handling by potentially high number of inexperienced users. For the BlockGrid prototype's implementation, minimizing cost was also a high priority.

b. Available Choices

There are three principal types of circuits that can be considered for a BlockGrid implementation. These are a COTS electronic device, a customized fully soldered circuit, and a customized circuit implemented on a breadboard. Within each of these broad categories there are innumerable individual options. However, each category has its own advantages and disadvantages that make the optimal solution dependent on the use case.

(1) COTS Electronic Device

One circuit option is to connect the microcomputer nodes to COTS electronic devices. These devices could be as simple as a lightbulb or as complicated as an LED screen, a piece of an industrial control unit, or a model train set. If the BlockGrid controls electronic devices that observers are familiar with, the clarity of the blockchain technology demonstration can been enhanced. COTS circuits can also simplify BlockGrid operation for the user, as all that is required with the circuits on system setup is to plug the devices in. COTS circuits are also likely to be more user-friendly and durable, as their commercial design and manufacturing quality have already been optimized for positive user experience and durability.

There are several drawbacks to the COTS circuit option, however. One issue is that COTS devices tend to be much larger than the microcomputer nodes that control them, which can complicate the physical interface between them. These devices can also significantly increase the overall size of the demonstration apparatus. Another challenge is that most COTS devices require more voltage than the 3.3V that the Raspberry Pi 3B can directly provide. This necessitates a more complicated gating mechanism to control the circuits' power, which can introduce troubleshooting difficulties and negatively impact user experience.

(2) Customized Fully Soldered Circuit

An alternative to using COTS devices for the BlockGrid's circuits is to create a customized electronic circuit. This could be done using a kit or through prototyping with electronics components, printed circuit boards, and 3D printed materials. These approaches can generate unique circuits for desired smartgrid applications that are fully soldered and assembled.

The benefit of this customized approach is that it allows circuits to be designed exactly for the application envisioned by the operator. Well-designed custom circuits can maximize the uniqueness and visual effectiveness of the demonstration apparatus to observers. This approach also provides student users with an opportunity to gain hardware design skills.

The most significant downside to customized circuits is that they are very difficult to troubleshoot when not performing properly. Since the circuits are unique, they can be difficult to fix or replace if they fail. The challenges of the circuit design process can also introduce a level of complication and difficulty beyond what is constructive for a novice user. For users involved in the system design process such as administrators or students, the additional burden of circuit design may negatively impact user experience.

(3) Customized Circuit Implemented on Breadboard

A third option for the BlockGrid's circuits is to create customizable circuits on a breadboard that are powered by the microcomputer nodes. Such circuits can be designed with simple electronics parts such as LEDs, resistors, and electrical switches. With this approach, the user can quickly design and implement a smartgrid circuit.

The biggest upside to breadboard smartgrid circuits is that they give users maximum control of the BlockGrid during operation. With breadboards, users can make real-time adjustments to their circuits and easily diagnose and troubleshoot problems circuits that malfunction. The simple circuits can be implemented with breadboards in this approach also allow student users to engage in circuit design at a more beginning level. This approach is also very low cost, with the electronics parts being both inexpensive and re-usable. Taking this approach minimizes the difficulty in creating, maintaining, and replace the circuits for a BlockGrid.

The chief drawbacks to this approach are that the circuits are more fragile and consequently less portable. Once a breadboard circuit is set up, it can be very easily disturbed by environmental factors. This makes moving the circuits between locations without disruption them challenging and requires the BlockGrid system to be handled more delicately during use. The breadboard circuits also limit the scale and complexity of the possible smartgrid designs, which may reduce the impressiveness of the demonstration to some extent.

c. Selection for Default Configuration

For the BlockGrid prototype, the customized circuit implemented on a breadboard option was chosen. As shown in Table 4, the three circuit types were ranked against one another with 1 being the best and 3 being the worst on the following criteria; microcomputer interface, visibility for demo, setup difficulty, adaptability, troubleshooting difficulty, durability, and cost. In this scoring system, the option with the lowest aggregate score is the best choice. The customized circuit implemented on a breadboard option had the lowest composite score, 11, and was chosen for the BlockGrid prototype's implementation.

Although the breadboard circuits were more fragile than other choices, they allowed easy troubleshooting and modification. If LEDs or jumper cables became detached, they could easily be repositioned or replaced. The low cost of the breadboard option also allowed a wide range of smartgrid designs to be tested during the development of the prototype.

Table 4. Ranking of Smartgrid Circuit Options

	COTS ELECTRONIC DEVICE	CUSTOMIZED FULLY-SOLDERED CIRCUIT	CUSTOMIZED CIRCUIT IMPLEMENTED ON BREADBOARD
MICROCOMPUTER INTERFACE	3	2	1
VISIBILITY FOR DEMO	1	2	3
SETUP DIFFICULTY	3	2	1
ADAPTABILITY	2	3	1
TROUBLESHOOTING DIFFICULTY	2	3	1
DURABILITY	1	2	3
COST	3	2	1
TOTAL SCORE	15	16	11

Additionally, though the breadboards do not allow for construction of large high-visibility circuits, individual LEDs were found to work well for simple applications like demonstrating the BlockGrid prototype's SmartSwitch smart contract. LEDs can easily turn off and on or change colors to signify the result of smart contract transactions. And in the event that incorporation of a higher voltage COTS device into the smartgrid is desired, the breadboard circuits can be easily converted to switches that allow the BlockGrid's smart contracts to gate the power flow to the larger devices.

C. SYSTEM SOFTWARE

The BlockGrid's fundamental software system elements are the system's blockchain software, the SCU OS, and the OS of the microcomputer nodes. There are a wide variety of choices for each element, and each has its own potential benefits and drawbacks. This section describes the most common software options currently available and explains the rationale for the choices that were made for the BlockGrid prototype's implementation.

1. Blockchain Software

The most fundamental software component of a BlockGrid is the blockchain protocol. This implementation choice has implications for the other system software and hardware elements, as all other system components must be compatible with the blockchain software selected. In order for the BlockGrid to meet system requirements, the chosen blockchain protocol must exhibit several important attributes.

a. Criteria

First, the blockchain protocol must run sufficiently fast on the platform to enable real-time demonstration of smartgrid transactions. To maximize transaction speeds, the blockchain protocol should allow parameters such as consensus algorithm difficulty to be changed. Additionally, as the BlockGrid system should be adaptable, the blockchain protocol should scale well to accommodate inclusion of more nodes. Finally, the protocol should support deployment of smart contracts and decentralized applications (DApps).

b. Available Choices

As the number of applications for blockchain technology has continued to grow, so too have the number of blockchain protocols. Increasingly, blockchain software is being designed for specific applications such as micropayments, IoT networks, and large-scale DApps. Consequently, each blockchain protocol option has its own benefits and drawbacks. The following are the most common protocols available at the time of this writing.

(1) Bitcoin

As the most established cryptocurrency in the world, Bitcoin is one of the most widely used blockchain protocols. To achieve consensus in the network, mining nodes validate blocks of transactions using the “Hashcash” Proof of Work algorithm. In addition to its primary cryptocurrency function, Bitcoin supports smart contracts written in its own programming language, Script [246].

Using Bitcoin as the blockchain protocol for the BlockGrid has several potential advantages. The protocol's primary utility as a cryptocurrency makes it well suited to a BlockGrid implementation with a transaction-based smartgrid. Permissions in the system could be granted with Bitcoin payments, as has already been envisioned on a larger scale [247]. Due to its very large developer base Bitcoin also has extensive developer support materials, which could ease implementation and troubleshooting of the system [248]–[250].

The biggest drawback to using Bitcoin is that it wasn't designed to support smart contracts. Bitcoin's chief programming language, Script, is not Turing-complete and does not allow for storage of system state [251]. Consequently, programming complex smart contacts for a smartgrid system with Bitcoin would be difficult. Additionally, Bitcoin faces throughput issues when scaling due to block size increases [252]. It might therefore be challenging to keep block commit times consistently low for a BlockGrid system.

(2) Ethereum

Ethereum is perhaps the most widely used blockchain protocol for smart contract, DApp, and custom coin deployment. The protocol currently runs its own custom "Ethash" Proof of Work consensus algorithm. There are several Ethereum blockchain interface clients available. Smart contracts for Ethereum are commonly written in the Solidity language, but can also be written in several other languages [253]. The protocol also has a JavaScript API, Web3, which can be called by locally running programs to query smart contract transaction results [254].

There are several advantages to using Ethereum as the blockchain protocol for a BlockGrid. The biggest is that Ethereum is designed specifically to support smart contracts. Solidity is a Turing-complete language, so any smart contract can be programmed for the Ethereum blockchain [255]. As Ethereum is so widely used, there are also extensive user support materials available [256]–[258]. There are a wide variety of example smart contracts and smart contract tutorials that can be used to guide beginners [259], [260]. There are also tutorials available for implementing Ethereum on a private network [261],

[262]. Additionally, on a private network it is possible to set the consensus algorithm difficulty to a desired level to optimize transaction speeds [263].

The biggest drawback to Ethereum is the difficulty in making the block commit times consistent and low enough to run a smartgrid. As the protocol currently runs a Proof of Work consensus algorithm, the speed of transaction validation is dependent on the computational power of the mining nodes on the network. Additionally, as the size of the distributed ledger grows, more time is required to update the nodes in the network. It might therefore be challenging to run an Ethereum blockchain on which block commit times were consistently and sufficiently low enough to be effective on a real-world smartgrid application. Another issue is that while it is possible to run a private Ethereum blockchain, the common blockchain interface software for Ethereum is designed to interface with the main public Ethereum blockchain. Consequently, it can be challenging to run some Ethereum interface software on a private chain, particularly if the system has no internet access.

(3) HydraChain

HydraChain is a blockchain protocol based on Ethereum that is focused on enabling development with permissioned private chains. HydraChain runs a Practical Byzantine Fault Tolerance (PBFT) consensus algorithm, as opposed to Ethereum's Proof of Work Ethash algorithm. It has full compatibility with Ethereum's development and interface tools. The protocol is also focused on smart contract development in Python [264].

There are several reasons that HydraChain would make a good choice for the BlockGrid's blockchain protocol. First, HydraChain's focus on permissioned private blockchains provides more development support for implementing a private blockchain on a BlockGrid system [265]. The PBFT consensus algorithm may also be better suited to run the BlockGrid's smartgrid than a Proof of Work protocol, as it is faster and would require less computational power to run on microcomputer nodes [266]. Additionally, the protocol's support of smart contracts written in Python might put less of a burden on students tasked with writing their own smart contracts than blockchain software that requires learning a new programming language.

The main drawback to using HydraChain is that it is not yet widely used. As such there are very few user support resources or troubleshooting guides available for the protocol. Consequently, it may be challenging for beginners to successfully implement a BlockGrid with this software, even though upon implementation this protocol offers significant performance and development benefits.

(4) Qtum

Qtum is a blockchain oriented toward business applications that is compatible with both the Bitcoin and Ethereum blockchains. The protocol runs Bitcoin's blockchain software with a Proof of Stake consensus algorithm along with the Ethereum Virtual Machine (EVM) to support smart contracts [267]. A Qtum x86 VM is currently being developed that will enable smart contract development in any language with an x86 compiler, but at the time of this writing has not been released [268]. Presently, Qtum smart contracts are mostly written in the Solidity programming language, although they can also be implemented with Bitcoin's Script language.

There are several potential advantages to running Qtum on a BlockGrid. The most significant advantage is that due to its use of Bitcoin and Ethereum software, a Qtum blockchain can run existing smart contracts for those platforms while also benefitting from the potentially faster performance of a Proof of Stake protocol. Qtum also offers lightweight mobile device-friendly chain interface software that could work well for an IoT-based BlockGrid [269]. Additionally, as Qtum combines the two most widely used blockchain technologies with an increasingly popular alternative to achieving distributed consensus via Proof of Work, its use provides an educational opportunity to cover the salient differences between blockchain protocols and consensus algorithms in the classroom.

The largest downside to using Qtum is that there are comparatively few user support resources available. It is still a fairly new platform, and there are not many guidelines or tutorials available that could help beginners implement the protocol on a BlockGrid. Additionally, as the protocol is new there may be major updates to the protocol before it becomes fully mature, and these might introduce backwards compatibility issues or require inconvenient updates to a BlockGrid.

(5) Hyperledger Fabric

Hyperledger Fabric is a versatile open source blockchain platform that specializes in the development of applications for permissioned blockchains [270]. As one of the five Hyperledger Business Blockchain Technologies, it is distinguished by its modularity. With Hyperledger Fabric developers can choose their desired consensus and permissioning mechanisms [271]. Smart contracts for Hyperledger Fabric can be written in Go, JavaScript, or Java [272].

There are several reasons why Hyperledger Fabric would make a good choice for a BlockGrid's blockchain. First, the system offers choice of consensus algorithm, which allows choice of a faster option such as Proof of Stake or PBFT to optimize block commit times for the system [273], [274]. The protocol's focus on supporting isolated permissioned chains also makes it easier to implement for an isolated BlockGrid system [275]. Additionally, Hyperledger Fabric's modularity allows a system to be modified more easily after implementation to accommodate changes to a BlockGrid's configuration.

The most significant drawback to using Hyperledger Fabric is that it is still a fairly new platform. As such there are not many user support resources available and troubleshooting may be challenging for beginning students. Additionally, as the blockchain software is coinless, there is no built-in mechanism for smartgrid transactions or permissions. Consequently, the smart contracts for the system may need to be more comprehensive to achieve the desired smartgrid functionality.

(6) Corda

Corda is an open-source blockchain protocol focused on supporting financial services and enabling blockchain-based financial contracts. Like Hyperledger Fabric, it is also focused on supporting permissioned blockchains. Corda is distinguished by its point-to-point architecture, in which transaction participants only store and view transaction records for the transactions they have participated in to create a multilateral ledger within the network. The protocol allows developers to choose the consensus algorithm for their network. Additionally, Corda offers the ability to run multiple “notary pools,” or subnetworks, within a network that each run their own consensus algorithm. Corda runs on the Java Virtual Machine and smart contracts are written in Java or Kotlin programming languages [276].

There are several reasons that Corda would be a good choice as the blockchain for a BlockGrid system. With its multilateral ledger, Corda minimizes the size of the blockchain that the individual nodes in the system have to update. This combined with the choice of a fast consensus algorithm like PBFT could minimize smart contract transaction times and improve smartgrid performance [277]. Programming smart contracts in Java may also be easier for students than having to learn a new language.

There are several disadvantages, however, to selecting Corda as the blockchain protocol for a BlockGrid. One issue is that Corda’s smart contract framework is designed to support industry financial transactions, and as such includes legal prose and contract formatting that is unnecessary for most BlockGrid smart contracts and may be confusing for students. Another issue is that the notary pool and point-to-point features of Corda network architecture require more work to configure and also require more intricate smart contracts, increasing the difficulty of implementation. Finally, Corda does not have many user support resources for small scale private chain implementation.

(7) Openchain

Openchain is a blockchain protocol that is designed to maximize scalability and support digital asset management. With the Openchain protocol, an administrator approves network nodes, or “accounts,” to form a “closed-loop” ledger within the administrator’s permissioned account. The protocol allows for hierarchical relationships between accounts that create a “ledger tree” that resembles an OS directory hierarchy in structure. Openchain uses a miner-less partitioned consensus mechanism that assigns only one validation authority, or “server,” to each node on an account. Servers can insert pointers within their ledgers to Bitcoin’s blockchain, also described as “pegging” the blockchain to Bitcoin, to achieve immutability. Smart contracts within the Openchain framework therefore function like their own server accounts and execute logic in response to payments. Smart contracts for the protocol can be programmed in any language [278].

There are several upsides to using Openchain as the blockchain software for a BlockGrid. The biggest advantage to using Openchain is that transactions are completed quickly and can trigger smart contract execution with minimal latency. This could allow for smart contract management of a smartgrid in close to real time. Another significant advantage of Openchain is that it is very fast and straightforward to implement accounts. Openchain provides Git repositories with preconfigured software, allowing network setup with a few simple downloads. Additionally, Openchain's software is sufficiently lightweight that it can be run in a Docker container [279]. While running a node in a Docker container has an associated computational overhead, far less RAM, CPU cycles, and hard drive space are required for Docker containers compared to a VM running a full operating system. This could allow microcomputer nodes to allocate more computation resources toward more advanced smartgrid functionality. The ability within the protocol to hierarchically organize different accounts could also allow for more complex smartgrid implementations in which nodes have different and overlapping subgrids.

The biggest downside to using Openchain as the blockchain protocol for a BlockGrid is that there are minimal user support resources for smart contract implementation. Since smart contracts in the Openchain framework apply only to those accounts that have been permissioned to access the contract, the contracts themselves are essentially local scripts that interface to the permissioned accounts via the Openchain API. Consequently, Openchain does not provide any smart contract development guidelines. Another result of this feature of the Openchain framework is that the smart contracts are not really decentralized. If the node upon which a smart contract runs is taken offline, the smart contract is no longer available to the accounts in the system that have been permissioned to access it.

(8) BigchainDB

BigchainDB (DB stands for Data Base) is an open-source blockchain technology that combines the features of blockchains and conventional database software. BigchainDB uses the Proof of Stake Tendermint blockchain protocol to generate a network of linked conventional database nodes [280]. Each node in the network runs its own MongoDB instance, giving the network as a whole the query and data organization features of a conventional database along with the decentralization, immutability, and fault tolerance of a blockchain network [281]. The protocol emphasizes low transaction latency, lots of permissioning options, and easy private network configuration. BigchainDB can store smart contract code but does not directly execute smart contracts. Smart contracts must be invoked by locally running scripts that query and then execute the stored code [282].

There are several reasons that BigchainDB might be useful to implement on a BlockGrid. First, the protocol permits easy setup of a private BigchainDB network [283]. BigchainDB allows users considerable freedom to define digital assets that are stored on the chain, which might be useful for research efforts aimed at exploring what types of information blockchains can hold. The protocol's Proof of Stake consensus mechanism could also lower transaction and data update times and thereby improve BlockGrid system performance. Finally, the robust database query functionality of BigchainDB might make it especially well-suited to efforts to model and demonstrate the application of blockchain technology to supply chain logistics in the classroom.

The most significant drawback of BigchainDB is its lack of smart contract support. While it is possible to implement smart contract functionality in a BigchainDB system, this is much more difficult to do with BigchainDB than it is for other blockchain protocols. Consequently, while BigchainDB may be very well-suited for digital asset management and supply chain applications, it is not particularly well suited for smart contract-mediated cyber-physical systems. If adopted for a BlockGrid, a beginning user might need to forgo smart contracts altogether and instead utilize local query scripts to activate smartgrid circuits rather than code stored on the chain.

(9) IOTA

IOTA is a distributed ledger technology (DLT) that utilizes a "tangle" architecture rather than a blockchain. Within the tangle architecture, system nodes are connected to one another as a directed acyclic graph (DAG) [284]. Distributed consensus is achieved in this protocol by having each node validate two previous transactions via a low-difficulty "Hashcash" Proof of Work algorithm prior to initiating transactions of their own. IOTA has recently begun supporting smart contracts, termed "Qubics," to enable computation using data acquired from external sources, termed "Oracles," as well as to enable nodes in the tangle to perform outsourced computation. Qubics are programmed in the Abra language and are run in duplicate by an "assembly" of validators so that the results confirmed via a quorum consensus mechanism that requires agreement among a minimum of two thirds of the validators. IOTA is distinguished by its scalability and low transaction times and is specialized for IoT implementations [285].

There are several potential upsides to utilizing IOTA as the DLT protocol for a BlockGrid. As IOTA is specialized to support IoT networks with its low latency architecture and minimal Proof of Work consensus mechanism, it could be an excellent choice for BlockGrid systems that feature IOT devices. IOTA can also be run in Dockers, which could maximize the amount of microcomputer node resources devoted to smartgrid functionality. As an avant-garde DLT architecture, IOTA also offers an opportunity to explore DLT alternatives to blockchain technology in the classroom and could be used in demonstrations to illustrate performance differences from conventional blockchain protocols.

The biggest drawback to IOTA is that it does not currently support private tangles. Consequently, internet access would be required to run IOTA on a BlockGrid system. In this case the Qubics for a BlockGrid would likely need to be implemented on the Testnet tangle that IOTA provides [286]. Another potential issue is that while IOTA's tangle architecture is scalable due to the increases in transaction speed as more nodes join the tangle, running a small number of nodes within a tangle may not yield good results.

(10) Lisk

Lisk is a new blockchain technology focused on enabling DApp development. The protocol uses a delegated Proof of Stake (DPOS) consensus algorithm in which a pool of 101 elected nodes validate blocks using a "broadhash" algorithm [287]. Owners of the Lisk mainchain currency LSK, i.e. "stake holders," vote to elect the validation nodes. Their vote has a value proportional to the amount of LSK currency in their account. The Lisk protocol is set up for programmers to develop and deploy DApps with permissioned sidechains. With Lisk the front-end of DApps can be programmed in any language, and DApp back-ends are programmed in JavaScript [288].

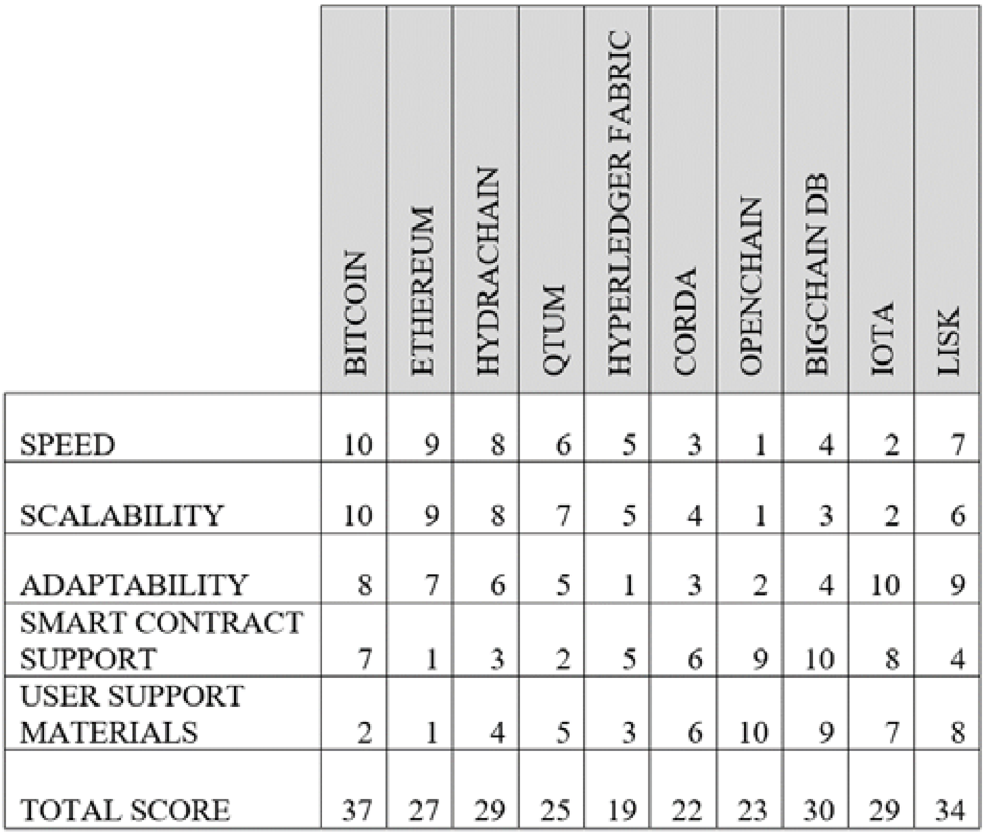
There are several advantages to using Lisk as the blockchain protocol for a BlockGrid. One significant advantage is due to its central focus on DApp development, the protocol is designed for rapid and straightforward setup of permissioned sidechains. This could allow a permissioned sidechain to be quickly implemented as the BlockGrid's network. Lisk also offers a suite of tools for DApp development which would be useful for a BlockGrid featuring DApps [289]. DApps could allow, for example, smartgrid functions to be executed from a web app. Another advantage to Lisk is that the back-end programming is done in JavaScript, a widely used language for web app development, which could save BlockGrid users from having to learn a new protocol-specific programming language. Additionally, the speed of the DPOS consensus algorithm could improve BlockGrid transaction times and system performance.

There are, however, a few disadvantages to running Lisk. One disadvantage is that implementing a fully functional DApp on a BlockGrid requires that the BlockGrid have internet access. Another issue is that is that at the time of this writing Lisk is not a fully mature platform. The protocol underwent a relaunch in 2018 and anticipated updates and upgrades to the Lisk software suite could cause backward compatibility issues and require inconvenient re-installs if implemented today on a BlockGrid. Moreover, the Lisk platform software development kit (SDK), which should simplify DApp development, has not been released at the time of this writing. There is also a dearth of user-support materials available. Lisk does have a dedicated documentation and academy section on their website, but the resources presently available in those sections are minimal [290].

c. Selection for Default Configuration

Ethereum was chosen as the blockchain protocol for the BlockGrid prototype. In Table 5 the ten blockchain protocols surveyed in this section are ranked against one another on the categories of speed, scalability, adaptability, smart contract support, and user support materials with 1 being the best score and 10 being the worst. Each protocol is given a composite score that is the sum of their rankings for each category, making the protocol with the lowest total score the “best” option.

Table 5. Ranking of Blockchain Protocol Options



	BITCOIN	ETHEREUM	HYDRACHAIN	QTUM	HYPERLEDGER FABRIC	CORDA	OPENCHAIN	BIGCHAIN DB	IOTA	LISK
SPEED	10	9	8	6	5	3	1	4	2	7
SCALABILITY	10	9	8	7	5	4	1	3	2	6
ADAPTABILITY	8	7	6	5	1	3	2	4	10	9
SMART CONTRACT SUPPORT	7	1	3	2	5	6	9	10	8	4
USER SUPPORT MATERIALS	2	1	4	5	3	6	10	9	7	8
TOTAL SCORE	37	27	29	25	19	22	23	30	29	34

With this scoring methodology, Ethereum only ranked as the 5th best protocol overall. However, Ethereum has, at present, the best smart contract support and user support materials by a significant margin. With Ethereum there many resources available that guide beginners through writing, compiling, and operating smart contracts [291]–[293]. There are also a significant number of companies that currently utilize smart contracts that run on the Ethereum mainchain [294]–[296]. These examples provide opportunities to explore real world applications of blockchain technology. Consequently, Ethereum is presently the best suited protocol for use in a classroom environment.

An additional benefit to using Ethereum is that there are tutorials available for how to implement a private Ethereum network on Raspberry Pi 3B microcomputers [297]. The Chainskills tutorials by Eloudrhiri also provide sample smart contracts and chain monitoring scripts [298]. These tutorials and software were heavily utilized in the implementation of the BlockGrid prototype.

2. System Control Unit OS

The second software element to implement for a BlockGrid is the OS for the SCU. The choice in OS can have implications for compatibility with software running on the system and the overall user experience. In order for a BlockGrid to function, the SCU OS must meet several criteria.

a. Criteria

For the SCU OS to be suitable, it must be compatible with the chosen blockchain protocol. The OS must also be compatible with any blockchain interfacing software such as blockchain wallets and smart contract compilers. Additionally, the OS should be easy to set up, operate, and troubleshoot.

b. Available Choices

With the proliferation of VM technology, it is easier than ever to select an OS with the attributes ideally suited to a particular application. Many operating systems today come preinstalled with software for specific tasks such as penetration testing or server hosting [299], [300]. There are benefits and drawbacks to using a given OS for a BlockGrid SCU. The following are the most common OS options that could be run on an SCU for a BlockGrid.

(1) Windows 10 VM

One option is to run a BlockGrid's SCU from a Windows 10 VM. While many users could configure a SCU on their own Windows 10 laptop, using a Windows 10 VM would allow administrators to preconfigure all system software and networking parameters on a disc image and distribute an identical SCU environment to users. This approach also provides the same OS environment for users who have laptops that natively run MAC (Macintosh) or Linux operating systems. Microsoft currently offers a 20 GB Windows 10 Enterprise disc image for developers with a free 90-day license [301]. A Windows 10 disc image can also be created with an active Windows license. Presently the Windows 10 Professional license costs approximately $200 [302].

Running a Windows 10 VM for the BlockGrid's SCU could have several benefits. As Windows is a widely used laptop OS, the OS environment would be familiar to a large percentage of users. Since most available blockchain software has been widely run on Windows, a Windows 10 SCU would also allow users to run some of the most user-tested versions of blockchain software. This could have positive effects for user experience. Additionally, having a Windows 10 SCU offers the opportunity to have more streamlined interfacing between all of the nodes in the BlockGrid if partnered with Windows IoT-Core for the OS of the system's microcomputer nodes. This could be especially helpful for a BlockGrid focused on DApp development or a BlockGrid that incorporates IoT devices in its smartgrid.

However, there are several disadvantages to running a Windows 10 VM as the OS for a BlockGrid SCU. The most notable drawback is the potential cost of the OS license compared to the free Linux alternatives. Avoiding the Microsoft license fee by using the free image offered by Microsoft for developers requires re-downloading and reconfiguring a Windows 10 Enterprise VM every 90 days. Also, Windows 10 is not a lightweight operating system so VM performance could be an issue.

(2) Linux VM

Another option is to run the BlockGrid's SCU through a Linux VM on the user's host computer. Linux operating systems are more commonly used to support virtualized applications and may therefore result in a more user-friendly virtualization experience than the Windows VM option. There are several different distributions of Linux that could be considered. Discussion of specific Linux variants follow.

(3) Kali Linux

Kali Linux is a Debian-based Linux distribution designed for penetration testing, digital forensics, and reverse engineering. Presently, it runs Linux Kernel v. 4.14. It is distinguished by its “single, root user” implementation in which the default user has root privileges. The OS comes packaged with a suite of 600 cybersecurity software tools. Kali also has a noteworthy focus on supporting wireless interfaces [303].

Using Kali Linux as the SCU OS might be useful for BlockGrid applications that are cybersecurity focused. For example, if the BlockGrid’s microcomputer nodes ran Kali ARM, it would be possible to use the Kali framework to test blockchain security on any node within the BlockGrid system. Additionally, Kali comes with the software required to create a software defined router and WAP should a BlockGrid need to deploy those features in its network implementation. Additionally, the default root user privileges in Kali could make BlockGrid installation and network management slightly less cumbersome for users.

The most significant disadvantage to running Kali as the SCU OS is that the OS is not widely used outside of the cybersecurity community. It may therefore be unfamiliar to beginning students and introduce an additional learning curve that negatively impacts the user experience. Additionally, few Linux OS-focused tutorials and troubleshooting guides for blockchain applications focus on Kali distributions.

(4) CentOS

CentOS is a free open-source Linux OS modeled from the Red Hat Enterprise Linux distribution. It comes installed with Linux Kernel version 4.11. The OS comes with a robust firewall and a suite of server software. CentOS is noted for its stability and it has a fairly long release cycle. It is widely used for enterprise and web hosting applications [304].

There are several potential benefits to running CentOS as the SCU OS for a BlockGrid. The features of the OS focused on server hosting could be advantageous for a BlockGrid running DApps that require a web server. The OS could also work well for administrators as its stability and long release cycle require comparatively few updates. Additionally, the enterprise-focused features of the OS such as its firewall might work well for a larger BlockGrid that incorporates subnets or a BlockGrid that needs to interface with other IT infrastructure.

There are several likely drawbacks, however, to using CentOS as the SCU OS for a BlockGrid. One disadvantage is that due to its long release cycle, CentOS can experience software compatibility issues when applications update for compatibility with more recent Linux OS versions. Another potential issue is that CentOS is a less commonly used Linux distribution. As with Kali Linux, it may introduce an additional learning curve for beginning students that negatively impacts user experience. Most Linux OS tutorials and troubleshooting guides are also not focused on CentOS distros.

(5) Ubuntu 18.04

Ubuntu version 18.04, aka Bionic Beaver, is the most current version of the Debian-based general-purpose Ubuntu Linux OS. It runs Linux kernel v 4.15. The OS is noteworthy for its security features, support of NVIDIA GPUs, and Bluetooth functionality. It also comes with OpenJDK installed. Ubuntu 18.04 only is only available in a 64-bit version and can only run 64 bit-compatible software [305].

There are several reasons why using Ubuntu 18.04 as the SCU OS could be beneficial. One advantage is that Ubuntu 18.04's 2018 release coincided with the 2018 release of many blockchain protocols, and much of the current blockchain software designed for Linux operating systems has documentation focused on Ubuntu 18.04. For this reason, Ubuntu 18.04 might be a good choice for a BlockGrid running recently released blockchain software. Ubuntu 18.04 may also be preferable for BlockGrid administrators because the OS will be supported until 2023. Additionally, Ubuntu is a widely used Linux distribution and may be more familiar to users, possibly lessening the learning curve required to operate the BlockGrid.

The most significant disadvantage to the Ubuntu 18.04 OS is that as the most recent Ubuntu distribution it is not yet widely adopted. Consequently, while much of the official developer-generated documentation for new blockchain software focuses on Ubuntu 18.04, most of the user community-generated tutorials and support forums that exist for blockchain software implementation address older versions of blockchain software and earlier Linux distributions. Ubuntu 18.04 is therefore less likely to be familiar to most users than an previous and more widely disseminated Ubuntu versions.

(6) Ubuntu 16.04

Ubuntu 16.04, also known as Xenial Xerus, is a widely used version of the Ubuntu OS that was released in 2016. It runs Linux Kernel 4.4. The OS also includes the “Snap” universal package manager for application downloads. Additionally, Ubuntu 16.04 comes with a large suite of desktop software [306].

The best attribute of the Ubuntu 16.04 OS is that it is very widely used. As a result, there are extensive tutorials and user support resources that make use of this Linux distribution. There are numerous resources for Linux implementations of Ethereum blockchain software hosted on this version of Ubuntu [307], [308]. Ubuntu 16.04 is also likely to be familiar to users and may reduce the learning curve for installing and operating the system.

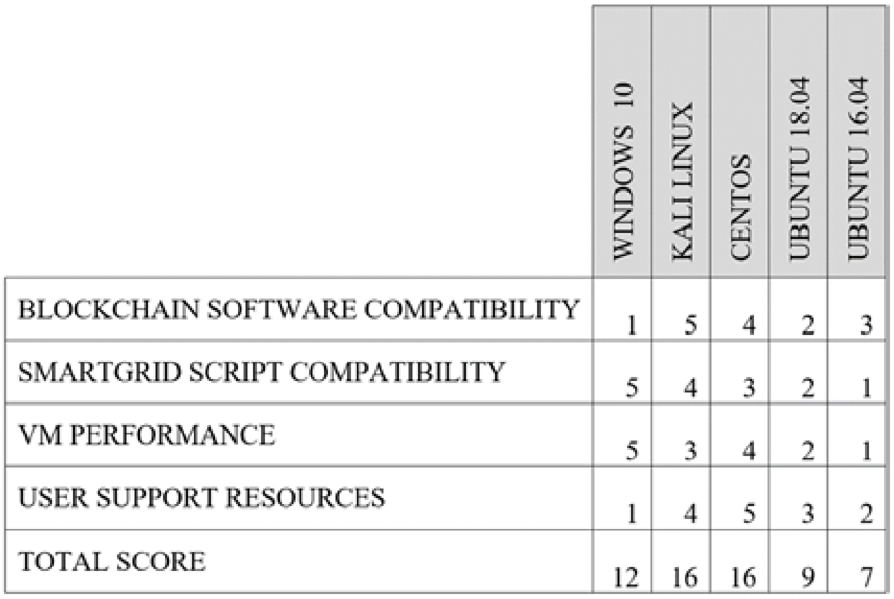
The most significant disadvantage of Ubuntu 16.04 is it isn’t the most current version of Ubuntu. Consequently, newly released user support resources for blockchain protocols, especially the newest blockchain protocols, are likely to focus on the more current versions of the Ubuntu OS moving forward. Additionally, Ubuntu 16.04 is only scheduled to receive maintenance updates until 2021 and no longer receives hardware updates, potentially requiring an administrator to install a new SCU OS before long.

c. Selection for Default Configuration

Ubuntu 16.04 VM was selected as the SCU OS for the BlockGrid prototype. As shown in Table 6, the operating systems were ranked against one another on the categories of blockchain software compatibility, smartgrid script compatibility, VM performance, and user support resources with 1 being the best and 5 being the worst. In this scoring system, the OS with the lowest total score is the best option.

Using this scoring system, Ubuntu 16.04 had the best score of all five operating systems. Although the scores were close between the two Ubuntu operating systems, Ubuntu 16.04 had significantly better user support resources. In particular, the user resources for implementing Ethereum blockchain software on Ubuntu 16.04 were much better than they were for Ubuntu 18.04. As a result, Ubuntu 16.04 was chosen.

Table 6. Comparison of SCU Operating Systems



	WINDOWS 10	KALI LINUX	CENTOS	UBUNTU 18.04	UBUNTU 16.04
BLOCKCHAIN SOFTWARE COMPATIBILITY	1	5	4	2	3
SMARTGRID SCRIPT COMPATIBILITY	5	4	3	2	1
VM PERFORMANCE	5	3	4	2	1
USER SUPPORT RESOURCES	1	4	5	3	2
TOTAL SCORE	12	16	16	9	7

3. System Node Microcomputer OS

The last software element for a BlockGrid is the OS for the microcomputer nodes. As the number of microcomputers and IoT devices available have continued to grow, operating systems have been created with characteristics suited to diverse applications such as programming electronics, embedded systems, multimedia playback, and IoT networks. In order for a microcomputer OS to work for a BlockGrid, it must meet several criteria.

a. Available Choices

There are a number of OS options available for microcomputers. Each has advantages and disadvantages when applied to a BlockGrid. The optimal microcomputer OS choice likely depends on the type of smartgrid functionality that a BlockGrid architect seeks to achieve. The following are several OS options with unique features that may be useful for different types of BlockGrid implementations.

b. Criteria

First, the OS must be compatible with the blockchain protocol for the system. The OS should also be compatible with the smartgrid functionality desired for the microcomputer and the locally run scripts necessary to support the smartgrid. To optimize the user experience, the OS should also be stable, easy to operate, and have sufficient user support resources.

(1) Android

Android is the most widely used OS in the world due to the high volume of Android mobile devices in use [309]. There are several versions of the OS available that might be useful for devices in a BlockGrid. The most current version of the OS for general purpose mobile devices like tablets and phones is Android Oreo. There are also Android OS versions for other device categories such as Android Wear OS for wearables and Android Things for IoT devices. All of these Android versions have APIs that allow them to run Android software [310].

Running Android OS on the nodes of a BlockGrid has several potential advantages. As many mobile devices natively run Android OS, using the OS for all system nodes could help achieve uniformity for a BlockGrid that incorporates COTS mobile devices in an IoT network. Android could also be a useful OS for a BlockGrid running DApps. Another advantage is that Android development is done in Java, which is a widely known language. Presently, Android cannot support a full node implementation of Ethereum, but a mobile app wallet could run in an Android OS and monitor smart contract transactions.

There are several disadvantages, however, to using Android OS. The most significant drawback is that it is not possible to implement a full node for most blockchain protocols on an Android device due to both the comparatively low processing power and memory for most mobile devices, as well as the lack of blockchain software designed for Android OS. This likely significantly limits the ability of a BlockGrid with Android microcomputers to fully explore and demonstrate features of blockchain technology. Another issue is that Android devices have limited ability to run software designed for desktop systems. This in turn likely limits the type of smartgrid functionality that microcomputers running Android can support. The OS may also be unfamiliar to users on non-mobile devices, potentially steepening a BlockGrid's learning curve.

(2) Windows IoT Core

Windows IoT Core is a lightweight implementation of Microsoft Windows 10 for embedded systems and IoT devices. It comes with a suite of IoT development software and specializes in supporting programming access to hardware. Apps in Windows IT Core are written in C# and can run on any Windows 10 device [311].

Running IoT Core on the BlockGrid's microcomputer nodes could have several advantages. For a BlockGrid running a Windows 10 OS on the SCU, running IoT Core would allow seamless interfacing and networking of the BlockGrid's nodes [312]. The IoT development software that comes with IoT Core could also be useful for programming smartgrid functionality. Windows may also be a more familiar OS environment for students.

The biggest downside of running Windows IoT Core is that it does not support full node implementation of most blockchain protocol software. Consequently, microcomputer node devices on the BlockGrid would need to run blockchain interface software to monitor and react to blockchain transactions. This would likely complicate operation of the BlockGrid and could negatively impact user experience with the platform.

(3) Armbian

Armbian is a minimalist Debian-based Linux distribution optimized for ARM board applications. It is distinguished by its modularity and shares a code base with the Ubuntu OS. Armbian offers a suite of OS customization and developer tools to enable customization of the OS for specific microcomputer applications. There are customized versions of the OS available for many common microcomputers [313].

There are several potential advantages to using Armbian as the microcomputer OS for a BlockGrid. First, Armbian's minimal size could improve speed of operation and system performance for a BlockGrid. For a BlockGrid running an Ubuntu OS for the SCU, running the Ubuntu-based Armbian OS on the microcomputer nodes might help with interfacing and software portability within the system. For a BlockGrid utilizing non-Raspberry Pi microcomputers it may be beneficial to run the customized Armbian OS offered for that specific device. Armbian might also be a good choice for a BlockGrid that incorporates nodes with customized hardware, as the OS could then be customized for those devices.

The largest drawback to using Armbian is that the OS is best used by experienced developers. Most user support resources are targeted toward experts seeking to customize the OS rather than beginners trying to set up systems for the first time. This may make the OS less ideal for use with inexperienced students.

(4) Moebius

Moebius is a lightweight Debian-based Linux distribution for the Raspberry Pi. It requires only 128MB to install and only 20 MB of RAM to run. In its default state the OS has no GUI and can only be controlled from the command line [314].

The most significant advantage to using Moebius as the microcomputer OS for a BlockGrid is how few computational resources it requires. This small size could make the OS well suited for IoT devices. Users may also find it familiar to use as a GUI-less OS because the OS is Debian-based.

The chief drawback to using Moebius is that it's not widely used. There are minimal user support resources. The documentation that does exist is aimed at experienced developers and does not provide tutorials for beginners. Additionally, students used to working with a GUI-based OS may also find it challenging to work entirely from the command line.

(5) Raspbian Stretch

Raspbian Stretch is a Debian-based Linux distribution that is the standard OS for the Raspberry Pi devices. It comes with a suite of software to support use of the Raspberry Pi as a fully functional computer. The OS can be run from an SD Card, onboard memory, an external disk, and over an Ethernet connection [315].

The biggest advantage to using Raspbian as the OS for the microcomputer nodes of a BlockGrid is that there are lots of user support resources [316], [317]. Due to the widespread use of the Raspberry Pi devices and the abundance of associated documentation, this operating system is comparatively user-friendly. The troubleshooting resources available for Raspbian are the most robust for any ARM operating system. There are physical computing libraries focused on Raspbian that enable the user to interface smart grid circuits more easily than with any other OS [318]. There are also numerous helpful Raspbian-focused blockchain tutorials [319], [320].

However, there are some disadvantages to the Raspbian OS. The biggest drawback is that Raspbian is not as modular as some of its alternatives. The OS also comes with a considerable amount of bloatware that is not likely to be useful for a BlockGrid node microcomputer [321]. Raspbian may therefore not be ideal for advanced smartgrid implementations or for use with experienced students.

(6) Kali ARM

Kali ARM is a version of the popular Linux penetration testing distribution Kali Linux for ARM processors. Like its pc counterpart, Kali Linux is a Debian-based Linux distribution designed for penetration testing, digital forensics, and reverse engineering. Presently, it runs Linux Kernel v. 4.14. Kali Linux comes with a suite of 600 cybersecurity software tools. The OS has a focus on supporting wireless interfaces. It is also a “single, root user” implementation in which the default user has root privileges [322].

Kali ARM could make a good choice as the microcomputer OS for a BlockGrid set up to explore blockchain cybersecurity. For a BlockGrid running Kali Linux as the SCU OS, the choice of Kali ARM for the microcomputer nodes would allow the Kali tool suite to be used for penetration testing from every node on the network. The preinstalled software tools for setting up WAPs could also be leveraged to support complex smartgrid implementations with subnets.

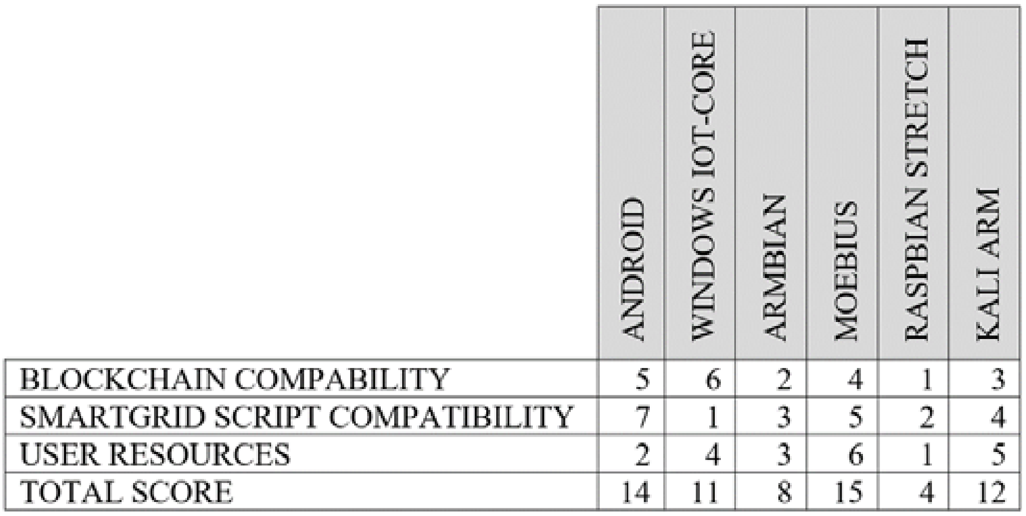
The chief disadvantage of running Kali ARM is that it is not widely implemented. Consequently, the user support resources are not well developed. The OS is also not familiar to many users and could introduce a steep learning curve for beginning students.

c. *Selection for BlockGrid Prototype*

Raspbian Stretch was the microcomputer OS selected for the BlockGrid prototype implementation. As shown in Table 7, the six OS options were graded against one another in the categories of blockchain compatibility, smartgrid script compatibility, and user resources with 1 being the best score and 6 being the worst. In this scoring system, the OS with the lowest total score is the best option.

Of the six OS options considered, Raspbian had the lowest total score. In comparison to the other options, Raspbian offered considerably more user-support resources. As the most commonly used OS for the Raspberry Pi, Raspbian offers the best opportunity for a familiar and positive user experience operating a BlockGrid.

Table 7. Comparison of Microcomputer Operating Systems



	ANDROID	WINDOWS IOT-CORE	ARMBIAN	MOEBIUS	RASPBIAN STRETCH	KALI ARM
BLOCKCHAIN COMPABILITY	5	6	2	4	1	3
SMARTGRID SCRIPT COMPATIBILITY	7	1	3	5	2	4
USER RESOURCES	2	4	3	6	1	5
TOTAL SCORE	14	11	8	15	4	12

D. CHAPTER SUMMARY

This chapter reviewed many of the available implementation options for BlockGrid system features. The rationale for the choices that were made in the BlockGrid prototype's implementation was also given. The next chapter provides a description of the prototype's design, function, and operation.

THIS PAGE INTENTIONALLY LEFT BLANK

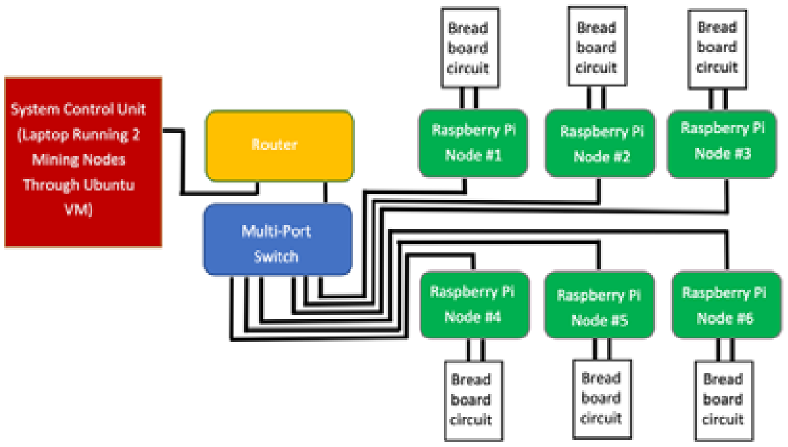
V. BLOCK GRID PROTOTYPE OVERVIEW

The BlockGrid prototype is a simple implementation of the BlockGrid requirements laid out in Chapter III that is optimized for portability and low cost. The prototype empowers an instructor to introduce blockchain technology in the classroom and to demonstrate smart contract functionality by using smart contracts to control a cyber-physical system. The goal of this chapter is to provide an overview of how the prototype works. The prototype's design, function, and operation are described in this chapter, and the technical details of the prototype's implementation are provided in Appendices A and B.

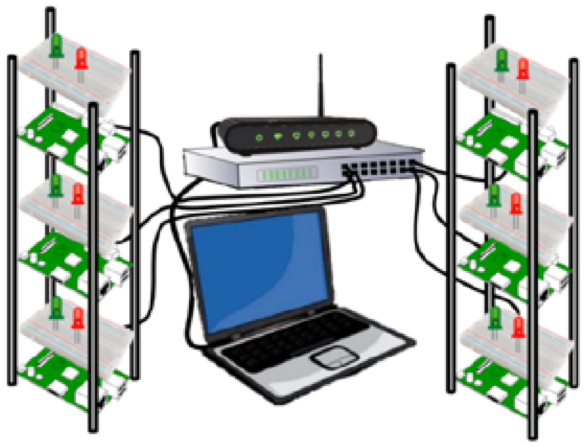
The user; i.e., the instructor, is in complete control of and initiates all BlockGrid events. The user interface is the System Control Unit (SCU). The BlockGrid Prototype's Ubuntu Linux VM runs on a laptop and acts as the SCU. The prototype is designed so that all user interactions with the system, including those with the Raspberry Pi nodes, are carried out through the SCU. This single-interface approach simplifies operation of the prototype and makes it suitable for demonstrations.

A. PROTOTYPE LAYOUT

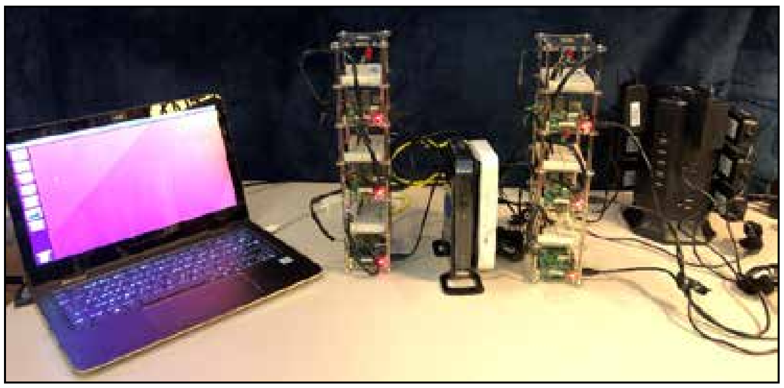
The BlockGrid Prototype has a seven-node hardware layout consisting of six Raspberry Pi microcomputer nodes and an Ubuntu Linux VM running on the user's laptop that serves as the System Control Unit (SCU). Each of the six Raspberry Pi nodes is attached by electrical jumper cables from its General Purpose Input/Output (GPIO) unit to its own breadboard circuit. Each of these breadboard circuits powers a pair of LEDs, one red and one green. The Raspberry Pi nodes and adjoined circuit boards are stacked vertically in two display towers with three Raspberry Pi's and three circuit boards each to minimize desktop surface area. The Raspberry Pi nodes and the laptop-residing VM are connected to one another on an isolated network using ethernet cables, a COTS router, and a multi-port ethernet switch.



16a. BlockGrid Prototype Network Scheme



16b. BlockGrid Prototype Layout



16c. Photo of BlockGrid Prototype

Figure 16. BlockGrid Prototype Network Scheme, Layout, and Photo.

Figure 16 provides three visual depictions of the prototype’s hardware configuration. In the diagram in Figure 16a a conceptual scheme of the hardware layout is given. Figure 16b shows how the hardware elements are arranged. In Figure 16c a photograph of the assembled prototype is shown.

Within this hardware layout the BlockGrid Prototype runs a private, isolated, eight-node Ethereum blockchain. The blockchain’s network consists of two mining nodes and six non-mining nodes. The two mining nodes, which execute the Proof of Work algorithm for the prototype’s blockchain, are virtual and run on the laptop-hosted Ubuntu Linux VM. The six non-mining nodes, which maintain an updated copy of the blockchain’s ledger but do no mining themselves, are each run from one of the prototype’s six Raspberry Pi microcomputers.

The BlockGrid Prototype’s layout has several features that are optimized for portability and classroom use: self-contained and isolated, portable, and sufficient for a variety of demonstrations. The prototype’s isolated network ensures that it has no external network connection dependencies. This allows it to be easily used in classroom or demonstration environments that lack internet access. The prototype’s relatively compact physical layout allows it to be set up in any location with at least 6 ft^2 of flat space and a nearby power supply. Additionally, the number of nodes is large enough to allow a variety of interactions between nodes to be demonstrated, including the resiliency of the blockchain when nodes are compromised or taken down. The number of nodes is still small enough, however, for information about the entire system to be easily displayed on a single Ethereum Wallet web application interface. When this interface is displayed via a screen projector, this allows currency transfers and smart contract transactions between the blockchain node accounts to be easily seen by large groups.

B. PROTOTYPE SYSTEM CONTROL UNIT OVERVIEW AND FUNCTION

The BlockGrid Prototype’s Ubuntu Linux VM runs on a laptop and acts as the System Control Unit (SCU). The prototype is designed so that all user interactions with the system, including those with the Raspberry Pi nodes, are carried out through the SCU. This single-interface approach simplifies operation of the prototype and makes it suitable for demonstrations.

1. SCU Control Mechanisms

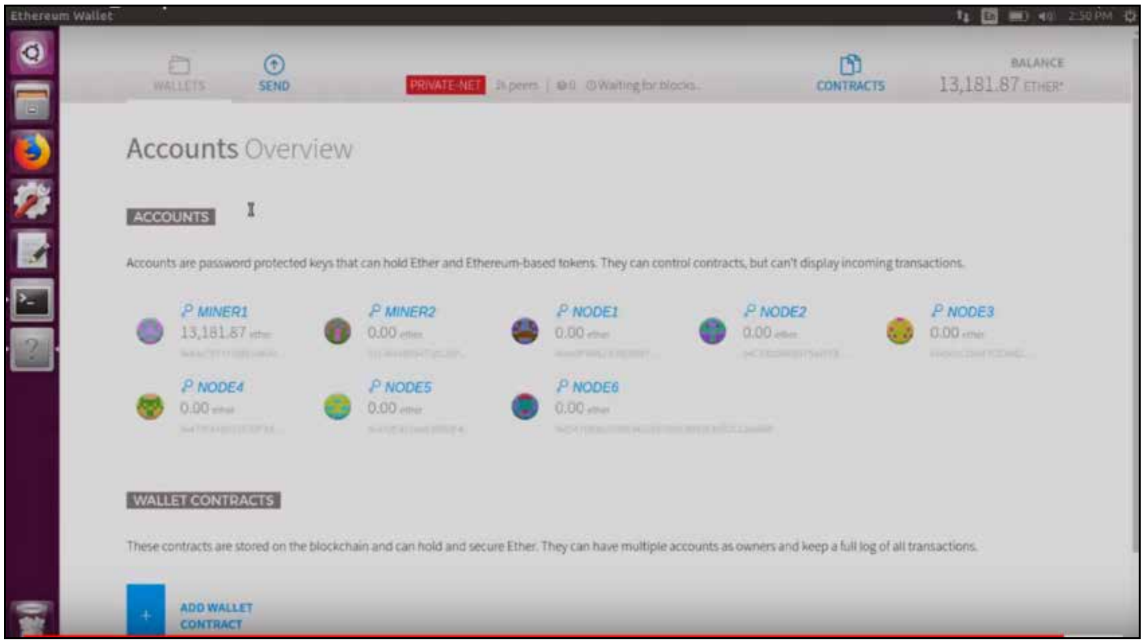
Control of the BlockGrid Prototype through the SCU is achieved primarily using three tools. The first tool is Secure Shell (SSH), a cryptographic network protocol used for remotely administering networks [323]. With SSH, the operating systems of the prototype's Raspberry Pi nodes can be accessed from the SCU via the command line interface (CLI) of the Ubuntu Linux VM's native Terminal application.

This access is necessary for a variety of setup, demonstration, and system administration tasks. For example, during the prototype's setup SSH connections are required to issue Linux terminal commands that synchronize the Raspberry Pi nodes' clocks with the mining nodes that run on the SCU. SSH connections are also required to initialize the Ethereum nodes hosted on each Raspberry Pi to establish the blockchain's network. During demonstrations, SSH connections from the SCU to the Raspberry Pi nodes allow the Ethereum blockchain to be showcased from each node's perspective. This can be used to illustrate blockchain technology principles such ledger redundancy, blockchain network resiliency, and n/2 attacks [324]–[326]. For system administration purposes, SSH connections allow the user to update, patch, and troubleshoot the Raspberry Pi nodes from the SCU.

The second tool used to control the BlockGrid Prototype from the SCU is Ethereum's Geth client. Geth is an implementation of the Ethereum protocol written in the Go language that enables blockchain operation and administrative functions from the command line [327]. Once the prototype's blockchain is up and running, Geth can be used to execute transactions of Ethereum's 'Ether' cryptocurrency between blockchain node accounts, as well as smart contract transactions. If a BlockGrid requires maintenance or augmentation, Geth can also be used to deploy and alter configurations for blockchain nodes and smart contracts.

The third tool for prototype control from the SCU is the Ethereum Wallet web application. Ethereum Wallet is a browser-hosted web application that enables blockchain operation and administrative activities via a GUI interface [328]. Ethereum Wallet has much of the same functionality as the Geth client, but its GUI interface is more intuitive for less technical observers, thus making it more suitable for demonstrations than the Geth CLI.

In the BlockGrid Prototype, Ethereum Wallet is run in the SCU VM's native Firefox web browser. The Ethereum Wallet GUI allows observers to see what is happening in all nodes in the system during transfers of the Ethereum blockchain's Ether cryptocurrency between node accounts or as a result of smart contract transactions. By using the GUI as a visual reference for the audience, a presenter can more clearly explain concepts like Proof of Work, account addresses, and transaction mechanics as the blockchain operates in real time. A screenshot of the interface is shown in Figure 17.



Accounts are password protected keys that can hold Ether and Ethereum-based tokens. They can control contracts, but can't display incoming transactions.

These contracts are stored on the blockchain and can hold and secure Ether. They can have multiple accounts as owners and keep a full log of all transactions.

Figure 17. Ethereum Wallet Interface Screenshot.

2. Operating the SCU

During setup of the BlockGrid prototype the user can initialize the Ethereum blockchain across all of the prototype's nodes via the SCU. Once the system is running, the operator can use the SCU to execute Ether currency transfers between the blockchain node accounts. These transactions can be visualized for demonstration purposes using the Ethereum Wallet web app installed on the VM. Through the Ethereum Wallet web app the operator is also able to execute smart contract transactions between the blockchain node accounts. The outcomes of these smart contract transactions in well-defined cases are reflected in state changes of the affected nodes' LEDs. From the SCU, the operator can also perform any required system administration functions while the blockchain is running.

3. SCU in BlockGrid Architecture

The VM also runs the blockchain's two mining nodes, which compete with one another to perform the Proof of Work calculations for the blockchain's Ethereum protocol. Since only the SCU VM has the computational power within the BlockGrid prototype to perform the Proof of Work algorithm for the system, these mining nodes are fixed in their role and do not change their miner-status during the lifetime of the blockchain's operation. Additionally, since the only way to gain Ether currency within the system is through mining and transactions, when the BlockGrid prototype is initialized, and prior to any transactions, only the miners accrue Ether currency. This Ether currency accumulates in the miner's blockchain accounts, or "wallets."[2]

A minimum of two mining nodes is required for the Ethereum protocol to run, even on a private network. Although the nodes run within the same VM, they are able to compete with each other, as the processing power allocated to each node is unequal and varies with each Proof of Work algorithm attempt. Over time, this results in a difference in the number of blocks added and Ether currency earned by each node.

While running the mining nodes on separate VM's or separate hardware would perhaps better simulate the distributed nature of real-world blockchains, running the mining nodes on the same VM allows for setup, maintenance, and troubleshooting of the blockchain network to take place from one central location during operation of the

[2] These accounts are indistinguishable from the accounts of conventional Ethereum blockchain users. They can transact Ether currency and participate in smart contract executions. Unlike conventional users, they can also accrue increasing amounts of Ether currency as they mine blocks successfully. In contrast, the accounts for non-mining nodes in the BlockGrid Prototype can only initially receive Ether currency via Ether transfer transactions from miner accounts as they generate no Ether currency independently.

prototype. Furthermore, running each mining node on a concurrently executing VM would overburden conventional user laptops, because multi-GB host RAM allocations were necessary. Alternatively, separating the mining nodes physically would require adding mining-capable hardware, such as another laptop, to the prototype. This would likely complicate the logistics and space required to use the prototype in demonstrations.

C. PROTOTYPE RASPBERRY PI NODE OVERVIEW

Each Raspberry Pi node has two electrical output cables that separately power a red and a green LED mounted on a breadboard. Every Raspberry Pi node is therefore capable of ‘switching’ a red or green LED on and off, in addition to performing other computing functions. Each Raspberry Pi node also serves as a non-mining node for the BlockGrid prototype’s Ethereum blockchain and maintains a full copy of the blockchain’s ledger. In addition to maintaining a full copy of the ledger every Raspberry Pi’s blockchain node has an account, also referred to as a “wallet,” on the blockchain. Through its account, each node can participate in Ether currency and smart contract transactions like any conventional Ethereum user.

As the Raspberry Pi nodes do not participate in the system’s Proof of Work mining competitions, they are unable to add new blocks to the blockchain and earn Ether currency. Consequently, when the system is initialized the Raspberry Pi node accounts have a starting value of zero Ether currency. As the prototype’s mining nodes have the only currency-accruing accounts in the system, the Raspberry Pi nodes can only acquire Ether currency initially through transfers from the mining node accounts. Furthermore, the Raspberry Pi node accounts are unable to participate in smart contract transactions until they possess enough Ether currency to pay the smart contract transaction fees required by the Ethereum protocol [329]. In the BlockGrid prototype’s default configuration, these transfers from the mining node accounts to the Raspberry Pi node accounts are not scripted and must be manually executed by the system’s operator after the system is initialized. If the prototype operator is an instructor giving a demonstration, this required initial task provides an opportunity to demonstrate the steps involved with blockchain cryptocurrency transactions.

Once Ether currency has been transferred by the BlockGrid prototype operator to the Raspberry Pi node accounts, these nodes can participate in smart contract transactions. The blockchain maintains a record of each node account's Ether currency balances and any account's associated smart contract data. In the BlockGrid prototype, scripts run locally on each Raspberry Pi that monitor the node account's smart contract data and execute functions that turn the node's attached LEDs on and off when specified smart contract conditions are met.

D. PROTOTYPE SMART CONTRACT OVERVIEW

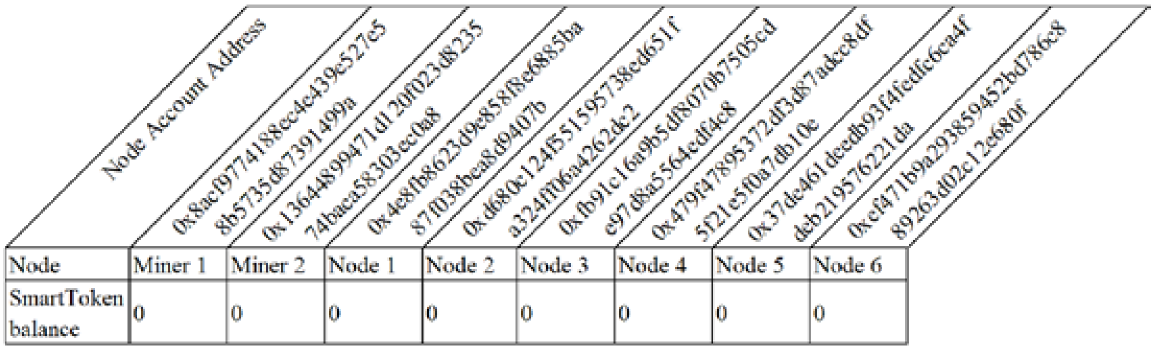
The BlockGrid prototype operates a basic smart contract, called "SmartSwitch," on the system's blockchain. The purpose of the SmartSwitch contract is to empower an instructor to easily demonstrate how smart contracts work by using a smart contract to control a cyber-physical system. Consequently, the BlockGrid prototype's SmartSwitch contract was designed to be as simple as possible, while enabling control of the LEDs attached to the prototype's Raspberry Pi nodes. Using the SmartSwitch contract to turn the red and green LEDs on and off in response to smart contract transactions executed on the blockchain provides a compelling visual aide to demonstrate how smart contracts work.

1. Structure of the SmartSwitch Smart Contract

The SmartSwitch contract achieves cyber-physical control of the BlockGrid prototype by issuing a "SmartToken" that can be used to turn the Raspberry Pi nodes' red and green LEDs on and off. A SmartToken can be abstractly conceived of as a unit of currency used in the context of a smart contract. Only one LED will be lit at any time. When the SmartToken balance in a node's account is greater than zero, the node's green LED is on. When a node's SmartToken balance is zero, the node's red LED is on. This smart contract implementation is intentionally simple to allow for easy demonstrations and avoids any of the complexities that are often incorporated into smart contract-defined tokens in real-world applications.

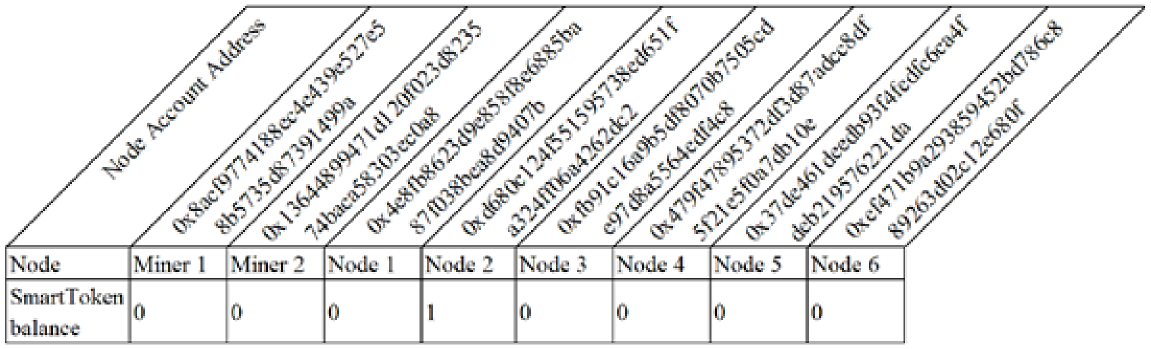
To implement the SmartToken concept, the SmartSwitch contract creates a two-dimensional array that links node account addresses with integer values. Within the array, integer values represent the number of SmartTokens possessed by a corresponding account.

The integer values associated with each Raspberry Pi node's account are analyzed to direct power flow to either the node's green or red LED. A conceptual example of the SmartSwitch contract's data structure is given in Figure 18.



Node Account Address	0x8acf9774188ec4e439e527e5 8b5735d8739149 9a	0x136448994 71d120f023d8235 74baca58303ec0a8	0x4e8fb8623d9e858f8e6885ba 87f038bea8d9407b	0xd680c124f551595738ed651f a324ff06a4262dc2	0xfb91c16a9b5df8070b7505cd e97d8a5564edf4c8	0x479f47895372df3d87adcc8df 5f21e5f0a7db10e	0x37de461deedb93f4fedfc6ea4f deb219576221da	0xef471b9a293859452bd786c8 89263d02c12e680f
Node	Miner 1	Miner 2	Node 1	Node 2	Node 3	Node 4	Node 5	Node 6
SmartToken balance	0	0	0	0	0	0	0	0

SmartSwitch data structure initial state: SmartSwitch contract data structure after initialization showing a SmartToken balance of 0 for each node account. As all the SmartToken balances are 0, all of the Raspberry Pis' red LEDs are powered on in this state.

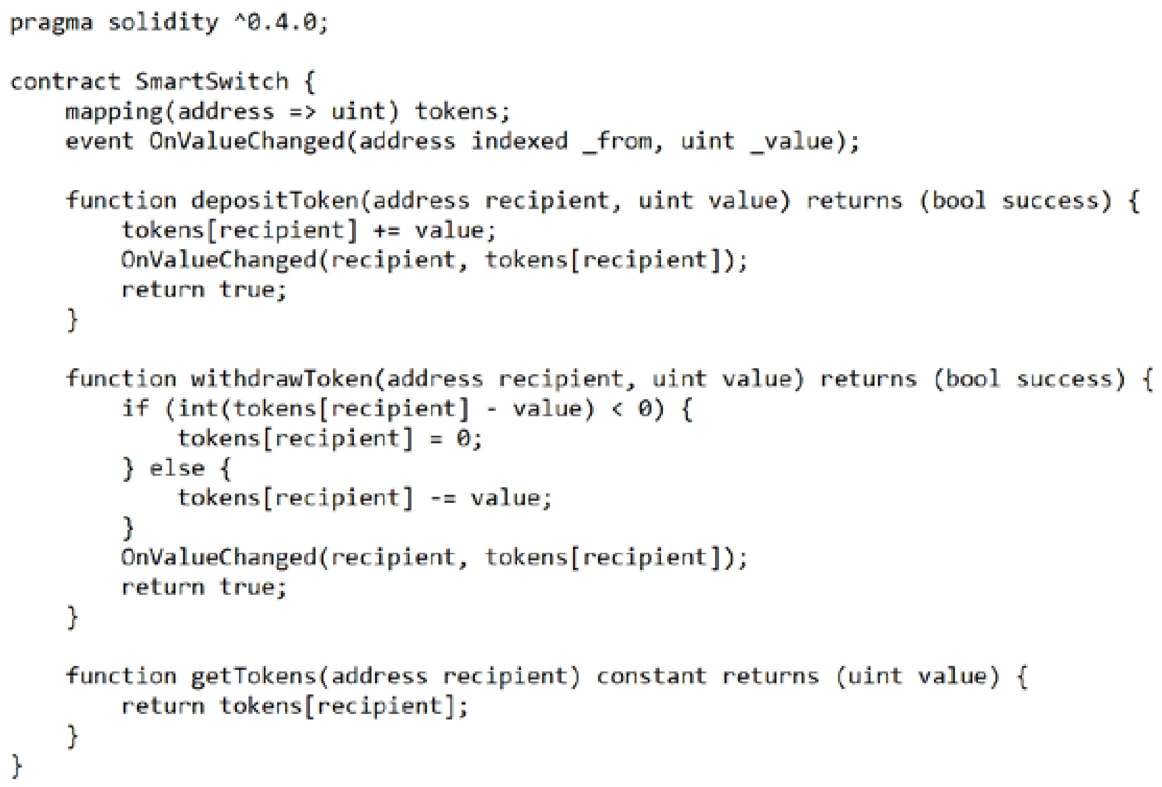


Node Account Address	0x8acf9774188ec4e439e527e5 8b5735d8739149 9a	0x136448994 71d120f023d8235 74baca58303ec0a8	0x4e8fb8623d9e858f8e6885ba 87f038bea8d9407b	0xd680c124f551595738ed651f a324ff06a4262dc2	0xfb91c16a9b5df8070b7505cd e97d8a5564edf4c8	0x479f47895372df3d87adcc8df 5f21e5f0a7db10e	0x37de461deedb93f4fedfc6ea4f deb219576221da	0xef471b9a293859452bd786c8 89263d02c12e680f
Node	Miner 1	Miner 2	Node 1	Node 2	Node 3	Node 4	Node 5	Node 6
SmartToken balance	0	0	0	1	0	0	0	0

SmartSwitch data structure after depositToken() transaction - SmartSwitch contract data structure after a depositToken() smart contract transaction function call is made for the account address 0xd680c124f551595738ed651fa324ff06a4262dc2, raising the SmartToken balance from 0 to 1 for Raspberry Pi Node 2. A transaction like this can be initiated by any node account in the system. After the transaction is processed, the green LED for Raspberry Pi Node 2 is powered on, while the remaining Raspberry Pi nodes still have their red LEDs powered on.

Figure 18. Conceptual Model of the SmartSwitch Contract Data Structure Before and After a depositToken() Smart Contract Transaction Function Call.

The SmartSwitch contract contains three functions that the prototype operator can execute to modify values within the contract's array data structure. The Solidity code for the SmartSwitch smart contract consists of three functions and is shown in Figure 19. The first function, depositToken(), increments the value of an account's associated integer; i.e., its number of SmartTokens, by an amount specified by the operator. The second function, withdrawToken(), decrements an account's SmartToken total by an amount specified by the operator. As written, withdrawToken() will not decrement the SmartToken value below zero, so there is no concept of SmartToken debt within the system. The third function, getTokens(), returns the number of SmartTokens currently in an account, allowing the prototype operator to query that value.



```
pragma solidity ^0.4.0;

contract SmartSwitch {
    mapping(address => uint) tokens;
    event OnValueChanged(address indexed _from, uint _value);

    function depositToken(address recipient, uint value) returns (bool success) {
        tokens[recipient] += value;
        OnValueChanged(recipient, tokens[recipient]);
        return true;
    }

    function withdrawToken(address recipient, uint value) returns (bool success) {
        if (int(tokens[recipient] - value) < 0) {
            tokens[recipient] = 0;
        } else {
            tokens[recipient] -= value;
        }
        OnValueChanged(recipient, tokens[recipient]);
        return true;
    }

    function getTokens(address recipient) constant returns (uint value) {
        return tokens[recipient];
    }
}
```

Figure 19. SmartSwitch Smart Contract Source Code. Adapted From the Eloudrhiri Chainskills Smart Contract Tutorial [330].

a. *Distinguishing Between Different SmartTokens and Ether Currency*

It is important to note that the SmartSwitch contract operates almost entirely separately and independent from the Ethereum blockchain's native Ether currency. SmartTokens are an arbitrary value created by the SmartSwitch contract, whereas Ether currency is the native cryptocurrency for the Ethereum protocol. While both SmartToken and Ether balances are tracked for each node on the Ethereum blockchain's ledger, they are not related to one another and are stored in different locations on the blockchain. As described in Section G of Chapter II, each node account's Ether currency balance is stored as its own field in the header of the account's data structure. By contrast, each node's SmartToken balance is stored within the data structure created by the SmartSwitch contract within the SmartSwitch contract's account. SmartToken balances are accessible to every node via the state trie stored along with each node's copy of the distributed ledger. This means that every node can see every other node's SmartToken balance, not just its own.

Executing the functions of the SmartSwitch contract to adjust SmartToken balances for node accounts does not involve use of Ether currency. Consequently, there is no required Ether currency exchange or interaction between the account initiating the transaction and the node that will process it. The only area of overlap between Ether currency and SmartTokens is that the account making a SmartSwitch contract function call to adjust the SmartToken balance for an particular account must pay the Gas tax for the transaction in Ether currency.

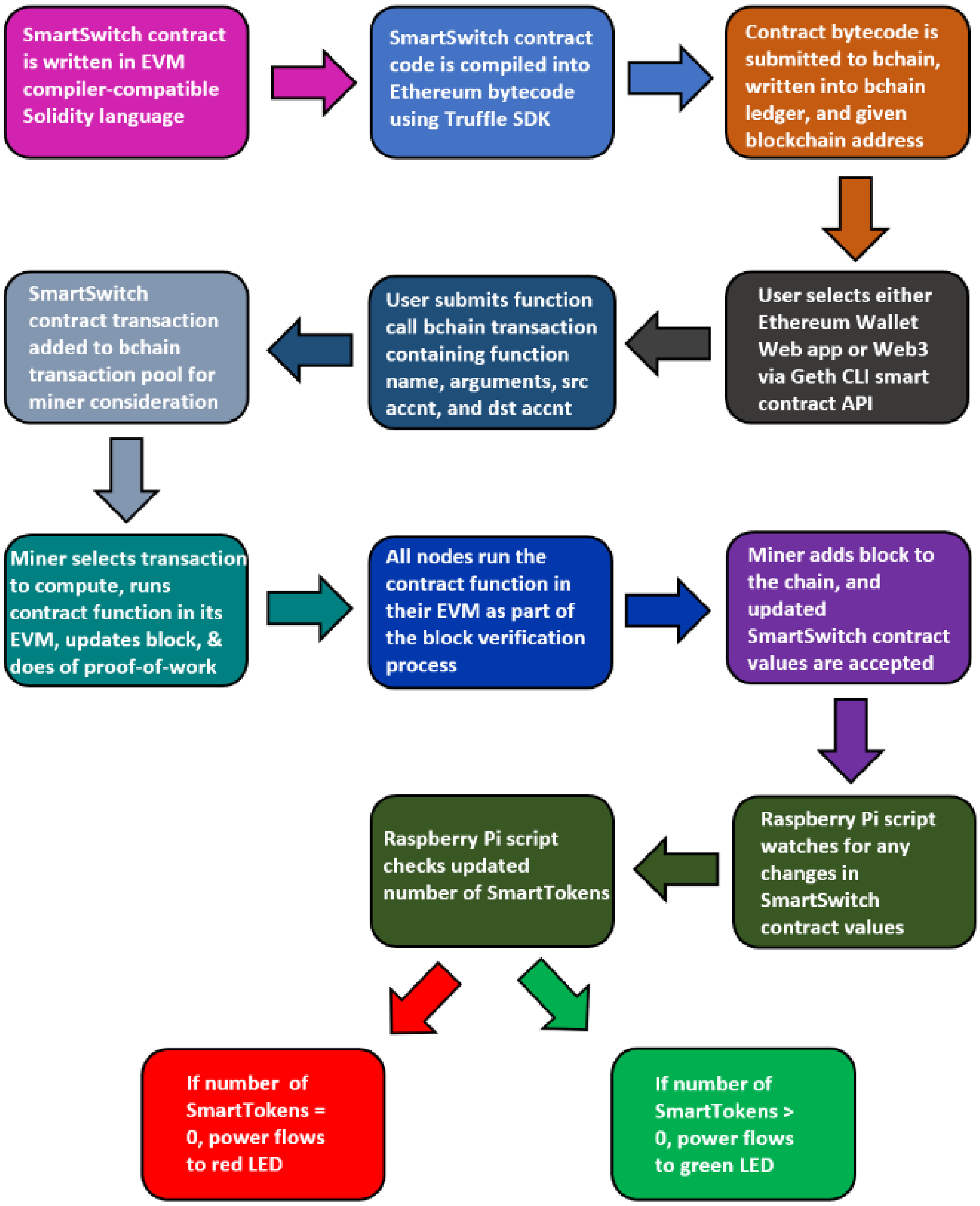
2. Deploying and Executing the SmartSwitch Smart Contract

Before the operator can make SmartSwitch contract function calls that are computed and updated throughout the blockchain network, a number of steps must be taken to deploy the smart contract and enable its execution. As shown in Figure 20, the SmartSwitch contract is initially written in the Solidity programming language. Then the code is compiled into Ethereum bytecode using the Truffle SDK. Many open-source compilers are available for Solidity, and Truffle is one of the most popular [331]. Once compiled, the contract's bytecode is submitted as a new account transaction to the blockchain, written into the blockchain's ledger, and given a corresponding address. At this point the SmartSwitch contract is fully deployed on the blockchain system and a node account can call its functions for execution.

Any node account in BlockGrid prototype can execute a SmartSwitch function call against any other node's account. Typically, these transactions are initiated by the prototype operator from the SCU VM but they may also be initiated by the operator from any of the Raspberry Pi nodes via an SSH connection from the SCU VM or directly from a Raspberry Pi itself. SmartTokens are not transferred between accounts and are not a fixed quantity within the system. There is no predeermined upper bound on the maximum number of SmartTokens that can be deposited within node's account, or that can exist within the system overall. The only upper bound is the maximum integer value permitted on the SmartToken contract's block in the blockchain's ledger, which is determined by the block's size. In the Ethereum protocol, block sizes can vary and the maximum size is ultimately governed by the Gas limit set by the system's miners [332]. In the BlockGrid prototype, the Gas limit is set to the maximum, which is 0xffffffff.The SmartToken is not like a currency in which there is a fixed number of tokens in circulation within the system. SmartTokens are not circulated between node accounts. They are simply integer values associated with an account address that can only be modified through the *depositToken()* or *withdrawToken()* function calls for a specific account.

To operate the system, the miner nodes must first be initialized within the SCU VM. The operator must then manually initialize the Raspberry Pi nodes and synchronize their system clocks to the SCU VM via SSH connections to get the entire blockchain network up and running. The operator can then open the Ethereum Wallet web application within the SCU VM's native Firefox browser to control the system.

Once the SmartSwitch contract is deployed, the operator can make a smart contract function calls to a blockchain API that allows transactions to read from and write to the blockchain. Within the BlockGrid prototype this is done using the Ethereum Wallet web application's built-in smart contract interface. These calls can also be made at the command line using the Geth command line tool by using Geth's JavaScript Web3 API [333].


SmartSwitch contract is written in EVM compiler-compatible Solidity language
SmartSwitch contract code is compiled into Ethereum bytecode using Truffle SDK
Contract bytecode is submitted to bchain, written into bchain ledger, and given blockchain address
User selects either Ethereum Wallet Web app or Web3 via Geth CLI smart contract API
User submits function call bchain transaction containing function name, arguments, src accnt, and dst accnt
SmartSwitch contract transaction added to bchain transaction pool for miner consideration
Miner selects transaction to compute, runs contract function in its EVM, updates block, & does of proof-of-work
All nodes run the contract function in their EVM as part of the block verification process
Miner adds block to the chain, and updated SmartSwitch contract values are accepted
Raspberry Pi script watches for any changes in SmartSwitch contract values
Raspberry Pi script checks updated number of SmartTokens
If number of SmartTokens = 0, power flows to red LED
If number of SmartTokens > 0, power flows to green LED

The term nodes in the flowchart refers to both the virtual mining nodes on the SCU VM and the non-mining Raspberry Pi nodes. Both run the smart contract code in their respective EVMs to update the SmartSwitch state in their ledger copy. Once the updated smart contract values are accepted and new transaction block is added to the chain by the miner, the new SmartToken balance is assessed by the script on the Raspberry Pi nodes.

Figure 20. SmartSwitch Contract Build and Execution Sequence.

The Ethereum Wallet web application offers the advantage of a GUI that showcases each step in the smart contract execution in a demonstration-friendly manner. It also offers conveniently stored smart contract addresses upon their initial deployment so that they can be referenced by clicking on the smart contract's name within the GUI rather than its lengthy blockchain address.

Once the blockchain API has been selected, the operator must submit a blockchain transaction providing several data elements.

These are:

(1) the address of the SmartSwitch contract (this is done automatically when the smart contract is selected),

(2) the requested function call and associated arguments,

(3) the address of the account making the function call (and paying the associated Gas tax), and

(4) the address of the account containing the SmartToken value to be queried or modified.

Once the smart contract transaction has been signed with the initiating account's private key, this transaction data is submitted to the blockchain's transaction pool where it can be accessed by miners assembling the next block.

The mining nodes competing to add the next block to the system's blockchain execute the smart contract functions within their node's Ethereum Virtual Machine (EVM) and update their copy of the blockchain ledger to reflect the outcomes of the function call made by the operator. For the SmartSwitch contract any of the contract's three functions can be used: *getToken()*, *depositToken()*, or *withdrawToken()*. Both mining nodes and non-mining nodes process the smart contract code in their EVMs. This means that within the BlockGrid prototype both the virtual mining nodes on the SCU VM and the Raspberry Pi nodes run the SmartSwitch code in their respective EVMs to update their respective copies of the system's ledger. When the mining nodes finish running all of the transactions in the transaction pool needed for the next block to be added to the system's blockchain, all of the other nodes in the system execute the smart contract code within their respective EVMs as part of the block verification process in the Ethereum protocol's consensus algorithm.

Once a block has been verified, additional steps are required to complete the transaction process. The mining node that ultimately wins the Proof of Work competition commits the new block to the blockchain. The Gas tax for the computation is assessed to the account that submitted the transaction by the Ethereum protocol. This Gas tax is then used by the protocol to compensate the miner's account. Finally, the new state of the smart contract is updated onto every copy of the blockchain's ledger duplicated throughout the system.

3. Monitoring SmartToken Balances from the Raspberry Pi Nodes

While the SmartSwitch contract operates on the BlockGrid prototype's Ethereum blockchain, a JavaScript program called smart_token.js runs continuously on each Raspberry Pi node to monitor for an interrupt signal indicating any change in the number of SmartTokens in that node's account. Whenever a change in the state of the SmartSwitch contract occurs as a result of a smart contract transaction executed by one of the system's node accounts, an interrupt signal is generated that is detected by smart_token.js and prompts smart_token.js to count the number of SmartTokens in the node's account. If the number of SmartTokens in the node's account is 0, the smart_token.js directs the Raspberry Pi to send power to the node's red LED through the Raspberry Pi's appropriate GPIO pin. If the number of tokens is found to be any value greater than 0, the program directs the Raspberry Pi to send power to the green LED through its associated GPIO pin.

With this simple functionality every Raspberry Pi node in the BlockGrid prototype system essentially functions as a smart contract-controlled power switch, or "smart switch." By depositing and withdrawing SmartTokens from the nodes' accounts via the SCU VM's Ether Wallet smart contract interface, the system operator can make the red and green LEDs of the system turn on and off. This visually demonstrates the effect of a smart contracts in operation. It opens the door for discussion of the use of smart contracts to operate logical gates or grant permissions for cyber-physical systems within a smartgrid or IoT network.

A detailed walkthrough of how to set up and operate the prototype is provided in Appendix C. Video walkthroughs of the setup and operation of the prototype are also provided as supplemental materials.

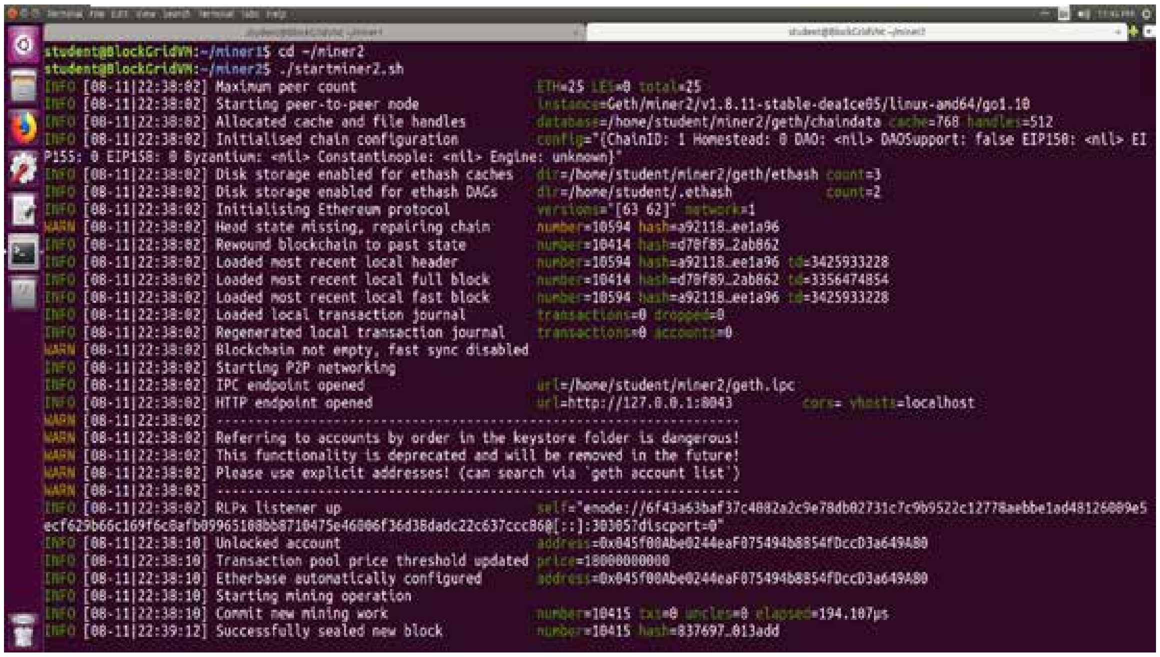
E. OPPORTUNITIES FOR INSTRUCTION

At every stage in the BlockGrid prototype operation workflow, there are opportunities to demonstrate key aspects of blockchain technology for observers. When the display of the laptop running the SCU VM is connected and duplicated to a projector, observers can see important aspects of system function in addition to observing the power switch between system nodes' red and green LEDs.

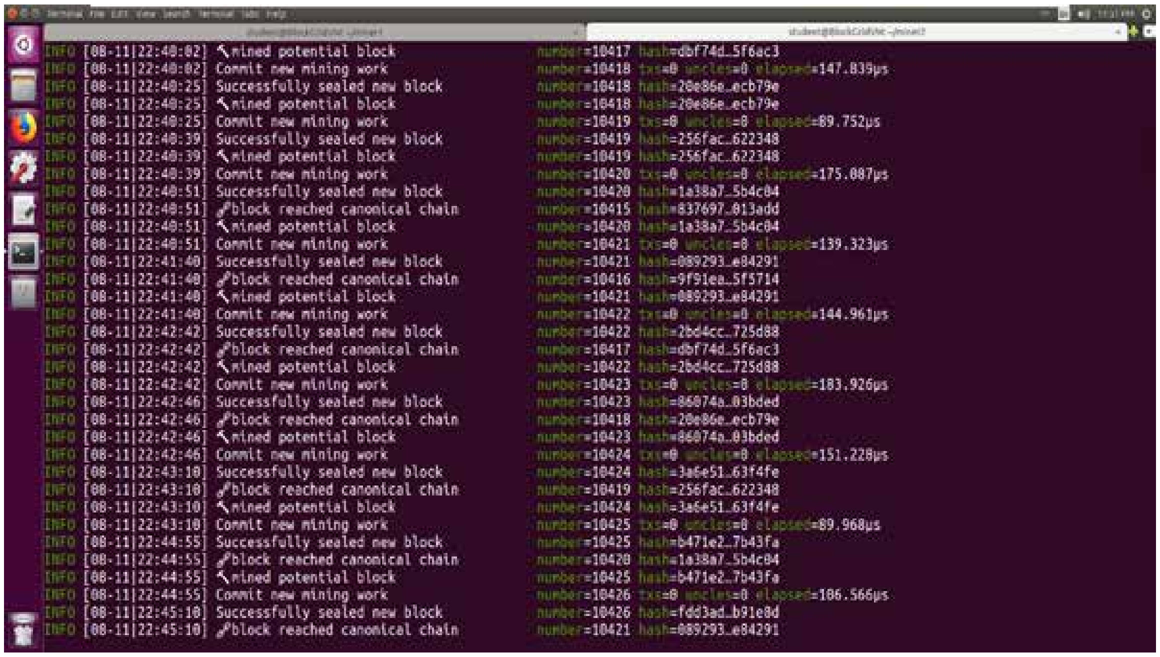
1. Miner Initialization on the SCU VM

Upon initialization, the miners on the SCU VM generate a continuous stream of system state output reports containing data such as blockchain size, time required to add the most recent block, and Proof of Work algorithm phase. The Proof of Work algorithm phases reported are high-level descriptions such as "mined potential block," "Commit new mining work," and "Successfully sealed new block." This information is displayed by the SCU VM's Terminal application.

Within the Terminal application window, each status report line in the stream includes several fields of information. The fist field is a timestamp. The second field is the Proof of Work algorithm phase. The third field is the block number, which defines the position of the block within the sequence of the blockchain (as in the first block would have block number 1 and the 7th in the blockchain would have block number 7). The fourth field is the number of transactions recorded in the block. The fifth field is the number of uncle blocks present in the current block's subtree of the growing blockchain. (Uncle blocks are discussed in Section G of Chapter II.) The sixth and final field is a another timestamp that reflects the time elapsed since the last Proof of Work status report.



```
student@BlockGridVM:~/miner1$ cd ~/miner2
student@BlockGridVM:~/miner2$ ./startminer2.sh
INFO [08-11|22:38:02] Maximum peer count                       ETH=25 LES=0 total=25
INFO [08-11|22:38:02] Starting peer-to-peer node               instance=Geth/miner2/v1.8.11-stable-dea1ce05/linux-amd64/go1.10
INFO [08-11|22:38:02] Allocated cache and file handles         database=/home/student/miner2/geth/chaindata cache=768 handles=512
INFO [08-11|22:38:02] Initialised chain configuration          config="{ChainID: 1 Homestead: 0 DAO: <nil> DAOSupport: false EIP150: <nil> EI
P155: 0 EIP158: 0 Byzantium: <nil> Constantinople: <nil> Engine: unknown}"
INFO [08-11|22:38:02] Disk storage enabled for ethash caches   dir=/home/student/miner2/geth/ethash count=3
INFO [08-11|22:38:02] Disk storage enabled for ethash DAGs     dir=/home/student/.ethash            count=2
INFO [08-11|22:38:02] Initialising Ethereum protocol           versions="[63 62]" network=1
WARN [08-11|22:38:02] Head state missing, repairing chain      number=10594 hash=a92118…ee1a96
INFO [08-11|22:38:02] Rewound blockchain to past state         number=10414 hash=d70f89…2ab862
INFO [08-11|22:38:02] Loaded most recent local header          number=10594 hash=a92118…ee1a96 td=3425933228
INFO [08-11|22:38:02] Loaded most recent local full block      number=10414 hash=d70f89…2ab862 td=3356474854
INFO [08-11|22:38:02] Loaded most recent local fast block      number=10594 hash=a92118…ee1a96 td=3425933228
INFO [08-11|22:38:02] Loaded local transaction journal         transactions=0 dropped=0
INFO [08-11|22:38:02] Regenerated local transaction journal    transactions=0 accounts=0
WARN [08-11|22:38:02] Blockchain not empty, fast sync disabled
INFO [08-11|22:38:02] Starting P2P networking
INFO [08-11|22:38:02] IPC endpoint opened                      url=/home/student/miner2/geth.ipc
INFO [08-11|22:38:02] HTTP endpoint opened                     url=http://127.0.0.1:8043         cors= vhosts=localhost
WARN [08-11|22:38:02] -----------------------------------------------------------------
WARN [08-11|22:38:02] Referring to accounts by order in the keystore folder is dangerous!
WARN [08-11|22:38:02] This functionality is deprecated and will be removed in the future!
WARN [08-11|22:38:02] Please use explicit addresses! (can search via `geth account list`)
WARN [08-11|22:38:02] -----------------------------------------------------------------
INFO [08-11|22:38:02] RLPx listener up                         self="enode://6f43a63baf37c4082a2c9e78db02731c7c9b9522c12778aebbe1ad48126009e5
ecf629b66c169f6c0afb09965108bb8710475e46006f36d38dadc22c637ccc860[::]:303057discport=0"
INFO [08-11|22:38:10] Unlocked account                         address=0x045f00Abe0244eaF075494b8854fDccD3a649A80
INFO [08-11|22:38:10] Transaction pool price threshold updated price=18000000000
INFO [08-11|22:38:10] Etherbase automatically configured       address=0x045f00Abe0244eaF075494b8854fDccD3a649A80
INFO [08-11|22:38:10] Starting mining operation
INFO [08-11|22:38:10] Commit new mining work                   number=10415 txs=0 uncles=0 elapsed=194.107µs
INFO [08-11|22:39:12] Successfully sealed new block            number=10415 hash=837697…013add
```



```
INFO [08-11|22:40:02] 🔨mined potential block                  number=10417 hash=dbf74d…5f6ac3
INFO [08-11|22:40:02] Commit new mining work                   number=10418 txs=0 uncles=0 elapsed=147.839µs
INFO [08-11|22:40:25] Successfully sealed new block            number=10418 hash=20e86e…ecb79e
INFO [08-11|22:40:25] 🔨mined potential block                  number=10418 hash=20e86e…ecb79e
INFO [08-11|22:40:25] Commit new mining work                   number=10419 txs=0 uncles=0 elapsed=89.752µs
INFO [08-11|22:40:39] Successfully sealed new block            number=10419 hash=256fac…622348
INFO [08-11|22:40:39] 🔨mined potential block                  number=10419 hash=256fac…622348
INFO [08-11|22:40:39] Commit new mining work                   number=10420 txs=0 uncles=0 elapsed=175.087µs
INFO [08-11|22:40:51] Successfully sealed new block            number=10420 hash=1a38a7…5b4c04
INFO [08-11|22:40:51] 🔗block reached canonical chain          number=10415 hash=837697…013add
INFO [08-11|22:40:51] 🔨mined potential block                  number=10420 hash=1a38a7…5b4c04
INFO [08-11|22:40:51] Commit new mining work                   number=10421 txs=0 uncles=0 elapsed=139.323µs
INFO [08-11|22:41:40] Successfully sealed new block            number=10421 hash=089293…e84291
INFO [08-11|22:41:40] 🔗block reached canonical chain          number=10416 hash=9f91ea…5f5714
INFO [08-11|22:41:40] 🔨mined potential block                  number=10421 hash=089293…e84291
INFO [08-11|22:41:40] Commit new mining work                   number=10422 txs=0 uncles=0 elapsed=144.961µs
INFO [08-11|22:42:42] Successfully sealed new block            number=10422 hash=2bd4cc…725d88
INFO [08-11|22:42:42] 🔗block reached canonical chain          number=10417 hash=dbf74d…5f6ac3
INFO [08-11|22:42:42] 🔨mined potential block                  number=10422 hash=2bd4cc…725d88
INFO [08-11|22:42:42] Commit new mining work                   number=10423 txs=0 uncles=0 elapsed=183.926µs
INFO [08-11|22:42:46] Successfully sealed new block            number=10423 hash=86074a…03bded
INFO [08-11|22:42:46] 🔗block reached canonical chain          number=10418 hash=20e86e…ecb79e
INFO [08-11|22:42:46] 🔨mined potential block                  number=10423 hash=86074a…03bded
INFO [08-11|22:42:46] Commit new mining work                   number=10424 txs=0 uncles=0 elapsed=151.228µs
INFO [08-11|22:43:10] Successfully sealed new block            number=10424 hash=3a6e51…63f4fe
INFO [08-11|22:43:10] 🔗block reached canonical chain          number=10419 hash=256fac…622348
INFO [08-11|22:43:10] 🔨mined potential block                  number=10424 hash=3a6e51…63f4fe
INFO [08-11|22:43:10] Commit new mining work                   number=10425 txs=0 uncles=0 elapsed=89.968µs
INFO [08-11|22:44:55] Successfully sealed new block            number=10425 hash=b471e2…7b43fa
INFO [08-11|22:44:55] 🔗block reached canonical chain          number=10420 hash=1a38a7…5b4c04
INFO [08-11|22:44:55] 🔨mined potential block                  number=10425 hash=b471e2…7b43fa
INFO [08-11|22:44:55] Commit new mining work                   number=10426 txs=0 uncles=0 elapsed=106.566µs
INFO [08-11|22:45:10] Successfully sealed new block            number=10426 hash=fdd3ad…b91e8d
INFO [08-11|22:45:10] 🔗block reached canonical chain          number=10421 hash=089293…e84291
```

Figure 21. Example Stream of Miner State Reports.

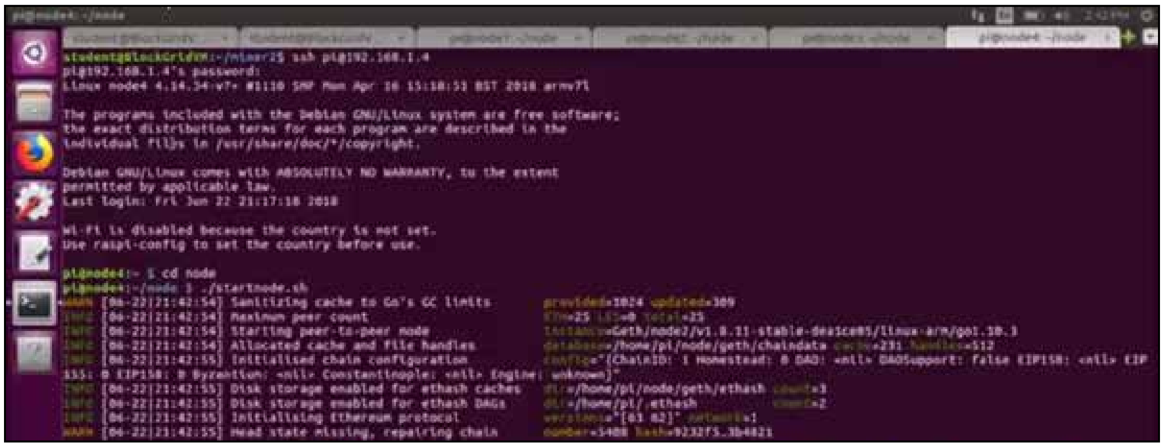
Two screenshots of the system state output report from a mining node upon startup and from a mining node that is fully operational are provided in Figure 21. The output shown in the first startup screenshot of Figure 21 shows 21 steps taken by the miner2 node to initialize the blockchain as well as several security advisory warnings about the blockchain's configuration. These steps and warnings are described in the white text of the left column. The second screenshot shows the status reports when the blockchain is up and running. A close examination of the sequence of block numbers in the left column of the second screenshot shows several gaps. These block number sequence gaps reflect instances in which the miner lost the Proof of Work competition for a block and a different miner added the block to the blockchain.

The Terminal output during the miner initialization provides an opportunity to discuss the fundamental features of blockchain protocols and consensus algorithms as these features are referenced in the system state output reports [334]–[336]. Launching the miners can also be a starting point from which to explore the specifics of Proof of Work consensus algorithms as well as the specific features of the Ethereum blockchain protocol such as the Ethash consensus algorithm, the role of the Ethereum Virtual Machine, and the state trie download [337]–[339].

2. Ethereum Node Initialization on the Raspberry Pis

After the Ethereum nodes on the BlockGrid prototype's Raspberry Pis are initialized during system setup, they too generate a constant stream of system state output reports. The state output information reported upon initialization includes descriptions of the steps the Ethereum node takes to prepare and sync with the blockchain such as "Starting peer-to-peer node," "Allocated cache and file handles," and "Disk storage enabled for ethash DAGs [340]." Once the node is fully operational the status reports alternate between states like "Imported new block headers," "Imported new state entries," and "Imported new block receipts" along with a timestamp, a count of the newly imported items, the time elapsed to perform the action, the block number, and the block's hash [341]. An example screenshot of the output data of a node upon startup is provided in Figure 22.

Initializing the Raspberry Pi nodes provides an opportunity to discuss fundamentals of blockchain architecture. It can also be a starting point for discussions of possible blockchain-connected devices. For example, one could discuss the suitability of turning sensors or ICS infrastructure to blockchain nodes [342], [343]. Launching the nodes may further provide an opportunity to discuss the potential applications of smart contracts in facilitating interactions within IoT networks and cyber-physical systems [344], [345].



```
student@BlockGridVM:~/miner2$ ssh pi@192.168.1.4
pi@192.168.1.4's password:
Linux node4 4.14.34-v7+ #1110 SMP Mon Apr 16 15:18:51 BST 2018 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Fri Jun 22 21:17:18 2018

Wi-Fi is disabled because the country is not set.
Use raspi-config to set the country before use.

pi@node4:~ $ cd node
pi@node4:~/node $ ./startnode.sh
WARN [06-22|21:42:54] Sanitizing cache to Go's GC limits       provided=1024 updated=309
INFO [06-22|21:42:54] Maximum peer count                       ETH=25 LES=0 total=25
INFO [06-22|21:42:54] Starting peer-to-peer node               instance=Geth/node2/v1.8.11-stable-dea1ce05/linux-arm/go1.10.3
INFO [06-22|21:42:54] Allocated cache and file handles         database=/home/pi/node/geth/chaindata cache=231 handles=512
INFO [06-22|21:42:55] Initialised chain configuration          config="{ChainID: 1 Homestead: 0 DAO: <nil> DAOSupport: false EIP150: <nil> EIP
155: 0 EIP158: 0 Byzantium: <nil> Constantinople: <nil> Engine: unknown}"
INFO [06-22|21:42:55] Disk storage enabled for ethash caches   dir=/home/pi/node/geth/ethash count=3
INFO [06-22|21:42:55] Disk storage enabled for ethash DAGs     dir=/home/pi/.ethash           count=2
INFO [06-22|21:42:55] Initialising Ethereum protocol           versions="[63 62]" network=1
WARN [06-22|21:42:55] Head state missing, repairing chain      number=3408 hash=923273…3b4821
```

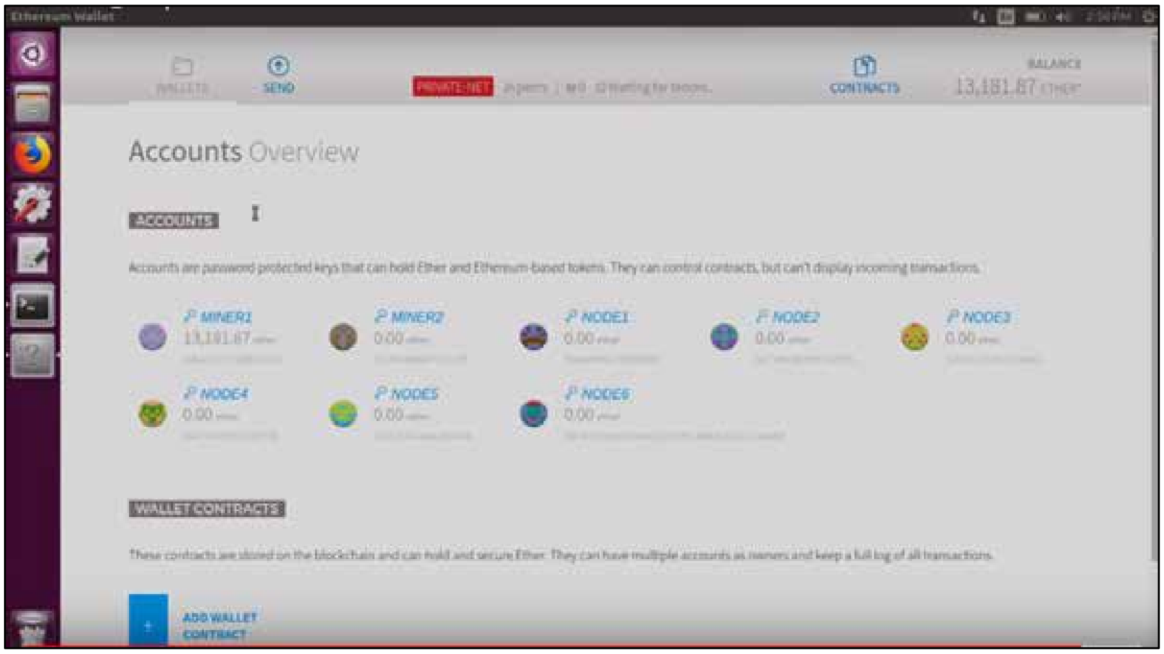
Figure 22. Example Stream of Raspberry Pi Node State Reports.

3. Ethereum Wallet Node Account Interface

When the system operator opens the Ethereum Wallet web browser application, a user interface shows all the node accounts on the system and their respective account balances for the blockchain protocol’s Ether currency. An example screenshot of the interface is provided in Figure 23. In this screenshot within the “Accounts” view all the account names within the prototype are listed along with their corresponding Ether currency balances. This view reflects the conditions upon system initialization, as only a miner node has a positive account balance.

When the system is initialized, only the miner accounts have non-zero balances, as they are the only nodes that accrue Ether currency as compensation for their mining work. However, the operator can manually execute Ether currency transfers from the mining node accounts to the Raspberry Pi node accounts using the Ethereum Wallet interface.Initial transfers to the Raspberry Pi node accounts from the miner accounts are necessary to provide the Raspberry Pi node accounts with enough Ether currency to pay the Gas tax required to execute subsequent smart contract transactions.

Navigating the Ethereum Wallet interface provides an opportunity to explain how the Ethereum protocol works, including the role of the protocol's currency in incentivizing miners and in the Gas tax applied to all Ethereum transactions. Exploring the Ethereum Wallet interface also provides an opportunity to discuss the nature of private chains for development and research purposes as well as the benefits of the prototype's internally networked system. In a classroom context it can also be a springboard for discussion on the role of blockchain technology in cryptocurrencies and other potential financial applications such as inter-bank transfers, currency exchange, financial regulations, and taxation [346]–[350].



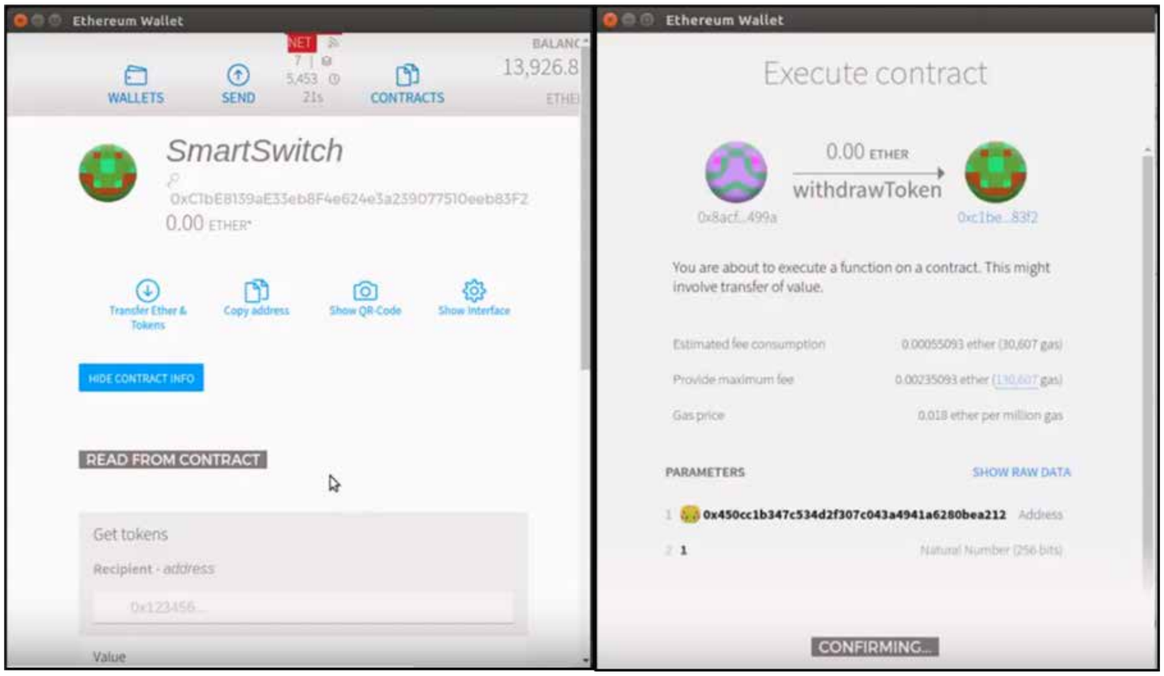
Accounts are password protected keys that can hold Ether and Ethereum-based tokens. They can control contracts, but can't display incoming transactions.

These contracts are stored on the blockchain and can hold and secure Ether. They can have multiple accounts as owners and keep a full log of all transactions.

Figure 23. Ethereum Wallet Interface Screenshot.

4. Smart Contract Transactions

From the Ethereum Wallet interface, the operator can switch the display to show a separate smart contract interface. The Ethereum Wallet web app displays the functions of the SmartSwitch contract and allows the operator to manually call those functions to execute smart contract transactions. Example screenshots of the Ethereum Wallet smart contract interface are given in Figure 24.



SmartSwitch

0xC1bE8139aE33eb8F4e624e3a239077510eeb83F2

0.00 ETHER*

Get tokens

Recipient - address

0x123456...

Value

You are about to execute a function on a contract. This might involve transfer of value.

Estimated fee consumption	0.00055093 ether (30,607 gas)
Provide maximum fee	0.00235093 ether (130,607 gas)
Gas price	0.018 ether per million gas

1	0x450cc1b347c534d2f307c043a4941a6280bea212	Address
2	1	Natural Number (256 bits)

Figure 24. SmartSwitch Contract Interface for a withdrawToken() Function Call Decrementing an Account's SmartToken Balance.

On the left in Figure 24, the smart contract interface shows the SmartSwitch contract name and address and lists the smart contract's functions for the operator. Having selected a function to execute and provided the recipient node account's address and the initiating node account's address, the operator is then prompted to enter the initiating node account's private key to execute the transaction and pay the associated Gas tax. On the right, the interface shows the *withdrawToken()* function call transaction execution that takes place immediately following the entry of the initiating node account's private key.

Navigating the Ethereum Wallet's smart contract interface to execute a smart contract transaction provides an opportunity to discuss the fundamental features of smart contracts: how they are developed and how they work. For example, the source code of the SmartSwitch contract can be analyzed concurrently with its execution on the prototype. This provides observers with the opportunity to learn how the smart contract is structured, as well as some of the syntax of Ethereum's Solidity programming language [351].

Another instructional possibility is to execute successive smart contract transactions from the Ethereum Wallet GUI and the command line Web3 API. This can demonstrate the crucial role of APIs in allowing individual nodes to communicate with the blockchain [352]. A third option is to walk through the steps involved in executing a smart contract transaction on the blockchain, from the smart contract function call to the execution of the code within every nodes' EVM [353].

Although the SmartSwitch contract does not create any dependencies between the nodes or utilize multi-node conditional logic to control the Raspberry Pi nodes' LED output, the BlockGrid prototype highlights several distributed computing attributes of blockchain technology, which can be demonstrated during smart contract executions. As explained previously, smart contract code is run by every node, both mining and non-mining, and is used to create consensus about the validity of the newly mined block containing the smart contract transaction. Additionally, the functions of the SmartSwitch contract can be invoked by any node's account to modify the SmartToken balance in any other node's account. Both mining nodes and non-mining nodes have accounts, and consequently both the virtual mining nodes and the non-mining Raspberry Pi nodes have accounts that can execute smart contract transactions. An instructor can therefore demonstrate that control of the LEDs in the BlockGrid prototype is distributed as they can be switched on and off by any node account and from either the SCU VM or any Raspberry Pi node's operating system.

For convenience, the BlockGrid prototype is designed so that SmartSwitch function calls can be made from different node accounts using a single interface: the smart contract interface of the SCU VM Ethereum Wallet web application. Initiating a function call from a specific node account anywhere in the system only requires providing the source account's blockchain address when the transaction is initiated and signing the transaction with transaction-initiating node account's private key. However, the distributed nature of the blockchain can also be demonstrated by executing the smart contract transaction from any of the system's Raspberry Pi nodes. The prototype does not automate this, but an operator can manually execute a smart contract transaction from any Raspberry Pi node in the system.[3]

Additionally, due to the distributed properties of the blockchain, the state of the SmartSwitch contract is accessible to all of the system's nodes independent of the status of the nodes that initiated or received the transaction. This has noteworthy cybersecurity implications that positively impact the availability and integrity of the smart contract data. For example, if a node that increments the SmartToken balance of a target node and is subsequently taken offline by a DDoS attack, there is no consequence to the target node and its LED status remains the same. If a node in the system is compromised by an attacker who corrupts or changes the SmartToken balances for all the node accounts within the local copy of the blockchain ledger, the distributed nature of the blockchain and the consensus algorithm of the blockchain protocol do not allow the attacker's actions to compromise the system. As described previously in Section H of Chapter II, to succeed in an attempt to control or DDoS the system, an attacker would need to launch an n/2 eclipse attack to gain control of more than half of the system's mining nodes. These availability and integrity issues can be effectively explored in demonstrations to address the security benefits of blockchains and how they impact potential applications of smart contracts.

Distributed computing features of blockchain technology can also be demonstrated with the prototype though Ether currency transactions between node accounts. The transaction history inscribed in the ledger can be examined from the perspective of each node to showcase how the ledger is distributed throughout the network. The transfers of Ether currency between accounts can also be executed from any node in the network provided that the sender and recipient addresses and their respective private keys are known.

Launching the smart contract interface can also provide a starting point for a discussion of the potential scope of smart contract functionality for different blockchain applications. Example smart contract applications for discussion could include voting

[3] This can be done using the Web3 API via the Geth CLI installed on the Raspberry Pi's OS, which, in turn, can be accessed directly by attaching a monitor and keyboard to the Raspberry Pi or from the SCU VM via an SSH connection.

systems, personnel records, insurance, healthcare records, and supply chain management [354]–[358].

5. LED color changes

As the system operator executes SmartSwitch transactions that make SmartSwitch function calls to modify node account SmartToken balances, the prototype's LEDs are powered on and off according to the conditional logic in the smart_token.js scripts that run on each Raspberry Pi. The smart_token.js scripts watch for any SmartSwitch transactions that change the SmartToken balance for the node's account. The conditional logic in the script directs power to the red LED whenever the Raspberry Pi's SmartToken balance is 0. When the account balance is greater than 0, power is directed to the green LED. When the 'smart switch' is thrown and power flow switches between the LEDs, the change is visible to observers and provides a concrete demonstration of the smart contract transaction process. Figure 25 shows a lighted red LED for one of the BlockGrid prototype's Raspberry Pi nodes.

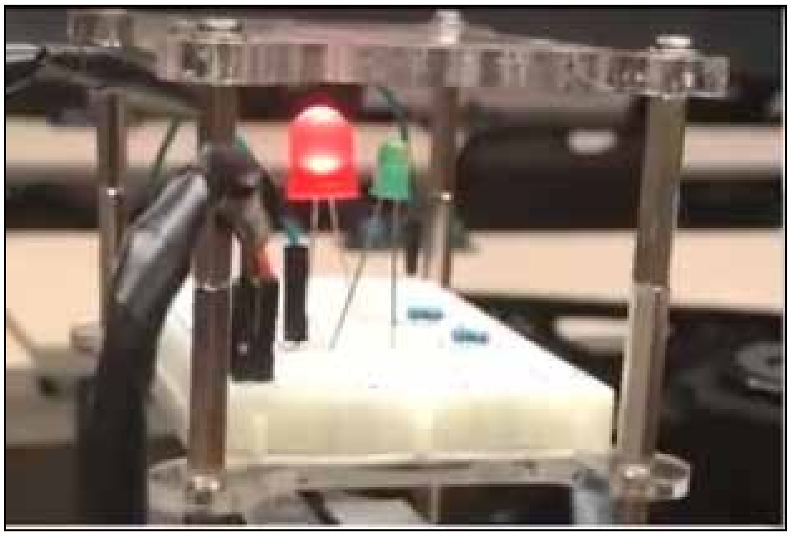


Figure 25. A Red LED Powered on for a BlockGrid Prototype Node.

Executing SmartSwitch transactions to switch the power flow between red and green LEDs provides an opportunity to discuss the consequences of blockchain features for smart contract functionality. Issues such as transaction times, n/2 attacks, and transaction cost can be explored. (In Chapter II, n/2 attacks are discussed in detail.) LED switching can also provide a starting point for discussions about the applicability of blockchain technology for cyber-physical systems generally. Examples include ICS applications, energy grids, satellite systems, and warfighting technology [359]–[363].

F. CHAPTER SUMMARY

This chapter described the BlockGrid prototype’s layout, components, and function in detail. Opportunities for instruction during the setup and operation of the prototype were also covered. The next chapter will address conclusions of the work and opportunities for future research.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSIONS AND FUTURE WORK

This chapter discusses the conclusions of the thesis project and identifies opportunities for future work.

A. CONCLUSIONS

Chapter I outlined three principal goals for this work:

1. To determine how a cyber-physical system could be constructed to enable demonstration of blockchain technology.

2. To construct a blockchain-controlled cyber-physical system that could function as a demonstration platform prototype.

3. To generate the guidelines necessary for educators to assemble and use the prototype as a blockchain technology demonstration platform.

The first goal was met through the research and testing conducted during the BlockGrid prototype's development. In Chapter II, the concept of a BlockGrid demonstration platform was defined and its system requirements were enumerated. Chapter III detailed the implementation options for each system feature were evaluated and system features options most consistent with portability, durability, low-cost, and ease-of-use were identified.

The second goal was accomplished through successful implementation of the BlockGrid prototype. Chapter V described how the BlockGrid prototype allows an instructor to demonstrate blockchain principles at a fundamental level for beginning students using a private Ethereum blockchain run on a cluster of Raspberry Pi's and a VM running on a user-provided laptop. The prototype provides observers with a visual demonstration of the steps involved in smart contract execution by allowing an operator to use a smart contract to control LEDs attached to the Raspberry Pis.

The BlockGrid prototype has been successfully demonstrated in several environments, including RIMPAC (Rim of the Pacific, a U.S. and Pacific allies annual naval exercise) 2018 and Discover NPS Day 2019. The prototype has also been used as a research tool for students within the NPS Computer Science Department.

The final goal will be accomplished through publication of this thesis and the included supporting materials. This thesis has been structured through its chapters to:

(1) present the motivation for creating an effective blockchain technology demonstration platform;

(2) define a BlockGrid as a suitable model to meet the demonstration platform need;

(3) explain the currently available implementation choices to consider when constructing a BlockGrid and why specific implementation choices were made for the BlockGrid prototype; and

(4) describe the prototype’s design, function, and operation.

This thesis, its appendices, and supplemental materials, provide educators, students, and researchers the information and tools necessary to pursue instruction and experimentation on the application of blockchain technology to cyber-physical systems.

B. FUTURE WORK

There are several opportunities for future work to extend the effort described here. The first is to investigate the BlockGrid prototype’s effectiveness as a teaching tool using quantifiable methods. For example, a pedagogical experiment could be constructed in which similar lessons on blockchain technology were delivered to student groups with and without a prototype demonstration component. Students could be assessed on their knowledge of blockchain technology before and after each lesson and the difference in the assessment results for lessons delivered with and without the use of the demonstration prototype could be compared. Such an experiment could yield evidence metrics regarding the efficacy of the prototype as a teaching tool. It could also yield insights into how the prototype could be improved to increase its effectiveness as a demonstration platform.

Second, ways to augment and revise the prototype or to construct new BlockGrid prototypes with different feature implementations could be explored. One design choice for the prototype worth considering, should the cost of high-performance microcomputers decrease, is deployment of more powerful microcomputer nodes in the BlockGrid network. All nodes could perform mining functions and system performance may be increased.

Such an upgrade might also provide a more compelling demonstration of the distributed features of blockchain technology.

Other work could develop blockchain course materials that incorporate the BlockGrid prototype and leverage its potential to make the course more interactive. Thus, the Blockgrid prototype could be used either for demonstrations, or for interactive laboratory lessons and small research projects.

In summary, this work has resulted in the successful creation of a cyber-physical system that enables concrete demonstration of blockchain technology. It provides tools to meet the need for USG personnel to learn about blockchain technology. This will help USG personnel effectively apply blockchain technology and incorporate it into the nation's cyber defense mechanisms and IT infrastructure.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. BLOCKGRID PROTOTYPE HARDWARE AND SOFTWARE TECHNICAL DETAILS

A. BLOCKGRID PROTOTYPE HARDWARE COMPONENTS

1. User-Provided Laptop

To ensure high performance, it is recommended that a laptop with at least 16 GB of RAM be used to host the SCU VM so that a minimum of 12GB of RAM can be allocated to the SCU VM.

2. Microcomputer Nodes

Six Raspberry Pi 3B microcomputers were used in the BlockGrid prototype.

Available at: https://www.adafruit.com/product/3055

3. Electrical Circuit Components

(1) Breadboards

Six eBoot 400-Point Solderless Circuit Breadboards were used in the BlockGrid prototype. One was used for each Raspberry Pi node.

Available at: https://www.amazon.com/gp/product/B071D7V9HD/

(2) Jumper Cables

RGBZONE Multicolored Dupont Wire Male-to-Female and Male-to-Male jumper cable were used to connect the GPIO units of the Raspberry Pis to their attached breadboard circuits (Female-to-Male) and to connect the ground lanes of the breadboard (Male-to-Male).

Available at: https://www.amazon.com/gp/product/B01M1IEUAF/

(3) Resistors

220 Ω Adafruit Through-Hole Resistors were used to control the current through the breadboard circuit and were placed between the LEDs and the grounding lane of each breadboard.

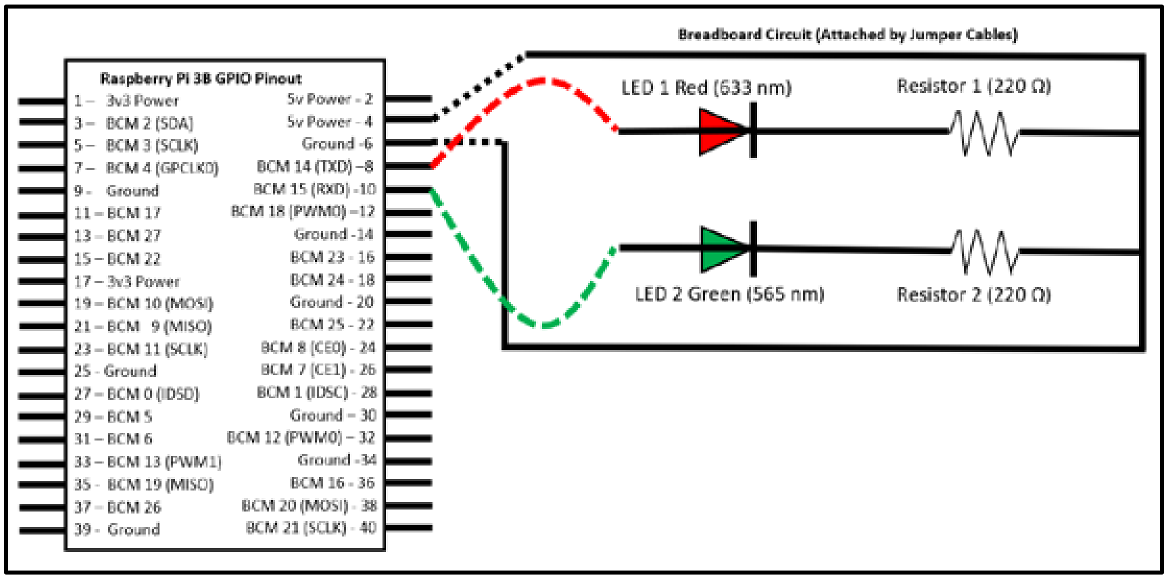
Available at: https://www.amazon.com/Adafruit-Through-Hole-Resistors-Pack-ADA2780/dp/B01GQFV79K/

(4) LEDs

One Red and one green GikFun 5mm LED was used for each of the six Raspberry Pi-attached breadboard circuits in the BlockGrid prototype.

Available at: https://www.amazon.com/gp/product/B01ER728F6/

(5) Circuit Diagram for Raspberry Pi-attached Breadboard Circuits



On the above pinout diagram, the abbreviations stand for the following:

BCM - Broadcom Mode	Ω – Ohm (resistance unit)	v – Volt
CE0 - Chip Select 0	PWM0 - Pulse Width Modulation 0	
CE1 - Chip Select 1	PWM1 - Pulse Width Modulation 1	
GPCLK0 - General Purpose Clock 0	RXD - Receive Data	
IDSD - Internal Data Serial Data	SCA - Single Connector Attachment	
IDSC - Internal Data Serial Control	SCLK - Serial Clock	
MOSI - Master Out Slave In	SD - Secure Digital	
MISO - Master In Slave Out	SDA - Serial Data	
nm – nanometers (10^{-9} meters)	TXD - Transmit Date	

Figure 26. Circuit Diagram For the RPI-attached Circuits in the BlockGrid Prototype. Adapted From [364] and [365].

4. Networking Hardware

(6) Router

A NetGear RangeMax N600 Dual Band Wi-Fi Router was used to route Ethernet traffic between the Raspberry Pi and SCU VM miner nodes in the prototype. The wireless functionality of the router was not used for the prototype.

Available at: https://www.bestbuy.com/site/netgear-rangemax-n600-dual-band-wi-fi-router-black/1208844.p?

(7) Switch

A NetGear 8-Port 10/100/1000 Mbps Gigabit Unmanaged Ethernet switch was used to connect the Raspberry Pi Nodes and SCU laptop to the router.

Available at: https://www.bestbuy.com/site/netgear-8-port-10-100-1000-mbps-gigabit-unmanaged-switch-white/7451964.p?

(8) Cables

· 1.5 ft CAT6 Ethernet Cables were used to connect the RPIs to the switch and to connect the switch to the router.

Available at: https://www.amazon.com/gp/product/B01F3CSJ3E/

· A 3ft CAT6 Ethernet Cable was used to connect the SCU laptop to the router.

Available at: https://www.amazon.com/StarTech-com-Yellow-Snagless-Patch-N6PATCH3YL/dp/B003TSTDIA/

· An Insignia USB 3.0-toGigabit Ethernet Adapter was used to connect the Ethernet cable to laptops lacking an Ethernet port.

Available at: https://www.insigniaproducts.com/pdp/NS-PU98635/3510527

5. Additional Hardware

(9) Power Tower

The BlockGrid prototype incorporates nine devices that require power from a wall outlet. These are the six Raspberry Pi nodes, the router and switch, and the user-provided laptop. In order to provide the required number of outlets for all of the BlockGrid prototypes devices, a LidLife Power Strip, Tower 12 Surge Protector Electric Charging Station with 5 USB ports was used. This Power Tower allows all of the BlockGrid prototype hardware to be powered from one wall outlet and to fit conveniently on a tabletop.

Available at: https://www.amazon.com/gp/product/B073S6SKDF/

(10) Raspberry Pi/Breadboard Scaffold

In order to enable the BlockGrid prototype's six Raspberry Pi nodes and attached breadboard circuits to be arranged conveniently on a conventional table top, two GeauxRobot 6-Layer Dog Bone Stack Clear Case Box Enclosures were used to stack the Raspberry Pi's and breadboard circuits vertically.

Available at: https://www.amazon.com/gp/product/B01D9130QC/

(11) Pelican Cases

To allow the BlockGrid prototype to be safely and conveniently transported between demonstration locations, two Pelican-style SRA 37–6-BK Watertight Cases with Foam 18 x 13 x 7 inches were customized to fit the BlockGrid protype's hardware components.

Available at: https://www.amazon.com/gp/product/B01MZ75AUW/

(12) Micro SD Memory Cards

To store the Raspberry Pi nodes' operating systems and the blockchain software stored therein, each Raspberry Pi node was provisioned an SD memory card. SanDisk Ultra 16GB Class 10 SDHC UHS-I Memory Card up to 80MB/s (SDSDUNC-016G-GN6IN) memory cards were used.

Available at: https://www.amazon.com/SanDisk-Class-UHS-I-Memory-SDSDUNC-016G-GN6IN/dp/B0143RTB1E/

B. BLOCKGRID PROTYPE SOFTWARE COMPONENTS

1. Blockchain Software

Ethereum was selected as the blockchain protocol for the BlockGrid prototype. Geth, the Go language implementation of Ethereum, was used for the prototype's private blockchain.

Available at: https://geth.ethereum.org/install/

It may also be convenient to perform the installation from device command line using a package manager such as Brew [52]. If using Brew, Ethereum can be installed with the following command:

```
brew install ethereum
```

2. System Control Unit OS

Ubuntu 16.04 (Xenial Xerus) was used as the OS for the SCU in the BlockGrid prototype. The SCU was run as a VM on the user-provided laptop. A copy of the VM used for the prototype is provided in the supplemental materials section for this thesis. The Ubuntu 16.04 OS can be downloaded directly as well.

Available at: https://releases.ubuntu.com/16.04/

3. Blockchain API Software

Two Blockchain APIs are available on the BlockGrid prototype through which Ether currency and smart contract transactions can be executed.

(1) Web3

Web3 is a Javascript API that can be conveniently used from the command line. Web3 came pre-installed with Geth on the SCU VM. It was installed manually on the Raspberry Pi nodes.

Available at: https://web3js.readthedocs.io/en/v1.2.7/getting-started.html

The installation is easiest to perform with a package manager such as NPM (Node Package Manager) with the following command:

```
npm install web3
```

The specific web3 version used in the BlockGrid prototype was 0.20.1 and can be installed from the command line with the following command [365]:

```
npm install web3@0.20.1
```

(2) Ethereum Wallet

Ethereum Wallet is a GUI-based API that provides easy-to-interpret visual displays of the steps involved in blockchain transactions. Ethereum Wallet was installed on the SCU VM.

Available at: https://github.com/ethereum/mist/releases

4. Smart Contract Development Software

The Truffle SDK was used to compile the Solidity-language SmartSwitch smart contract. It was installed on the SCU VM [365].

Available at: https://www.trufflesuite.com/

Truffle can also be installed from the command line with a package manager such as NPM with the following command:

```
npm install -g truffle
```

5. SmartSwitch Smart Contract

The SmartSwitch smart contract for the BlockGrid prototype was taken from the "Create and Deploy a Smart Contract" Chainskills tutorial by Said Eloudrhiri referenced in Appendix B [330]. In the Tutorial the smart contract is given the name SmartToken, and was renamed SmartSwitch in the BlockGrid prototype to more clearly convey the intention to use the smart contract's functions to turn the prototype's LEDs on and off.

Available at: https://chainskills.com/2017/04/03/create-and-deploy-a-smart-contract-66/

SmartSwitch Code (from Eloudrhiri tutorial):

```
pragma solidity ^0.4.0;

contract SmartSwitch {
     mapping(address => uint) tokens;
     event OnValueChanged(address indexed _from, uint _value);

     function depositToken(address recipient, uint value) returns
     (bool success) {
          tokens [recipient] += value;
          OnValueChanged(recipient, tokens [recipient]);
          return true;
     }

     function withdrawToken(address recipient, uint value) returns
     (bool success) {
          if (int(tokens [recipient] - value) < 0) {
               tokens [recipient] = 0;
          } else {
               tokens [recipient] -= value;
          }
          OnValueChanged(recipient, tokens [recipient]);
          return true;
     }

     function getTokens(address recipient) constant returns (uint
     value) {
          return tokens [recipient];
     }
}
```

6. Microcomputer Software

(3) Raspberry Pi Operating System

Raspbian was chosen to be the OS for the Raspberry Pi nodes.

Available at: https://www.raspberrypi.org/downloads/raspberry-pi-os/

(4) Additional Software

- Node.js – Node.js is a Javascript Runtime Environment that was installed so that the smart_token.js script could run on the Raspberry Pi nodes.

Available at: https://nodejs.org/en/

Node.js can be installed from the Raspbian Terminal Application command line with the following command [365]:

```
sudo apt-get install nodejs
```

· onoff – onoff is a Node.js module that was installed on the Raspberry Pi nodes to allow the smart_token.js script to control the Raspberry Pi's GPIO pins.

Available at: https://www.npmjs.com/package/onoff

onoff can be installed from the Raspbian Terminal Application command line using a package manager such as npm with the following command [365]:

```
npm install onoff
```

b. smart_token.js script

The smart_token.js script for the BlockGrid prototype was adapted from the "Raspberry Pi and Ethereum: The Epilogue" Chainskills tutorial by Said Eloudrhiri referenced in Appendix B [365]. The script was copied to every Raspberry Pi node's home directory.

Available at: https://chainskills.com/2017/04/10/raspberry-pi-and-ethereum-the-epilogue/

smart_token.js script code (adapted from Eloudrhiri tutorial):

```
// Interaction with Raspberry Pi GPIO
var Gpio = require('onoff').Gpio

// Interaction with Ethereum
var Web3 = require('web3')
var web3 = new Web3()

// connect to the local node
web3.setProvider(new
web3.providers.HttpProvider('http://localhost:8042'))

// SmartSwitch Smart Contract Address
var contractAddress = '0xc1be8139ae33eb8f4e624e3a239077510eeb83f2'

// Define the ABI (Application Binary Interface)
var ABI = JSON.parse('[{"anonymous":false,"inputs":
[{"indexed":true,"name":"_from","type":"address"},
{“indexed”:false,”name”:”_value”,”type”:”uint256”}],
“name”:”OnValueChanged”,”type”:”event”},
{“constant”:false,”inputs”:[{“name”:”recipient”,”type”:”address”}
,
{“name”:”value”,”type”:”uint256”}],
“name”:”depositToken”,”outputs”:[{“name”:”success”,”type”:”bool”}
],
“payable”:false,”stateMutability”:”nonpayable”,”type”:”function”}
,
{“constant”:false,”inputs”:[{“name”:”recipient”,”type”:”address”}
,
{“name”:”value”,”type”:”uint256”}],”name”:”withdrawToken”,
“outputs”:[{“name”:”success”,”type”:”bool”}],
“payable”:false,”stateMutability”:”nonpayable”,”type”:”function”}
,
{“constant”:true,”inputs”:[{“name”:”recipient”,”type”:”address”}]
,
“name”:”getTokens”,”outputs”:[{“name”:”value”,”type”:”uint256”}],
“payable”:false,”stateMutability”:”view”,”type”:”function”}]’)

// Create a contract object
var contract = web3.eth.contract(ABI).at(contractAddress)

// components connected to the RPI
var redLed = new Gpio(8, ‘out’)
var greenLed = new Gpio(10, ‘out’)


// display initial state
showStatus()

// wait for an event triggered on the Smart Contract
var     onValueChanged     =     contract.OnValueChanged({_from:
web3.eth.coinbase});

onValueChanged.watch(function(error, result) {
     if (!error) {
          showStatus()
     }
})

// power the LED according the value of the token
function showStatus() {

     // retrieve the value of the token
     var token = contract.getTokens(web3.eth.coinbase)

     // display the LED according the value of the token
     if (token > 0) {
          // Green: you have enough token
```

```
                redLed.writeSync(0)
                greenLed.writeSync(1)

        } else {
                // Red: not enough token
                greenLed.writeSync(0)
                redLed.writeSync(1)
        }
}

// release process
process.on(‘SIGINT’, function () {
        greenLed.unexport()
        redLed.unexport()
        button.unexport()
})
```

APPENDIX B. BLOCKGRID PROTOTYPE INSTALLATION AND CONFIGURATION GUIDE

To install and configure the BlockGrid Prototype, the Chainskills tutorial series "Create a private Etheruem blockchain with IoT devices" by Said Eloudrhiri was followed with some deviations and adaptions to suit the BlockGrid prototype's software and physical requirements [52], [298], [330], [365]–[368]. The deployment of the Ethereum blockchain, smart contract, and attached circuit described in these tutorials should allow a reader to generate a BlockGrid of their own preference or reproduce the BlockGrid prototype.

The Chainskills course consists of a series of 7 tutorials that walk through how to deploy a private Ethereum blockchain using a laptop and a Raspberry Pi. The course provides a sample smart contract SmartToken, a script smart_token.js, and a circuit schematic that were all adapted with minor changes for use in the BlockGrid prototype. The deviations from each tutorial necessary to construct the BlockGrid protype are described in this Appendix. PDFs of the tutorials have also been included in the supplemental materials.

1. "Create a private Etheruem blockchain with IoT devices" [298]

Available at: https://chainskills.com/2017/02/24/create-a-private-ethereum-blockchain-with-iot-devices-16/

While the BlockGrid prototype makes no use of internet connections once assembled, during the initial setup of prototype it is necessary to download software components for Raspbian and Ethereum from the internet during installations. If internet access is restricted at the assembly site, the necessary software can be installed from a USB drive or SD card on the Raspberry Pi, with acquisition of the necessary software following facility protocols. If a USB drive or SD card is used, Step 5 of the tutorial "Setup Wi-Fi" may be skipped.

If replicating the prototype with its isolated Ethernet Network, Step 6 will also differ from the tutorial. To replicate the prototype the COTS router described in Appendix A must be configured to assign a static IP address to each port. For the BlockGrid prototype, these were assigned 192.168.1.1 through 192.168.1.9. SSH will still be used to connect the Nodes to the SCU as described in Step 6, but this will happen across an Ethernet connection rather than WiFi.

To fully replicate the prototype, repeat the installation steps to install Ethereum on all six Raspberry Pi nodes.

2. "Install an Ethereum node on a computer" [52]

Available at: https://chainskills.com/2017/03/03/install-a-ethereum-node-on-a-computer-26/

If using the SCU VM provided, the Ethereum installation is already complete and this tutorial does not need to be followed. If not, the tutorial can be followed step for step as no deviations are required to install Ethereum on a BlockGrid SCU.

3. "Set up the private chain - miners" [366]

Available at: https://chainskills.com/2017/03/10/part-3-setup-the-private-chain-miners/

If using the SCU VM provided, the miners installation is already complete and this tutorial does not need to be followed. Note the password for the miner node accounts on the SC VM is "password."

If running through the tutorial, omit the ending "`--ipcpath "~/Library/Ethereum/geth.ipc"`" of the miner setup command and the startminer1.sh script to ensure proper initiation of the miners.

4. "Pair the miners" [367]

Available at: https://chainskills.com/2017/03/17/pair-the-miners-46/

If using the SCU VM provided, the miner pairing installation is already complete and the installations in this tutorial are unnecessary. However, even though the installation is unnecessary if using the SCU VM, it is important to do steps 1.1-1.3 to "Clean your miners" periodically to synchronization and transaction times for the prototype.

If following the tutorial to create a BlockGrid, it is necessary to add "admin" and "personal" to the –rpcapi flag list in the startminer.sh scripts in order to successfully run the suggested admin.nodeInfo.enode and personal.unlockAccount commands. The list should read:

```
–rpcapi "admin,personal,db,eth,net,web3,miner"
```

5. “Synchronize the Raspberry PI with the Private Blockchain” [368]

Available at: https://chainskills.com/2017/03/27/synchronize-the-raspberry-pi-with-the-private-blockchain-56/

Some minor deviations were made in the instructions for this tutorial to set up the BlockGrid prototype. To initialize a BlockGrid node, run a “`geth --datadir ~/node init ChainSkills/genesis.json`” command rather than the “`geth --datadir ~/ChainSkills/node init ../genesis.json`” command recommended by the tutorial. Additionally, use this this version of startnode.sh script rather than the one recommended by the tutorial:

```
#!/bin/bash
geth --identity "node3" --fast --networkid 4533 --datadir
/home/pi/node --nodiscover --rpc --rpcport "8042" --rpcapi
"personal,admin,db,eth,net,web3,miner"  --port  "30303"  --
unlock 0 --password "/home/pi/node/password.sec" --ipcpath
/home/pi/.ethereum/geth.ipc
```

Also note that to sync multiple nodes to the chain the clocks on all nodes have to be synchronized. This can be achieved with the following command:

```
ssh pi <ip addr> sudo date – set \”$(date)\”
```

6. Tutorial 6- “Create and deploy a Smart Contract” [330]

https://chainskills.com/2017/04/03/create-and-deploy-a-smart-contract-66/

If using the SCU VM provided, the installation of the SmartSwitch smart contract is already complete and the installations in this tutorial are unnecessary.

If following the tutorial to deploy a smart contract, several deviations from the tutorial are recommended. First, it is recommended to define and lower both Gas and gasPrice in order for the “truffle migrate –reset” command to work. The truffle.js file should look something like this:

```
module.exports = {
  networks: {
     development: {
     host: "127.0.0.1",
     port: 8042,
     network_id: "4533", // Match this to the network id
     gas: 1000000, // this can be adjusted
     gasPrice: 1 // this can be adjusted
     }
  }
};
```

Another recommended deviation is to install Ethereum Wallet chain-monitoring GUI software rather the Mist software recommended by the tutorial. To migrate raw keys to enable Ethereum Wallet to interface with the blockchain, use myEtherWallet software to generate the needed keys. Once generated, they can be migrated into the blockchain using the following command:

```
web3.personal.importRawKey("<private key from
        myEtherWallet>","<password>")
```

7. "Raspberry Pi and Ethereum: The Epilogue" [365]

Available at: https://chainskills.com/2017/04/10/raspberry-pi-and-ethereum-the-epilogue/

To replicate the BlockGrid Prototype, a circuit will need to be created for each of the six Raspberry Pi Nodes. The supplies required to build the BlockGrid Prototype circuit apparatus are detailed in Section A of Appendix A.

To create circuit layouts like those found on the BlockGrid prototype, some deviations need to be made from the tutorial. For example, the prototype Raspberry Pi nodes utilize GPIO pins 8 and 10 on the Raspberry Pi nodes rather than those specified by the tutorial. Additionally, the tutorial incorporates a switch element into the breadboard circuit that was not used with the BlockGrid prototype.

For software, the smart_token.js script used for the prototype, given in section B of Appendix A, differs from the script presented in the tutorial. For example, the ABI definition in the script used for the prototype is:

```
var ABI = JSON.parse('[{"anonymous":false,"inputs":
[{"indexed":true,"name":"_from","type":"address"},
{"indexed":false,"name":"_value","type":"uint256"}],
“name”:”OnValueChanged”,”type”:”event”},{“constant”:false,”
inputs”:[{“name”:”recipient”,”type”:”address”},{“name”:”val
ue”,”type”:”uint256”}],“name”:”depositToken”,”outputs”:[{“n
ame”:”success”,”type”:”bool”}],“payable”:false,”stateMutabi
lity”:”nonpayable”,”type”:”function”},{“constant”:false,”in
puts”:[{“name”:”recipient”,”type”:”address”},{“name”:”value
”,”type”:”uint256”}],”name”:”withdrawToken”,“outputs”:[{“na
me”:”success”,”type”:”bool”}],“payable”:false,”stateMutabil
ity”:”nonpayable”,”type”:”function”},{“constant”:true,”inpu
ts”:[{“name”:”recipient”,”type”:”address”}],“name”:”getToke
ns”,”outputs”:[{“name”:”value”,”type”:”uint256”}],“payable”
:false,”stateMutability”:”view”,”type”:”function”}]’)
```

Rather than the ABI given by the tutorial:

```
var ABI= JSON.parse('[{"constant":false,"inputs":[{"name":
"recipient","type":"address"},{"name":"value","type":"uint2
56"}],"name":"depositToken","outputs":[{"name":"success","t
ype":"bool"}],"payable":false,"type":"function"},{"constant
":true,"inputs":[{"name":"recipient","type":"address"}],"na
me":"getTokens","outputs":[{"name":"value","type":"uint256"
}],"payable":false,"type":"function"},{"constant":false,"in
puts":[{"name":"recipient","type":"address"},{"name":"value
","type":"uint256"}],"name":"withdrawToken","outputs":[{"na
me":"success","type":"bool"}],"payable":false,"type":"funct
ion"},{"anonymous":false,"inputs":[{"indexed":false,"name":
"status","type":"uint256"}],"name":"OnStatusChanged","type"
:"event"},{"anonymous":false,"inputs":[{"indexed":true,"nam
e":"_from","type":"address"},{"indexed":false,"name":"_valu
e","type":"uint256"}],"name":"OnValueChanged","type":"event
"}]')
```

The prototype script also uses GPIO pins 8 and 10 rather than the 14, 15, and 18 pins in the tutorial script. The following switch button logic is omitted from the prototype script as well:

```
// watch event on the button
button.watch(function (err, value) {
     if (err) {
          throw err
      }
showStatus()
})
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. BLOCKGRID SETUP AND OPERATION INSTRUCTIONS

This appendix is intended to serve as an instructor's manual to enable setup and operation of the BlockGrid prototype. Detailed instructions for the physical setup and software instructions are provided, along with accompanying video files that walk through each step. Links to YouTube-hosted copies of the videos are provided. Mp4 files of the video walkthroughs are also included as supplemental materials within the online thesis repository.

A. BLOCKGRID SETUP INSTRUCTIONS

A video of the setup instructions, prepared by the author, is viewable available at: https://www.youtube.com/watch?v=jCZKYr2eoeE&t=4s)

An mp4 file of the setup instructions video has also been included in the supplemental materials for this thesis.

The BlockGrid prototype comes stored in two Pelican cases, as shown in Figure 27:

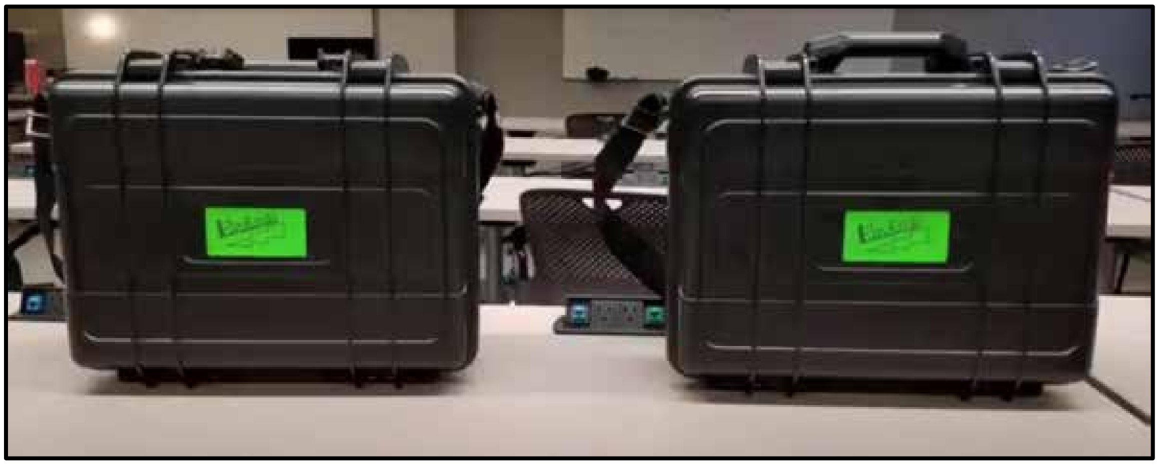


Figure 27. Screenshot From Prototype Setup Instructional Video of BlockGrid Prototype Pelican Cases

One case contains two Raspberry Pi (RPI)/breadboard towers, an Ethernet switch, and power adapters. See Figure 28 and 00:07 through 00:10 of the BlockGrid Setup Instructions video for guidance. Figure 28 shows the contents of the first case.

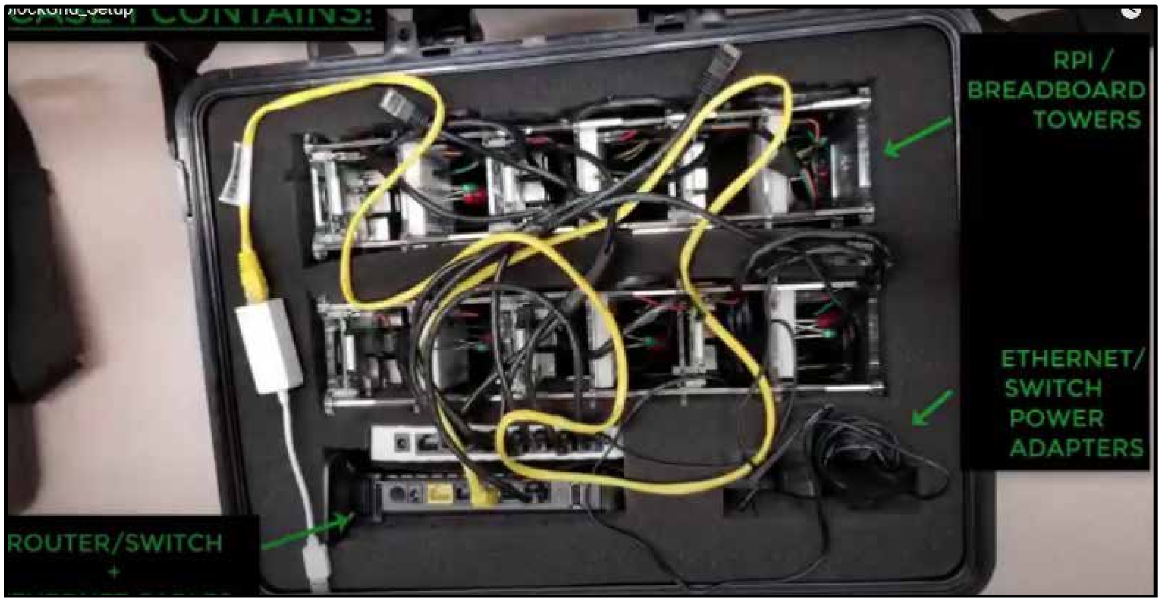


Figure 28. Screenshot from Prototype Setup Instructional Video of Pelican Case Interior Showing RPi Tower and Router Equipment.

The other case contains Raspberry Pi power adapters and spare parts. See Figure 29 and 00:12 through 00:15 of the BlockGrid Setup Instructions video for guidance. Figure 29 shows the contents of the second case.

Connect the BlockGrid Prototype Hardware

Step 1 - After unpacking, position the power tower and plug in the Raspberry Pi and Router/Switch power adapters. See Figure 30 and 00:17 through 00:45 of the BlockGrid Setup Instructions video for guidance. Figure 30 shows Raspberry Pi power adapters being plugged into the power tower.

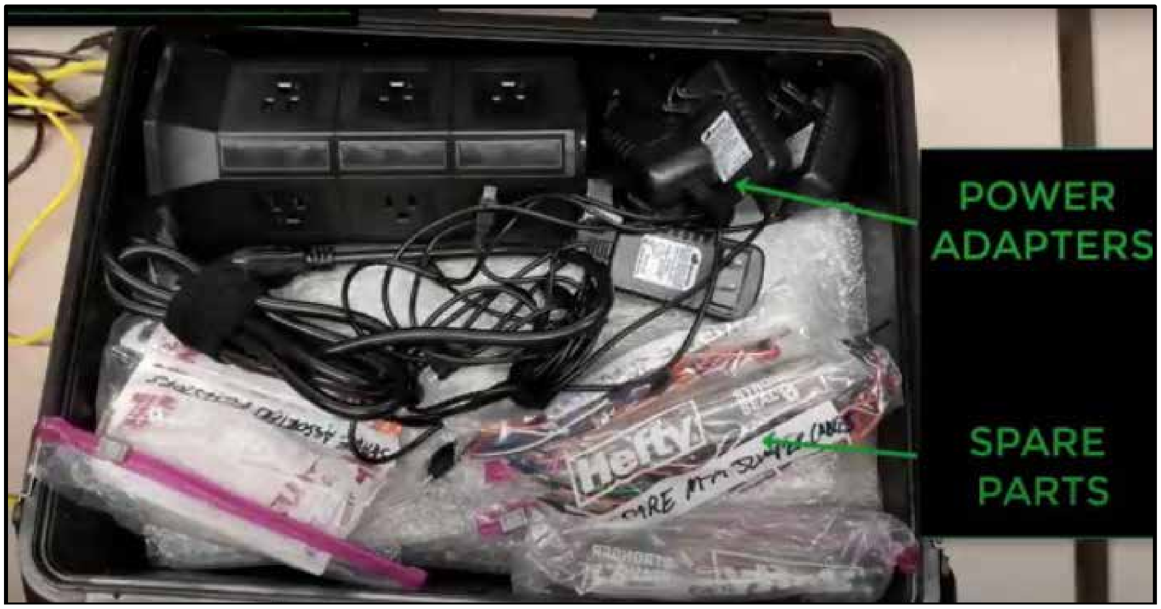


Figure 29. Screenshot from Prototype Setup Instructional Video of Pelican Case Interior Showing Power Adapters and Spare Parts.

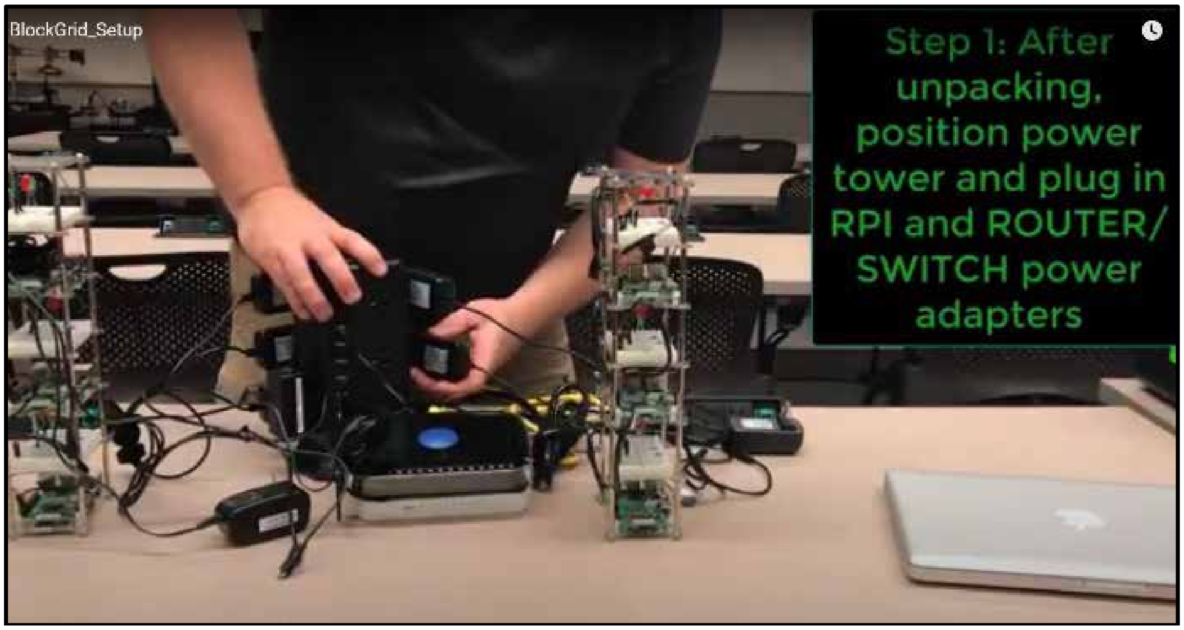

BlockGrid_Setup
Step 1: After unpacking, position power tower and plug in RPI and ROUTER/ SWITCH power adapters

Figure 30. Screenshot from Prototype Setup Instructional Video Showing how to position RPi Tower and Router Equipment.

Step 2 - Plug in Router/Switch and connect Ethernet cables. See Figure 31 and 00:45 through 01:20 of the BlockGrid Setup Instructions video for guidance. Figure 31 shows Ethernet cables being connected from the Raspberry Pi nodes to the Ethernet switch.

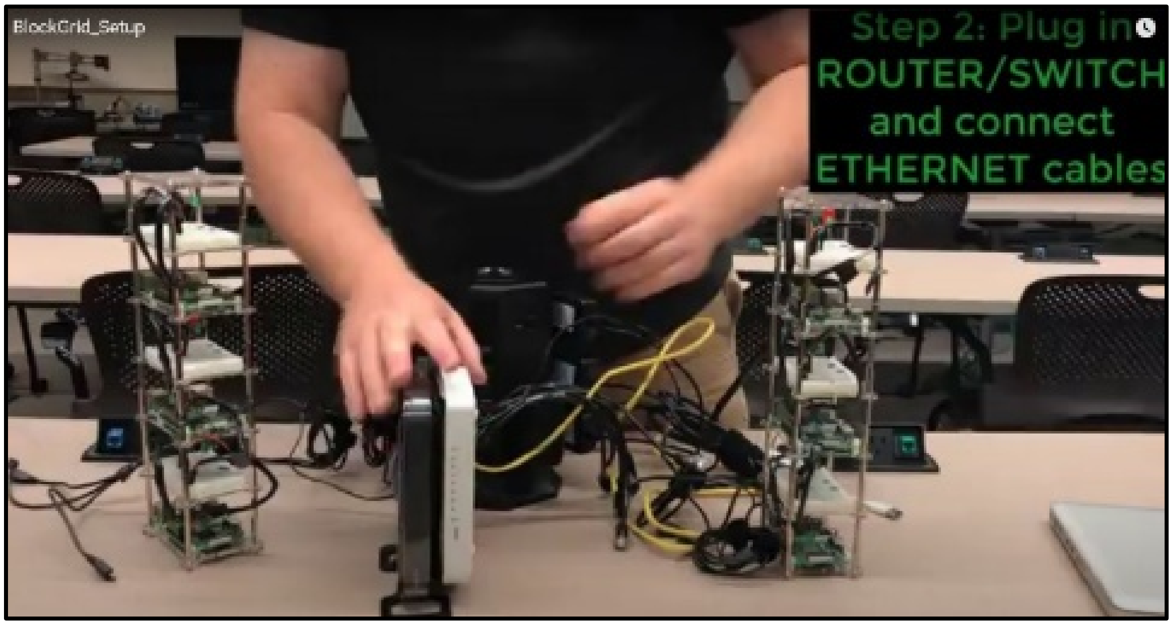


Figure 31. Screenshot from Prototype Setup Instructional Video Showing How to Connect Ethernet Cables to Create Internal Network.

Step 3 - Plug in the Raspberry Pi power adapters to the Raspberry Pis. See Figure 32 and 01:21 through 01:43 of the BlockGrid Setup Instructions video for guidance. Figure 32 shows Raspberry Pi power adapters being plugged into the Raspberry Pis.

Step 4 - Plug in the Ethernet cable to laptop. See Figure 33 and 01:43 through 01:48 of the BlockGrid Setup Instructions video for guidance. Figure 33 shows an Ethernet cable connecting the COTS router to the SCU laptop being plugged in.

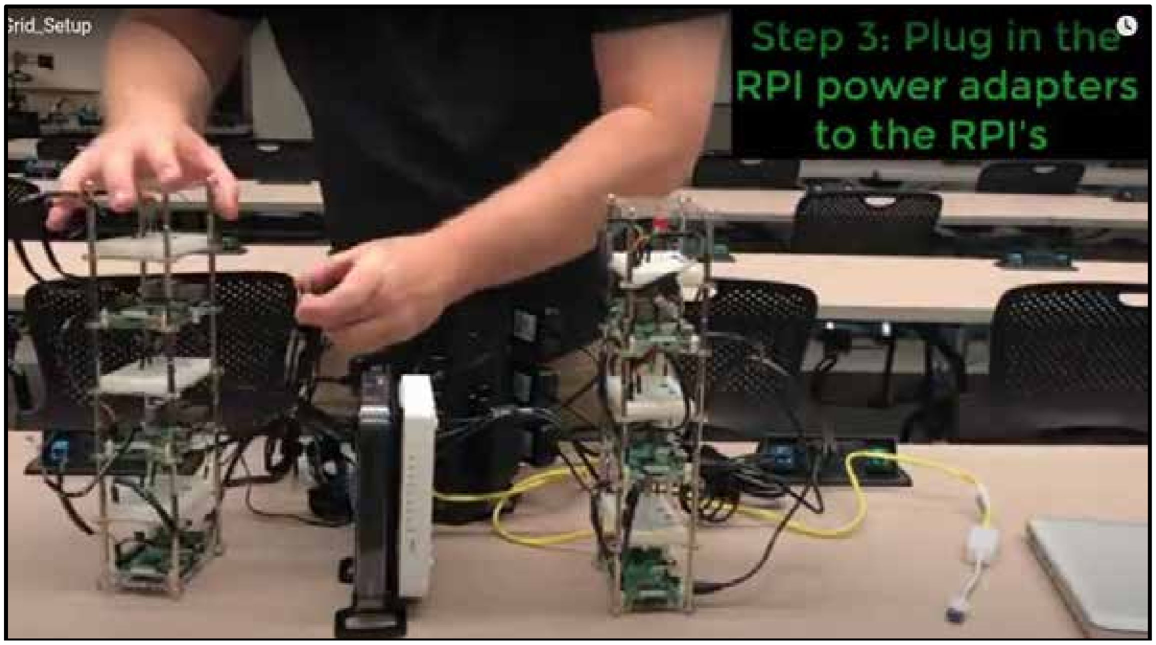


Figure 32. Screenshot from Prototype Setup Instructional Video Showing How to Power on Raspberry Pis.

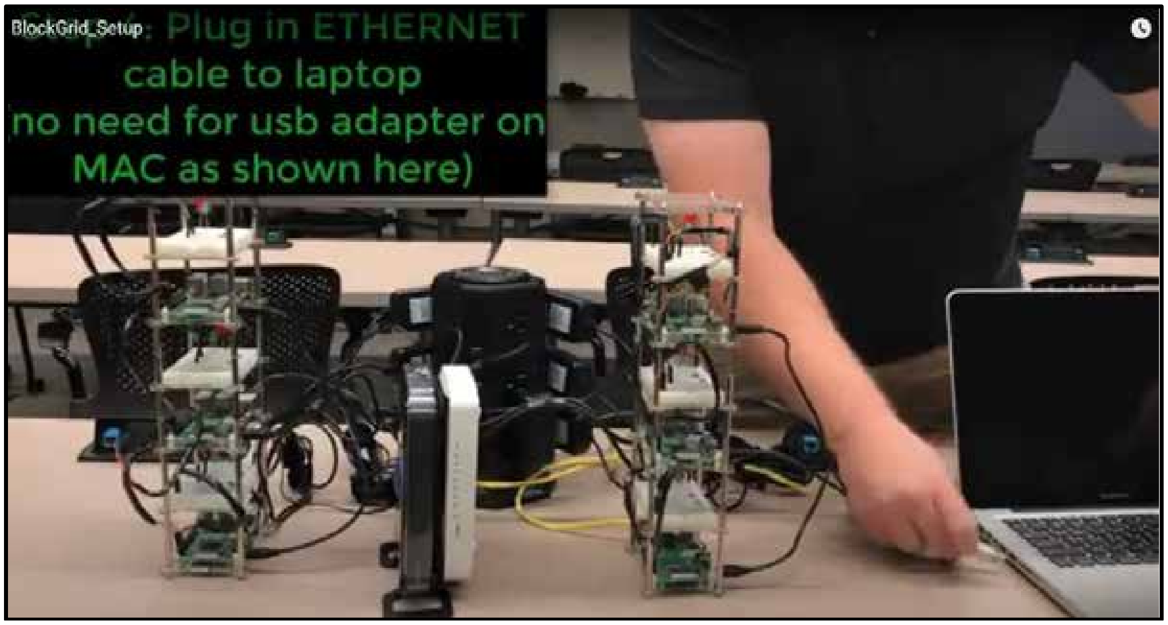


Figure 33. Screenshot from Prototype Setup Instructional Video Showing How to Connect Ethernet Cables to User Laptop to Complete Internal Network.

Initialize the Ethereum Blockchain from within the SCU VM

Step 1 – From within the SCU VM's Terminal Application, synchronize RPI clocks to the mining VM. See Figure 34 and 01:49 through 02:23 of the BlockGrid Setup Instructions video for guidance. Figure 34 shows a screenshot of the time syncing commands being entered into the SCU VM. Perform the sync with the following command:

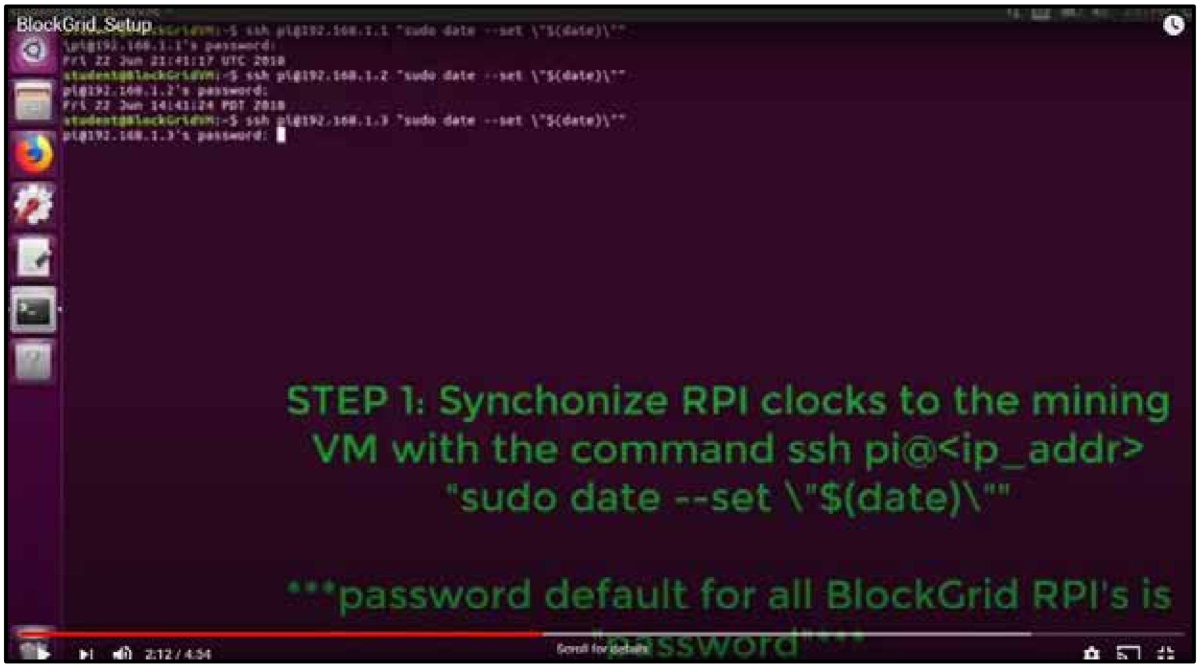
```
ssh pi <ip addr> sudo date – set \”$(date)\”
```

The IP addresses for the Raspberry Pi nodes within the prototype's internal network are as follows (default password for all BlockGrid RPI's is "password"):

Raspberry Pi Node	IP Address	Login Password
Node 1	192.168.1.1	password
Node 2	192.168.1.2	password
Node 3	192.168.1.3	password
Node 4	192.168.1.4	password
Node 5	192.168.1.5	password
Node 6	192.168.1.6	password

Step 2 - Start the miner nodes. See Figure 35 and 02:24 through 02:35 of the BlockGrid Startup Instructions video for guidance. Figure 35 shows a screenshot of miner2 being initialized in the SCU VM. To start the miner nodes, navigate to the ~/miner<#> directory and run the following commands:

```
cd miner<#>
./startminer<#>.sh
```



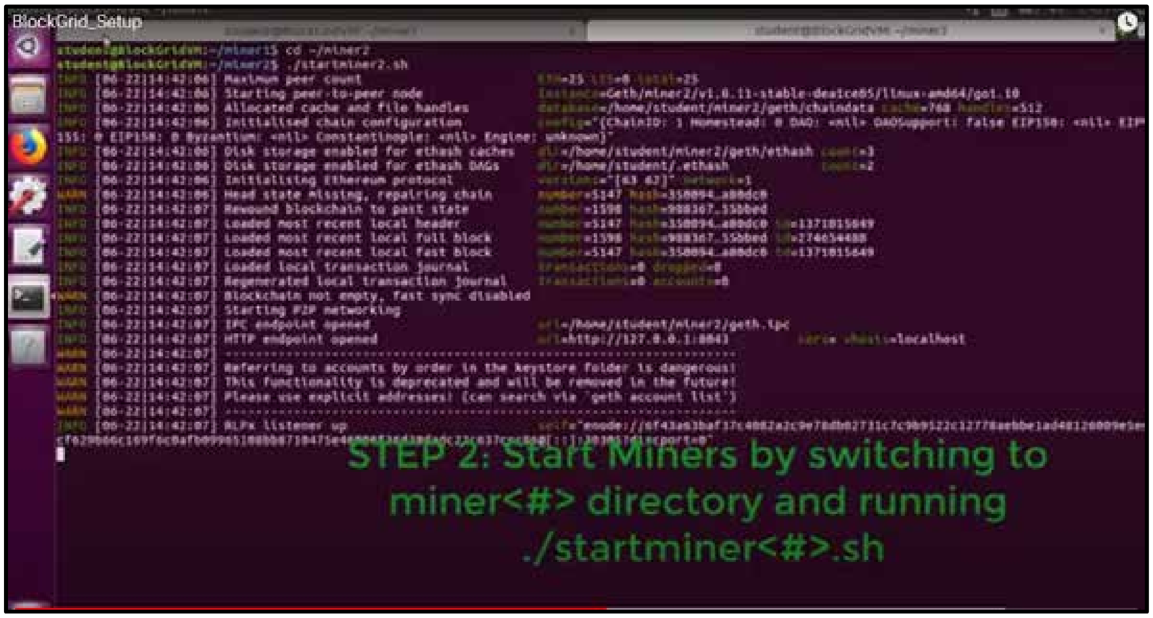
BlockGrid_Setup

```
student@BlockGridVM:~$ ssh pi@192.168.1.1 "sudo date --set \"$(date)\""
\pi@192.168.1.1's password:
Fri 22 Jun 21:41:17 UTC 2018
student@BlockGridVM:~$ ssh pi@192.168.1.2 "sudo date --set \"$(date)\""
pi@192.168.1.2's password:
Fri 22 Jun 14:41:24 PDT 2018
student@BlockGridVM:~$ ssh pi@192.168.1.3 "sudo date --set \"$(date)\""
pi@192.168.1.3's password:
```

STEP 1: Synchonize RPI clocks to the mining VM with the command ssh pi@<ip_addr> "sudo date --set \"$(date)\""

password default for all BlockGrid RPI's is "password"

Figure 34. Screenshot from Prototype Setup Instructional Video Showing How to Synchronize RPI clocks.



BlockGrid_Setup

```
student@BlockGridVM:~/miner1$ cd ~/miner2
student@BlockGridVM:~/miner2$ ./startminer2.sh
INFO [06-22|14:42:06] Maximum peer count                       ETH=25 LES=0 total=25
INFO [06-22|14:42:06] Starting peer-to-peer node               instance=Geth/miner2/v1.8.11-stable-dea1ce05/linux-amd64/go1.10
INFO [06-22|14:42:06] Allocated cache and file handles         database=/home/student/miner2/geth/chaindata cache=768 handles=512
INFO [06-22|14:42:06] Initialised chain configuration          config="{ChainID: 1 Homestead: 0 DAO: <nil> DAOSupport: false EIP150: <nil> EIP
155: 0 EIP158: 0 Byzantium: <nil> Constantinople: <nil> Engine: unknown}"
INFO [06-22|14:42:06] Disk storage enabled for ethash caches   dir=/home/student/miner2/geth/ethash count=3
INFO [06-22|14:42:06] Disk storage enabled for ethash DAGs     dir=/home/student/.ethash            count=2
INFO [06-22|14:42:06] Initialising Ethereum protocol           versions="[63 62]" network=1
WARN [06-22|14:42:06] Head state missing, repairing chain      number=5147 hash=350094…a00dc0
INFO [06-22|14:42:07] Rewound blockchain to past state         number=1598 hash=908367…55bbed
INFO [06-22|14:42:07] Loaded most recent local header          number=5147 hash=350094…a00dc0 td=1371015049
INFO [06-22|14:42:07] Loaded most recent local full block      number=1598 hash=908367…55bbed td=274654408
INFO [06-22|14:42:07] Loaded most recent local fast block      number=5147 hash=350094…a00dc0 td=1371015049
INFO [06-22|14:42:07] Loaded local transaction journal         transactions=0 dropped=0
INFO [06-22|14:42:07] Regenerated local transaction journal    transactions=0 accounts=0
WARN [06-22|14:42:07] Blockchain not empty, fast sync disabled
INFO [06-22|14:42:07] Starting P2P networking
INFO [06-22|14:42:07] IPC endpoint opened                      url=/home/student/miner2/geth.ipc
INFO [06-22|14:42:07] HTTP endpoint opened                     url=http://127.0.0.1:8043          cors= vhosts=localhost
WARN [06-22|14:42:07] ----------------------------------------------------------------
WARN [06-22|14:42:07] Referring to accounts by order in the keystore folder is dangerous!
WARN [06-22|14:42:07] This functionality is deprecated and will be removed in the future!
WARN [06-22|14:42:07] Please use explicit addresses! (can search via `geth account list`)
WARN [06-22|14:42:07] ----------------------------------------------------------------
INFO [06-22|14:42:07] RLPx listener up                         self="enode://6f43a63baf37c4082a2c9e78db02731c7c9b9122c12778aebbe1ad4812609e5e
cf629b66c169f6c0afb09965180b0871047 5e4[illegible]@[::]:30303?discport=0"
```

STEP 2: Start Miners by switching to miner<#> directory and running ./startminer<#>.sh

Figure 35. Screenshot from Prototype Setup Instructional Video Showing How to Start Miner Nodes.

Step 3 - Start the RPI nodes. See Figure 36 and 02:36 through 03:20 of the BlockGrid Startup Instructions video for guidance. To start the RPI nodes, connect via SSH to the nodes from the SCU VM Terminal application with the following command:

```
ssh pi@<ip_addr>
```

Once logged into the pi (recall the login password is "password for all prototype RPIs), navigate to the node directory with the following command:

```
cd node
```

Once in the node's ~/node directory, run the startup script with the following command:

```
./startnode.sh
```

Step 4 – Initiate the local smart_token.js scripts on the RPI nodes to begin watching SmartSwitch. See Figure 37 and 03:23 through 03:53 of the BlockGrid Startup Instructions video for guidance. Figure 37 shows a screenshot of the steps required to initialize the smart_token.js scripts on the Raspberry Pi nodes from the SCU VM. To initiate the scripts, fist connect via SSH to the nodes with the following command:

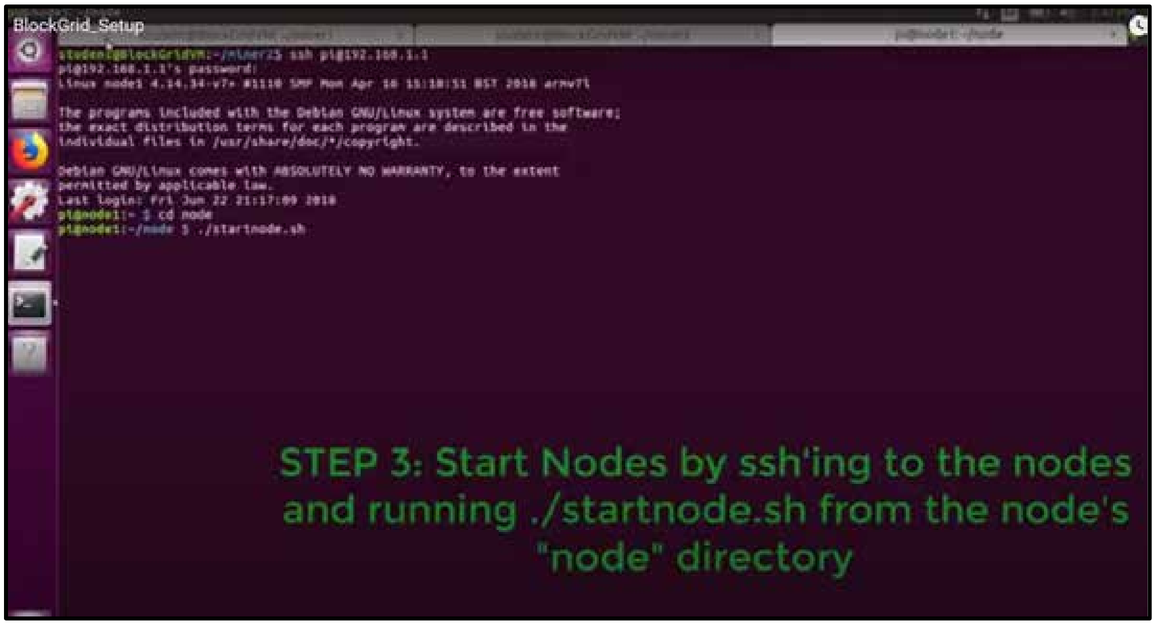
```
ssh pi@<ip_addr>
```

Once logged into the pi (recall the login password is "password for all prototype RPIs), run the following command from each RPI node's home directory:

```
node smart_token.js
```

Step 5 – Launch the Ethereum Wallet chain-monitoring GUI software. See Figure 38 and 03:54 through 04:54 of the BlockGrid Startup Instructions video for guidance. Figure 38 provides two screenshots showing how to launch Ethereum Wallet from the SCU VM. The first screenshot shows the command required launch the software. The second screenshot shows the delay encountered as Ethereum Wallet initializes. Launch Ethereum Wallet by running the following command from the SCU VM's home directory (This takes a very long time to load, approximately 3 minutes):

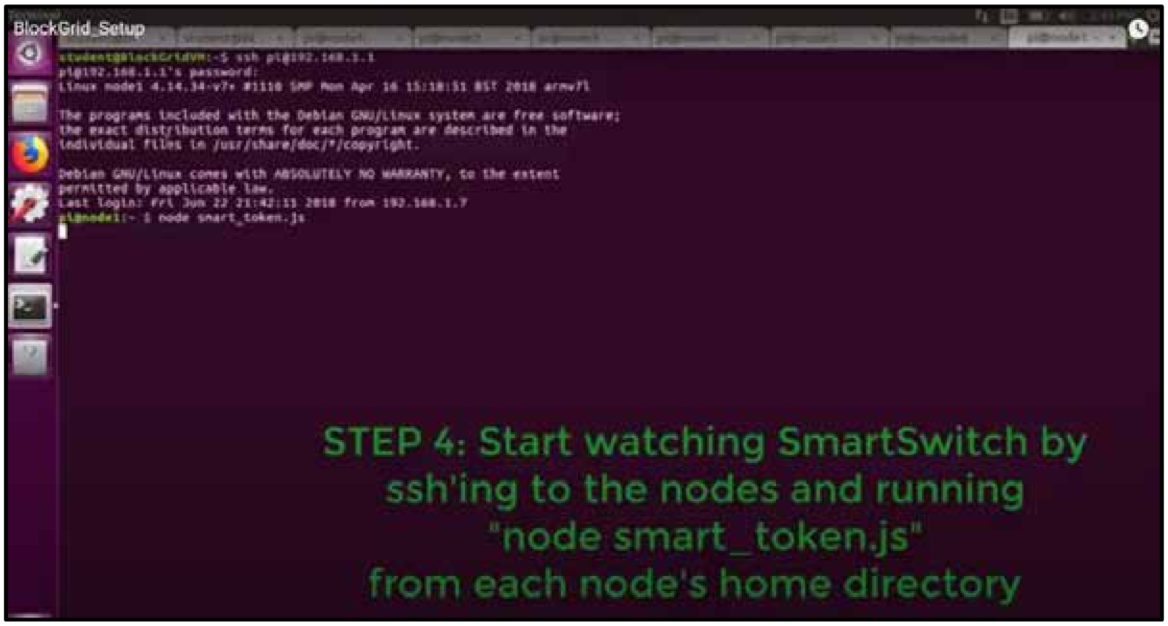
```
./ethereumwallet --network 1 –rpc ~/miner1/geth.ipc
```



```
student@BlockGridVM:~/miner2$ ssh pi@192.168.1.1
pi@192.168.1.1's password:
Linux node1 4.14.34-v7+ #1110 SMP Mon Apr 16 15:18:51 BST 2018 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Fri Jun 22 21:17:09 2018
pi@node1:~ $ cd node
pi@node1:~/node $ ./startnode.sh
```

STEP 3: Start Nodes by ssh'ing to the nodes and running ./startnode.sh from the node's "node" directory

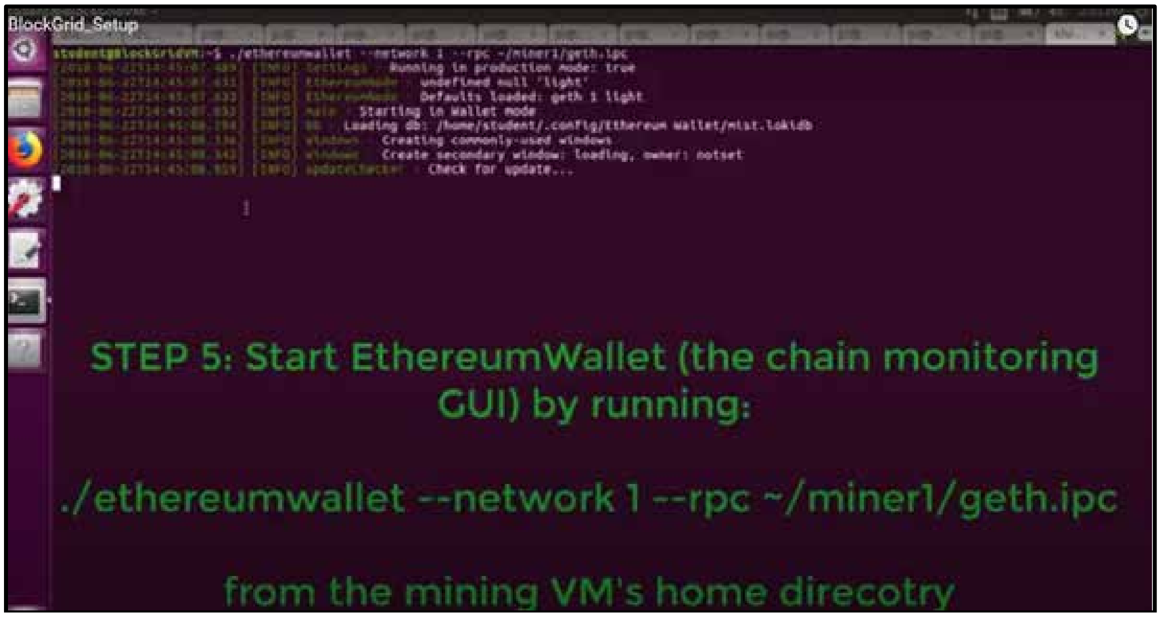
Figure 36. Screenshot from Prototype Setup Instructional Video Showing How to Start Raspberry Pi nodes.



```
student@BlockGridVM:~$ ssh pi@192.168.1.1
pi@192.168.1.1's password:
Linux node1 4.14.34-v7+ #1110 SMP Mon Apr 16 15:18:51 BST 2018 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Fri Jun 22 21:42:11 2018 from 192.168.1.7
pi@node1:~ $ node smart_token.js
```

STEP 4: Start watching SmartSwitch by ssh'ing to the nodes and running "node smart_token.js" from each node's home directory

Figure 37. Screenshot from Prototype Setup Instructional Video Showing How to Start SmartSwitch-watching Local Script.

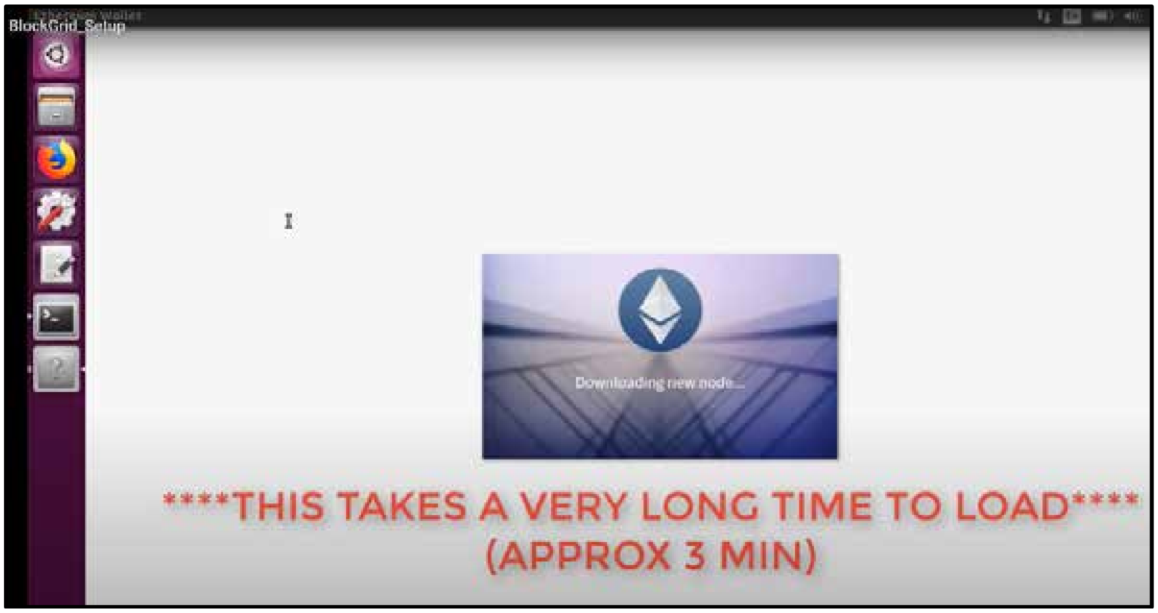


```
student@BlockGridVM:~$ ./ethereumwallet --network 1 --rpc ~/miner1/geth.ipc
[2018-06-22T14:45:07.489] [INFO] Settings - Running in production mode: true
[2018-06-22T14:45:07.631] [INFO] EthereumNode - undefined null 'light'
[2018-06-22T14:45:07.633] [INFO] EthereumNode - Defaults loaded: geth 1 light
[2018-06-22T14:45:07.832] [INFO] Main - Starting in Wallet mode
[2018-06-22T14:45:08.294] [INFO] db - Loading db: /home/student/.config/Ethereum Wallet/mist.lokidb
[2018-06-22T14:45:08.336] [INFO] Windows - Creating commonly-used windows
[2018-06-22T14:45:08.342] [INFO] Windows - Create secondary window: loading, owner: notset
[2018-06-22T14:45:08.929] [INFO] updateChecker - Check for update...
```

STEP 5: Start EthereumWallet (the chain monitoring GUI) by running:

./ethereumwallet --network 1 --rpc ~/miner1/geth.ipc

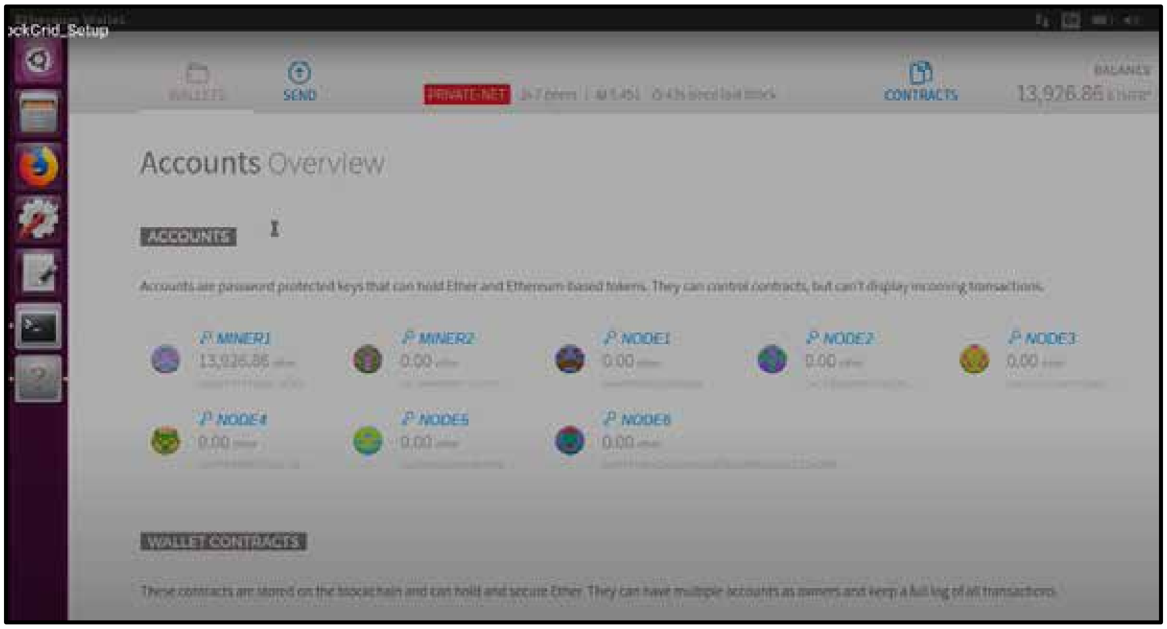
from the mining VM's home direcotry



****THIS TAKES A VERY LONG TIME TO LOAD****
(APPROX 3 MIN)

Figure 38. Screenshots from Prototype Setup Instructional Video Showing How to Launch Ethereum Wallet Web App.

Once the Ethereum Wallet GUI has successfully launched, the BlockGrid prototype is fully operational. Figure 39 shows the Ethereum Wallet interface once the BlockGrid Prototype is operational.



Accounts are password protected keys that can hold Ether and Ethereum-based tokens. They can control contracts, but can't display incoming transactions.

These contracts are stored on the blockchain and can hold and secure Ether. They can have multiple accounts as owners and keep a full log of all transactions.

Figure 39. Screenshot from Prototype Setup Instructional Video Showing Ethereum Wallet Web App GUI interface.

B. BLOCKGRID PROTOTYPE OPERATION

A video of the prototype operation instructions, prepared by the author, is viewable available at: https://www.youtube.com/watch?v=8-lMhEzy2tc

An mp4 file of the prototype operation instructions video has also been included in the supplemental materials for this thesis.

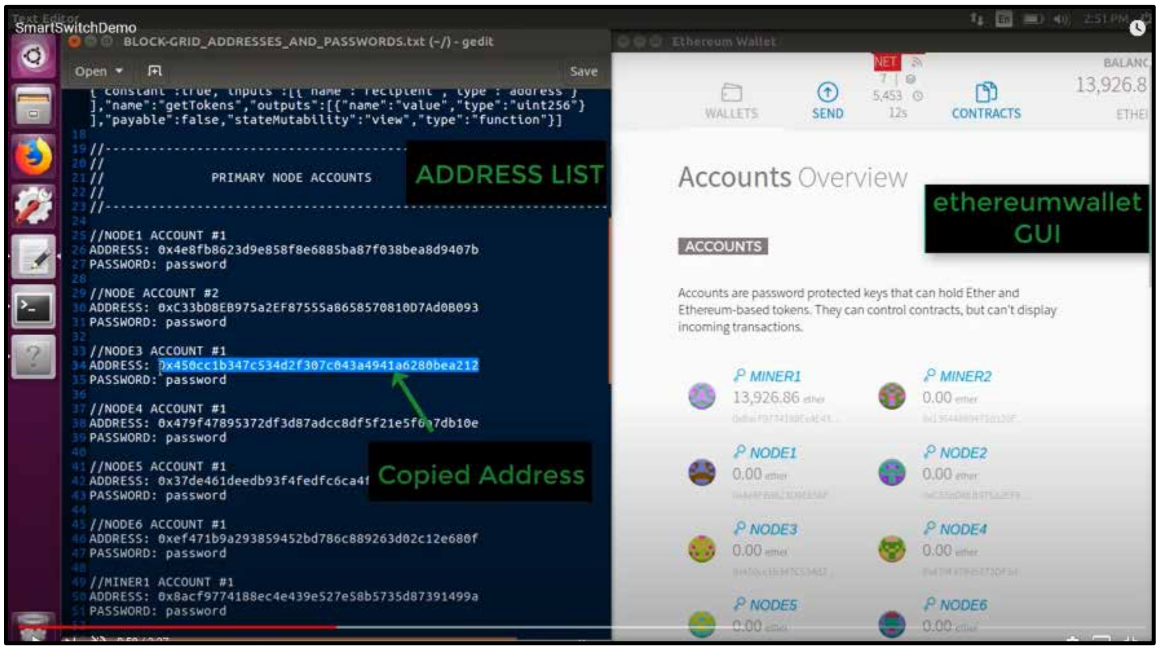
How the SmartSwitch smart contract works

When a function from the SmartSwitch contract is executed to add or withdraw SmartTokens from an RPI node's account on the blockchain, the smart_token.js script running on the node observes the change and switches the power output to a different GPIO output pin on the node (causing the light to switch between RED and GREEN)

How to Execute a SmartSwitch Smart Contract Transaction

Step 1 - Obtain a node account's address from the BLOCKGRID_ACCOUNTS _AND_ADDRESSES.txt file in the SCU VM's home directory so that it can be entered into the SmartSwitch function input fields in the Ethereum Wallet GUI to execute a transaction. See 00:21 through 01:01 of the prototype operation instructions video and

Figure 40 for guidance. Figure 40 provides a screenshot showing how to locate a node address for smart contract execution.

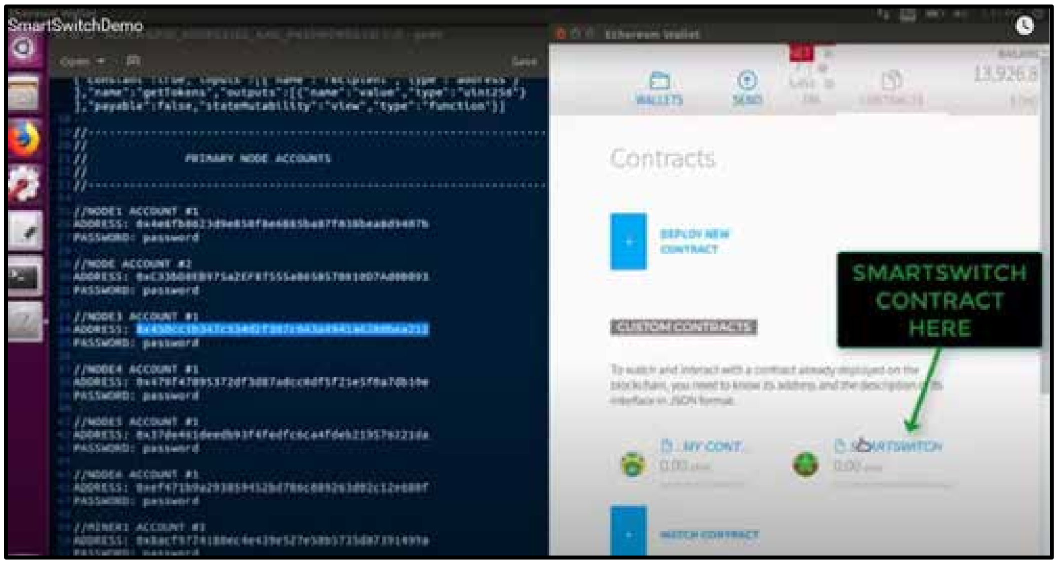


```
[ constant :true, inputs :[{ name : recipient , type : address }
],"name":"getTokens","outputs":[{"name":"value","type":"uint256"}
],"payable":false,"stateMutability":"view","type":"function"}]

//------------------------------------
//
//              PRIMARY NODE ACCOUNTS
//
//--------------------------------------------------------------

//NODE1 ACCOUNT #1
ADDRESS: 0x4e8fb8623d9e858f8e6885ba87f038bea8d9407b
PASSWORD: password

//NODE ACCOUNT #2
ADDRESS: 0xC33bD8EB975a2EF87555a8658570810D7Ad0B093
PASSWORD: password

//NODE3 ACCOUNT #1
ADDRESS: 0x450cc1b347c534d2f307c043a4941a6280bea212
PASSWORD: password

//NODE4 ACCOUNT #1
ADDRESS: 0x479f47895372df3d87adcc8df5f21e5f6 7db10e
PASSWORD: password

//NODE5 ACCOUNT #1
ADDRESS: 0x37de461deedb93f4fedfc6ca4f
PASSWORD: password

//NODE6 ACCOUNT #1
ADDRESS: 0xef471b9a293859452bd786c889263d02c12e680f
PASSWORD: password

//MINER1 ACCOUNT #1
ADDRESS: 0x8acf9774188ec4e439e527e58b5735d87391499a
PASSWORD: password
```

Accounts are password protected keys that can hold Ether and Ethereum-based tokens. They can control contracts, but can't display incoming transactions.

Figure 40. Screenshot from Prototype Setup Instructional Video Showing How to Access Blockchain Account Addresses.

Step 2 - From within the Ethereum Wallet GUI, open the SmartSwitch contract. See 01:04 through 01:18 of the prototype operations instruction video and Figure 41 for guidance. Figure 41 gives a screenshot showing how to select the SmartSwitch contract from the Ethereum Wallet GUI.



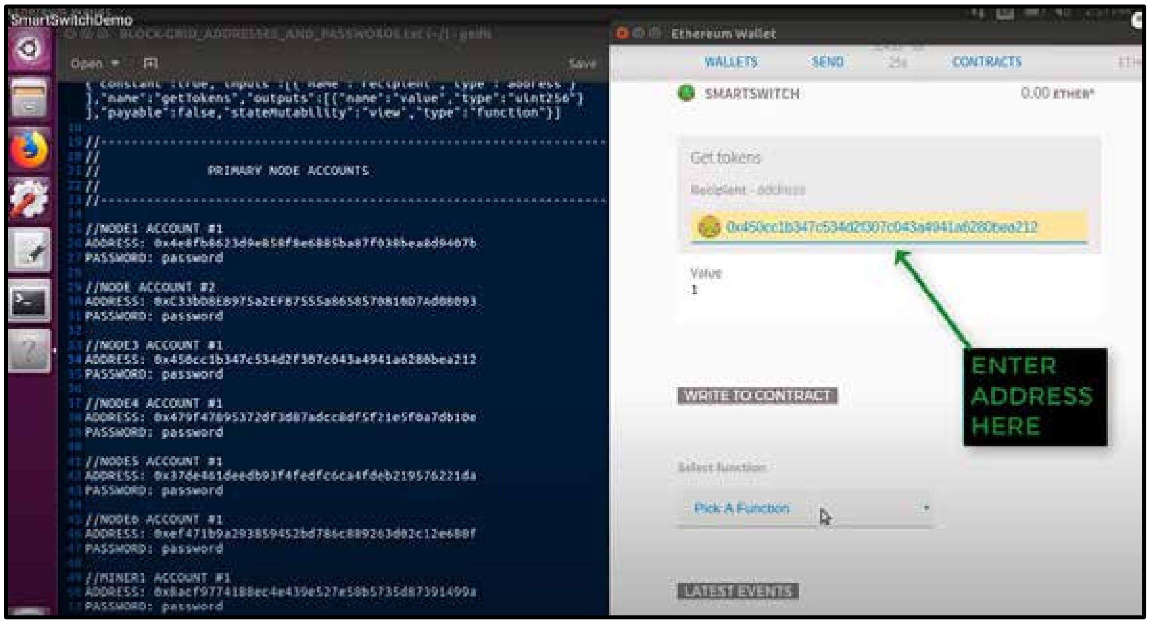
```
],"name":"getTokens","outputs":[{"name":"value","type":"uint256"}
],"payable":false,"stateMutability":"view","type":"function"}]

//----------------------------------------------------------------
//
//              PRIMARY NODE ACCOUNTS
//
//----------------------------------------------------------------

//NODE1 ACCOUNT #1
ADDRESS: 0x4e6fb8623d9e858f8e6865ba87f638bea8d9487b
PASSWORD: password

//NODE ACCOUNT #2
ADDRESS: 0xC33b08EB975a2EF8f555a8658578810D7Ad08093
PASSWORD: password

//NODE3 ACCOUNT #1
ADDRESS: 0x450cc1b347c534d2f307c043a4941a6280bea212
PASSWORD: password

//NODE4 ACCOUNT #1
ADDRESS: 0x479f47895372df3d87adcc8df5f21e5f0a7db10e
PASSWORD: password

//NODE5 ACCOUNT #1
ADDRESS: 0x37de461deedb93f4fedfc6ca4fdeb213579221da
PASSWORD: password

//NODE6 ACCOUNT #1
ADDRESS: 0xef671b9a293859452bd786c089263d02c12e080f
PASSWORD: password

//MINER1 ACCOUNT #1
ADDRESS: 0xbacf9774180ec4e439e527e58b5735d87391495a
PASSWORD: password
```

To watch and interact with a contract already deployed on the blockchain, you need to know its address and the description of its interface in JSON format.

Figure 41. Screenshot from Prototype Setup Instructional Video Showing How to Access the SmartSwitch Contract.

Step 3 – Enter the address for a desired account to the Get tokens field to see the account's current SmartToken Balance. See 01:18 through 01:28 of the prototype operations instructional video and Figure 42 for guidance. Figure 42 gives a screenshot that shows Get tokens being invoked for account 0x450cc1b347c534d2f307c043a4941a6280bea212 and returning a SmartToken balance of 1.

Step 4 – Select the desired SmartToken balance adjustment function, either Deposit Token or Withdraw Token, from the Select Function pulldown menu. Then enter the desired node's blockchain account address into the Recipient address field and enter the desired SmartToken balance adjustment in the Value field. See 01:37 through 01:43 of the prototype instructional video and Figure 43 for guidance. Figure 43 gives a screenshot showing Withdraw Token being invoked for account 0x450cc1b347c534d2f307c043a4941a6280bea212 with an adjustment value of 1.

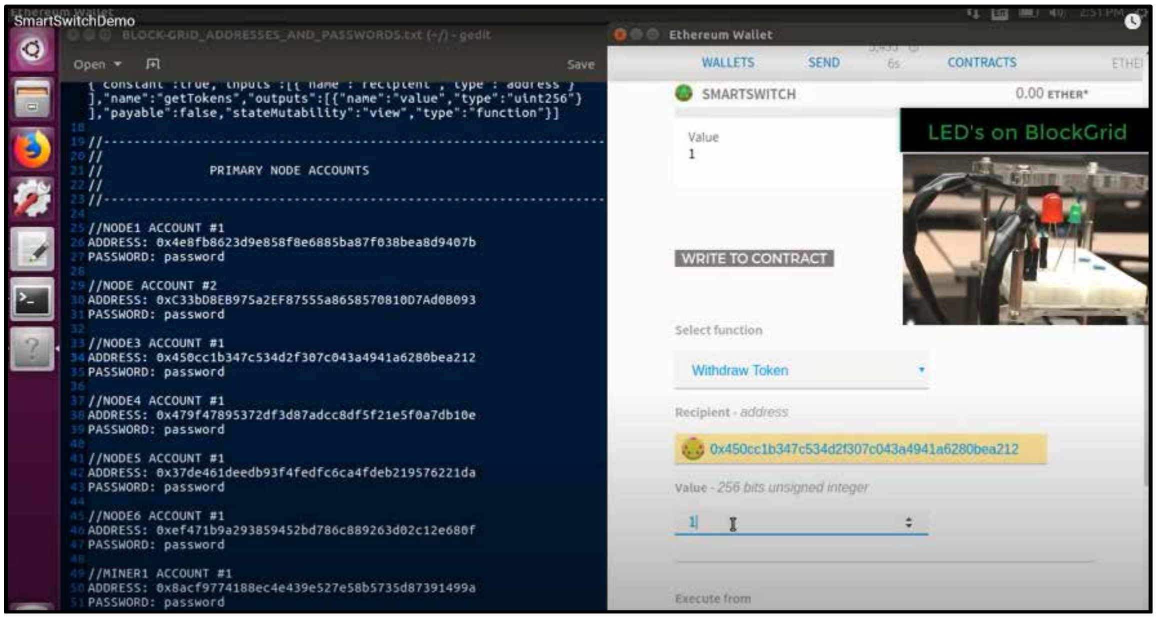


```
{"constant":true,"inputs":[{"name":"recipient","type":"address"}
],"name":"getTokens","outputs":[{"name":"value","type":"uint256"}
],"payable":false,"stateMutability":"view","type":"function"}]

//-------------------------------------------------------------
//
//              PRIMARY NODE ACCOUNTS
//
//-------------------------------------------------------------

//NODE1 ACCOUNT #1
ADDRESS: 0x4e8fb8623d9e858f8e6885ba87f038bea8d9407b
PASSWORD: password

//NODE ACCOUNT #2
ADDRESS: 0xC33bD8EB975a2EF87555a8658570810D7Ad0B093
PASSWORD: password

//NODE3 ACCOUNT #1
ADDRESS: 0x450cc1b347c534d2f307c043a4941a6280bea212
PASSWORD: password

//NODE4 ACCOUNT #1
ADDRESS: 0x479f47895372df3d87adcc8df5f21e5f0a7db10e
PASSWORD: password

//NODE5 ACCOUNT #1
ADDRESS: 0x37de461deedb93f4fedfc6ca4fdeb219576221da
PASSWORD: password

//NODE6 ACCOUNT #1
ADDRESS: 0xef471b9a293859452bd786c889263d02c12e680f
PASSWORD: password

//MINER1 ACCOUNT #1
ADDRESS: 0x8acf9774188ec4e439e527e58b5735d87391499a
PASSWORD: password
```

Recipient - address: 0x450cc1b347c534d2f307c043a4941a6280bea212

Value: 1

Figure 42. Screenshot from Prototype Setup Instructional Video Showing How to Assess an Account's SmartToken Balance.



```
{"constant":true,"inputs":[{"name":"recipient","type":"address"}
],"name":"getTokens","outputs":[{"name":"value","type":"uint256"}
],"payable":false,"stateMutability":"view","type":"function"}]

//-------------------------------------------------------------
//
//              PRIMARY NODE ACCOUNTS
//
//-------------------------------------------------------------

//NODE1 ACCOUNT #1
ADDRESS: 0x4e8fb8623d9e858f8e6885ba87f038bea8d9407b
PASSWORD: password

//NODE ACCOUNT #2
ADDRESS: 0xC33bD8EB975a2EF87555a8658570810D7Ad0B093
PASSWORD: password

//NODE3 ACCOUNT #1
ADDRESS: 0x450cc1b347c534d2f307c043a4941a6280bea212
PASSWORD: password

//NODE4 ACCOUNT #1
ADDRESS: 0x479f47895372df3d87adcc8df5f21e5f0a7db10e
PASSWORD: password

//NODE5 ACCOUNT #1
ADDRESS: 0x37de461deedb93f4fedfc6ca4fdeb219576221da
PASSWORD: password

//NODE6 ACCOUNT #1
ADDRESS: 0xef471b9a293859452bd786c889263d02c12e680f
PASSWORD: password

//MINER1 ACCOUNT #1
ADDRESS: 0x8acf9774188ec4e439e527e58b5735d87391499a
PASSWORD: password
```

Value: 1

Select function: Withdraw Token

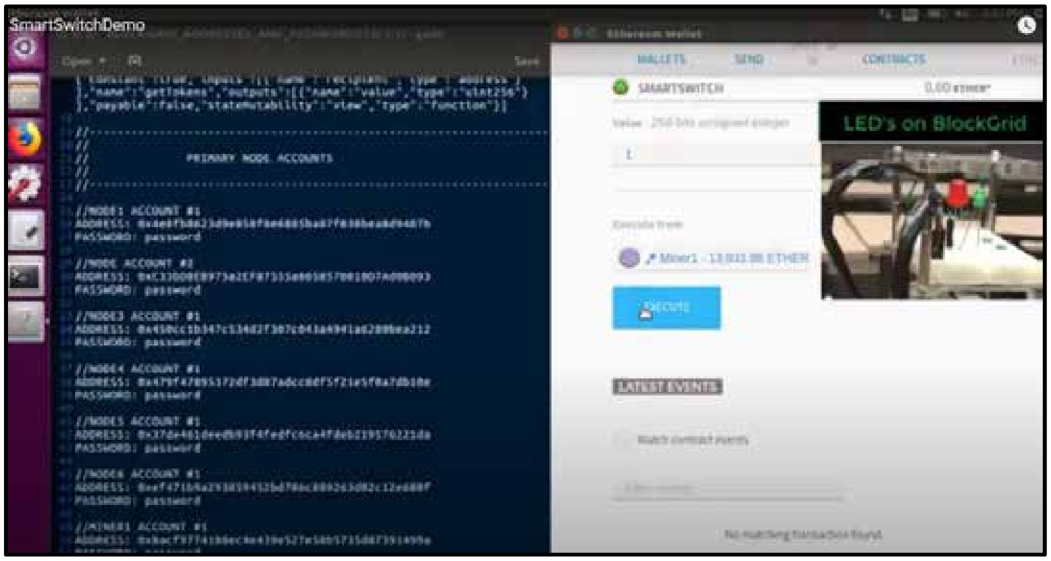
Recipient - address: 0x450cc1b347c534d2f307c043a4941a6280bea212

Value - 256 bits unsigned integer: 1

Figure 43. Screenshot from Prototype Setup Instructional Video Showing How to Select a SmartSwitch Function to Execute and With the Appropriate Value.

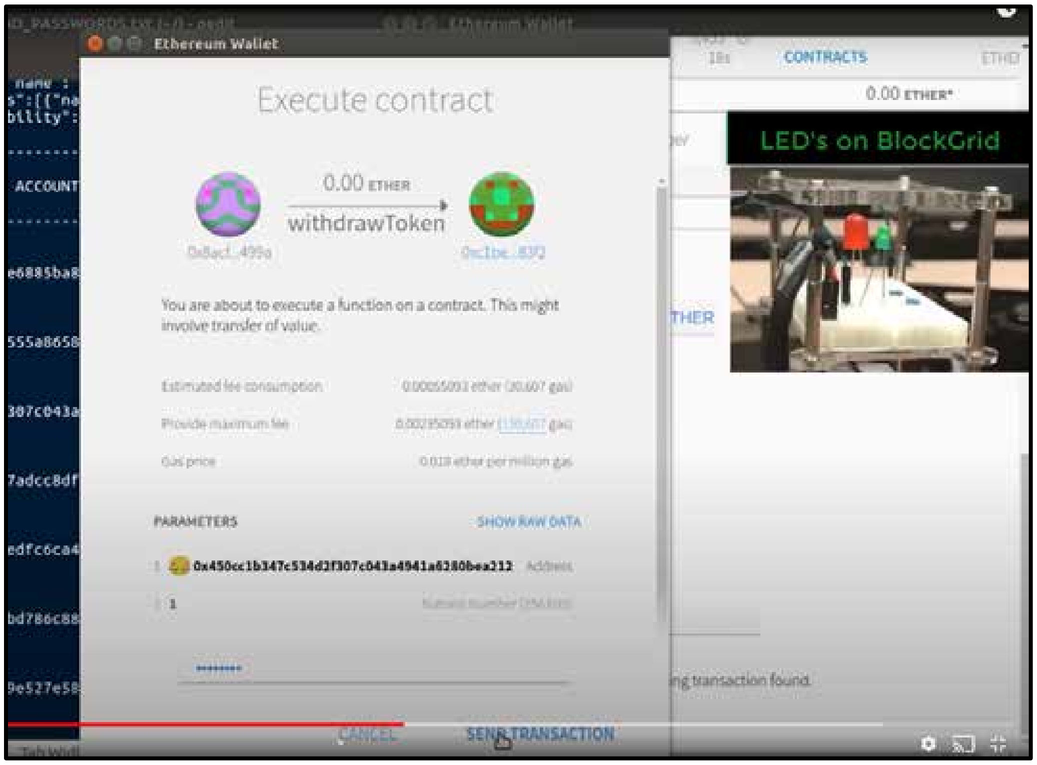
Step 5 - Initiate the SmartSwitch transaction by clicking the Execute button and entering in the node account password "password." See 01:43 though 02:00 of the prototype instructional video and Figure 44 for guidance. Figure 44 gives two screenshots that show the Ethereum Wallet GUI during a Withdraw Token execution for account 0x450cc1b347c534d2f307c043a4941a6280bea212 with an adjustment value of 1. The first screenshot shows how to initiate the smart contract transaction within the Ethereum Wallet GUI. The second screenshot shows the verification pop-up window that auto-generates following execution initiation, and which requires entry of the node password "password" to complete the transaction.

Step 6 – Observe the change in the SmartToken balance for the account and the corresponding change in LED color. See 02:01 through 03:27 of the prototype instructional video and Figure 45 for guidance. Figure 45 gives two screenshots showing the decrement in SmartToken balance for account 0x450cc1b347c534d2f307c043a4941a6280bea212 from 1 to 0 as a result of the Withdraw Token smart contract execution and the corresponding rerouting of LED current from the Raspberry Pi node's green LED to the red LED. The first screenshot shows the node's SmartToken balance to be 1 and the green LED powered on. The second screenshot shows the node's SmartToken balance to be 0 and the red LED powered on.



SmartSwitchDemo

```
],"name":"getTokens","outputs":[{"name":"value","type":"uint256"}
],"payable":false,"stateMutability":"view","type":"function"}]

//----------------------------------------------------------------
//
//              PRIMARY NODE ACCOUNTS
//
//----------------------------------------------------------------

//NODE1 ACCOUNT #1
ADDRESS: 0xae8fb8e23d9e858f0e6885ba07f838beadd9487b
PASSWORD: password

//NODE ACCOUNT #2
ADDRESS: 0xC33D08EB973e2EF87555a8658570810D7A00B093
PASSWORD: password

//NODE3 ACCOUNT #1
ADDRESS: 0x450cc1b347c534d2f307c043a4941a6280bea212
PASSWORD: password

//NODE4 ACCOUNT #1
ADDRESS: 0x479f47895172df3d87adcc8df5f21e5f0a7db10e
PASSWORD: password

//NODE5 ACCOUNT #1
ADDRESS: 0x37de461deedb93f4fedfc6ca4fdeb219176221da
PASSWORD: password

//NODE6 ACCOUNT #1
ADDRESS: 0xef471b6a29385945 2bd786c889263d82c12ed88f
PASSWORD: password

//MINER1 ACCOUNT #1
ADDRESS: 0x8acf97741b8ec4e439e527e58b5735d87391499a
```



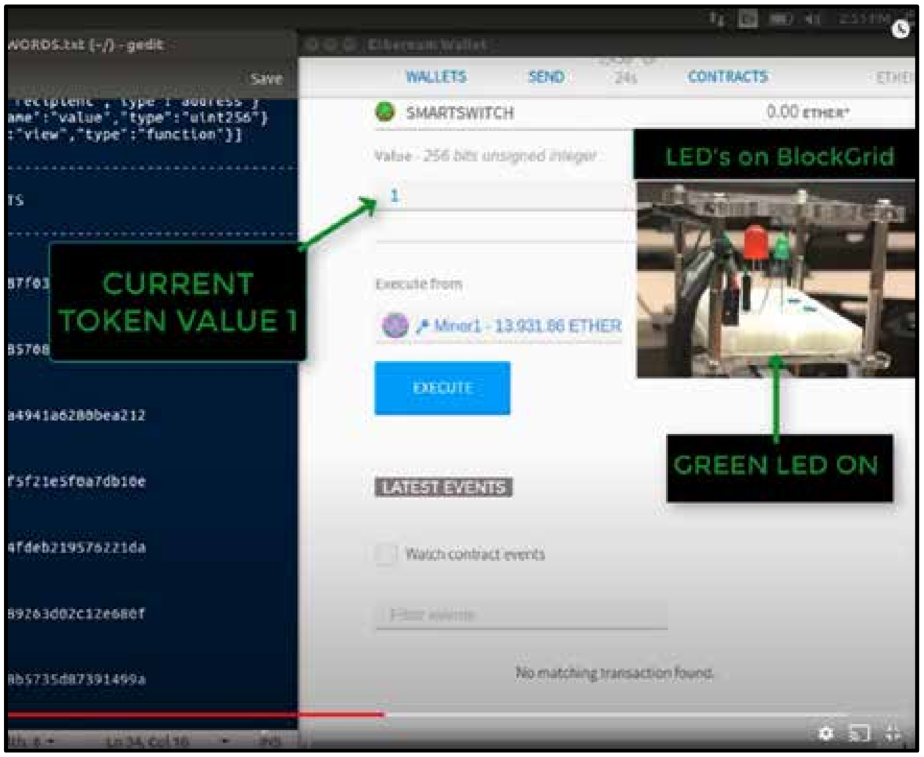
You are about to execute a function on a contract. This might involve transfer of value.

Estimated fee consumption	0.00055093 ether (30,607 gas)
Provide maximum fee	0.00235093 ether (130,607 gas)
Gas price	0.018 ether per million gas

PARAMETERS

1	0x450cc1b347c534d2f307c043a4941a6280bea212	Address
2	1	Natural number (256 bits)

Figure 44. Screenshots from Prototype Setup Instructional Video Showing How to Execute a SmartSwitch Function.



```
 recipient","type":"address"},
ame":"value","type":"uint256"}
:"view","type":"function"}]
```

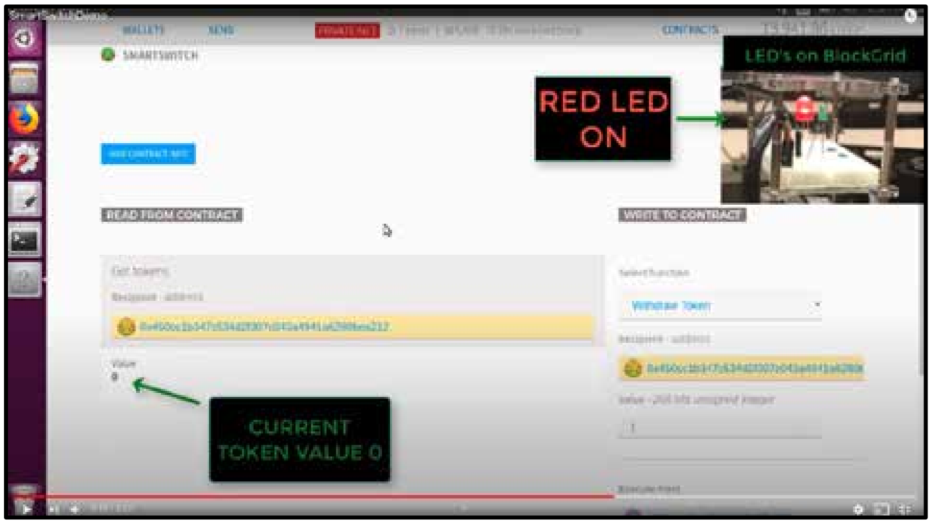


Figure 45. Screenshots from Prototype Setup Instructional Video Showing SmartSwitch Contract Execution.

THIS PAGE INTENTIONALLY LEFT BLANK

SUPPLEMENTAL MATERIALS

(1) Ethereum Blockchain Data Structure Diagram – PDF document file

As noted in Section G of Chapter 2 and Reference [166], the Coinmonks article "Ethereum Primitives #1.2" includes a link to a diagram of the Ethereum Data Structure attributed to Lee Thomas that is too large to include in the document. A PDF file of the diagram is provided as a supplemental material and can be retrieved by contacting the NPS Dudley Knox Library directly, or by checking the supplemental materials link posted along with this thesis in the Calhoun NPS Institutional Archive [53].

(2) BlockGrid Prototype VM – VirtualBox Image file

As detailed in Chapter V and listed in Appendix A, a VM was used for the BlockGrid prototype's SCU. A copy of this VM in VirtualBox Image file form is included in this supplemental materials section to enable reconstruction of the prototype. The VM runs an Ubuntu 16.04 OS and is preconfigured with the requisite network settings and installed scripts necessary to run the BlockGrid prototype. The VM image file can be retrieved by contacting the NPS Dudley Knox Library directly or by checking the supplemental materials link posted along with this thesis in the Calhoun NPS Institutional Archive [53].

(3) Eloudrhiri Tutorials for Deploying Ethereum, a Smart Contract, and Cyber-Physical LED system on a Raspberry Pi – PDF document files

As described in Appendix B and throughout the thesis, the Chainskills tutorials by Eloudrhiri for deploying Ethereum on a Raspberry Pi were central to creation of the BlockGrid prototype [52], [298], [330], [365]–[368]. There seven tutorials in the series, and PDF files of all seven tutorials are provided to give guidance for building the fundamental elements of a BlockGrid. The PDF files can be retrieved by contacting the NPS Dudley Knox Library directly or by checking the supplemental materials link posted along with this thesis in the Calhoun NPS Institutional Archive [53].

(4) BlockGrid Prototype Setup Instructions Video Walk-Through - mp4 video file

As described in Section A of Appendix C, this mp4 video file walks through the steps necessary to assemble both the hardware and software of the BlockGrid prototype to enable operation. The mp4 file can be retrieved by contacting the NPS Dudley Knox Library directly or by checking the supplemental materials link posted along with this thesis in the Calhoun NPS Institutional Archive [53].

(5) BlockGrid Prototype Operation Instructions Video Walk-Through – mp4 video file

As described in Section B of Appendix C, this mp4 video file walks through the steps necessary to operate the BlockGrid prototype and execute a SmartSwitch function to demonstrate blockchain technology through cyber-physical control of the prototype. The mp4 file can be retrieved by contacting the NPS Dudley Knox Library directly or by checking the supplemental materials link posted along with this thesis in the Calhoun NPS Institutional Archive [53].

LIST OF REFERENCES

[1] M. Mylrea and S. N. G. Gourisetti, “Blockchain for smart grid resilience: Exchanging distributed energy at speed, scale and security,” in *2017 Resilience Week (RWS)*, 2017, pp. 18–23. [Online]. [Online]. Available: https://ieeexplore.ieee.org/document/8088642/

[2] M. Crosby, P. Pattanayak, S. Verma, and V. Kalyanaraman, “Blockchain technology: Beyond bitcoin,” *Applied Innovation*, vol. 2, no. 6–10, p. 71, 2016. [Online]. Available: https://ieeexplore.ieee.org/document/8246573

[3] T. T. A. Dinh, R. Liu, M. Zhang, G. Chen, B. C. Ooi, and J. Wang, “Untangling blockchain: A data processing view of blockchain systems,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 7, pp. 1366–1385, 2018. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8246573

[4] V. Gatteschi, F. Lamberti, C. Demartini, C. Pranteda, and V. Santamaria, “To blockchain or not to blockchain: That is the question,” *IT Professional*, vol. 20, no. 2, pp. 62–74, 2018. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8338007

[5] D. Tapscott and A. Tapscott, *Blockchain Revolution: How the Technology Behind Bitcoin Is Changing Money, Business, and the World*. London, England, UK: Penguin, 2016.

[6] M. Swan, *Blockchain: Blueprint for a New Economy*. Sebastopol, CA, USA: O’Reilly Media, Inc., 2015.

[7] D. Randall, P. Goel, and R. Abujamra, “Blockchain applications and use cases in health information technology,” *Journal of Health & Medical Informatics*, vol. 8, no. 3, pp. 8–11, 2017. [Online]. Available: http://www.avidscience.com/wp-content/uploads/2017/10/blockchain-applications-and-use-cases-in-health-information-technology.pdf

[8] A. B. Ayed, “A conceptual secure blockchain-based electronic voting system,” *International Journal of Network Security & Its Applications*, vol. 9, no. 3, pp. 01–09, 2017. [Online]. Available: https://aircconline.com/ijnsa/V9N3/9317ijnsa01.pdf

[9] A. Schumacher, “Reinventing healthcare on the blockchain—Toward a new era in precision medicine,” *Blockchain Res. Inst.(BRI), Toronto. Big Whitepapers*, 2018. [Online]. Available: https://s3.us-east-2.amazonaws.com/brightline-website/downloads/reports/Brightline_Schumacher_Reinventing-Healthcare-on-the%20Blockchain_Blockchain-Research-Institute.pdf

[10] Y. Xu, J. Ren, G. Wang, C. Zhang, J. Yang, and Y. Zhang, “A blockchain-based nonrepudiation network computing service scheme for industrial IoT,” *IEEE Transactions on Industrial Informatics*, vol. 15, no. 6, pp. 3632–3641, 2019. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8633331

[11] S. Rathore, B. W. Kwon, and J. H. Park, “BlockSecIoTNet: Blockchain-based decentralized security architecture for IoT network,” *Journal of Network and Computer Applications*, vol. 143, pp. 167–177, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S1084804519302243

[12] A. Sudhan and M. J. Nene, “Employability of blockchain technology in defense applications,” in *2017 International Conference on Intelligent Sustainable Systems (ICISS)*, 2017, pp. 630–637. [Online]. Available: https://ieeexplore.ieee.org/document/8389247/

[13] NCI-NATO Newsroom, “NCI Agency Innovation Challenge,” *NATO Communications and Information Agency*, 25 Apr. 2016. [Online]. Available: https://www.ncia.nato.int/about-us/newsroom/nci-agency-innovation-challenge.html

[14] R. D. S. Kulshrestha and I. Navy, “Military applications of blockchain technology,” in *The Age of Blockchain: A Collection of Articles*, vol. 21, IndraStra Global, 2018. [Online]. doi: 10.5281/zenodo.1202390

[15] J. Pan and Z. Yang, “Cybersecurity challenges and opportunities in the new ‘Edge Computing + IoT’ world,” in *Proceedings of the 2018 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, 2018, pp. 29–32. [Online]. Available: http://delivery.acm.org/10.1145/3190000/3180470/p29-pan.pdf

[16] N. Kshetri, “Can blockchain strengthen the internet of things?,” *IT Professional*, vol. 19, no. 4, pp. 68–72, 2017. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8012302

[17] Defense Advanced Research Projects Agency, “Ocean of things aims to expand maritime awareness across open seas,” *DARPA-News and Events*, Dec. 06, 2017. [Online]. Available: www.darpa.mil/news-events/2017-12-06.

[18] Department of Homeland Security Science and Technology Directorate, “Blockchain portfolio.” Accessed September 04, 2020. [Online]. Available: https://www.dhs.gov/science-and-technology/blockchain-portfolio

[19] Department of Homeland Security Science and Technology Directorate, “News release: DHS awards $199K for blockchain credential life cycle management,” *DHS S&T News Room*, Nov. 14, 2019. [Online]. Available: https://www.dhs.gov/science-and-technology/news/2019/11/14/news-release-dhs-awards-199k-blockchain-tech

[20] Department of Homeland Security Science and Technology Directorate, “News release: DHS awards $198K for raw material import tracking using blockchain,” *DHS S&T News Room*, Nov. 08, 2019. [Online]. Available: https://www.dhs.gov/science-and-technology/news/2019/11/08/news-release-dhs-awards-198k-raw-material-import-tracking

[21] Defense Advanced Research Projects Agency, “Secure messaging platform,” *SBIR*, Apr. 22, 2016. [Online]. Available: www.sbir.gov/sbirsearch/detail/1144411.

[22] Data Foundation, “Bringing blockchain into government: a path forward for creating effective federal blockchain initiatives,” June 2019. [Online]. Available: https://www.datafoundation.org/bringing-blockchain-into-government

[23] National Center for Manufacturing Sciences, “NCMS, Moog sign agreement to perform work supporting adaptation of blockchain technology,” Society of Manufacturing Engineers (SME), Apr. 04, 2018. [Online]. Available: https://www.sme.org/ncms-moog-sign-agreement-to-perform-work-supporting-adaptation-of-blockchain-technology

[24] K. Bernstein, M. Kane, and C. Lantman, “Shield supply chain assurance technology. DARPA, Electronics Resurgence Initiative Summit,” presented at the Electronics Resurgence Initiative, San Francisco, CA, USA Jul. 23–25, 2018. [Online]. Available: https://eri-summit.darpa.mil/docs/20180725_1545_SHIELD.pdf

[25] Eversden, Andrew, “The important lessons of federal blockchain projects,” *Federal Times*, 10 Jun. 2019. [Online]. Available: https://www.federaltimes.com/it-networks/2019/06/10/the-important-lessons-of-federal-blockchain-projects/

[26] General Services Administration, “GSA’s Making it Easier to do Business with the Government,” Sep. 2017. [Online]. Available: https://www.gsa.gov/cdnstatic/making_it_september_17_final_508.pdf

[27] Gilbert, Jackie, “HHS Grant Solutions is already leveraging blockchain technology,” *FedHealth IT*, Jun. 11, 2019. [Online]. Available: https://www.fedhealthit.com/2019/06/hs-grant-solutions-is-already-leveraging-blockchain-technology/

[28] GCN Staff, “HHS gets ATO for blockchain-based acquisition system,” *Government Computer News*, Dec. 11, 2018. [Online]. Available: https://gcn.com/articles/2018/12/11/arrieta-hhs-accelerate.aspx

[29] United States, Congress, House, Armed Services Committee. “H.R.2810 - National Defense Authorization Act for Fiscal Year 2018,” Government Printing Office, 2018. [Online]. Available: https://www.congress.gov/bill/115th-congress/house-bill/2810/text

[30] Norquist, David L, "DOD Digital Modernization Strategy: DOD Information Resources Management Strategic Plan FY19-23," Washington, DC, USA, OSD, 2019. [Online]. Available: https://media.defense.gov/2019/Jul/12/2002156622/-1/-1/1/DOD-DIGITAL-MODERNIZATION-STRATEGY-2019.PDF

[31] M. Mylrea and S. N. G. Gourisetti, "Blockchain for supply chain cybersecurity, optimization and compliance," in *2018 Resilience Week (RWS)*, 2018, pp. 70–76. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8473517

[32] M. Mylrea and S. Gourisetti, "Blockchain: Next generation supply chain security for energy infrastructure and NERC critical infrastructure protection (CIP) compliance," in *Resilience Week*, USA, 2018. [Online]. Available: http://www.iiisci.org/Journal/CV$/sci/pdfs/SA513XS18.pdf

[33] S. Chow and M. E. Peck, "The bitcoin mines of China," *IEEE Spectrum*, vol. 54, no. 10, pp. 46–53, 2017. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8048840

[34] G. Hileman and M. Rauchs, "Global blockchain benchmarking study," *Cambridge Centre for Alternative Finance*, *University of Cambridge*, vol. 122, 2017. [Online]. Available: https://j2-capital.com/wp-content/uploads/2017/11/GLOBAL-BLOCKCHAIN.pdf

[35] B. Kaiser, M. Jurado, and A. Ledger, "The looming threat of china: An analysis of chinese influence on bitcoin," *arXiv preprint arXiv:1810.02466*, 2018. [Online]. Available: https://arxiv.org/abs/1810.02466

[36] F. Dai, Y. Shi, N. Meng, L. Wei, and Z. Ye, "From Bitcoin to cybersecurity: A comparative study of blockchain application and security issues," in *2017 4th International Conference on Systems and Informatics (ICSAI)*, 2017, pp. 975–979. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8248427

[37] Cant, Joeri, "CEO of Zimbabwean crypto exchange loses password to Bitcoin Cold Wallet," *Cointelegraph*, Nov. 01, 2019. [Online]. Available: https://cointelegraph.com/news/ceo-of-zimbabwean-crypto-exchange-loses-password-to-bitcoin-cold-wallet

[38] Rich, Nathaniel, "Ponzi schemes, private yachts, and a missing $250 million in crypto: The strange tale of quadriga," *Vanity Fair*, Nov. 22, 2019. [Online]. Available: https://www.vanityfair.com/news/2019/11/the-strange-tale-of-quadriga-gerald-cotten

[39] Mitre Corportation, Common Vulnerabilities and Exposures Database. Accessed, September 06, 2020. [Online]. Available: https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=bitcoin

[40] Mitre Corportation, Common Vulnerabilities and Exposures Database. Accessed, September 06, 2020. [Online] Available: https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=ethereum

[41] Y. Xu, G. Wang, J. Yang, J. Ren, Y. Zhang, and C. Zhang, "Towards secure network computing services for lightweight clients using blockchain," *Wireless Communications and Mobile Computing*, vol. 2018, 2018. [Online]. Available: https://www.hindawi.com/journals/wcmc/2018/2051693/

[42] X. L. Yu, O. Al-Bataineh, D. Lo, and A. Roychoudhury, "Smart contract repair," *arXiv preprint arXiv:1912.05823*, 2019. [Online]. Available: https://arxiv.org/abs/1912.05823

[43] Hertig, Alyssa, "The latest Bitcoin bug was so bad, developers kept its full details a secret," *Coindesk*, Sep. 21, 2018. [Online]. Available: https://www.coindesk.com/the-latest-bitcoin-bug-was-so-bad-developers-kept-its-full-details-a-secret

[44] R. van den Dam, "Blockchain in telecom: from concept to reality," *IBM-Telecom, Media, Entertainment Blog*, Jan. 03, 2018. [Online]. Available: www.ibm.com/blogs/insights-on-business/telecom-media-entertainment/blockchain-telecom-concept-reality/.

[45] Verizon Communications Inc, "Verizon enterprise solutions will create global blockchain platform services," *Verizon News*, Feb. 13, 2018. [Online]. Available: www.verizon.com/about/news/verizon-enterprise-solutions-will-create-global-blockchain-platform-services.

[46] M. Gray, "Introducing Project Bletchley and elements of blockchain born in the Microsoft Cloud," *Microsoft Developer Blog*, Jun. 15, 2016. [Online]. Available: https://azure.microsoft.com/en-us/blog/bletchley-blockchain/

[47] Cisco Systems, "Blockchain by Cisco." Accessed July 15, 2018. [Online]. Available: https://www.cisco.com/c/dam/en/us/solutions/collateral/digital-transformation/blockchain-whitepaper.pdf

[48] G. Bell. J. Downing, E.J., G. M., M. N., L.P., M. Kilani, K. Rahbari, S.S., C.T., and K. Woods, "Blockchain and its suitability for government applications," United States Department of Homeland Security, Public-Private Analytic Exchange Program 2018. [Online]. Available: https://www.dhs.gov/sites/default/files/publications/2018_AEP_Blockchain_and_Suitability_for_Government_Applications.pdf

[49] N. G. Vovchenko, E. N. Tishchenko, T. V. Epifanova, and M. B. Gontmacher, "Electronic currency: the potential risks to national security and methods to minimize them," 2017. [Online]. Available: ftp://dlib.info/opt/ReDIF/RePEc/ers/papers/17_1_p3.pdf

[50] G. Tziakouris, “Cryptocurrencies—a forensic challenge or opportunity for law enforcement? an Interpol perspective,” *IEEE Security & Privacy*, vol. 16, no. 4, pp. 92–94, 2018. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8425619/

[51] J. Baron, A. O’Mahony, D. Manheim, and C. Dion-Schwarz, “National security implications of virtual currency: examining the potential for non-state actor deployment,” RAND Corporation-NDRI, Santa Monica United States, 2015. [Online]. Available: http://www.dtic.mil/dtic/tr/fulltext/u2/1000534.pdf

[52] S. Eloudrhiri, “Install an Ethereum node on a computer,” *Chainskills*, Mar. 03, 2017. [Online]. Available: https://chainskills.com/2017/03/03/install-a-ethereum-node-on-a-computer-26/

[53] Naval Postgraduate School, “Calhoun: the NPS Institutional Archive,” *Dudley Knox Library*. Accessed September 13, 2020. [Online]. Available: https://calhoun.nps.edu/

[54] S. Meunier, “Blockchain 101: What is blockchain and how does this revolutionary technology work?,” in *Transforming climate finance and green investment with Blockchains*, Elsevier, 2018, pp. 23–34. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128144473000033

[55] J. Wanguba, “How many cryptocurrencies are there in 2020?,” *Altcoin News*, Sep. 08, 2020. [Online]. Available: https://e-cryptonews.com/how-many-cryptocurrencies-are-there-in-2020

[56] L. Lamport, R. Shostak, and M. Pease, “The Byzantine Generals Problem,” in *Concurrency: the Works of Leslie Lamport*, 2019, pp. 203–226. [Online]. Available: https://people.eecs.berkeley.edu/~luca/cs174/byzantine.pdf.

[57] L. Lamport, “The Weak Byzantine Generals Problem,” *Journal of the ACM (JACM)*, vol. 30, no. 3, pp. 668–676, 1983. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/2402.322398

[58] L. Lamport and M. Fischer, “Byzantine generals and transaction commit protocols,” Technical Report 62, SRI International, 1982. [Online]. Available: https://www.microsoft.com/en-us/research/uploads/prod/2016/12/Byzantine-Generals-and-Transaction-Commit-Protocols.pdf

[59] L. Lamport and P. M. Melliar-Smith, “Byzantine clock synchronization,” in *Proceedings of the third annual ACM symposium on Principles of distributed computing*, 1984, pp. 68–74. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/800222.806737

[60] D. L. Chaum, “Computer systems established, maintained and trusted by mutually suspicious groups,” The Electronics Research Laboratory, University of California, Berkeley, CA, USA, 22 Feb. 1979. [Online]. Available: https://chaum.com/publications/techrep.pdf

[61] A. T. Sherman, F. Javani, H. Zhang, and E. Golaszewski, “On the origins and variations of blockchain technologies,” *IEEE Security & Privacy*, vol. 17, no. 1, pp. 72–77, 2019. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8674176

[62] Y. I. Kang, W. Lovegrove, and J. D. Spragina, “An architecture for factory automation and information management systems,” in *Pacific Computer Communications Symposium*, 1985. [Online]. Available: https://sites.google.com/site/internethistoryasia/lib/pccs

[63] A. Adlemo, S.-A. Andréasson, T. Andreasson, and C. Carlsson, “Achieving fault tolerance in factory automation systems by dynamic configuration,” in *Systems Integration ’90. Proceedings of the First International Conference on Systems Integration*, 1990, pp. 396–402. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=138705

[64] T. Youngs and C. Taylor, “Trident and the future of the British nuclear deterrent,” Standard Note: SN/IA/3706, House of Commons Library, London, England, 2005. [Online]. Available: http://www.acronym.org.uk/old/archive/docs/0507/hoc_library.pdf

[65] V. A. Megna, “Tactical airborne distributed computing and networks,” *Dispersed Sensor Processing Mesh Project, AGARD-CCP-303*, pp. 30–1, 1981. [Online]. Available: https://apps.dtic.mil/dtic/tr/fulltext/u2/a109274.pdf

[66] L. ABBOTT, “Test experience on an ultrareliable computer communication network,” in *Digital Avionics Systems Conference*, 1984, p. 2649. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.1984-2649

[67] M. B. Bertelloni, “The Cypherpunk vision of techno-politics,” *St. Anne’s Academic Review*, p. 1, 1998. [Online]. Available: http://st-annes-mcr.org.uk/wp-content/uploads/2013/09/STAAR-7-2017.pdf#page=8

[68] A. Narayanan, “What happened to the crypto dream?, part 1,” *IEEE security & privacy*, vol. 11, no. 2, pp. 75–76, 2013. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6493328

[69] S. Levy, “Crypto rebels,” *Wired Magazine*, 01 Feb. 1993. https://www.wired.com/1993/02/crypto-rebels/

[70] D. Chaum, “Security without identification: Transaction systems to make big brother obsolete,” *Communications of the ACM*, vol. 28, no. 10, pp. 1030–1044, 1985. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/4372.4373

[71] P. Ludlow, *Crypto Anarchy, Cyberstates, and Pirate Utopias*. Cambridge, MA, USA: MIT Press, 2001. [Online]. Available: https://ieeexplore.ieee.org/document/6299667

[72] W. Dai, “b-money,” published at http://www.eskimo.com/˜weidai/bmoney.txt, Nov. 1998. [Online]. Available: http://www.weidai.com/bmoney.txt

[73] H. Qureshi, “The Cypherpunks,” *Nakamoto*, 29 Dec. 2019. [Online]. Available: https://nakamoto.com/the-cypherpunks/

[74] C. Dwork and M. Naor, “Pricing via processing or combatting junk mail,” in *Annual International Cryptology Conference*, 1992, pp. 139–147. [Online]. Available: https://dl.acm.org/doi/10.5555/646757.705669

[75] M. Jakobsson and A. Juels, “Proofs of work and bread pudding protocols,” in *Secure information networks*, Springer, 1999, pp. 258–272. [Online]. Available: https://link.springer.com/chapter/10.1007/978-0-387-35568-9_18

[76] K. Curran and J. Honan, “Eliminating the Volume of Spam E-Mails Using a Hashcash-Based Solution,” *Inf. Secur. J. A Glob. Perspect.*, vol. 15, no. 2, pp. 22–41, 2006. [Online]. Available: https://search.proquest.com/docview/229508636

[77] H. Finney, “RPOW- Reusable Proofs of Work,” Aug. 15, 2004. [Online]. Available: https://nakamotoinstitute.org/finney/rpow/index.html

[78] N. Szabo, “Bit Gold,” Dec. 28, 2005. [Online]. Available: https://nakamotoinstitute.org/bit-gold/

[79]. S. Haber and W. S. Stornetta, “How to time-stamp a digital document,” in *Conference on the Theory and Application of Cryptography*, 1990, pp. 437–455. [Online]. Available: https://link.springer.com/content/pdf/10.1007%2F3-540-38424-3_32.pdf

[80] S. Haber and W. S. Stornetta, “Secure names for bit-strings,” in *Proceedings of the 4th ACM Conference on Computer and Communications Security*, 1997, pp. 28–35. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/266420.266430

[81] S. A. Haber and W. S. Stornetta Jr, “Method for secure time-stamping of digital documents,” US Patent 5,136,547, Aug. 04 1992. [Online]. Available: https://patents.google.com/patent/US5136647A/en

[82] S. A. Haber and W. S. Stornetta Jr, “Digital document time-stamping with catenate certificate,” US Patent 5,136,646, Aug. 04 1992. [Online]. Available: https://patents.google.com/patent/US5136646A/en

[83] S. A. Haber and W. S. Stornetta Jr, “Digital document authentication system,” US Patent 5,781,629, Jul. 14 1998. [Online]. Available: https://patents.google.com/patent/US5781629A/en

[84] D. Bayer, S. Haber, and W. S. Stornetta, “Improving the efficiency and reliability of digital time-stamping,” in *Sequences Ii*, Springer, 1993, pp. 329–334. [Online]. Available: https://link.springer.com/chapter/10.1007/978-1-4613-9323-8_24

[85] R. C. Merkle, “Secrecy, authentication, and public key systems,” Ph.D. dissertation, Dept. of Comp. Sci., Stanford University, Palo Alto, CA, 1979. [Online]. Available: https://dl.acm.org/doi/book/10.5555/909000

[86] R. C. Merkle, “Method of providing digital signatures,” US Patent 4,309,569, Jan. 5, 1982. [Online]. Available: https://patents.google.com/patent/US4309569A/en

[87] M. Bosamia and D. Patel, “Current Trends and Future Implementation Possibilities of the Merkel Tree,” *International Journal of Computer Sciences and Engineering*, vol. 6, no. 8, pp. 294–301, 2018. [Online]. Available: https://www.researchgate.net/profile/Mansi_Bosamia/publication/327601654_Current_Trends_and_Future_Implementation_Possibilities_of_the_Merkel_Tree/links/5b9914d4a6fdcc59bf8b1229/Current-Trends-and-Future-Implementation-Possibilities-of-the-Merkel-Tree.pdf

[88] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” Bitcoin, 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[89] R. Klomp and A. Bracciali, “On symbolic verification of Bitcoin’s script language,” in *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, Springer, 2018, pp. 38–56. [Online]. Available: https://dspace.stir.ac.uk/bitstream/1893/27845/1/symbolic-verification-bitcoins.pdf

[90] H. A. Kalodner, M. Carlsten, P. Ellenbogen, J. Bonneau, and A. Narayanan, “An Empirical Study of Namecoin and Lessons for Decentralized Namespace Design.,” in *WEIS*, 2015. [Online]. Available: https://www.cs.princeton.edu/~arvindn/publications/namespaces.pdf

[91] V. Buterin, “Mastercoin: a second-generation protocol on the Bitcoin blockchain,” *Bitcoin Magazine*, Nov. 04, 2013. [Online]. Available: https://bitcoinmagazine.com/articles/mastercoin-a-second-generation-protocol-on-the-bitcoin-blockchain-1383603310

[92] V. Buterin, “A prehistory of the Ethereum protocol,” *Vitalik Buterin’s website*, Sep. 14, 2017. [Online]. Available: https://vitalik.ca/general/2017/09/14/prehistory.html

[93] K. Croman *et al.*, “On scaling decentralized blockchains,” in *International conference on financial cryptography and data security*, 2016, pp. 106–125. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-662-53357-4_8

[94] A. Rosic, “Blockchain Scalability: When, Where, How?,” *Blockgeeks*, 21 May 2017. [Online]. Available: https://blockgeeks.com/guides/blockchain-scalability/

[95] N. Atzei, M. Bartoletti, and T. Cimoli, “A survey of attacks on Ethereum smart contracts.,” *IACR Cryptol. ePrint Arch.*, vol. 2016, p. 1007, 2016. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-662-54455-6_8

[96] S. Bistarelli, G. Mazzante, M. Micheletti, L. Mostarda, and F. Tiezzi, “Analysis of ethereum smart contracts and opcodes,” in *International Conference on Advanced Information Networking and Applications*, 2019, pp. 546–558. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-15032-7_46

[97] A. Bogner, M. Chanson, and A. Meeuw, “A decentralised sharing app running a smart contract on the ethereum blockchain,” in *Proceedings of the 6th International Conference on the Internet of Things*, 2016, pp. 177–178. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/2991561.2998465

[98] Tang, Huimin, Yong Shi, and Peiwu Dong. “Public blockchain evaluation using entropy and TOPSIS.” *Expert Systems with Applications*, vol. 117, pp. 204–210, Mar. 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417418306250

[99] S. Pongnumkul, C. Siripanpornchana, and S. Thajchayapong, “Performance analysis of private blockchain platforms in varying workloads,” in *2017 26th International Conference on Computer Communication and Networks (ICCCN)*, 2017, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8038517/

[100] M. Liu, K. Wu, and J. J. Xu, “How will blockchain technology impact auditing and accounting: permissionless versus permissioned blockchain,” *Current Issues in Auditing*, vol. 13, no. 2, pp. A19–A29, 2019. [Online]. Available: https://aaapubs.org/doi/full/10.2308/ciia-52540

[101] Q. Nasir, I. A. Qasse, M. Abu Talib, and A. B. Nassif, “Performance analysis of hyperledger fabric platforms,” *Security and Communication Networks,* vol. 2018, pp. 3976093, Sep. 2018. [Online]. Available: https://www.hindawi.com/journals/scn/2018/3976093/abs/

[102] M. Raikwar, D. Gligoroski, and K. Kralevska, “SoK of used cryptography in blockchain,” *IEEE Access*, vol. 7, pp. 148550–148575, 2019. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8865045

[103] S. Zhang and J.-H. Lee, “Analysis of the main consensus protocols of blockchain,” *ICT Express*, vol. 6, no. 2, pp. 93–97, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S240595951930164X

[104] H. Vranken, “Sustainability of bitcoin and blockchains,” *Current opinion in environmental sustainability*, vol. 28, pp. 1–9, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877343517300015

[105] C. Baraniuk. “Bitcoin’s energy consumption ‘equals that of Switzerland.’” *BBC News*, Jul. 03, 2019. [Online]. Available: https://www.bbc.com/news/technology-48853230

[106] V. Denisova, A. Mikhaylov, and E. Lopatin, “Blockchain infrastructure and growth of global power consumption,” *International Journal of Energy Economics and Policy*, vol. 9, no. 4, p. 22, 2019. [Online]. Available: https://search.proquest.com/docview/2256121380

[107] R. L. Rana, P. Giungato, A. Tarabella, and C. Tricase, “Sustainability of bitcoins and blockchain,” in *Proceedings of BASIQ International Conference on New Trends in Sustainable Business and Consumption*, 2019, pp. 771–777. [Online]. Available: https://www.researchgate.net/profile/Ann_Katrin_Arp/publication/333902657_Study_on_European_funding_programmes_for_sustainable_development

[108] D. Larimer, “Transactions as proof-of-stake,” *Crypto Chain University*, Nov. 28, 2013. [Online]. Available: https://cryptochainuni.com/wp-content/uploads/Invictus-Innovations-Transactions-As-Proof-Of-Stake.pdf

[109] E. Deirmentzoglou, G. Papakyriakopoulos, and C. Patsakis, “A survey on long-range attacks for proof of stake protocols,” *IEEE Access*, vol. 7, pp. 28712–28725, 2019. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8653269/

[110] V. Buterin and V. Griffith, “Casper the friendly finality gadget,” *arXiv preprint arXiv:1710.09437*, 2017. [Online]. Available: https://arxiv.org/abs/1710.09437

[111] EthHub, “Proof of Stake (PoS),” *Ethereum 2.0*. Accessed September 15, 2020. [Online]. Available: https://docs.ethhub.io/ethereum-roadmap/ethereum-2.0/proof-of-stake/

[112] W. Foxley, “Ethereum 2.0 countdown begins with release of deposit contract,” *Coindesk*, Nov. 04, 2020. [Online]. Available: https://www.coindesk.com/ethereum-2-0-contract-deposit-mainnet

[113] M. Castro and B. Liskov, “Practical Byzantine Fault Tolerance,” in *OSDI*, 1999, vol. 99, no. 1999, pp. 173–186. [Online]. Available: http://www.pmg.lcs.mit.edu/papers/osdi99.pdf

[114] X. Hao, L. Yu, L. Zhiqiang, L. Zhen, and G. Dawu, “Dynamic Practical Byzantine Fault Tolerance,” in *2018 IEEE Conference on Communications and Network Security (CNS)*, 2018, pp. 1–8. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8433150/

[115] O. Onireti, L. Zhang, and M. A. Imran, “On the viable area of wireless Practical Byzantine Fault Tolerance (PBFT) blockchain networks,” in *2019 IEEE Global Communications Conference (GLOBECOM)*, 2019, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9013778

[116] X. Fan, “Scalable practical byzantine fault tolerance with short-lived signature schemes,” in *Proceedings of the 28th Annual International Conference on Computer Science and Software Engineering*, 2018, pp. 245–256. [Online]. Available: https://v1.iotex.io/academics-pbft.pdf

[117] D. Mohanty, *R3 Corda for Architects and Developers: With Case Studies in Finance, Insurance, Healthcare, Travel, Telecom, and Agriculture*. New York, NY, USA: Apress, 2019. [Online]. Available: https://www.google.com/books/edition/R3_Corda_for_Architects_and_Developers/T_-eDwAAQBAJ?hl=en&gbpv=1&dq=Mohanty,+Debajani.+R3+Corda+for+Architects+and+Developers:+With+Case+Studies+in+Finance,+Insurance,+Healthcare,+Travel,+Telecom,+and+Agriculture.+Apress,+2019.&pg=PR3&printsec=frontcover

[118] M. K. Shrivas and D. T. Yeboah, “The Disruptive Blockchain: Types Platforms and Applications,” in *Fifth Texila World Conference for Scholars (TWCS) on Transformation: The Creative Potential of Interdisciplinary & Mutlidisciplinary Knowledge Exchange*, 2018. [Online]. Available: https://pdfs.semanticscholar.org/a0c2/c517e8869493219fa739e3418d7caa825845.pdf

[119] C. Cachin and M. Vukolić, “Blockchain consensus protocols in the wild,” *arXiv preprint arXiv:1707.01873*, 2017. [Online]. Available: https://arxiv.org/abs/1707.01873

[120] C. D. Clack, V. A. Bakshi, and L. Braine, “Smart contract templates: foundations, design landscape and research directions,” *arXiv preprint arXiv:1608.00771*, 2016. [Online]. Available: https://arxiv.org/abs/1608.00771

[121] A. Serada, T. Sihvonen, and J. T. Harviainen, “CryptoKitties and the new ludic economy: how blockchain introduces value, ownership, and scarcity in digital gaming,” *Games and Culture*, Feb. 20, 2020. [Online]. Available: https://journals.sagepub.com/doi/abs/10.1177/1555412019898305

[122] M. Raikwar, S. Mazumdar, S. Ruj, S. S. Gupta, A. Chattopadhyay, and K.-Y. Lam, “A blockchain framework for insurance processes,” in *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, 2018, pp. 1–4. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8328731/

[123] B. K. Mohanta, S. S. Panda, and D. Jena, “An overview of smart contract and use cases in blockchain technology,” in *2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, 2018, pp. 1–4. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8494045/

[124] K. Christidis and M. Devetsikiotis, “Blockchains and smart contracts for the internet of things,” *Ieee Access*, vol. 4, pp. 2292–2303, 2016. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7467408/

[125] W. Metcalfe, “Ethereum, Smart Contracts, DApps,” in *Blockchain and Crypt Currency*, Springer, Singapore, 2020, pp. 77–93. [Online]. Available: https://link.springer.com/chapter/10.1007/978-981-15-3376-1_5

[126] W. Cai, Z. Wang, J. B. Ernst, Z. Hong, C. Feng, and V. C. Leung, “Decentralized applications: The blockchain-empowered software system,” *IEEE Access*, vol. 6, pp. 53019–53033, 2018. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8466786/

[127] D. Dolenc, J. Turk, and M. Pustišek, “Distributed Ledger Technologies for IoT and Business DApps,” in *2020 International Conference on Broadband Communications for Next Generation Networks and Multimedia Applications (CoBCom)*, 2020, pp. 1–8. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9174188/

[128] D. Vujičić, D. Jagodić, and S. Ranđić, “Blockchain technology, bitcoin, and Ethereum: A brief overview,” in *2018 17th International Symposium Infoteh-Jahorina (infoteh)*, 2018, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8345547/

[129] D. Yaga, P. Mell, N. Roby, and K. Scarfone, “Blockchain technology overview,” *arXiv preprint arXiv:1906.11078*, 2019. [Online]. Available: https://arxiv.org/abs/1906.11078

[130] L. Kiffer, D. Levin, and A. Mislove, “Stick a fork in it: Analyzing the ethereum network partition,” in *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, 2017, pp. 94–100. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/3152434.3152449

[131] R. R. Vokerla *et al.*, "An Overview of Blockchain Applications and Attacks," in *2019 International Conference on Vision Towards Emerging Trends in Communication and Networking (ViTECoN)*, 2019, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8899450/

[132] Werner, Sam M. et al. "Uncle traps: harvesting rewards in a queue-based ethereum mining pool." in *Proceedings of the 12th EAI International Conference on Performance Evaluation Methodologies and Tools*, 2019. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/3306309.3306328

[133] Ethereum, "Design rationale – Uncle incentivization," *Ethereum Wiki*. [Online]. Available: https://eth.wiki/en/fundamentals/design-rationale

[134] Z. Wang, J. Liu, Q. Wu, Y. Zhang, H. Yu, and Z. Zhou, "An analytic evaluation for the impact of uncle blocks by selfish and stubborn mining in an imperfect Ethereum network," *Computers & Security*, vol. 87, p. 101581, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404818314172

[135] S.-Y. Chang, Y. Park, S. Wuthier, and C.-W. Chen, "Uncle-block attack: Blockchain mining threat beyond block withholding for rational and uncooperative miners," in *International Conference on Applied Cryptography and Network Security*, 2019, pp. 241–258. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-21568-2_12

[136] M. Zamani, M. Movahedi, and M. Raykova, "Rapidchain: Scaling blockchain via full sharding," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 931–948. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/3243734.3243853

[137] M. Drake, "Understanding Database Sharding." *Digital Ocean*, 7 Feb. 2019. [Online]. Available: https://www.digitalocean.com/community/tutorials/understanding-database-sharding

[138] G. Wang, Z. J. Shi, M. Nixon, and S. Han, "Sok: Sharding on blockchain," in *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, 2019, pp. 41–61. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/3318041.3355457

[139] H. Dang, T. T. A. Dinh, D. Loghin, E.-C. Chang, Q. Lin, and B. C. Ooi, "Towards scaling blockchain systems via sharding," in *Proceedings of the 2019 international conference on management of data*, 2019, pp. 123–140. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/3299869.3319889

[140] L. Brankovic, C. S. Iliopoulos, R. Kundu, M. Mohamed, S. P. Pissis, and F. Vayani, "Linear-time superbubble identification algorithm for genome assembly," *Theoretical Computer Science*, vol. 609, pp. 374–383, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0304397515009147

[141] S. Tikhomirov, “Ethereum: state of knowledge and research perspectives,” in *International Symposium on Foundations and Practice of Security*, 2017, pp. 206–221. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-75650-9_14

[142] H. Pervez, M. Muneeb, M. U. Irfan, and I. U. Haq, “A comparative analysis of DAG-based blockchain architectures,” in *2018 12th International Conference on Open Source Systems and Technologies (ICOSST)*, 2018, pp. 27–34. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8632193/

[143] L. Hofer, “DAG vs. blockchain – technologies for different use cases,” *ICO.li*, 23 Apr. 2019. [Online]. Available: https://www.ico.li/dag-vs-blockchain/

[144] Y. Li *et al.*, “Direct Acyclic Graph-based ledger for Internet of Things: performance and security analysis,” *IEEE/ACM Transactions on Networking*, 2020. [Online]. Available: https://www.researchgate.net/profile/Daquan_Feng/publication/333418335_Direct_Acyclic_Graph_based_Blockchain_for_Internet_of_Things_Performance_and_Security_Analysis

[145] S. Amani, M. Bégel, M. Bortin, and M. Staples, “Towards verifying ethereum smart contract bytecode in Isabelle/HOL,” in *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs*, 2018, pp. 66–77. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/3167084

[146] P. Hegedűs, “Towards analyzing the complexity landscape of solidity based ethereum smart contracts,” *Technologies*, vol. 7, no. 1, p. 6, 2019. [Online]. Available: https://www.mdpi.com/2227-7080/7/1/6

[147] Ethereum Wiki, “Dagger Hashimoto,” *Github*, Aug. 21, 2018. [Online]. Available: https://github.com/ethereum/wiki/wiki/Dagger-Hashimoto

[148] T. Chen *et al.*, “An adaptive Gas cost mechanism for Ethereum to defend against under-priced DoS attacks,” in *International Conference on Information Security Practice and Experience*, 2017, pp. 3–24. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-72359-4_1

[149] S. M. Werner, P. J. Pritz, and D. Perez, “Step on the Gas? a better approach for recommending the Ethereum Gas price,” *arXiv preprint arXiv:2003.03479*, 2020. [Online]. Available: https://arxiv.org/abs/2003.03479

[150] M. Schäffer, M. Di Angelo, and G. Salzer, “Performance and scalability of private Ethereum blockchains,” in *International Conference on Business Process Management*, 2019, pp. 103–118. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-30429-4_8

[151] MyEtherWallet, “What is Gas?” *MEW Help Center*. [Online]. Available: https://kb.myetherwallet.com/en/transactions/what-is-gas/

[152] F. Ritz and A. Zugenmaier, “The impact of uncle rewards on selfish mining in Ethereum,” in *2018 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, 2018, pp. 50–57. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8406560

[153] V. Buterin, “Toward a 12-second Block Time,” *Ethereum Blog*, Jul. 11, 2014. [Online]. Available: https://blog.ethereum.org/2014/07/11/toward-a-12-second-block-time/

[154] F. Liu, X. Wang, Z. Li, J. Xu, and Y. Gao, “Effective Gas Price prediction for carrying out economical Ethereum transaction,” in *2019 6th International Conference on Dependable Systems and Their Applications (DSA)*, 2020, pp. 329–334. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9045840

[155] H. Zhang, C. Jin, and H. Cui, “A method to predict the performance and storage of executing contract for Ethereum consortium-blockchain,” in *International Conference on Blockchain*, 2018, pp. 63–74. [Online]. Available: https://link.springer.com/chapter/10.1007%2F978-3-319-94478-4_5

[156] P. E. Black, “Dictionary of algorithms and data structures,” Gaithersburg, MD, USA, National Institute of Standards and Technology, 2004. [Online]. Available: https://xlinux.nist.gov/dads/

[157] D. Brickwood, “Understanding trie databases in Ethereum,” *Shyft*, 02 Aug. 2018. [Online]. Available: https://medium.com/shyft-network-media/understanding-trie-databases-in-ethereum-9f03d2c3325d

[158] G. I. Hotchkiss, “The 1.x files: The state of stateless Ethereum,” *Ethereum Blog*, 30 Dec. 2019. [Online]. Available: https://blog.ethereum.org/2019/12/30/eth1x-files-state-of-stateless-ethereum/

[159] E. Hildenbrandt *et al.*, “Kevm: A complete formal semantics of the Ethereum Virtual Machine,” in *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*, 2018, pp. 204–217. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8429306

[160] A. M. Antonopoulos and G. Wood, *Mastering ethereum: building smart contracts and dapps*. Sebastopol, CA, USA: O’Reilly Media, 2018. [Online]. Available: https://github.com/ethereumbook/ethereumbook

[161] Ethereum, “Fundamentals – RLP,” *Ethereum Wiki*. [Online]. Available: https://eth.wiki/en/fundamentals/rlp

[162] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014. [Online]. Available: https://ethereum.github.io/yellowpaper/paper.pdf

[163] M. Tavares, A. Guerreiro, C. Coutinho, F. Veiga, and A. Campos, "WalliD: Secure your ID in an Ethereum wallet," in *2018 International Conference on Intelligent Systems (IS)*, 2018, pp. 714–721. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8710547

[164] V. Thakur, M. N. Doja, Y. K. Dwivedi, T. Ahmad, and G. Khadanga, "Land records on blockchain for implementation of land titling in India," *International Journal of Information Management*, vol. 52, p. 101940, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0268401219303329

[165] C. Yue *et al.*, "Analysis of indexing structures for immutable data," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 925–935. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/3318464.3389773

[166] Jack of all trades, "Ethereum Primitives #1.2," *Coinmonks*. 27 Aug. 2020. [Online]. Available: https://medium.com/coinmonks/ethereum-primitives-1-2-e7ce0fa0a84c

[167] P. Fairley, "Ethereum will cut back its absurd energy use," *IEEE spectrum*, vol. 56, no. 1, pp. 29–32, 2018. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8594790/

[168] V. Buterin, "Ethereum 2.0 mauve paper," in *Ethereum developer conference*, 2016, vol. 2. [Online]. Available: https://cdn.hackaday.io/files/10879465447136/Mauve%20Paper%20Vitalik.pdf

[169] B. Pirus, "Vitalik: We underestimated how long Proof-of-Stake and sharding would take to complete," *Cointelegraph*, 03 Jul. 2020. [Online]. Available: https://cointelegraph.com/news/vitalik-we-underestimated-how-long-proof-of-stake-and-sharding-would-take-to-complete

[170] Ethereum, "Latest status," *Ethereum 2*.0, 14, Oct. 2020. [Online]. Available: https://ethereum.org/en/eth2/

[171] Buterin, Vitalik et al. "Incentives in Ethereum's hybrid Casper protocol." 2019 IEEE international conference on blockchain and cryptocurrency (ICBC). IEEE, 2019. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8751241/

[172] V. Buterin *et al.*, "Combining GHOST and Casper," *arXiv preprint arXiv:2003.03052*, 2020. [Online]. Available: https://arxiv.org/abs/2003.03052

[173] B. Prashant and I. Makrant, “Migration from POW to POS for Ethereum.” *engrXiv Preprints*, 01, Oct. 2019. [Online]. Available: https://engrxiv.org/ad8en/

[174] Ethereum, “Upgrading Ethereum to radical new heights.” Accessed Novemer 20, 2020. [Online]. Available: https://ethereum.org/en/eth2/

[175] M. Saad *et al.*, “Exploring the attack surface of blockchain: A systematic overview,” *arXiv preprint arXiv:1904.03487*, 2019. [Online]. Available: https://arxiv.org/abs/1904.03487

[176] S. Sayeed and H. Marco-Gisbert, “Assessing blockchain consensus and security mechanisms against the 51% attack,” *Applied Sciences*, vol. 9, no. 9, p. 1788, 2019. [Online]. Available: https://www.mdpi.com/2076-3417/9/9/1788

[177] I.-C. Lin and T.-C. Liao, “A survey of blockchain security issues and challenges.,” *IJ Network Security*, vol. 19, no. 5, pp. 653–659, 2017. [Online]. Available: http://ijns.jalaxy.com.tw/contents/ijns-v19-n5/ijns-2017-v19-n5-p653-659.pdf

[178] M. Bastiaan, “Preventing the 51%-attack: a stochastic analysis of two phase proof of work in bitcoin,” presented at 22nd Twente Student Conf. on IT Jan. 23rd, Enschede, The Netherlands, 2015. [Online]. Available: https://fmt.ewi.utwente.nl/media/175.pdf

[179] S. Shanaev, A. Shuraeva, M. Vasenin, and M. Kuznetsov, “Cryptocurrency value and 51% attacks: evidence from event studies,” *The Journal of Alternative Investments*, vol. 22, no. 3, pp. 65–77, 2019. [Online]. Available: https://www.mdpi.com/2076-3417/9/9/1788

[180] B. Bambrough, “Bitcoin rival suffers devastating attack,” *Forbes*, Jan. 28, 2020. [Online]. Available: https://www.forbes.com/sites/billybambrough/2020/01/28/bitcoin-rival-suffers-devastating-attack/

[181] D. Efanov and P. Roschin, “The all-pervasiveness of the blockchain technology,” *Procedia Computer Science*, vol. 123, pp. 116–121, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050918300206

[182] I. Eyal and E. G. Sirer, “Majority is not enough: Bitcoin mining is vulnerable,” in *International conference on financial cryptography and data security*, 2014, pp. 436–454. [Online]. Available: https://www.cs.cornell.edu/~ie53/publications/btcProcFC.pdf

[183] A. Sapirshtein, Y. Sompolinsky, and A. Zohar, “Optimal selfish mining strategies in bitcoin,” in *International Conference on Financial Cryptography and Data Security*, 2016, pp. 515–532. [Online]. Available: https://arxiv.org/abs/1507.06183

[184] M. Saad, L. Njilla, C. Kamhoua, and A. Mohaisen, “Countering selfish mining in blockchains,” in *2019 International Conference on Computing, Networking and Communications (ICNC)*, 2019, pp. 360–364. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8685577

[185] R. Zhang, J. Zhang, Y. Chen, N. Qin, B. Liu, and Y. Zhang, “Making eclipse attacks computationally infeasible in large-scale DHTs,” in *30th IEEE International Performance Computing and Communications Conference*, 2011, pp. 1–8. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6108091/

[186] G. Bissias, B. N. Levine, A. P. Ozisik, and G. Andresen, “An analysis of attacks on blockchain consensus,” *arXiv preprint arXiv:1610.07985*, 2016. [Online]. Available: https://arxiv.org/abs/1610.07985

[187] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, “Eclipse attacks on bitcoin’s peer-to-peer network,” in *24th {USENIX} Security Symposium ({USENIX} Security 15)*, 2015, pp. 129–144. [Online]. Available: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/heilman

[188] G. Xu *et al.*, “Am I eclipsed? A smart detector of eclipse attacks for Ethereum,” *Computers & Security*, vol. 88, p. 101604, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404818313798

[189] A. Ganon, “Creating a Bitcoin node using an ODROID-HC2,” *Odroid Magazine*, May 01, 2019. [Online]. Available: https://magazine.odroid.com/article/creating-a-bitcoin-node-using-an-odroid-hc2/

[190] T. Okada, “Handle smart contract on Ethereum with Arduino or ESP32,” *Medium*, Mar. 02, 2018. [Online]. Available: https://medium.com/@takahirookada/handle-smart-contract-on-ethereum-with-arduino-or-esp32-1bb5cbaddbf4

[191] S. Provoost, “Bitcoin on an Orange Pi (using Armbian),” *Medium*, Jun. 28, 2018. [Online]. Available: https://medium.com/provoost-on-crypto/bitcoin-on-an-orange-pi-using-armbian-66c3523bbfc0

[192] W. F. Slater, “Introduction to setting up Ethereum on a small Raspberry Pi network,” *Chicago Blockchain Project*, Apr. 21, 2018. [Online]. Available: http://www.billslater.com/blockchain/EthereumonRaspberryPibyWmFSlaterIIIv01.0.pdf

[193] K. McCullen and M. Walters, “Computer science and robotics using the Raspberry Pi, Arduino, and other SBCS,” *Journal of Computing Sciences in Colleges*, vol. 33, no. 6, pp. 157–159, 2018. [Online]. Available: https://dl.acm.org/doi/abs/10.5555/3205191.3205213

[194] B. Qureshi and A. Koubaa, “On energy efficiency and performance evaluation of SBC based clusters: A Hadoop case study,” *arXiv preprint arXiv:1903.06648*, 2019. [Online]. Available: https://arxiv.org/abs/1903.06648

[195] S. J. Johnston *et al.*, “Commodity single board computer clusters and their applications,” *Future Generation Computer Systems*, vol. 89, pp. 201–212, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X18301833

[196] L. Buechley and M. Eisenberg, “The LilyPad Arduino: Toward wearable engineering for everyone,” *IEEE Pervasive Computing*, vol. 7, no. 2, pp. 12–15, 2008. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/4487082

[197] S. Ferdoush and X. Li, “Wireless sensor network system design using Raspberry Pi and Arduino for environmental monitoring applications,” *Procedia Computer Science*, vol. 34, pp. 103–110, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050914009144

[198] A. Araújo, D. Portugal, M. S. Couceiro, and R. P. Rocha, “Integrating Arduino-based educational mobile robots in ROS,” *Journal of Intelligent & Robotic Systems*, vol. 77, no. 2, pp. 281–298, 2015. [Online]. Available: https://link.springer.com/article/10.1007/s10846-013-0007-4

[199] Arduino, “Arduino Due,” Accessed July 15, 2018. [Online]. Available: https://store.arduino.cc/usa/due

[200] Arduino, “Getting started with Arduino Due,” Accessed July 15, 2018. [Online]. Available: https://www.arduino.cc/en/Guide/ArduinoDue

[201] Arduino, “Arduino Forum,” Accessed July 15, 2018. [Online]. Available: https://forum.arduino.cc/

[202] Arduino, “The Arduino Playground,” Accessed July 15, 2018 [Online] Available: https://playground.arduino.cc/

[203] AliExpress, “Shenzhen Xunlong Software CO.,Limited.” Accessed July 21, 2018 [Online]. Available: https://www.aliexpress.com/item/32761481418.html

[204] Orange Pi, “What’s Orange Pi PC 2?.” Accessed July 21, 2018. [Online]. Available: http://www.orangepi.org/orangepipc2/

[205] AliExpress, “NanoPi A64.” Accessed July 01, 2018. [Online]. Available: https://www.aliexpress.com/item/32813409357.html

[206] SocialCompare, “Nano PCs – Comparaison des Nano PCs.” Accessed July 08, 2018. [Online]. Available: https://socialcompare.com/en/comparison/nano-pcs-3v4onooz

[207] V. Vujović and M. Maksimović, “Raspberry Pi as a Sensor Web node for home automation,” *Computers & Electrical Engineering*, vol. 44, pp. 153–171, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0045790615000257

[208] C. W. Zhao, J. Jegatheesan, and S. C. Loon, “Exploring IoT application using Raspberry Pi,” *International Journal of Computer Networks and Applications*, vol. 2, no. 1, pp. 27–34, 2015. [Online]. Available: https://ijcna.org/Manuscripts/Volume-2/Issue-1/Vol-2-issue-1-M-04.pdf

[209] R. Emani, E. J. Glantz, C. Gamrat, and M. K. Hills, “Using the Raspberry Pi in IT Education,” in *Proceedings of the 20th Annual SIG Conference on Information Technology Education*, 2019, pp. 153–153. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/3349266.3351373

[210] Raspberry Pi, “Raspberry Pi 3 Model B.” Accessed May 31, 2018. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

[211] Raspberry Pi, “Board Index – Community.” Accessed May 31, 2018. [Online]. Available: https://www.raspberrypi.org/forums/

[212] Wolfram, “Community – Groups – Raspberry Pi.” Accessed July 01, 2020. [Online]. Available: https://community.wolfram.com/content?curTag=raspberry%20pi

[213] Embedded Linux Wiki, “Raspberry Pi Wiki | Hub.” Accessed July 08, 2018. [Online]. Available: https://elinux.org/RPi_Hub

[214] The Pi, “How to install Raspbian on the Raspberry Pi.” Accessed March 08, 2018. [Online]. Available: https://thepi.io/how-to-install-raspbian-on-the-raspberry-pi/

[215] J. Van Schoonveld, “Build physical projects with Python on the Raspberry Pi,” *Real Python*, 1 Jun. 2020. [Online]. Available: https://realpython.com/python-raspberry-pi/

[216] W. M. Lee and C. Chng, “Introduction to IoT using the Raspberry Pi,” *Code Magazine*, 2018. [Online]. Available: https://www.codemag.com/Article/1607071/Introduction-to-IoT-Using-the-Raspberry-Pi

[217] Core-Geth, “How to setup an Ethereum node on Raspberry Pi.” Accessed April 15, 2018. [Online]. Available: https://core-geth.org/setup-on-raspberry-pi

[218] E. Fernando, “Blockchain technology implementation in Raspberry Pi for private network,” in *2019 International Conference on Sustainable Information Engineering and Technology (SIET)*, 2019, pp. 154–158. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8986053

[219] A. Beck, "Exploring Ethereum with Raspberry Pi - Part 1: Getting Started," *DesignSpark*, Jan. 31, 2018. [Online]. Available: https://www.rs-online.com/designspark/exploring-ethereum-with-raspberry-pi-part-1-getting-started

[220] A. Goranović, M. Meisel, S. Wilker, and T. Sauter, "Hyperledger fabric smart grid communication testbed on raspberry PI ARM architecture," in *2019 15th IEEE International Workshop on Factory Communication Systems (WFCS)*, 2019, pp. 1–4. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8758000

[221] A. Kiayias and G. Panagiotakos, "On trees, chains and fast transactions in the blockchain," in *International Conference on Cryptology and Information Security in Latin America*, 2017, pp. 327–351. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-25283-0_18

[222] Banana Pi Open Source Project, "BPI-M64." Accessed July 02, 2018. [Online]. Available: http://www.banana-pi.org/m64.html

[223] BPI Wiki, "Banana Pi BPI-M64," Accessed July 02, 2018. [Online]. Available: http://wiki.banana-pi.org/Banana_Pi_BPI-M64

[224] PINE64, "ROCKPro64: A 64-bit Hexa-Core SBC with PCIe, USB 3.0, USB-C and gigabit Ethernet featuring up-to 4GB of LDDDR4 RAM." Accessed July 05, 2018. [Online]. Available: https://www.pine64.org/rockpro64/

[225] PINE64, "PINE64." Accessed July 05, 2018. [Online]. Available: https://forum.pine64.org/

[226] HardKernal, "ODROID-XU4." Accessed July 12, 2018. [Online]. Available: https://www.hardkernel.com/shop/odroid-xu4-special-price/

[227] R. Roy, "ODROID-XU4 Manual." *Odroid Magazine*, 01 Jan. 2017. [Online]. Available: https://magazine.odroid.com/odroid-xu4/

[228] ODROID Wiki, "ODROID-XU4." Accessed July 15, 2018. [Online]. Available: https://wiki.odroid.com/odroid-xu4/odroid-xu4

[229] UP shop, "UP board Series." Accessed August 01, 2018. [Online]. Available: https://up-shop.org/up-board-series.html

[230] UP, "UP – bridge the gap." Accessed August 01, 2018. [Online]. Available: https://up-board.org/

[231] LattePanda, "LattePanda4G/64G." Accessed July 22, 2018. [Online] Available: https://www.lattepanda.com/products/3.html

[232] LattePanda, “LattePanda – A Windows10 Development Board For Everything.” Accessed July 22, 2018. [Online]. Available: https://www.lattepanda.com/

[233] Sam, “Arduino with LattePanda,” *Core Electronics*, 16 Jun. 2017. [Online]. Available: https://core-electronics.com.au/tutorials/arduino-with-lattepanda.html

[234] LattePanda, “LattePanda Community Center.” Accessed July 22, 2018. [Online]. Available: https://www.lattepanda.com/forum/

[235] Udoo, “Udoo X86 II Ultra.” Accessed August 08, 2018. [Online]. Available: https://shop.udoo.org/udoo-x86-ii-ultra.html

[236] Udoo, “Udoo X86 II.” Accessed August 08, 2018. [Online]. Available: https://www.udoo.org/udoo-x86/

[237] Udoo x86 Documentation. “Getting Started with Arduino 101.” Accessed August 08, 2018. [Online]. Available: https://www.udoo.org/docs-x86/Arduino_101_(Intel_Curie)/Getting_Started_with_Arduino_101.html

[238] Udoo x86 Documentation. “GPIOs.” Accessed August 08, 2018. [Online]. Available: https://www.udoo.org/docs-x86/Hardware_&_Accessories/GPIOs.html

[239] A. Rizzo, G. Burresi, F. Montefoschi, M. Caporali, and R. Giorgi, “Making IoT with UDOO.,” *IxD&A*, vol. 30, pp. 95–112, 2016. [Online]. Available: http://www.mifav.uniroma2.it/inevent/events/idea2010/doc/30_6.pdf

[240] BBC News, “Raspberry Pi used to steal data from NASA lab,” Jun. 24, 2019. [Online]. Available: https://www.bbc.com/news/technology-48743043

[241] D’Herdt, J. “Detecting crypto currency mining in corporate environments,” *SANS Institute InfoSec Reading Room*, Feb. 04, 2015. [Online]. Available: https://www.sans.org/reading-room/whitepapers/threats/paper/35722

[242] S. Eskandari, A. Leoutsarakos, T. Mursch, and J. Clark, “A first look at browser-based cryptojacking,” in *2018 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, 2018, pp. 58–66. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8406561

[243] I. H. Saruhan, “Detecting and preventing rogue devices on the network,” *SANS Institute InfoSec Reading Room*, Aug. 13, 2007. [Online]. Available: https://www.sans.org/reading-room/whitepapers/detection/paper/1866

[244] L. Watkins, R. Beyah, and C. Corbett, “A passive approach to rogue access point detection,” in *IEEE GLOBECOM 2007-IEEE Global Telecommunications Conference*, 2007, pp. 355–360. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/4410983

[245] B. Alotaibi and K. Elleithy, “Rogue access point detection: Taxonomy, challenges, and future directions,” *Wireless Personal Communications*, vol. 90, no. 3, pp. 1261–1290, 2016. [Online]. Available: https://link.springer.com/article/10.1007/s11277-016-3390-x

[246] S. Lande and R. Zunino, “SoK: unraveling Bitcoin smart contracts,” *Principles of Security and Trust LNCS 10804*, p. 217, 2018. [Online]. Available: https://library.oapen.org/bitstream/handle/20.500.12657/27744/1002261.pdf?sequence=1#page=225

[247] M. T. Alam, H. Li, and A. Patidar, “Bitcoin for smart trading in smart grid,” in *The 21st IEEE International Workshop on Local and Metropolitan Area Networks*, 2015. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7114742

[248] Bitcoin, “Bitcoin Communities.” Accessed August 12, 2018. [Online]. Available: https://bitcoin.org/en/community

[249] BitcoinDeveloper, “Developer Guides.” Accessed August 12, 2018. [Online]. Available: https://developer.bitcoin.org/devguide/

[250] S. Roche, “Introduction to Bitcoin programming with BitcoinJS and Bitcoin Core,” *Bitcoin Developer Network*. 2020. [Online]. Available: https://bitcoindev.network/introduction-to-bitcoin-programming-with-bitcoinjs-and-bitcoin-core/

[251] S. Huh, S. Cho, and S. Kim, “Managing IoT devices using blockchain platform,” in *2017 19th International Conference on Advanced Communication Technology (ICACT)*, 2017, pp. 464–467. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7890132

[252] J. Mallett, “Scaling and consensus in monetary systems.,” Dept. of Comp. Sci., Reykjavik University, Reykjavik, Iceland, Feb. 10, 2020. [Online]. Available: https://assets.pubpub.org/i0dxgael/21581338973988.pdf

[253] F. C. Maldonado, *Introduction to Blockchain and Ethereum: Use distributed ledgers to validate digital transactions in a decentralized and trustless manner*. Birmingham, England, UK: Packt Publishing Ltd, 2018. [Online]. Available: https://books.google.com/books?hl=en&lr=&id=fP1wDwAAQBAJ&oi=fnd&pg=PR1&dq=+Parker,+David+W.+Introduction+to+Blockchain+and+Ethereum:+Use+Distributed+Ledgers+to+Validate+Digital+Transactions+in+a+Decentralized+and+Trustless+Manner.+Packt,+2018.&ots=pIRH1ZmE9X&sig=xHHaPsYp76qITP001rIV0y-R9_E

[254] Web3.js. “web3.js - Ethereum JavaScript API.” Accessed July 7, 2018. [Online]. Available: https://web3js.readthedocs.io/en/v1.2.11/

[255] C. Dannen, “Smart Contracts and Tokens,” in *Introducing Ethereum and Solidity*, Springer, 2017, pp. 89–110. [Online]. Available: https://link.springer.com/chapter/10.1007/978-1-4842-2535-6_5

[256] Ethereum, “Ethereum Community Forum.” Accessed July 08, 2018. [Online]. Available: https://forum.ethereum.org/

[257] Ethereum, “Smart Contracts and DApps.” Accessed July 08, 2018. [Online]. Available: https://forum.ethereum.org/categories/services-and-decentralized-applications

[258] Ethereum, “Solidity.” Accessed July 08, 2018. [Online]. Available: https://forum.ethereum.org/categories/solidity

[259] Ethereum, “Ethereum Development Tutorials.” Accessed July 08, 2018. [Online]. Available: https://ethereum.org/en/developers/tutorials/

[260] Tutorialspoint, “Ethereum Smart Contracts.” Accessed July 04, 2018. [Online]. Available: https://www.tutorialspoint.com/ethereum/ethereum_smart_contracts.htm

[261] Go Ethereum, “Private Network Tutorial.” Accessed July 04, 2018. [Online]. Available: https://geth.ethereum.org/getting-started/private-net

[262] T. Filatov, “Setting up your private Ethereum blockchain,” *Dappros*, 08 Apr. 2020. [Online]. Available: https://www.dappros.com/202004/setting-up-your-private-ethereum-blockchain/

[263] V. Saini, “How to reduce block difficulty in Ethereum private testnet,” *Hackernoon*, Jun. 08, 2018. [Online]. Available: https://hackernoon.com/how-to-reduce-block-difficulty-in-ethereum-private-testnet-2ad505609e82

[264] Hydrachain, “Project description.” Accessed August 05, 2018. [Online]. Available: https://pypi.org/project/hydrachain/

[265] W.-T. Tsai, R. Blower, Y. Zhu, and L. Yu, “A system view of financial blockchains,” in *2016 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, 2016, pp. 450–457. [Online]. Available: https://ieeexplore.ieee.org/document/7473060

[266] P. Sajana, M. Sindhu, and M. Sethumadhavan, “On blockchain applications: Hyperledger fabric and Ethereum,” *International Journal of Pure and Applied Mathematics*, vol. 118, no. 18, pp. 2965–2970, 2018. [Online]. Available: https://acadpubl.eu/jsi/2018-118-18/articles/18c/84.pdf

[267] Qtum, “Qtum x86 virtual machine & Qtum Enterprise version — progress update,” June 02, 2018. [Online]. Available: https://blog.qtum.org/qtum-x86-virtual-machine-qtum-enterprise-version-progress-update-2ff249968e2b

[268] Qtum, “Qtum -Defining the blockchain economy.” Accessed July 16, 2018. [Online]. Available: https://qtum.org/en

[269] A. Norta, P. Dai, N. Mahi, and J. Earls, “A public, blockchain-based distributed smart-contract platform enabling mobile lite wallets using a Proof-of-Stake consensus algorithm,” in *International Conference on Business Information Systems*, 2018, pp. 368–380. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-04849-5_33

[270] Hyperledger, “Hyperledger Fabric.” Accessed August 03, 2018. [Online]. Available: https://www.hyperledger.org/use/fabric

[271] C. Cachin, “Architecture of the Hyperledger blockchain fabric,” in *Workshop on distributed cryptocurrencies and consensus ledgers*, 2016, vol. 310, no. 4. [Online]. Available: https://pdfs.semanticscholar.org/f852/c5f3fe649f8a17ded391df0796677a59927f.pdf

[272] M. Valenta and P. Sandner, “Comparison of Ethereum, Hyperledger Fabric and Corda,” working paper, Frankfurt School Blockchain Center, Frankfurt am Main, Germany, 2017. [Online]. Available: http://www.smallake.kr/wp-content/uploads/2017/07/2017_Comparison-of-Ethereum-Hyperledger-Corda.pdf

[273] L. S. Sankar, M. Sindhu, and M. Sethumadhavan, “Survey of consensus protocols on blockchain applications,” in *2017 4th International Conference on Advanced Computing and Communication Systems (ICACCS)*, 2017, pp. 1–5. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8014672

[274] H. Sukhwani, J. M. Martínez, X. Chang, K. S. Trivedi, and A. Rindos, “Performance modeling of PBFT consensus process for permissioned blockchain network (hyperledger fabric),” in *2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS)*, 2017, pp. 253–255. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8069090

[275] E. Androulaki *et al.*, “Hyperledger fabric: a distributed operating system for permissioned blockchains,” in *Proceedings of the thirteenth EuroSys conference*, 2018, pp. 1–15. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/3190508.3190538

[276] R. G. Brown, J. Carlyle, I. Grigg, and M. Hearn, “Corda: an introduction,” *R3 CEV*, vol. 1, p. 15, Aug., 2016. [Online]. Available: https://www.researchgate.net/profile/Ian_Grigg/publication/308636477_Corda_An_Introduction/links/57e994ed08aed0a291304412.pdf

[277] R. M. Nadir, "Comparative study of permissioned blockchain solutions for enterprises," in *2019 International Conference on Innovative Computing (ICIC)*, 2019, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8966735

[278] Openchain, "Blockchain technology for the enterprise." Accessed August 05, 2018. [Online]. Available: https://www.openchain.org/

[279] Openchain, "Deploying Openchain in a production environment." Accessed August 05, 2018 [Online]. Available: https://docs.openchain.org/en/latest/general/nginx-deployment.html

[280] T. McConaghy, R. Marques, A. Müller, D. De Jonghe, T. McConaghy, G. McMullen, R. Henderson, S. Bellemare, A. Granzotto, "BigchainDB: a scalable blockchain database," BigchainDB GmbH, Berlin, Germany, Jun. 08, 2016. [Online]. Available: https://mycourses.aalto.fi/pluginfile.php/378362/mod_resource/content/1/bigchaindb-whitepaper.pdf

[281] BigchainDB, "BigchainDB Features & Use Cases." Accessed August 06, 2018. [Online]. Available: https://www.bigchaindb.com/features/

[282] BigchainDB, "BigchainDB 2.0 The Blockchain Database," BigchainDB GmbH, Berlin, Germany, May 2018. [Online]. Available: https://www.bigchaindb.com/whitepaper/bigchaindb-whitepaper.pdf

[283] BigchainDB Server. "How to Set Up a BigchainDB Network." Accessed August 06, 2018. [Online]. Available: http://docs.bigchaindb.com/projects/server/en/latest/simple-deployment-template/network-setup.html

[284] F. M. Benčić and I. P. Žarko, "Distributed ledger technology: Blockchain compared to directed acyclic graph," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, 2018, pp. 1569–1570. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8416434

[285] B. Shabandri and P. Maheshwari, "Enhancing IoT security and privacy using distributed ledgers with IOTA and the tangle," in *2019 6th International Conference on Signal Processing and Integrated Networks (SPIN)*, 2019, pp. 1069–1075. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8711591

[286] IOTA Foundation, "Introducing Pollen: the first decentralized testnet for IOTA 2.0," *Medium*, 30 Jun. 2020. [Online]. Available: https://blog.iota.org/introducing-pollen-the-first-decentralized-testnet-for-iota-2-0-349f63f509a1

[287] I. Bashir, *Mastering Blockchain*. Birmingham, England, UK: Packt Publishing Ltd, 2017. [Online]. Available: https://books.google.com/books?hl=en&lr=&id=urkrDwAAQBAJ&oi=fnd&pg=PP1&dq=Bashir,+Imran.+Mastering+blockchain.+Packt+Publishing+Ltd,+2017.&ots=Iw8l-m7zYQ&sig=d8qM4C8Ij62sf0ucINAtZrYi9Ds

[288] Alves, Davi, "A strategy for mitigating Denial of Service attacks on nodes with delegate account of Lisk blockchain," in *Proceedings of the 2020 The 2nd International Conference on Blockchain Technology*, pp. 7-12, 2020. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/3390566.3391684

[289] Lisk, "Build Your Blockchain Application in JavaScript." Accessed July 16, 2018. [Online]. Available: https://lisk.io/sdk

[290] Lisk, "A place to Connect." Accessed July 16, 2018. [Online]. Available: https://lisk.io/community

[291] X. Wu, Z. Zou, and D. Song, *Learn Ethereum: Build Your Own Decentralized Applications with Ethereum and Smart Contracts*. Birmingham, England, UK: Packt Publishing, 2019. [Online]. Available: https://play.google.com/store/books/details/Xun_Brian_Wu_Learn_Ethereum?id=RFqxDwAAQBAJ

[292] M. Mukhopadhyay, *Ethereum Smart Contract Development: Build Blockchain-Based Decentralized Applications Using Solidity*. Birmingham, England, UK: Packt Publishing Ltd, 2018. [Online]. Available: https://www.google.com/books/edition/Ethereum_Smart_Contract_Development/AOlODwAAQBAJ?hl=en&gbpv=1&dq=Mukhopadhyay,+Mayukh.+Ethereum+Smart+Contract+Development:+Build+blockchain-based+decentralized+applications+using+solidity.+Packt+Publishing+Ltd,+2018.&pg=PP1&printsec=frontcover

[293] R. Modi, *Solidity Programming Essentials: A Beginner's Guide to Build Smart Contracts for Ethereum and Blockchain*. Birmingham, England, UK: Packt Publishing Ltd, 2018. [Online]. Available: https://www.google.com/books/edition/Solidity_Programming_Essentials/H99YDwAAQBAJ?hl=en&gbpv=1&dq=Modi,+Ritesh.+Solidity+Programming+Essentials:+A+beginner%27s+guide+to+build+smart+contracts+for+Ethereum+and+blockchain.+Packt+Publishing+Ltd,+2018.&pg=PP1&printsec=frontcover

[294] N. Popper, "Business giants to announce creation of a computing system based on Ethereum," *New York Times*, Feb. 27, 2017. [Online]. Available: https://www.nytimes.com/2017/02/27/business/dealbook/ethereum-alliance-business-banking-security.html

[295] M. d. Castillo, "Blockchain 50: Billion dollar babies," *Forbes*, Apr. 16, 2019. [Online]. Available: https://www.forbes.com/sites/michaeldelcastillo/2019/04/16/blockchain-50-billion-dollar-babies/

[296] CB Insights, “80+ corporations working on blockchain and distributed ledgers,” *Research Briefs*, Jan. 03, 2019. [Online]. Available: https://www.cbinsights.com/research/organizations-corporates-test-blockchains-distributed-ledgers/

[297] L. Moore, “[Part One] Building an Ethereum node on a Raspberry Pi 3B,” *Medium*, 19 Feb. 2018. [Online]. Available: https://medium.com/@thelucasmoore/part-one-building-an-ethereum-node-on-a-raspberry-pi-3b-481104974cf7

[298] S. Eloudrhiri, “Create a private Ethereum blockchain with IoT devices,” *ChainSkills*, 24 Feb. 2017. [Online]. Available: https://chainskills.com/2017/02/24/create-a-private-ethereum-blockchain-with-iot-devices-16/

[299] Parrot Security, “Parrot Security - All-in-one framework for cyber security, software development and privacy defense.” Accessed August 16, 2018. [Online]. Available: https://www.parrotsec.org/

[300] Red Hat, “Linux Platforms – Red Hat Enterprise Linux.” Accessed August 17, 2018. [Online]. Available: https://www.redhat.com/en/technologies/linux-platforms/enterprise-linux

[301] Microsoft Evaluation Center, “Windows products & resources – Windows 10 Enterprise.” Accessed August 04, 2018. [Online]. Available: https://www.microsoft.com/en-us/evalcenter/evaluate-windows-10-enterprise

[302] Microsoft, “The best Windows ever – Shop Windows 10.” Accessed August 04, 2018. [Online]. Available: https://www.microsoft.com/en-us/store/b/windows

[303] Offensive Security, “Latest Kali Linux news and tutorials.” Accessed July 20, 2018. [Online]. Available: https://www.kali.org/

[304] CentOS, “The CentOS Project.” Accessed July 23, 2018. [Online]. Available: https://www.centos.org/

[305] Ubuntu, “Ubuntu 18.04.5 LTS (Bionic Beaver).” Accessed July 23, 2018. [Online]. Available: https://releases.ubuntu.com/18.04.5/

[306] Ubuntu, “Ubuntu 16.04.7 LTS (Xenial Xerus).” Accessed July 23, 2018. [Online]. Available: https://releases.ubuntu.com/16.04/

[307] N. Congleton, “Begin mining Ethereum on Ubuntu 16.04 Xenial Xerus Linux,” *LinuxConfig*, Aug. 21, 2018. [Online]. Available: https://linuxconfig.org/begin-mining-ethereum-on-ubuntu-16-04-xenial-xerus-linux

[308] J. Munde, “Steps to setup private Ethereum network on Ubuntu version 16.04 and above,” *Medium*, Nov. 04, 2018. [Online]. Available: https://medium.com/@jivanmunde15/steps-to-setup-private-ethereum-network-on-ubuntu-version-16-04-and-above-28eb520c3a65

[309] Ms. Smith, “Android now the world’s most popular operating system,” *CSO*, Apr. 03, 2017. [Online]. Available: https://www.csoonline.com/article/3187011/android-is-now-the-worlds-most-popular-operating-system.html

[310] J. R. Raphael, “Android versions: A living history from 1.0 to 11,” *ComputerWorld*, Jun. 26, 2020. [Online]. Available: https://www.computerworld.com/article/3235946/android-versions-a-living-history-from-1-0-to-today.html

[311] Microsoft, “An overview of Windows IOT Core,” *Windows for IoT*, Jan. 18, 2018. [Online]. Available: https://docs.microsoft.com/en-us/windows/iot-core/windows-iot-core

[312] Microsoft, “Windows for Internet of Things.” Accessed July 11, 2018. [Online]. Available: https://developer.microsoft.com/en-us/windows/iot/

[313] Armbian, “Linux ARM development boards.” Accessed July 24, 2018. [Online]. Available: https://www.armbian.com/

[314] Moebius, “Moebius – Minimal Linux distribution for your Raspberry Pi.” Accessed July 24, 2018. [Online]. Available: http://moebiuslinux.sourceforge.net/

[315] Raspberry Pi, “Raspberry Pi OS.” [Online]. Available: https://www.raspberrypi.org/documentation/raspbian/

[316] HowtoForge Linux Tutorials, “Raspberry Pi basics: installing Raspbian and getting it up and running.” Accessed July 28, 2018. [Online]. Available: https://www.howtoforge.com/tutorial/howto-install-raspbian-on-raspberry-pi/

[317] Raspbian, “Raspbian Forums.” Accessed July 28, 2018. [Online]. Available: https://www.raspbian.org/RaspbianForums

[318] M. Horne, “Raspberry Pi physical computing library GPIO Zero gets an upgrade to v1.5 – includes physical pin numbering!,” *Raspberry pipod and micro: bit base*, Feb. 13, 2019. [Online]. Available: https://www.recantha.co.uk/blog/?p=19358

[319] C. Cunningham, “PiMiner Raspberry Pi Bitcoin miner use a Raspberry Pi to control & monitor your USB bitcoin miners,” *Adafruit*, Jun. 20, 2013. [Online]. Available: https://learn.adafruit.com/piminer-raspberry-pi-bitcoin-miner

[320] G. Phillips, “6 easy Raspberry Pi crypto projects that anyone can follow,” *Blocks Decoded*, Oct. 11, 2019. [Online]. Available: https://blocksdecoded.com/raspberry-pi-crypto-projects/

[321] T. Klosowski, “Instantly free up almost 1GB on your Raspberry Pi by ditching LibreOffice and Wolfram,” *Lifehacker*, Apr. 29, 2016. [Online]. Available: https://lifehacker.com/instantly-free-up-almost-1gb-on-your-raspberry-pi-by-di-1773831271

[322] Offensive Security, “Kali on ARM.” Accessed July 27, 2018. [Online]. Available: https://www.kali.org/docs/arm/

[323] SSH, “SSH (Secure Shell).” Accessed August 03, 2018. [Online]. Available: https://www.ssh.com/ssh/#the-ssh-protocol

[324] K. Wüst and A. Gervais, “Do you need a blockchain?,” in *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, 2018, pp. 45–54. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8525392

[325] M. Mylrea and S. N. G. Gourisetti, “Blockchain: A path to grid modernization and cyber resiliency,” in *2017 North American Power Symposium (NAPS)*, 2017, pp. 1–5. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8107313

[326] C. Ye, G. Li, H. Cai, Y. Gu, and A. Fukuda, “Analysis of security in blockchain: Case study in 51%-attack detecting,” in *2018 5th International Conference on Dependable Systems and Their Applications (DSA)*, 2018, pp. 15–24. [Online]. Available: https://www.mdpi.com/2076-3417/9/9/1788

[327] Go Ethereum, “Go Ethereum – Official Go implementation of the Ethereum protocol.” Accessed July 18, 2018. [Online]. Available: https://geth.ethereum.org/

[328] E. Fraga, “Ethereum Wallet and Mist Beta 0.11.1 - windows hotfix,” *GitHub*, Jul. 23, 2018. [Online]. Available: https://github.com/ethereum/mist/releases

[329] N. Grech, M. Kong, A. Jurisevic, L. Brent, B. Scholz, and Y. Smaragdakis, “Madmax: Surviving out-of-Gas conditions in ethereum smart contracts,” *Proceedings of the ACM on Programming Languages*, vol. 2, no. OOPSLA, pp. 1–27, 2018. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/3276486

[330] S. Eloudrhiri, “Create and deploy a smart contract,” *ChainSkills*, Apr. 03, 2017. [Online]. Available: https://chainskills.com/2017/04/03/create-and-deploy-a-smart-contract-66/

[331] D. Mohanty, “Frameworks: Truffle and Embark,” in *Ethereum for Architects and Developers*, Springer, 2018, pp. 181–195. [Online]. Available: https://link.springer.com/chapter/10.1007/978-1-4842-4075-5_7

[332] S. Tikhomirov, E. Voskresenskaya, I. Ivanitskiy, R. Takhaviev, E. Marchenko, and Y. Alexandrov, “Smartcheck: Static analysis of Ethereum smart contracts,” in *Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain*, 2018, pp. 9–16. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/3194113.3194115

[333] W.-M. Lee, “Using the web3. JS APIs,” in *Beginning Ethereum Smart Contracts Programming*, Springer, 2019, pp. 169–198. [Online]. Available: https://link.springer.com/chapter/10.1007/978-1-4842-5086-0_8

[334] L. M. Bach, B. Mihaljevic, and M. Zagar, “Comparative analysis of blockchain consensus algorithms,” in *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2018, pp. 1545–1550. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8400278

[335] M. S. Ferdous, M. J. M. Chowdhury, M. A. Hoque, and A. Colman, “Blockchain consensuses algorithms: A survey,” *arXiv preprint arXiv:2001.07091*, 2020. [Online]. Available: https://arxiv.org/abs/2001.07091

[336] B. P. Rankhambe and H. K. Khanuja, “A comparative analysis of blockchain platforms–Bitcoin and Ethereum,” in *2019 5th International Conference On Computing, Communication, Control And Automation (ICCUBEA)*, 2019, pp. 1–7. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9129332

[337] K. Wu *et al.*, “Memory-bound Proof-of-Work acceleration for blockchain applications,” in *Proceedings of the 56th Annual Design Automation Conference 2019*, pp. 1–6, 2019. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/3316781.3317862

[338] F. Ma *et al.*, “EVM*: from offline detection to online reinforcement for Ethereum Virtual Machine,” in *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2019, pp. 554–558. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8668038

[339] J.-Y. Kim, J.-M. Lee, Y.-J. Koo, S.-H. Park, and S.-M. Moon, “Ethanos: Lightweight bootstrapping for Ethereum,” *arXiv preprint arXiv:1911.05953*, 2019. [Online]. Available: https://arxiv.org/abs/1911.05953

[340] Go-Ethereum, “Geth Documentation.” Accessed August 12, 2020. [Online]. Available: https://geth.ethereum.org/docs/

[341] V. Saini, “Getting deep into Geth: Why syncing Ethereum node is slow,” *Hackernoon*, Jul. 29, 2018. [Online]. Available: https://hackernoon.com/getting-deep-into-geth-why-syncing-ethereum-node-is-slow-1edb04f9dc5

[342] A. Moinet, B. Darties, and J.-L. Baril, “Blockchain based trust & authentication for decentralized sensor networks,” *arXiv preprint arXiv:1706.01730*, 2017. [Online]. Available: https://arxiv.org/abs/1706.01730

[343] A. Maw, S. Adepu, and A. Mathur, “ICS-BlockOpS: Blockchain for operational data security in industrial control system,” *Pervasive and Mobile Computing*, vol. 59, p. 101048, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S1574119218307041

[344] R. Xu, Y. Chen, E. Blasch, and G. Chen, “Blendcac: A smart contract enabled decentralized capability-based access control mechanism for the IoT,” *Computers*, vol. 7, no. 3, p. 39, 2018. [Online]. Available: https://www.mdpi.com/2073-431X/7/3/39

[345] R. Gupta, S. Tanwar, F. Al-Turjman, P. Italiya, A. Nauman, and S. W. Kim, “Smart contract privacy protection using AI in cyber-physical systems: tools, techniques and challenges,” *IEEE Access*, vol. 8, pp. 24746–24772, 2020. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8976143

[346] T. Wu and X. Liang, “Exploration and practice of inter-bank application based on blockchain,” in *2017 12th International Conference on Computer Science and Education (ICCSE)*, 2017, pp. 219–224. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8085492

[347] R. Bhattacharya, M. White, and N. Beloff, “A blockchain based peer-to-peer framework for exchanging leftover foreign currency,” in *2017 Computing Conference*, 2017, pp. 1431–1435. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8252284

[348] X. Zhang, M. Aranguiz, D. Xu, X. Zhang, and X. Xu, “Utilizing blockchain for better enforcement of green finance law and regulations,” in *Transforming Climate Finance and Green Investment with Blockchains*, Elsevier, 2018, pp. 289–301. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128144473000215

[349] D. A. Wijaya, J. K. Liu, D. A. Suwarsono, and P. Zhang, “A new blockchain-based value-added tax system,” in *International conference on provable security*, 2017, pp. 471–486. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-68637-0_28

[350] L. van Rijswijk, H. Hermsen, and R. Arendsen, “Exploring the future of taxation: A blockchain scenario study,” *Journal of Internet Law*, vol. 22, no. 9, pp. 1–31, 2019. [Online]. Available: https://openaccess.leidenuniv.nl/handle/1887/68772

[351] S. Bragagnolo, H. Rocha, M. Denker, and S. Ducasse, “SmartInspect: solidity smart contract inspector,” in *2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*, 2018, pp. 9–18. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8327566/

[352] G. McCubbin, “Intro to Web3.js – Ethereum Blockchain Developer Crash Course,” *Dapp University*, 05 Aug. 2020. [Online]. Available: https://www.dappuniversity.com/articles/web3-js-intro

[353] Y. Fu, M. Ren, F. Ma, Y. Jiang, H. Shi, and J. Sun, “Evmfuzz: Differential fuzz testing of ethereum virtual machine,” *arXiv preprint arXiv:1903.08483*, 2019. [Online]. Available: https://arxiv.org/abs/1903.08483

[354] F. Þ. Hjálmarsson, G. K. Hreiðarsson, M. Hamdaqa, and G. Hjálmtýsson, “Blockchain-based e-voting system,” in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, 2018, pp. 983–986. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8457919

[355] P. Mell, J. Dray, and J. Shook, “Smart contract federated identity management without third party authentication services,” *arXiv preprint arXiv:1906.11057*, 2019. [Online]. Available: https://arxiv.org/abs/1906.11057

[356] L. Bader, J. C. Bürger, R. Matzutt, and K. Wehrle, “Smart contract-based car insurance policies,” in *2018 IEEE Globecom Workshops (GC Wkshps)*, 2018, pp. 1–7. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8644136

[357] T. Mikula and R. H. Jacobsen, “Identity and access management with blockchain in electronic healthcare records,” in *2018 21st Euromicro conference on digital system design (DSD)*, 2018, pp. 699–706. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8491888

[358] S. E. Chang, Y.-C. Chen, and M.-F. Lu, “Supply chain re-engineering using blockchain technology: A case of smart contract based tracking process,” *Technological Forecasting and Social Change*, vol. 144, pp. 1–11, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0040162518305547

[359] J. Lee, M. Azamfar, and J. Singh, “A blockchain enabled cyber-physical system architecture for Industry 4.0 manufacturing systems,” *Manufacturing Letters*, vol. 20, pp. 34–39, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S2213846319300264

[360] S. Schorradt, E. Bajramovic, and F. Freiling, “On the feasibility of secure logging for industrial control systems using blockchain,” in *Proceedings of the Third Central European Cybersecurity Conference*, 2019, pp. 1–6. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/3360664.3360668

[361] C. Pop, T. Cioara, M. Antal, I. Anghel, I. Salomie, and M. Bertoncini, "Blockchain based decentralized management of demand response programs in smart energy grids," *Sensors*, vol. 18, no. 1, p. 162, 2018. [Online]. Available: https://pdfs.semanticscholar.org/3e92/1b2a61649b70209bb93aeee73645beeec6c4.pdf

[362] M. Li, L. Wang, and Y. Zhang, "A framework for rocket and satellite launch information management systems based on blockchain technology," *Enterprise Information Systems*, pp. 1–15, 2019. [Online]. Available: https://www.tandfonline.com/doi/abs/10.1080/17517575.2019.1669828

[363] D. K. Tosh, S. Shetty, P. Foytik, L. Njilla, and C. A. Kamhoua, "Blockchain-empowered secure Internet-of-Battlefield Things (IoBT) architecture," in *MILCOM 2018–2018 IEEE Military Communications Conference (MILCOM)*, 2018, pp. 593–598. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8599758

[364] Gadgetoid, "Raspberry Pi pinout." Accessed August 20, 2020. [Online]. Available: https://pinout.xyz/

[365] S. Eloudrhiri, "Raspberry Pi and Ethereum – The epilogue," *Chainskills*, Apr. 10, 2017. [Online]. Available: https://chainskills.com/2017/04/10/raspberry-pi-and-ethereum-the-epilogue/

[366] S. Eloudrhiri, "Set up the private chain – miners," *Chainskills*, Mar. 10, 2017. [Online]. Available: https://chainskills.com/2017/03/10/part-3-setup-the-private-chain-miners/

[367] S. Eloudrhiri, "Pair the miners," *Chainskills*, Mar. 17, 2017. [Online]. Available: https://chainskills.com/2017/03/17/pair-the-miners-46/

[368] S. Eloudrhiri, "Synchronize the Raspberry Pi with the private blockchain," *Chainskills*, Mar. 27, 2017. [Online]. Available: https://chainskills.com/2017/03/27/synchronize-the-raspberry-pi-with-the-private-blockchain-56/

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California