# REPORT DOCUMENTATION PAGE

*Form Approved*
*OMB No. 0704-0188*



| **1. REPORT DATE** *(DD-MM-YYYY)* | **2. REPORT TYPE** | **3. DATES COVERED** *(From - To)* |
|---|---|---|
| 10/01/2018 | Technical Report | |

**4. TITLE AND SUBTITLE**
Blockchain Protocol Security Analysis

**5a. CONTRACT NUMBER**
W56KGU-17-D-0004

**5b. GRANT NUMBER**

**5c. PROGRAM ELEMENT NUMBER**

**6. AUTHOR(S)**
Bryson, David E; Burgin, Kelley W; Serrao, Gloria J;

**5d. PROJECT NUMBER**

**5e. TASK NUMBER**

**5f. WORK UNIT NUMBER**

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**
The MITRE Corporation
300 Sentinel Drive
Suite 500 & 600
Annapolis Junction, MD 20701

**8. PERFORMING ORGANIZATION REPORT NUMBER**
PRS-19-1044

**9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)**
National Security Agency (NSA) Cybersecurity Solutions Organization (CSO)

**10. SPONSOR/MONITOR'S ACRONYM(S)**
NSA/CSS

**11. SPONSOR/MONITOR'S REPORT NUMBER(S)**
PP-19-0073

**12. DISTRIBUTION/AVAILABILITY STATEMENT**
DISTRIBUTION STATEMENT A. Approved for public release: distribution unlimited.

**13. SUPPLEMENTARY NOTES**

**14. ABSTRACT**
This paper is a timely look at the security properties of two major components: the consensus algorithm and the peer-to-peer (P2P) protocol. The goal is to familiarize cybersecurity experts with the technical components of blockchain and recommend areas for future security evaluations. MITRE analyzed and tested the Byzantine Fault Tolerant (BFT) characteristics of the Istanbul consensus protocol and the security properties of two P2P protocols. It is the first time that Blockchain P2P algorithms have been analyzed using the Cryptographic Protocol Shapes Analyzer (CPSA).

**15. SUBJECT TERMS**
Information Security Architecture; key management; IoT; distributed ledger technologies; finance; government; cybersecurity; Byzantine Fault Tolerant; information sharing; blockchain; BFT; Internet of Things; protocol;

| **16. SECURITY CLASSIFICATION OF:** | | | **17. LIMITATION OF ABSTRACT** | **18. NUMBER OF PAGES** | **19a. NAME OF RESPONSIBLE PERSON** |
|---|---|---|---|---|---|
| **a. REPORT** | **b. ABSTRACT** | **c. THIS PAGE** | | 43 | Susan Carpenito |
| | | | | | **19b. TELEPHONE NUMBER** *(Include area code)* 781-271-7646 |

MTR180406
MITRE TECHNICAL REPORT

MITRE

# Blockchain Protocol Security Analysis

Sponsor: NSA/CSS
Project No.: 0718N874-BC



**Ft. Meade, MD**

## Security Considerations for Distributed Ledger Technologies

**Authors: Dave Bryson**
**Kelley Burgin**
**Gloria Serrao**

**October 2018**

# Abstract

Blockchain is a distributed ledger technology of interest to financial and government entities for various use cases. This paper is a timely look at the security properties of two major components: the consensus algorithm and the peer-to-peer (P2P) protocol. The goal is to familiarize cybersecurity experts with the technical components of blockchain and recommend areas for future security evaluations. This work prepares cybersecurity experts to respond to proposals from U.S. Government customers to implement blockchain in areas such as: supply chain, key management, information sharing and Internet of Things (IoT) security.

MITRE analyzed and tested the Byzantine Fault Tolerant (BFT) characteristics of the Istanbul consensus protocol and the security properties of two P2P protocols. It is the first time that Blockchain P2P algorithms have been analyzed using the Cryptographic Protocol Shapes Analyzer (CPSA).

# Executive Summary

Time to maturity of Blockchain Technology is estimated to be 5 – 10 years. [1] Blockchain development is open-source, consists of numerous consensus and peer-to-peer constructs, and is sponsored by multiple forums. There is no known method to measure the security strength of consensus (which is game theory logic, and not cryptographic in nature). The underlying P2P protocols are not standardized, but contain familiar cryptographic functions. Blockchain technology is changing at a rapid pace, making traditional security analysis methods difficult.

This report provides foundational knowledge about blockchain security principles to inform cybersecurity experts and prepare them to perform similar analysis, participate in standards bodies, and perform security engineering to guide blockchain implementations for the U.S. government.

This analysis is divided between consensus and P2P components of blockchain. The appendices contain additional details about the RLPx and Tendermint protocol analysis, some information about RLPx eclipse attacks and current thinking about how Quantum computing will affect this technology.

This paper should be used for rationale and background to engage with other U.S. Government agencies and Industry. Through such engagement, cohesive security requirements and best implementations can be pursued. Blockchain will soon move from research and piloting to full implementation, so security experts must engage in open source development efforts and be open to new ways to achieve interoperability,

The test results for the Istanbul consensus algorithm demonstrate Byzantine Fault Tolerance and expected behavior with a faulty node present during consensus. The RLPx P2P protocol was found to be secure and the Tendermint P2P protocol has a known Man-In-The-Middle (MITM) attack which is detailed in this analysis. Cybersecurity organizations should closely monitor confidentiality and privacy solutions to identify the most promising consensus and P2P protocol developments.

## Acknowledgments

The authors would like to thank the MITRE CPSA team, in particular Paul Rowe, Moses Liskov, and John Ramsdell, for their expert assistance in the modeling and analysis of the P2P protocols.

We would also like to thank the following whose careful review led to an improved product: Bonnie Martin, Paul Rowe and Linda Shields.

# Table of Contents

# List of Figures

# 1 Goals

The National Security Agency (NSA) Cybersecurity Solutions Organization (CSO) tasked The MITRE Corporation to analyze the foundational security protocols of blockchain technology. This work will advise existing U.S. Government pilots and is in sync with an overall increased U.S. Government interest in blockchain. Examples are DHS funded grants of $9.7M awarded in 2017 and the 2018 GSA establishment of a U.S. Emerging Citizen Technologies program [2].

This paper is intended to inform systems security engineers, innovation leaders and cryptographic experts in the NSA CSO and prepare them to:

- Understand the technology, its use of cryptography, and security advantages and/or vulnerabilities
- Recommend security measures for NSA use of blockchain
- Provide security guidance to NSA's customers who are implementing blockchain within National Security Systems

The goal is to position NSA as a recognized security expert for blockchain technology. The analysis approach described here can be adopted for future blockchain proposals and implementations.

We reviewed permissioned blockchain implementations compatible with the sponsor's direction, their intended uses and piloting efforts. We analyze the Istanbul consensus protocol used by Ethereum, analyze cryptographic protocols within Ethereum's RLPx peer-to-peer (P2P) protocol and Tendermint's P2P protocol, and make recommendations for future security analysis and best implementation practices for blockchain technology.

# 2 Introduction

Distributed ledger technology, or Blockchain, is the underlying technology of crypto-currencies such as Bitcoin [3], and is gaining attention from investors, banks and commercial entities such as Walmart [4]. It can be defined as "a decentralized, distributed, immutable and public digital ledger that persists transactions based on a quorum strategy or consensus algorithm lined and secured with cryptography"[1] As a distributed and decentralized ledger, it allows untrusted parties to securely exchange transactions. Blockchain provides a way for computers to validate, settle and agree on a record of transactions. It maintains transaction records across many computers simultaneously. A group of transactions (a block) is securely appended to the existing ledger (or chain) by way of linking cryptographic hashes to the previous block. This results in a ledger that cannot be changed. Blockchain can take the place of a centralized storage solution such as a database which creates a single point of failure. A blockchain is distributed over multiple nodes using an underlying P2P network protocol for node discovery and communication.

The basic components of blockchain are:

[1] Chainhaus, Blockchain Masterclass, 2018

- Cryptography: hash functions that link blocks together providing integrity of the chain, and digital signatures providing integrity for the transactions.
- Consensus Algorithm: The process by which parties to a blockchain decide on the ordering and presence of transactions on the ledger.
- Distributed Ledger: A distributed, replicated, representation of all transactions.
- P2P Protocol: The protocol that manages the peer nodes of the network that support blockchain. It performs communication between nodes, flow control, node discovery, and framing.
- Smart Contracts: business rules or logic that can extend the functionality of a blockchain.

For additional introductory information describing blockchain technology see: MITRE's technical report "Blockchain for Government" [5] and an Information report, "Blockchain Technology Overview" by the National Institute of Science and Technology [6]

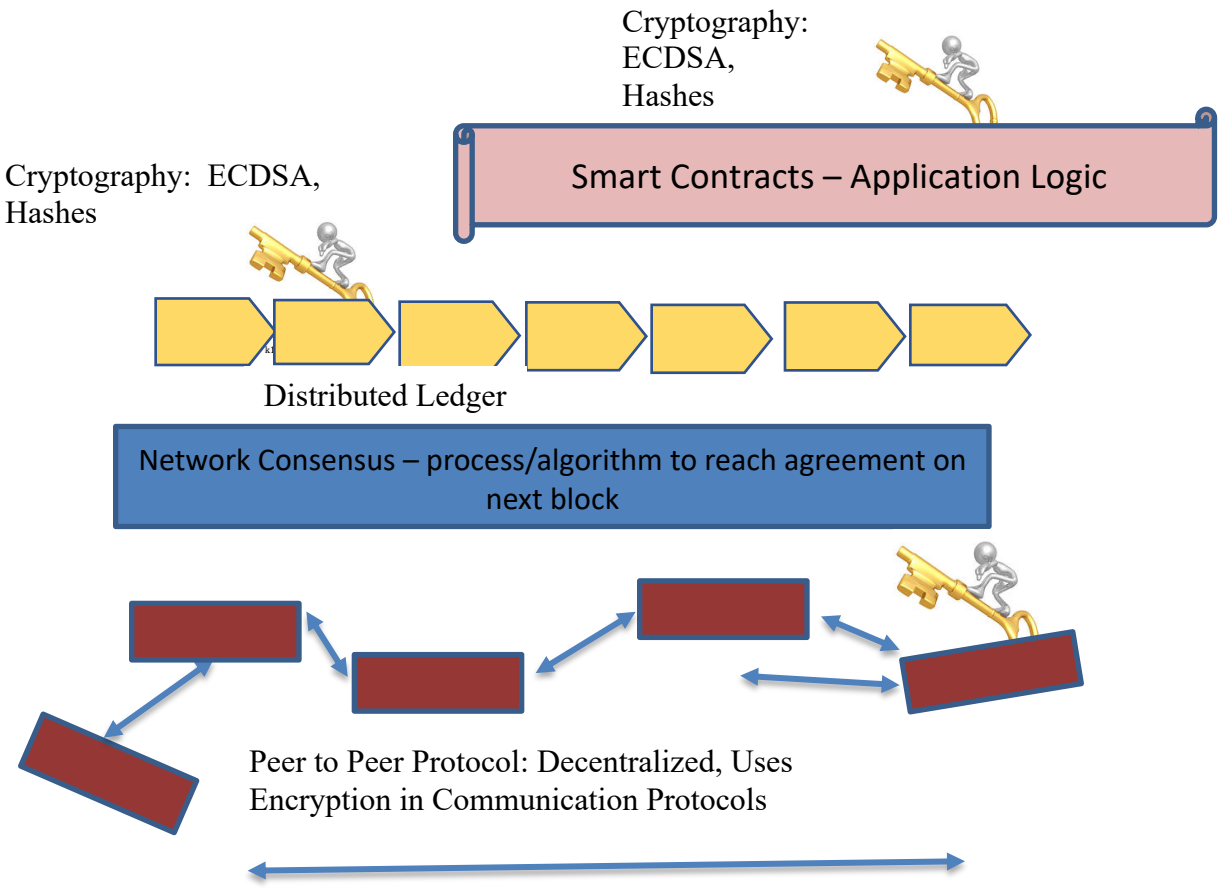


**Figure 1 Blockchain Components**

This work focuses on one consensus algorithm (Ethereum's Istanbul) and two P2P protocols (Ethereum's RLPx and Tendermint's P2P protocol.)

## 2.1 Problem Statement

Labeled as a "disruptive" technology by industry leaders, Blockchain goes beyond technology to a new way of distributing information without intermediaries which has great impact on business processes and decision making. Blockchain technology has just passed the peak of the hype stage of the Gartner Hype Cycle for Emerging Technologies in 2017. (Figure 2)

In the 2018 Gartner trend predictions, Blockchain ranks number 8 of 10 top trends. Gartner projects time to maturity to be 5 – 10 years. As Gartner cautions, implementers must "be sure that your team has the cryptographic skills to understand what is and isn't possible. Identify integration points with existing infrastructures and monitor the platform evolution and maturation." [7]

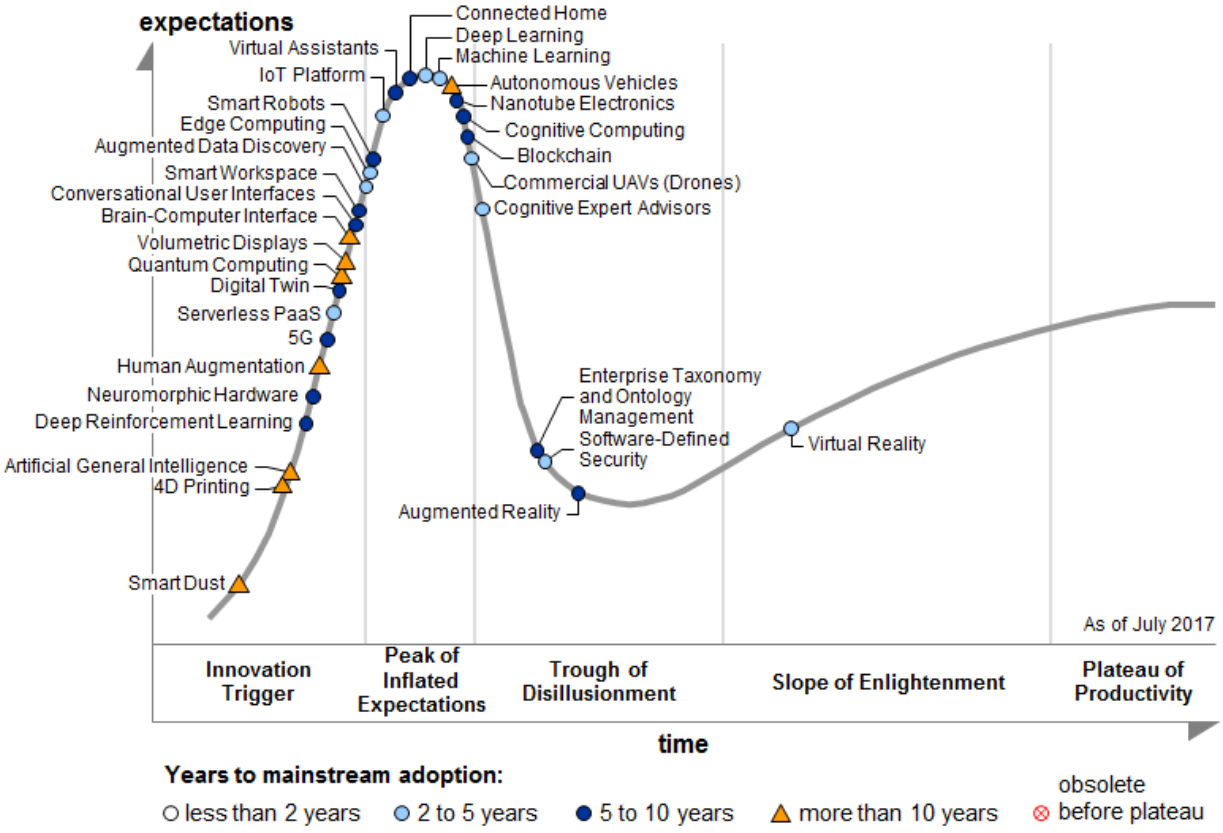


**Figure 2 Gartner Hype Cycle for 2017**

There are numerous blockchain consensus mechanisms and each is implemented differently; most are open-source projects. There is no known method to measure the security strength of consensus (which is game theory logic, not cryptographic in nature). The underlying peer-to-peer protocols are also not standardized. Blockchain technology is changing at a rapid pace, making traditional security analysis methods difficult.

## 2.2 Scope

To perform the security analysis, we first chose an Ethereum implementation of blockchain that supports permissioned blockchains and is compatible with the sponsor's piloting efforts. A permissioned blockchain shares transactional information with a designated set of interested parties. Blockchain qualities are most advantageous when the set of interested parties are trusted at different levels. Our analysis focuses on a specific Ethereum consensus algorithm, Istanbul, and the RLPx P2P protocol. In addition to these Ethereum components, the Tendermint P2P protocol was analyzed to gain the value of comparative analysis.

Ethereum, touted as a "world computer," can support multiple applications built on top of the blockchain via an embedded, deterministic virtual machine called the Ethereum Virtual Machine (EVM.) The EVM supports the development of "Smart Contracts," which are state transitioning programs that can extend the applicability of Ethereum into many areas such as health care and educational records by applying business logic for processing the stored records.

Ethereum Blockchain applications use a distributed, peer-to-peer networking protocol (RLPx) and its components such as: User Datagram Protocol (UDP) for node discovery, an Elliptic Curve Diffie Hellman (ECDH)-type key exchange for encrypted handshake, and Elliptic Curve Digital Signature Algorithm (ECDSA) for digital signatures. These components are separately well-understood and widely used. However, RLPx itself has not been standardized but is an open-source Ethereum library. The integration and slight changes in these components could have security impacts. For example, RLPx uses a Kademlia-"like" node discovery method which simplifies node ID creation.

The Tendermint open-source project originated in 2014. The goal of Tendermint was to address the speed, scalability and resource issues of proof-of-work (PoW) consensus algorithms. Tendermint is a Byzantine fault-tolerant (BFT) consensus engine with an Application Blockchain Interface (ABCI) for application developers which is compatible with any programming language. [8] The Tendermint consensus algorithm is out-of-scope but the underlying P2P protocol was analyzed.

The Tendermint P2P protocol uses an authenticated encryption scheme to establish connections between nodes on the network and to secure communications from adversaries attempting to gain access to the messages being sent between nodes. The protocol is based on the Station-to-Station protocol, which is known to be secure, but changes are made which bring into question the security of the protocol. The developers of the protocol acknowledge a potential man-in-the-middle attack and are evolving the security of the protocol to address it.

Our analysis of RLPx and Tendermint's P2P protocol focus solely on the cryptographic security of the protocols. Some information and references are provided about the RLPx communications channels and known attack vectors to those channels in Appendix C.

Smart Contracts are also out of scope. As software, they are not entirely immune to programmer error resulting in vulnerabilities. However, because they can't be amended based on the immutability of the blockchain, these vulnerabilities can persist. Ethereum contracts are written in a language such as Solidity, and compiled to bytecode. There have been bugs in the bytecode due to optimization errors during compilation. The bugs caused the bytecode to differ from the

intent of the higher-level language. The low-level, stack-based bytecode language is called "EVM code."

# 3 Security Principles and Blockchain

Security requirements are used by NSA CSO System Security Engineers (SSEs) to ensure the confidentiality, integrity and availability (CIA) of the information being processed, stored or transmitted by information systems. Additionally, non-repudiation is a security requirement that is sometimes applicable. In this section we address how these security requirements are met for information stored on a blockchain.

## 3.1 Integrity

### 3.1.1 Data and Transaction Integrity

Transaction and ledger integrity are key security functions of a blockchain. A *transaction* is a recording of a transfer of assets between parties. Transaction integrity is achieved by the calculation of a cryptographic hash for each transaction and a digital signature validating the transaction. A block is made up of multiple transactions and the contents of a block is also hashed for block integrity. Ledger integrity is based on the linking of completed blocks to previous blocks by inserting a cryptographic hash of the previous block into the newly created block.

### 3.1.2 Message Integrity

Integrity can also be applied to messages sent by nodes in the underlying P2P protocol. At the P2P network level, message integrity is achieved by RLPx but not Tendermint's P2P protocol. In RLPx, message integrity in the node discovery phase is achieved by applying digital signatures and hashes to the data being sent. After the encryption and Message Authentication Code (MAC) keys have been derived during the encrypted handshake phase, all frames containing packets of data are authenticated using a MAC of the frame. The Tendermint P2P protocol provides node authentication and key confirmation, but not message integrity.

## 3.2 Non-repudiation

Non-repudiation is the assurance that the sender of the information is provided with proof of delivery and the recipient is provided with proof of the sender's identity, so neither can later deny having processed the information. [9]

National Institute of Standards and Technology (NIST) Special Publication (SP) 800-53 [10] describes non-repudiation as protection against an individual falsely denying having performed a particular action. Non-repudiation provides the capability to determine whether a given individual took a particular action such as creating information, sending a message, approving information and receiving a message.

Within a blockchain, digital signatures provide proof of an entity performing a transaction. There is no other proof of the sender's identity and because identities are pseudo-anonymous, it

is not easily determined by viewers of the blockchain. The trustworthiness of digital signatures is dependent on secure storage of private keys. A "recipient" in the blockchain sense is not well defined because many entities view the blockchain and thus receive the information. Any party viewing the blockchain could be considered a recipient. In a strict sense, proof of delivery and non-repudiation to a particular recipient is not provided by blockchain. However, the visibility and permanence of a blockchain contributes to a non-repudiation solution.

## 3.3 Availability

The availability of the blockchain depends on the underlying P2P network and is achieved by distributing the blockchain across those peers. Ensuring resiliency of the information means to ensure resiliency of the distributed network. Public blockchains are resilient because of the large number of nodes. Resiliency of permissioned blockchains requires having the correct number of nodes with a calculated risk of a specific set of faulty nodes.

Permissioned blockchains have a much smaller number of distributed nodes than a public blockchain. They cannot withstand the same network conditions; they are not as resilient to a denial of service attack. Simply put, the likelihood of destroying all of the nodes simultaneously is easier if the network is smaller.

## 3.4 Authentication

Authentication is the act of "Verifying the identity of a user, process, or device, often as a prerequisite to allowing access to resources in an information system." [11] [12] Authentication is required for creating transactions and adding blocks of transactions to the blockchain. It is achieved via digital signatures by each transaction originator and by the creator of each block. The verification of an identity prior to blockchain access is not applicable to public blockchains. However, in permissioned blockchains, identities are verified prior to establishment of the blockchain. Keys created for authorized users must be protected. It is important that the process of authenticating parties does not result in a centralized authority so as to nullify the benefits of a distributed ledger.

## 3.5 Confidentiality

Confidentiality is "Preserving authorized restrictions on information access and disclosure, including means for protecting personal privacy and proprietary information." [9] For blockchain, transaction information and parties to the transaction are candidates for confidentiality.

### 3.5.1 Public Blockchains

Technologies to hide identities and transaction contents are actively being researched such as stealth addresses, zero-knowledge proof and homomorphic encryption. [13] Zero-knowledge Succinct Non-interactive Arguments of Knowledge (zk-SNARKS) is an implementation that shields coins by allowing them to be private. In most public blockchains, all participants see all transactions taking place. This fact combined with the absence of a central point of control is the basis of decentralization and distributed trust. Confidentiality is not usually a security property of a public blockchain. Entities are most often designated by public keys, referred to as

"addresses." While there is not a formal mapping of public key to identities, association with other information and identifying parties to the transactions is not difficult.

### 3.5.2 Permissioned Blockchains

Permissioned blockchains are designed for a more controlled environment than public blockchains. Participating parties are selected and given authority to provide validation of blocks or transactions or to participate in the consensus mechanism. They usually do not provide confidentiality of transaction information or party identities. Identities and transactions are usually visible to all parties of a permissioned blockchain.

One method of achieving confidentiality is to encrypt sensitive data on the blockchain. As with all encryption, however, this data would be exposed if there were a cryptographic attack. Because blockchain is a distributed, permanent record, an attack on the cryptography would expose all traffic to all parties at one time.

Solutions are emerging for protecting information "off-chain." Quorum, for example, has a parameter that can be included within a transaction to indicate it is private. Quorum uses a special node called a "constellation" node which contains a transaction manager. The transaction manager has the encrypted payload which is indexed by a hash on the blockchain. Only a SHA3-512 digest of the encrypted payloads are exchanged between the peers. Transaction contents are not visible except to authorized nodes. Storing information "off-chain" means that the blockchain itself is no longer the single, shared "source of truth." In some cases the parties each must maintain their own additional record of stored transaction information or move to a trusted third party, negating many of the advantages of using a blockchain.

# 4 Istanbul Analysis

## 4.1 Considerations for Selection

A consensus algorithm and its selection is of primary importance for blockchain security. The fundamental goal of adding new blocks securely to the blockchain without a centralized authority requires a consensus to be reached in order to add items to the blockchain. Numerous blockchain consensus algorithms exist and new ones are evolving. All consensus algorithms require voting of some sort. Examples of different ways to reach this consensus are: round-robin voting, majority voting and selection of a random group of signers in each round.

Below we provide rationale for our selection of Istanbul and a high-level overview of other types of consensus algorithms.

The consensus algorithm analyzed for this task is Istanbul, which is in the proposal state and is part of the Ethereum platform. It is documented on github. [13] Istanbul runs on the Ethereum permissioned blockchain. It was created by a joint venture of financial institutions in Taiwan, AMIS. It is used on J.P. Morgan's Quorum platform. Although still in the proposal stage, Istanbul met two important criteria for our security analysis: It is Byzantine Fault Tolerant (BFT) and it is a permissioned blockchain.

### 4.1.1 Byzantine Fault Tolerance (BFT) Requirement

A Byzantine fault is when one or more nodes in a distributed network exhibits arbitrary behavior. This puts trust in a distributed ledger at risk because true consensus or the ability to make a unified decision is lacking. This arbitrary behavior can be from malicious behavior or software errors. The behavior looks the same whether the nodes are delayed or faulty.

This fault tolerant behavior gets its name from a 1982 paper which is titled and describes "The Byzantine Generals' Problem." [14] The paper illustrates the issue when parts of a computer system fail or act incorrectly. As an illustration, it describes multiple generals planning an attack and forced to rely on oral communication to agree on the attack details. One general is a traitor and provides wrong information. The authors determined that true agreement was met only if more than 2/3 of the generals are loyal. A single traitor can confound 2 loyal generals.

This logic problem applies to distributed computing and to achieving consensus with a blockchain. A "traitor" within a distributed ledger is an entity sending out inaccurate information or acting in a way to cause confusion. This calls into question the reliability of the blockchain. The requirement of more than 2/3 loyal generals has been translated for distributed systems to require more than $3f+1$ nodes for that system to operate properly. The variable $f$ represents the number of faulty nodes that can be tolerated. Simply put, the number of malicious nodes cannot equal or exceed 1/3 of the overall nodes on the network. BFT is not a new principle, but it has been applied to blockchain technology within the last 5 years.

It is important that the consensus algorithm for a blockchain implementation is BFT and that consensus is not reached with less than $3f+1$ nodes. There are multiple methods to provide BFT but all are based on keeping all nodes honest and with a stake in the reliability of the blockchain. To ensure distributed leger entities are honest, they must prove their honesty by some stake they have in the process of consensus. BFT is achieved by several consensus methods:

- Proof of Work (PoW): Most familiar because it is used by Bitcoin, this consensus algorithm ensures randomness of block creation by requiring entities (miners) to solve a hard math problem involving multiple rounds of a hash function to achieve a designated nonce. Once the correct nonce is created, the miner receives a transaction fee of new bitcoins. Based on the "work performed", the winning entity (miner) proposes a block and a vote is held agreeing that the winner can add the block. The block is always added to the longest chain. Proof of work ensures that one organization cannot gain control of the blockchain (50.1% of the hashing power) because of the resource costs (mining) required by participants. Proof of Work also plays a major role in preventing Sybil attacks on public blockchains. A Sybil attack is a way to subvert a system by forging identities.

  The incentive of participants in a PoW blockchain is economic. Each entity is equally invested in ensuring the blockchain's success to create and spend coins.

- Proof of Stake (PoS): This method is based on the "stake" that each node has in the correct operation of the blockchain. Entities have a stake based on pre-allocated currency and not their investment in computational power. In proof of stake you are more likely to be chosen to add the next block based on the amount of coins you have invested in the system. After a proposer creates a block, it must be committed to the blockchain. PoS

systems vary in the particular details of the block validation. Some require each node in the system to sign until majority is reached, in others a random group is chosen for validation. PoS implementations may have the "nothing at stake" issue which is when a malicious entity could forge signing blocks on two separate forks because they are not investing resources (power, etc.) They can then collect two transaction fees.

- Proof of Activity: A hybrid approach combining PoW and PoS. PoW is done to mine the block but then a random group of validators is chosen to sign it.
- Proof of Elapsed Time: Using an Intel trusted execution environment, this algorithm ensures that blocks get produced randomly but their production does not require large amounts of computation because of a guaranteed wait time used in the trusted execution environment. [15]

### 4.1.2 Permissioned Blockchain Requirement

As MITRE has surveyed U.S. Government sponsors about their use cases for blockchain technology, most are interested in implementations that are shared with a designated set of interested parties. These parties might be trusted at different levels (for example coalition partners). Pilots across the Government are in early stages and most often involve supply chain, coalition sharing, or in one example, the Air Force is investigating how a permissioned blockchain can improve data flow and resiliency in a sensor network operating on tactical timelines.

## 4.2 Istanbul Overview

Keeping in mind that consensus is an agreement on the shared state of the system, Istanbul pre-defines a set of rules used to determine what transactions are valid and the ordering of the blocks in the chain. Istanbul has a three phase consensus (`PRE-PREPARE, PREPARE, and COMMIT`). The participants in the Istanbul consensus are: *Proposers* and *Validators.*

The steps forming a consensus in Istanbul are:

1. A proposer is selected by default in a round-robin fashion
2. The proposer proposes a new block and broadcasts it to the validators with a `PRE-PREPARE` message
3. Validators receive the `PRE-PREPARE` message and enter the state of *pre-prepared*
4. Validators broadcast `PREPARE` message.
5. If 2F+1 (hereafter referred to as 2/3) of `PREPARE` messages are received, a validator enters the state of *prepared* and broadcasts `COMMIT` message. (The validator goal is to inform peers that it accepts a proposed block and is going to insert the block into the chain.)
6. Validators wait for 2/3 of `COMMIT` messages.
7. When received, they enter the *committed* state and insert block into the chain.

Blocks in Istanbul are final, which means that there are no forks and any valid block must be somewhere in the main chain. To prevent a faulty node from generating a totally different chain from the main chain, each validator appends 2/3 received `COMMIT` signatures to a field in the block header before inserting it into the chain. Thus blocks are self-verifiable and a light client can be supported as well. However, the additional data in the header could cause an issue on block hash

calculation. Since the same block from different validators can have different set of `COMMIT` signatures, the same block can have different block hashes as well. To solve this, the block hash is calculated by excluding the `COMMIT` signatures part. Therefore, we can still keep the block/block hash consistency as well as put the consensus proof in the block header.

### 4.2.1 Consensus states

Istanbul BFT is a state machine replication algorithm. Each validator maintains a state machine replica in order reach block consensus.

States:

- `NEW ROUND`: Proposer to send new block proposal. Validators wait for `PRE-PREPARE` message.
- `PRE-PREPARED`: A validator has received `PRE-PREPARE` message and broadcasts `PREPARED` message. Then it waits for 2/3 of `PREPARE` or `COMMIT` messages.
- `PREPARED`: A validator has received 2/3 of `PREPARE` messages and broadcasts `COMMIT` messages. Then it waits for 2/3 of `COMMIT` messages.
- `COMMITTED`: A validator has received 2/3 of `COMMIT` messages and is able to insert the proposed block into the blockchain.
- `FINAL COMMITTED`: A new block is successfully inserted into the blockchain and the validator is ready for the next round.
- `ROUND CHANGE`: A validator is waiting for 2/3 of `ROUND CHANGE` messages on the same proposed round number.

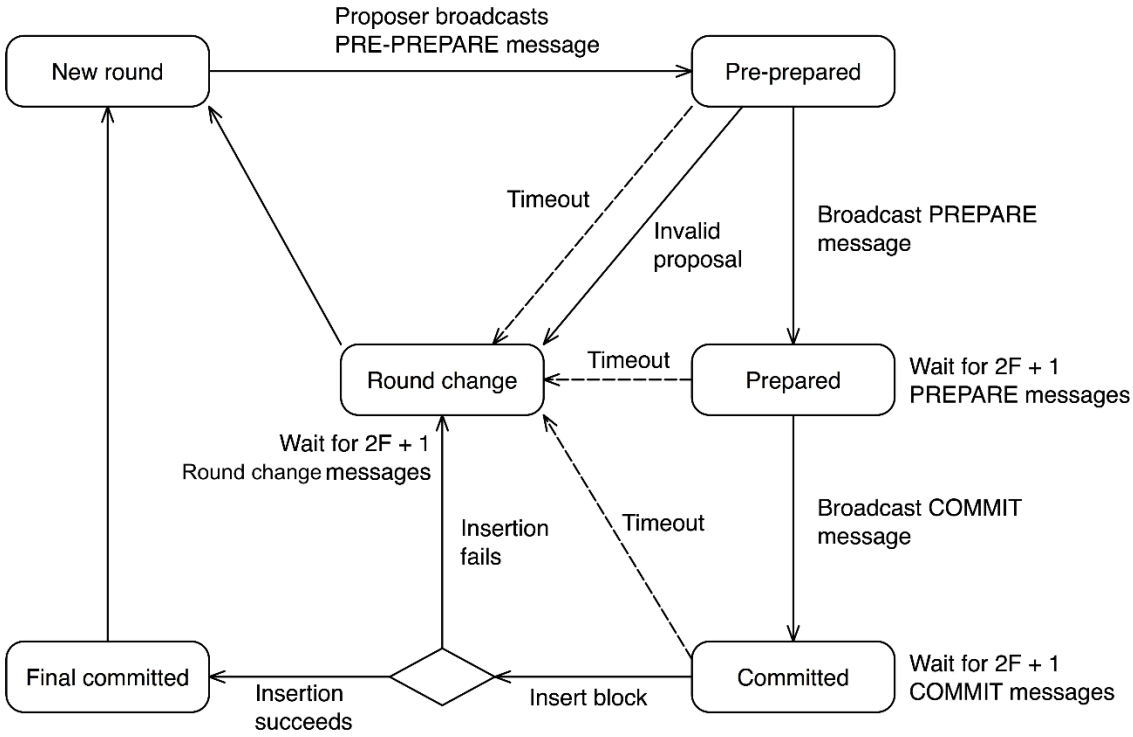


**Figure 1 – Stages of Istanbul Consensus**

## 4.3 Istanbul Test Set Up

For MITRE's analysis, a development environment was created using docker containers. Docker containers and virtualization enable a consistent environment independent of the underlying operating system. Containerization allows for the quick creation and take-down of an application environment.

To have full control of the test environment, the MITRE team created a Python-based application programming interface (API) that dynamically creates a custom network and a configurable number of Ethereum nodes. This allows us to programmatically interact with the nodes and docker environment to manipulate the nodes as well as track their current state. The environment was also configured for the deployment of both normal Ethereum nodes as well as faulty nodes to test Byzantine properties. Istanbul provides a "faulty node instantiation" reflecting behavior such as delays or dropped messages. This was used in the test and created realistic behaviors where the underlying cause of the fault could not be dictated by the test environment.

The overall environment provides an interactive console to interact with the blockchain as well as a browser based UI to monitor blockchain activity. The tests performed were adapted from Istanbul Ethereum internal tests.

## 4.4 Testing Parameters

Testing focused on two parameters: 1.) what happens when less than the required number of good nodes exist and the blockchain network slows or stops and 2.) what occurs if the required number of consensus votes are not achieved.

### 4.4.1 Faulty Node Limit:

Istanbul is based on traditional Practical BFT concepts. Therefore, to operate in a BFT environment you must have a minimum of $3f+1$ nodes, where $f$ is the number of faulty nodes. $3f+1$ provides the ability to handle traditional crash faults as well as potential Byzantine nodes that may be "lying" to other nodes on the network. So, the minimal network size should be 4 nodes. An estimated maximum network size at this stage in Istanbul development is 100. There is active, on-going research to improve that number. The large number of message exchanges required is why the limitation now exists.

### 4.4.2 Votes required for Consensus:

For nodes to make progress, they must receive votes from more than 2/3 of the network nodes on the various stages of the protocol. So as a general rule, validator nodes rely on 2/3 messages in a $3f+1$ network.

**Test 1: Basic Consensus**

Goal:

- Show that Istanbul achieves the promised 1 second block time under normal conditions

Test:

- Run 4 good nodes and track block creation time

Finding:

- Achieves on average 1 sec or less Block time

Note: When things are working as they should, you'll get approximately 1 second block times. As the network degrades this will increase until you reach a threshold and it stops; we were able to observe this in Tests 2 and 3. Block time directly impacts transaction speed as the transactions are processed in batch (block). The transaction speed on a healthy network is dependent on the complexity of the application logic and (potentially) transaction size. In a very simple application, you can get potentially 1000-1200 transactions per second.

**Test 2: Observe Istanbul when Faulty Node Limit is Reached**

Goal:

- Show Istanbul abides by $3f+1$ BFT requirement

Test:

- Start 3 “good” and 1 “bad” node – show network continues to make progress, but exhibits varying block time
- Add another bad node – show consensus becomes choppy/halts and behavior is visible to the network
- Remove bad node and show the protocol can adapt

Finding:

- Istanbul Follows $3f+1$ rule.
    - Note, this does not take into consideration adversarial network conditions, such as an adversary having control over the network and manipulating network protocols.

**Test 3: Validator Quorum Requirement**

Goal:

- Show Istanbul abides by 2/3 quorum requirement among validators

Test:

- Start 4 good validator nodes
- Stop 1 validator. Network slows but still working == 2/3 quorum
- Stop another. Network begins to stall <= 2/3 quorum
- Incrementally restart validators – show network recovers

Findings:

- Istanbul follows the 2/3 rule. Note, this does not take into consideration adversarial network conditions.

Other potential issues identified but not yet confirmed by testing.

- Testing should be conducted with simulated network traffic to see how network traffic increases and congestion affects the Istanbul consensus protocol.
- Conflicting Blocks:
    - 1 validator receives 2/3 commits and inserts the block. But, the other nodes (maybe due to network latency) do not receive 2/3 so they begin another round on the same block height. This could lead to conflicting blocks at the same height.
    - Also, if 2 good nodes for one reason or another “lock” on 2 different blocks, they may not receive the required 2/3 votes, resulting in an endless cycle for a portion of the protocol that would prevent progress on consensus.
- Istanbul, like other BFT algorithms using non-proof of work, are weakly synchronous. This means they rely on a timing mechanism to deal with network latency. They assume a somewhat reliable network. Permissioned blockchains, therefore, cannot withstand the same network conditions (partial partitions, etc.) that public blockchains such as Bitcoin and Ethereum can withstand. Permissioned blockchains are designed for a more controlled environment than public blockchains.

Overall, it’s recognized that Istanbul does not yet deal with issues related to weak synchrony. When they do finally address these issues, the result will closely resemble the Tendermint consensus algorithm.

# 5 P2P Cryptographic Protocol Analysis

A P2P network distributes the computing and processing burden among nodes on the network, in contrast to a client-server model where some nodes are more privileged than others. There are many Internet Engineering Task Force (IETF) P2P protocol standards [16], [17]. Some are optimized for file transfers and many of the common routing protocols such as Border Gateway Protocol (BGP) [18] and Routing Information Protocol (RIP) [19] are P2P protocols.

In the following, we present descriptions and analysis processes of two P2P protocols: Ethereum's RLPx protocol and Tendermint's P2P protocol.

## 5.1 Ethereum RLPx P2P Protocol

RLPx is a cryptographic P2P network and protocol suite used by Ethereum that is designed to meet the requirements of decentralized applications. Its main responsibility is to propagate transaction and block information across the network. It also maintains connectivity, communicates the consensus protocol, and filters malformed or non-verified transactions. It uses User Datagram Protocol (UDP) for node discovery and Transmission Control Protocol (TCP) for all ensuing sub-protocols of RLPx (to be explained in the next section).

Often, P2P protocols use a Distributed Hash Table (DHT) to calculate the distance between nodes and find the most efficient nodes for interaction. RLPx uses some elements of a common type of DHT, Kademlia. The main function of Kademlia is to find multiple files or content distributed across nodes. In the case of RLPx, the Blockchain is the single item of distributed content.

In Kademlia, a node ID is a large random number unique to the node that identifies the node and is used to locate distributed content in the network. Kademlia learns of neighbor nodes by querying closer and closer nodes. It calculates the distance to another node by performing a bitwise XOR of the two node IDs interpreted as an integer: distance (a,b) = a XOR b. The result is the closest set of nodes containing the desired content.

RLPx contains many features that are not included in common Kademlia implementations.

- Packets are signed; verification is performed by recovering the public key from the signature and checking that it matches the expected value.
- Node IDs are static 512-bit ECDSA public keys.
- DHT-related features are excluded. FIND_VALUE and STORE packets are not implemented.
- The XOR distance metric is based on SHA3 hash of the node ID (which is a public key).

### 5.1.1 Cryptographic Protocols in RLPx

This paper addresses the analysis of the cryptographic sub-protocols in RLPx. Packets are dynamically framed, prefixed with a Recursive Length Prefix (RLP) encoded header, encrypted, and authenticated. Multiplexing is achieved via the frame header which specifies the destination protocol of a packet. Cryptographic operations are based on ECDSA with secp256k1 parameters, SHA3-256 for hashing, AES256 for encryption, and Message Authentication Codes (MACs).

Each node maintains a static ECDSA key pair which is saved and restored between sessions. The node is able to generate ephemeral key pairs for the Encrypted Handshake phase. An RLPx implementation consists of several sub-protocols: Node Discovery, Encrypted Handshake, Framing, and Flow Control.

A primary function of any P2P communications protocol is neighbor discovery. Essential properties of neighbor discovery are: new nodes can reliably find nodes in the network; the network topology information is available to connect to other nodes; and node IDs are random.

The Node Discovery phase is where network formation occurs and is based on UDP Kademlia routing that has been repurposed as a P2P neighbor discovery protocol. There is no limit on the number of UDP connections a node can have, but there is a limit of 16 concurrent connections. RLPx Node Discovery uses static 512-bit ECDSA public keys as node IDs. A Node Discovery packet contains the packet-type, packet-data, a signature over the packet-type and packet-data, and the hash of the signature, packet-type, and packet-data. There are four packet-types, all of which include a timestamp.

- PingNode: checks responsiveness
- Pong: provides responsiveness; as of geth v1.8.0, contains the hash of the PingNode message to which the node is responding
- FindNeighbors: contains the nodeID of the desired target node
- Neighbors: response to a FindNeighbors request; returns a list of nodes (with the IP address, UDP port, and TCP port) closest to the target node

After the Node Discovery phase has completed, TCP connections are established via a handshake and, once established, packets are encapsulated as frames which are encrypted using AES-256 in CTR mode. By default, the maximum number of TCP connections at any given time is 25 which is determined by the `maxpeers` parameter. The handshake is carried out in two phases. The first phase is key exchange and the second phase is authentication and protocol negotiation. The key exchange is a pair of Elliptic Curve Integrated Encryption Scheme (ECIES) public key-encrypted messages which include ephemeral ECDSA keys for Perfect Forward Secrecy (PFS) and fresh nonces. Key material for encrypting future sessions is derived using a Key Derivation Function (KDF) with the nonces and Elliptic Curve Diffie-Hellman Ephemeral (ECDHE)-derived keying material. The second phase of the handshake is a pair of Hello messages from one node to the other that contain the capabilities that each node supports.

After the first phase of the handshake, packets (including the Hello message) are dynamically framed, prefixed with an RLP encoded header, encrypted with AES256, and authenticated with a MAC where the AES encryption key and MAC key are derived from the handshake. Multiplexing is achieved via the frame header which specifies the destination protocol of a packet.

A representation of the above sub-protocols using a common notation for encryption, signing, and hashing is provided in Appendix C.1.

### 5.1.2 Modeling Choices

The Cryptographic Protocol Shapes Analyzer (CPSA) tool (described in Section 5.3) was used to analyze the P2P protocols. This requires the user to model the protocols in a specific format and

specify the key assumptions of the protocols. Additionally, CPSA does not provide the full functionality needed to model the protocols as implemented, so some modeling choices were made when creating the input to CPSA for the protocols. The approximations below still allow for accurate modeling of the protocols as described. The output of the tool presents a valid analysis of the protocols.

#### 5.1.2.1 XOR function

All instances of the XOR function in RLPx occur within a hash function. This is true when XOR appears in a signature since the first step of signing a message is to hash the message. Since the commutativity and associativity properties of XOR are not used in any instance of XOR in RLPx, we represent the hash of the XOR of two values hash(a xor b) as the hash of the concatenation of the two values hash(a || b).

#### 5.1.2.2 ECDSA

CPSA provides the ability to model protocols that include Diffie-Hellman operations, however the Diffie-Hellman functionality does not allow modeling ECDSA as the tool does not permit using a Diffie-Hellman exponent to be used to sign a message as explained further below. One can use the Diffie-Hellman functionality in CPSA to represent ECDSA keys by modeling the ECDSA private key with the Diffie-Hellman private key and modeling the ECDSA public key with the Diffie-Hellman public key. A problem with this representation arises when modeling ECDSA in CPSA: the digital signature of a message using a private key is represented in CPSA by the encryption of the message with the private key. The problem that arises is that the Diffie-Hellman private key is of type “exponent” in CPSA which cannot be used as an asymmetric encryption key.

The chosen solution to the problem is to not use the Diffie-Hellman functionality available in CPSA, but rather to represent the ECDSA public and private keys using the basic functionality provided by CPSA. In this case, the public key is represented by an asymmetric key type and the private key is represented by the inverse of the public key. Now the signature of a message using a private key is a valid input to CPSA.

#### 5.1.2.3 Message Authentication Codes (MACs)

MACs can be implemented using encryption or hashing of the message. We have chosen to represent the MAC of a message as the encryption of the message with the MAC secret key derived in the Encrypted Handshake sub-protocol. Both choices will produce the same security results in CPSA since CPSA treats a hashed message as an encryption using the message as the secret key.

## 5.2 Tendermint P2P Protocol

The Tendermint P2P protocol uses an authenticated encryption scheme based on the Station-to-Station Protocol [20] which is known to be secure in particular against man-in-the-middle (MITM) attacks. However, the Tendermint P2P protocol makes sufficient changes to the Station-to-Station protocol that the resulting protocol is susceptible to MITM attacks if the persistent

signature public key of the peer node is not known in advance. This attack can be mitigated by adding a trusted third party such as a certificate authority to certify the public key is associated with a particular node. This of course dampens the decentralized spirit of blockchain. An alternative to using a trusted third party is to use the blockchain as a decentralized certificate authority to ensure that nodes are connected to at least one validator. A description of the protocol follows.

The Tendermint P2P protocol uses the Edwards-curve Digital Signature Algorithm (EdDSA) with ED25519 parameters. These parameters include using SHA2-512/256 (computing the SHA2-512 hash of the input then truncating the output to 256 bits) and elliptic curve Curve25519. These parameters seem to be chosen to optimize speed of signature generation. A signature key pair generated with these parameters is called an ED25519 key pair.

It is assumed that each node has a persistent static ED25519 key-pair and uses the corresponding public key as its ID. The protocol begins with the initiator node and peer node each generating an ephemeral ED25519 key pair and sending the corresponding public key to the other node over a newly established TCP connection.

A challenge is generated by both nodes by taking the SHA2-256 hash of the concatenation of the initiator node ephemeral public key with the peer node ephemeral public key. Each node signs the challenge with its persistent static private key and sends the other node a message consisting of the node's persistent static public key and the signature of the challenge. All future communications are encrypted.

Next two nonces are generated; one for encrypting messages to the other node and one for decrypting messages from the other node. To compute these nonces, the Ripemd160 hash of the concatenating of the initiator node ephemeral public key with the peer node ephemeral public key is used. The output of the Ripemd160 hash is 20 bytes long. Two 24 byte nonces are generated by appending four bytes to the hash output. One set of four bytes is zeros 0x0000 and the other is 0x0001. To determine which nonce the nodes will use, the two ephemeral public keys are sorted and the node with the lower public key uses the nonce with 0x0000 appended.

To encrypt messages between each other, the nodes use the Diffie-Hellman shared secret derived from the ephemeral keys by multiplying the node's ephemeral private key with the other node's ephemeral public key. As a result of the underlying mathematics, the result is the same for both nodes. The shared secret is used as the symmetric key for encryption. All future communications are encrypted using the shared secret and the generated nonces, where each nonce is incremented by one each time it is used. The communications maintain PFS, as the ephemeral key pair is used to derive the symmetric key used for encryption.

### 5.2.1 Modeling Choices

As in the case of modeling RLPx for input to the CPSA tool, there are issues with using a Diffie-Hellman exponent to sign a message. However, in the Tendermint P2P protocol, only the persistent key is used to sign a message, so the ephemeral keys can be modeled using the Diffie-Hellman functionality in CPSA. This makes the modeling much simpler as is seen in the details in Appendix D-1 and D-2.

## 5.3 Tools Used

The CPSA tool was used to model and analyze the RLPx and Tendermint P2P protocols to determine whether the protocols that contain cryptographic operations achieve their desired secrecy and authentication goals. CPSA is a software tool designed to assist in the design and analysis of cryptographic protocols. The repository for CPSA is: *git://github.com/mitre/cpsa.git.* It takes as input a protocol definition and a partial description of an execution of the protocol. CPSA then produces descriptions of all essentially different executions of the protocol that complete the partial description; these descriptions are called shapes. These descriptions are provided in both textual and graphical output files; some graphical output is provided in Appendix D as examples of what the tool produces. CPSA is consistent with the presence of a powerful network adversary capable of breaking a secrecy or authentication property that the protocol was intended to achieve. As a note, CPSA reveals only structural flaws in the protocol; it cannot detect flaws in underlying cryptographic algorithms or in protocol implementations.

# 6 Findings and Recommendations

## 6.1 Istanbul Consensus Algorithm

Based on our testing and analysis, Istanbul demonstrates good byzantine fault tolerance behavior. It, however, still needs improvements to be production-ready. Because Istanbul will resemble Tendermint when the required changes are made, we recommend Tendermint consensus for current NSA or NSA customer use, based on MITRE internal research and testing. Tendermint P2P protocol, as our analysis shows, should be closely monitored as developers are making improvements regularly.

## 6.2 P2P Protocols

### 6.2.1 RLPx Analysis Results

We describe below the results of the CPSA analysis of RLPx. Some examples of the graphical output of the analysis are provided in Appendix C-2. Some known non-cryptographic attacks on RLPx are presented in Appendix C-3. Notes on the effects of a quantum computer are provided in Appendix C-4.

#### 6.2.1.1 Node Discovery

The first sub-protocol of RLPx is Node Discovery which, as mentioned in Section 5.1.1, contains an ECDSA signature of the packet-type and packet-data (using the node's long term static ECDSA private key) and a SHA3-256 hash of the signature, packet-type and packet-data. There are four packet types which come in send / response pairs: PingNode / Pong and FindNeighbors / Neighbors. Both message pairs were modeled and analyzed separately in CPSA with the output descriptions consistent with the protocol expectations; there is no anomalous behavior in any possible execution of the protocol.

#### 6.2.1.2 Encrypted Handshake

The next sub-protocol is the Encrypted Handshake. Recall that the first phase of the Encrypted Handshake is a message pair between two nodes that wish to communicate securely. If this is a new connection between the nodes involved, the initiating node sends the responder node a message containing 1. the signature using an ephemeral ECDSA private key of the static Diffie-Hellman derived key XORed with a nonce created by the initiating node, 2. the hash of its ephemeral public key, 3. its long term static ECDSA public key, 4. the nonce, and 5. a flag "0x0"; all of this is ECIES encrypted in the long term public ECDSA key of the responder node. The responder node then sends the initiator node a message containing 1. its ephemeral public key, 2. a nonce created by the responder node, and 3. a flag "0x0"; all of this is ECIES encrypted in the long term public ECDSA key of the initiator node. In the case where the two nodes have communicated before, they will have a token that was derived during the previous handshake that is used in place of the Diffie-Hellman derived key during part 1 of the message from the initiator to the responder, and the flag in the responder message to the initiator is "0x1" representing that the responder node found the token. The ephemeral keys are used to calculate the Diffie-Hellman shared secret which, along with the nonces, are used to calculate the AES and MAC keys used in all future communications between the nodes.

Both phases of the Encrypted Handshake were modeled and analyzed together as one protocol where the Hello message in the second phase is encrypted using the derived material from the first phase. As with the Node Discovery protocol, the output of the CPSA tool showed that anomalous behavior cannot occur in an execution of the sub-protocol.

#### 6.2.1.3 Framing

All packets after the first phase of the Encrypted Handshake are framed, encrypted, and authenticated where the encryption and MAC keys are derived from the first phase and the MAC keys are updated with each use. Again, the output of CPSA showed that this sub-protocol is secure.

### 6.2.2 Tendermint P2P Protocol Analysis Results

We describe below the results of the CPSA analysis of the Tendermint P2P protocol. Some examples of the graphical output of the analysis are provided in Appendix D-2. A non-cryptographic attack on the implementation of the protocol is presented in Appendix D-3. Notes on the effects of a quantum computer are provided in Appendix D-4.

As mentioned in Section 5.2, the protocol is susceptible to a MITM attack. CPSA detected such attacks from both the initiator and responder point of view as described in Appendix D-2 but found no other weakness in the protocol. The protocol was modified by the Tendermint developers to include a certificate authority issuing certificates binding the initiator's and responder's names to their respective persistent public keys and both parties sending these certificates along with their public keys and signed messages. In this case, it was verified that the protocol completes as expected without a MITM attack; in fact, the modified protocol contains no structural flaws.

## 6.3 General Recommendations

- U.S. Government agencies should collaborate and leverage independent research entities to follow developments in Blockchain technology. Using this paper as a foundational understanding of blockchain security, we recommend developing a set of probable use cases, security characteristics needed and strategies to capitalize on the most mature blockchain developments. Proprietary solutions should be viewed cautiously so as not to be locked into a specific vendor without fully understanding the underlying components or the ability to tailor the code for specific DoD and IC needs.

- Grow a body of NSA expertise in blockchain technology so that the NSA can inform the DoD and IC community about best security practices and proper applications of blockchain technology. This knowledge should include developer skills and familiarization with open source blockchain development activities. Ideally, NSA experts should be from both the Cybersecurity Capabilities and Research Organizations.

- A new approach is needed for engagement with the open-source community. Active participation in the development should be sought. NSA developers should begin to contribute to blockchain efforts and other cybersecurity experts should promote standardization and interoperability of the different blockchain instances, especially P2P protocols. The two efforts must occur in parallel because of the fast pace of blockchain development.

- At the P2P layer, Ethereum is recommended for general use as its RLPx protocol is proven to be secure against structural attacks and most of the eclipse attacks described in Appendix C.3 have been mitigated in the latest geth version. However there is a concern about RLPx: the ECDSA key pair is used for both signing in the node discovery phase and public key encrypting in the encrypted handshake phase. It is generally recommended that different keys be used for different cryptographic operations.

- Investigate blockchain interoperability. One project built with the Tendermint ABCI is the Cosmos Network. The vision is an interoperable multi-chain network providing trustless exchange of cryptographic assets across independent blockchains, called zones.

- The NSA Cybersecurity organization should closely track emerging off-chain encryption schemes and provide guidance on best implementation practices and security mechanisms as well as identifying possible use cases for blockchain technologies combined with off-chain encryption.

## 6.4 Technical Areas to Investigate

Permissioned blockchains best match U.S. Government requirements. We suggest that the following technical issues be investigated more thoroughly:

- The Tendermint MITM Attack, addressed in this paper is well known and published. NSA should participate in framing a solution that does not utilize a centralized authority. NSA should follow these developments and advise customers not to accept centralized solutions.

- As NSA monitors Quantum computing and its effects on current solutions, they should also have a plan for implementing blockchain using Quantum resistant algorithms. (See Appendix C, Section C.4 for details.)

- Current permissioned blockchain instantiations assume a weakly synchronous network. This means, that predictable occurrences of server crashes, etc. are assumed. However, this does not account for non-predictable behavior that could result in network fragmentation such as cyber, kinetic or poor intermittent connectivity. How resilient a blockchain is in a truly asynchronous network should be studied and applied to blockchain implementations across the U.S. Government.

- Ethereum smart contracts, although not within the scope of this paper, should be of interest to NSA cybersecurity experts and they should advocate for best practices to assure the software in these contracts because of their permanence on the blockchain. NSA should guide components and programming languages used and approaches to security and threat analysis. [21]

# 7 References


[1] Gartner, "Smarter with Gartner," [Online]. Available: https://www.gartner.com/smarterwithgartner/gartner-top-10-strategic-technology-trends-for-2018/). [Accessed 15 February 2018].

[2] DHS, "Press Release: DHS S&T awards $9.7M for 13 Phase II Small Business Innovation Research Projects," 2017. [Online]. Available: https://www.dhs.gov/science-and-technology/news/2017/05/02/press-release-dhs-st-awards-97m-13-phase-ii-sbir-projects.

[3] GSA, "Emerging Citizen Technology," 2018. [Online]. Available: https://www.gsa.gov/technology/government-it-initiatives/emerging-citizen-technology.

[4] Bitcoin, "Bitcoin is an innovative payment network and a new kind of money," 2018. [Online]. Available: https://bitcoin.org.

[5] Bloomberg, "Walmart Is Getting Suppliers to Put Food on the Blockchain," 2018. [Online]. Available: https://www.bloomberg.com/news/articles/2018-04-23/walmart-is-getting-suppliers-to-put-food-on-blockchain-to-track.

[6] D. Bryson, D. Penney, D. Goldenberg and G. Serrao, "Blockchain Technology for Government," The MITRE Corporation, 2017.

[7] NIST NISTIR 8202, "Blockchain Technology Overview," 2018.

[8] Cosmos, "Consensus Compare: Tendermint BFT vs. EOS dPOS," 29 Sept 2017. [Online]. Available: https://blog.cosmos.network/consensus-compare-tendermint-bft-vs-eos-dpos-46c5bca7204b. [Accessed 20 August 2018].

[9] Committee on National Security Systems, 26 April 2010. [Online]. Available: https://www.cdse.edu/documents/toolkits-issm/cnssi4009.pdf.

[10] NIST SP800-53 rev. 4, "Security and Privacy Controls for Federal Information Systems and Organizations," 2015.

[11] NIST SP80053A rev. 4, "Assessing Security and Privacy Controls in Federal Information Systems and Organizations: Building Effective Assessment Plans," 2014.

[12] NIST NISTIR 7298, "Glossary of Key Information Security Terms," 2013.

[13] D. G. J. W.-O. Z. Young, "Survey of Confidentiality and Privacy Preserving Technologies for Blockchains," R3, 2016.

[14] Open Source, "Istanbul Byzantine Fault Tolerance," 22 June 2017. [Online]. Available: https://github.com/ethereum/EIPs/Issues/650. [Accessed February 2018].

[15] L. Lamport, R. Shostak and M. Pease, "The Byzantine Generals Problem," ACM Transactions on Programming Languages and Systems, Vol 4, No. 3, July 1982.

[16] A. Castor, "A (Short) Guide to Blockchain Consensus Protocols," 4 March 2017. [Online]. Available: https://www.coindesk.com/short-guide-blockchain-consensus-protocols/. [Accessed March 2018].

[17] G. Camarillo, "RFC 5694: Peer-to-Peer (P2P) Architecture: Definition, Taxonomies, Examples, and Applicability," Network Working Group, IETF, 2009.

[18] D. Bryan, P. Matthews, E. Shim, D. Willis and S. Dawkins, "RFC 7890: Concepts and Terminology for Peer-to-Peer SIP (P2PSIP)," IETF, 2016.

[19] K. Lougheed and Y. Rekhter, "RFC 4271: A Border Gateway Protocol 4 (BGP-4)," IETF, 2006.

[20] G. Malkin, "RFC 2453: RIP Version 2," IETF, 1998.

[21] W. Diffie, V. Oorschot and M. Wiener, "Authentication and Authenticated Key Exchanges," *Designs, Codes and Cryptography,* vol. 2, pp. 107-125, 1992.

[22] Ethereum, "Thinking About Smart Contract Security," 2016. [Online]. Available: https://blog.ethereum.org/2016/06/19/thinking-smart-contract-security/.

[23] Y. Marcus, E. Heilman and S. Goldberg, "Low-Resource Eclipse Attacks on Ethereum's Peer-to-Peer Network," Boston University and University of Pittsburgh, 2018.

[24] S. Pappas and B. Rodenburg, "BlockChain and Quantum Computing," MITRE, 2017.

[25] A. Brink, "Potentially a malicious peer can prevent nodes from connecting to each other by lying about their peer id #2033," 2018. [Online]. Available: https://github.com/tendermint/tendermint/issues/2033.

[26] M. Castro and B. Liskov, "Practical Byzantine Fault Tolerance," in *Third Symposium on Operating Systems Design and Implementation*, New Orleans, 1999.

[27] NIST SP800-160 Vol. 1, "Systems Security Engineering: Considerations for a Multidisciplinary Approach in the Engineering of Trustworthy Secure Systems," 2018.

[28] NIST FIPS 200, "Minimum Security Requirements for Federal Information and Information Systems," 2006.

[29] NIST SP800-18, "Guide for Developing Security Plans for Federal Information Systems," 2006.

[30] NIST SP800-37 rev. 2, "Risk Management Framework for Information Systems and Organizations: A System Life Cycle Approach for Security and Privacy," 2018.

[31] NIST SP800-30 rev. 1, "Guide for Conducting Risk Assessments," 2012.

[32] NIST FIPS 199, "Standards for Security Categorization of Federal Information and Information Systems," 2004.

[33] NIST SP800-60 Vol. 1 rev. 1, "Guide for Mapping Types of Information and Information Systems to Security Categories," 2008.

This page intentionally left blank.

# Appendix A Glossary

| | |
|---|---|
| Backlog | The storage to keep future consensus messages. |
| Consensus Proof | The commitment signatures of a block that can prove the block has gone through the consensus process. |
| Proposal | New block generation proposal which is undergoing consensus processing. |
| Proposer | A block validation participant that is chosen to propose block in a consensus round. |
| Round | Consensus round. A round starts with the proposer creating a block proposal and ends with a block commitment or round change. |
| Round State | Consensus messages of a specific sequence and round, including pre-prepare message, prepare message, and commit message. |
| Sequence | Sequence number of a proposal. A sequence number should be greater than all previous sequence numbers. Currently each proposed block height is its associated sequence number. |
| Snapshot | The validator voting state from last epoch. |
| Validator | Block validation participant. |

# Appendix B Abbreviations and Acronyms

| | |
|---|---|
| AES | Advanced Encryption Standard |
| API | Application Programming Interface |
| BFT | Byzantine Fault Tolerant |
| BGP | Border Gateway Protocol |
| CPSA | Cryptographic Protocol Shapes Analyzer |
| CSO | Cybersecurity Solutions Organization |
| CTR | Counter mode |
| DHT | Distributed Hash Table |
| DoD | Department of Defense |
| ECDH | Elliptic Curve Diffie Hellman |
| ECDHE | Ephemeral Elliptic Curve Diffie Hellman |
| ECDSA | Elliptic Curve Digital Signature Algorithm |
| ECIES | Elliptic Curve Integrated Encryption Scheme |
| EdDSA | Edwards-curve Digital Signature Algorithm |
| EVM | Ethereum Virtual Machine |
| IETF | Internet Engineering Task Force |
| IPFS | InterPlanetary File System |
| KDF | Key Derivation Function |
| MAC | Message Authentication Code |
| MITM | Man-in-the-Middle |
| NSA | National Security Agency |
| NTP | Network Time Protocol |
| P2P | Peer-to-Peer |
| PFS | Perfect Forward Secrecy |
| PKI | Public Key Infrastructure |
| PoS | Proof of Stake |
| PoW | Proof of Work |
| RIP | Routing Information Protocol |
| RLP | Recursive Length Prefix |
| RLPx | P2P Protocol underlying Ethereum |
| RPC | Remote Procedure Call |
| SHA | Secure Hash Algorithms |
| TCP | Transmission Control Protocol |
| UDP | User Datagram Protocol |
| WAN | Wide Area Network |

# Appendix C Additional Information on RLPx

## C.1 The RLPx Sub-protocols

### C.1.1 Node Discovery

These messages have the form

Hash || Signature || packet-type || packet-data

There are four types of packet data with packet types:

1. Ping, 2. Pong, 3. FindNeighbors, 4. Neighbors.

More specifically, the messages have the form

Hash(Signature || packet-type || packet-data) || Signature = Sign(Hash(packet-type || packet-data)) || packet-type || packet-data

**Note**: Hash is SHA3-256 and the signature is computed using ECDSA with the node's long term persistent ECDSA private key.

Figure C-1 illustrates the Ping/Pong and FindNeighbors/Neighbors exchanges in typical protocol description format (the symbol "|" is used instead of "||" for concatenation to save room in the figure).

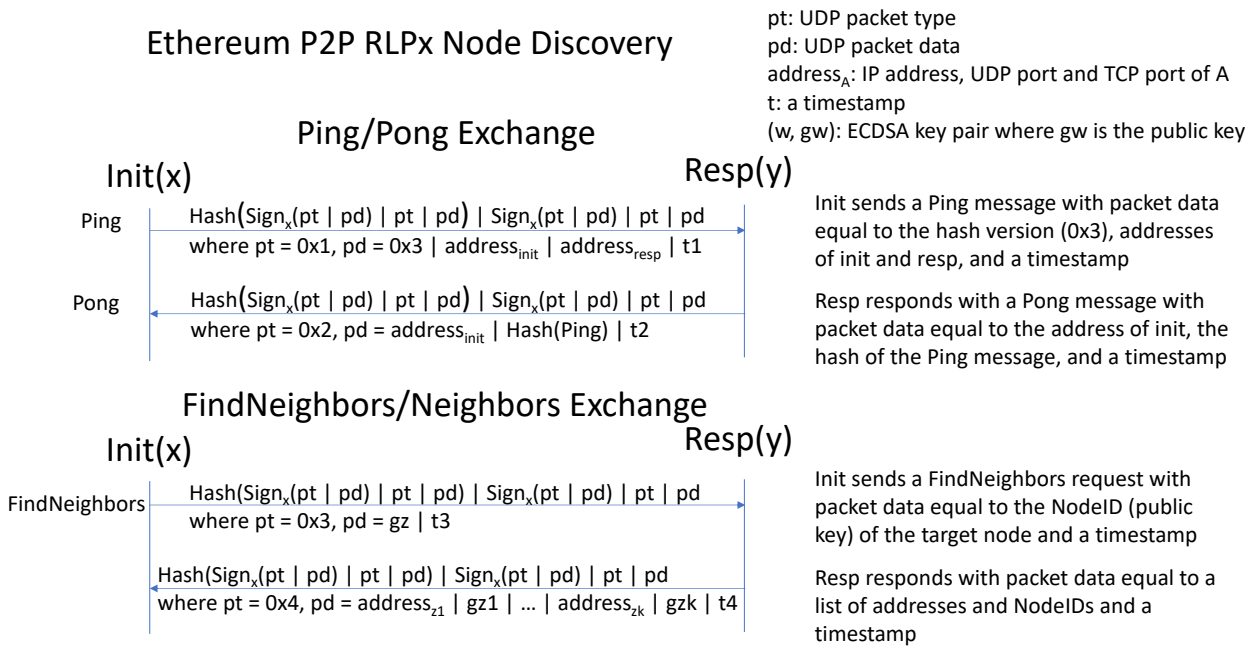


**Figure C-1 RLPx Node Discovery message exchanges**

### C.1.2 Encrypted Handshake

We use the following notation in this section.

Pubk A is the persistent static public key of A
Pubk B is the persistent static public key of B
K is the static Diffie-Hellman shared secret
(Pubk Ae, Privk Ae) is the ephemeral key pair of A
Pubk Be is the ephemeral public key of B
$N_A$ and $N_B$ are nonces generated by A and B resp.
$E_K(m)$ is the encryption of message m with key K
$Sign_{Privk\ A}(m)$ is the ECDSA signature of message m with the private key of A

**New connection**

Init -> Resp: $E_{Pubk\ B}(Sign_{Privk\ Ae}(Hash(K \text{ xor } N_A)) \| Hash(Pubk\ Ae) \| Pubk\ A \| N_A \| 0x0)$
Resp -> Init: $E_{Pubk\ A}(Pubk\ Be \| N_B \| 0x0)$

**Connection with known node**

Init -> Resp: $E_{Pubk\ B}(Sign_{Privk\ Ae}(Hash(\mathbf{token} \text{ xor } N_A)) \| Hash(Pubk\ Ae) \| Pubk\ A \| N_A \| 0x0)$
Resp -> Init: $E_{Pubk\ A}(Pubk\ Be \| N_B \| 0x1)$

**Note**: Hashing a message before signing it is an implicit part of digital signatures that is explicitly modeled here for purposes of representing Hash(x xor y) as Hash(x || y).

Figure C-2 illustrates the above message exchange for a new connection in typical protocol description format.

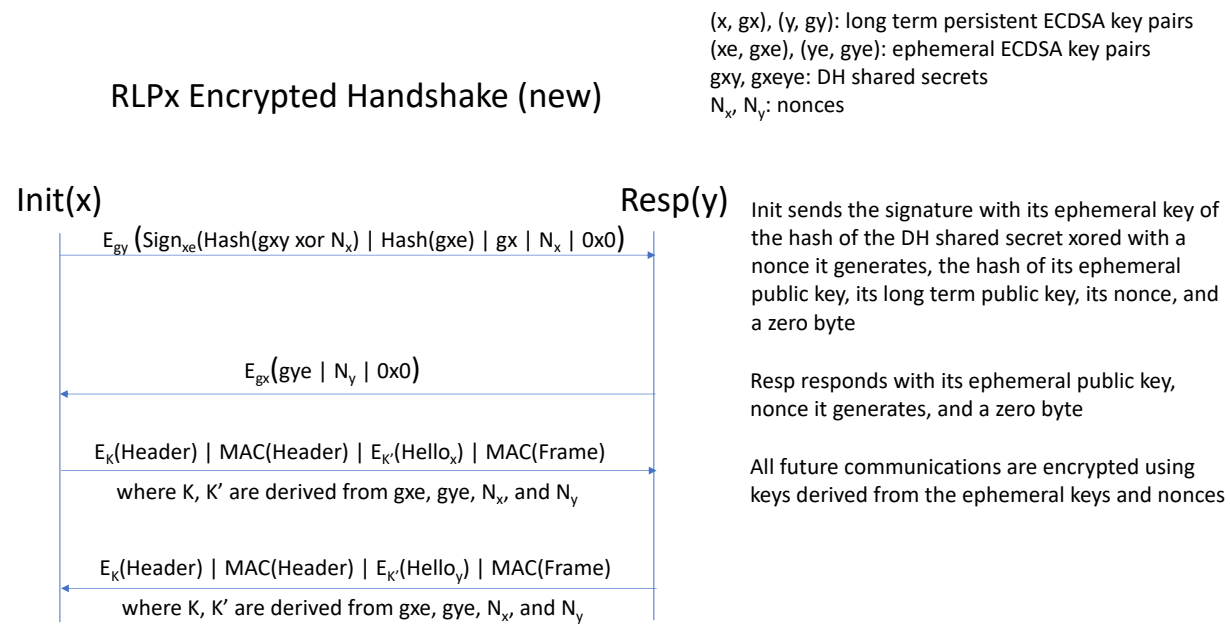


**Figure C-2 RLPx Encrypted Handshake message exchanges**

The ephemeral keys and nonces are used to generate the symmetric and MAC keys used in all future communications.

**Key Calculations**

Ke = Ephemeral shared secret
Shared secret = Hash(Ke || Hash($N_B$, $N_A$))
Token = Hash(shared secret)
AES_secret = Hash(ephemeral shared secret || shared secret)
MAC_secret = Hash(ephemeral shared secret || AES secret)
A MAC_ingress = B MAC_egress = Hash(MAC_secret xor $N_A$ || auth-sent-init)
A MAC_egress = B MAC_ingress = Hash(MAC_secret xor $N_B$ || auth-recvd-ack)

where auth-sent-init and auth-recvd-ack are the messages in the key exchange.

### C.1.3 Framing

There are two frame types: a single frame packet and a multi-frame packet. Both have the same structure:

**Note**: MAC_egress(t) and MAC_ingress(t) are the MACs of the input at a given time t. This applies to both the header and frame MACs separately. The egress and ingress MACs are updated with each frame by XORing the header or frame (depending on which MAC is being updated) with the encrypted output of the current MAC. That is, for time t, the next MAC at time t+1 is given by

MAC_egress(header)(t+1) ← $E_{MAC\_secret}$(MAC_egress(Header)(t) xor $E_{AES\_secret}$(Header))
MAC_egress(frame)(t+1) ← $E_{MAC\_secret}$(MAC_egress(Frame)(t) xor $E_{AES\_secret}$(Frame))

## C.2 Some Examples of CPSA Analysis Graphical Output

In this section, we provide some example output of the CPSA tool when analyzing the Node Discovery sub-protocol, specifically the exchange of PingNode and Pong messages. The output is a tree of graphs which demonstrate executions of the protocol consistent with the partial execution and key assumptions provided in the model. For example, the assumptions made in the PingNode/Pong exchange are that keys are generated randomly and the private key is kept secret. The graphical output of the PingNode/Pong exchange is provided in Figures C-3 and C-4.

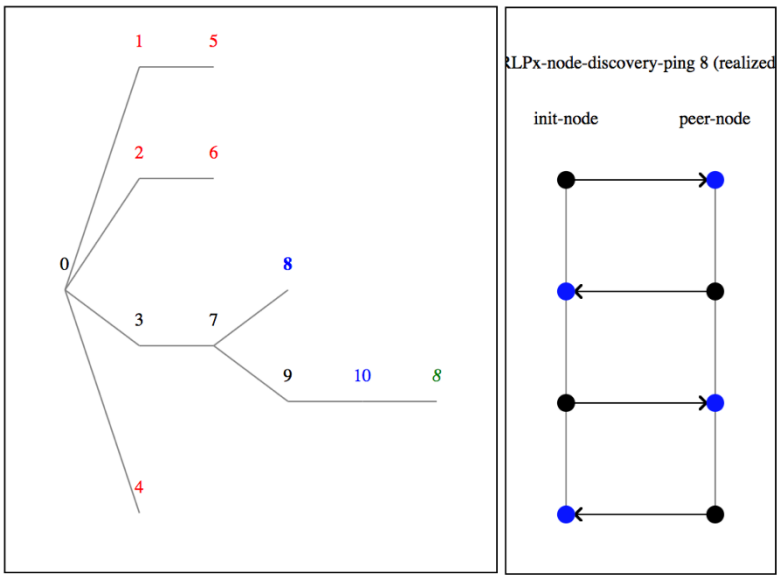


**Figure C-3 CPSA output from the initiator point of view of Ping/Pong exchange**

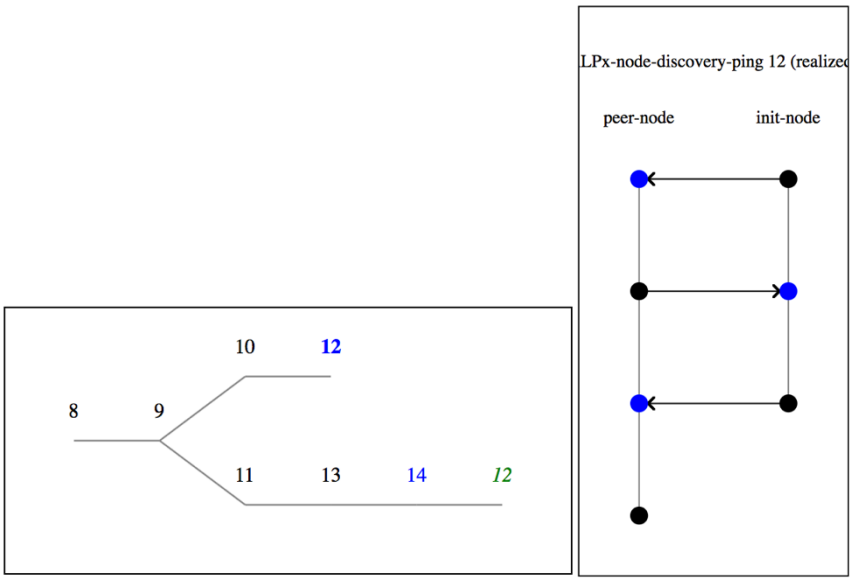


**Figure C-4 CPSA output from the responder point of view of Ping/Pong exchange**

On the left side of Figures C-3 and C-4 is a tree of graphs; each numbered node in the tree represents a graph such as the ones on the right side of the Figures. A node is black if it needs to be refined by CPSA, red if it is inconsistent with the partial description and assumptions, blue if it is an completion of the protocol consistent with the partial description and assumptions, and green if it has already been seen in the tree. As can be seen in the Figures, there is only one blue leaf node in each of the trees (these are the graphs of interest). The graph represented by the blue node for each tree is on the right side of the figure.

In Figure C-3, we see the graphical representation of the Node Discovery sub-protocol that we expected, where the vertical lines represent time going downward and the horizontal lines represent messages being sent from one participant to the other. There are four messages instead of the two PingNode and Pong messages because the first two messages represent the two nodes sending their public keys to the other node in order to model the message exchange in CPSA. In Figure C-4, the last message (Pong) is missing. This is because the responder has no way of knowing whether the initiator received the message (there is no ack).

Since the only graph that satisfies our conditions also meets our expectations, there are no vulnerabilities in the protocol.

## C.3 Known Non-Cryptographic Issues

Implementations of RLPx were not analyzed as a part of this task as CPSA only reveals structural flaws in a protocol. This section provides an overview of known attacks that are discussed in a January 2018 technical paper [22] describing eclipse attacks on the RLPx implementation in Ethereum geth version 1.6.6 released on June 23, 2017. An eclipse attack isolates a node from correctly seeing the other nodes in the network and the true state of the blockchain. An attack on the underlying P2P protocol affects the consensus protocol since nodes can be made to have a flawed view of the network causing the consensus to be formed on that incorrect view. It is important to note that most of these attacks have been mitigated in an the geth 1.8 version of Ethereum.

Three attacks are outlined below; two of them leverage the RLPx modification of Kademlia making node ID's simply 512 bit ECDSA public keys. This subsequently allows an attacker to create an unlimited number of Ethereum nodes associated with different node IDs from one IP address using a public key generation algorithm.

### C.3.1 Eclipse by Connection Monopolization

This attack on a target node occupies all its allocated TCP connections with the node IDs associated with an attacker-generated node. The attack exploits the fact that all TCP connections may be incoming (initiated by other nodes). It is implemented as follows:

1. Attacker creates at least 25 nefarious nodes (ECDSA public keys) from possibly a single IP address.
2. Force a reboot of the target node (at reboot a node has no incoming or outgoing connections).
3. Attacker sends incoming TCP connections from the nefarious nodes until all the target's TCP connection slots (25 by default) are filled before the target establishes any outgoing connections.
4. The target node is isolated.

The proposed countermeasure (included in geth version 1.8) is to have an upper limit on incoming TCP connections allowing nodes to make both incoming and outgoing connections.

### C.3.2 Eclipse by Owning the Table

This attack exploits the fact that a process used to seed the client hash table does nothing if the table is non-empty.

1. Attacker creates nefarious nodes (public keys). These nodes are created specifically to fill a certain number of last buckets in the victim's hash table. This can be done because the mapping of node IDs to buckets is public. The `logdist` function results in most nodes mapping into the last few buckets.
2. Attacker uses nefarious node IDs to continually ping the target node resulting in the target node database being filled with nefarious node IDs.
3. Force a reboot of the target node (its table is empty upon reboot).
4. Attacker aggressively pings the victim using nefarious node IDs. This fills the victim's table with the attacker node IDs and all outgoing TCP connections are with the nefarious nodes.
5. Attacker monopolizes remaining connection slots with incoming TCP connections.

### C.3.3 Attack by Manipulating Time

This attack uses the fact that UDP messages are timestamped and messages more than 20 seconds old are dropped.

1. Attacker changes the target's clock to be more than 20 seconds in the future (using known Network Timing Protocol (NTP) attacks).
2. Target will reject any honest UDP message as expired.
3. Target "forgets" all other nodes and is "forgotten" by all other nodes because it doesn't receive pong and neighbor responses from honest nodes.
4. Honest nodes are evicted from target's table (local store of nodes) and database (longer store).

At this point, the target has not been eclipsed because it can still make TCP connections. However, the target can be easily eclipsed using the above eclipse attacks since the table and database are sparsely populated because all nodes have been "forgotten".

## C.4 The Effects of a Quantum Computer

The cryptographic functions that are used in RLPx are hashes, ECDSA digital signatures, elliptic curve public key encryption (ECIES), AES symmetric key encryption, and MACs. We describe below the attacks on these functions using a quantum computer and the resulting consequences in the protocol security goals. Information used in the following was obtained from [23].

There are two algorithms that can be implemented on a quantum computer that provide a drastic speed-up in attacks on some cryptography. Grover's algorithm can be used to find hash collisions such that for a hash of length k bits there is a speedup by a factor of $2^{k/2}$ resulting in the hash being only as secure as one half the bits in the hash. Shor's algorithm can be used to find the private key from an asymmetric private/public key pair given the public key with an exponential speed-up in time. This means that an adversary with a quantum computer can

decrypt messages encrypted in others' public keys and forge signatures. In RLPx, only Shor's algorithm is needed to break the security goals of the protocol.

We now examine specifically how Shor's algorithm affects the RLPx protocol. Recall, in the Node Discovery function of RLPx, messages include an ECDSA signature over the packet-type and packet-data and a hash of the signature, packet-type, and packet-data. Assuming an adversary can use Shor's algorithm to recover the private key of the message sender, the adversary can change the packet-type and/or packet-data, sign that data with the recovered private key, and hash the result to create a different message that appears to come from the message sender.

In the Encrypted Handshake, the two messages between an initiator node and a peer node are encrypted in the persistent static ECDSA public key of the message recipient. Using Shor's algorithm, an adversary can recover the corresponding persistent static private keys to decrypt both messages. The first message from the initiator node to the peer node contains an ECDSA signature using the initiator's ephemeral ECDSA private key. The signature verification process reveals the initiator's ephemeral public key. The adversary can then use Shor's algorithm to recover the initiator's ephemeral private key. Likewise, the adversary can recover the responder's ephemeral private key since the responder's ephemeral public key is sent as the first part of the second message from the peer node to the initiator node.

The adversary now has two attack options. The first is to forge messages from either node to the other node since the adversary has all the private keys needed to properly construct one of the two messages. The other is to compute the derived keys used in all future encrypted communications between the initiator node and the peer node. The adversary can decrypt any messages between the two nodes or forge encrypted messages from one node to the other.

In the Framing function of RLPx, the header and frames are encrypted and authenticated using key material derived from the key exchange. As mentioned above, an adversary can use Shor's algorithm to recover the information needed to compute the Diffie-Hellman shared secret and therefore the key material used to encrypt and MAC the header and frame. However, the MAC keys are updated with each use, so they will become out of sync with the initiator and/or peer node.

# Appendix D Additional Information on Tendermint P2P Protocol

## D.1 Details of the Protocol

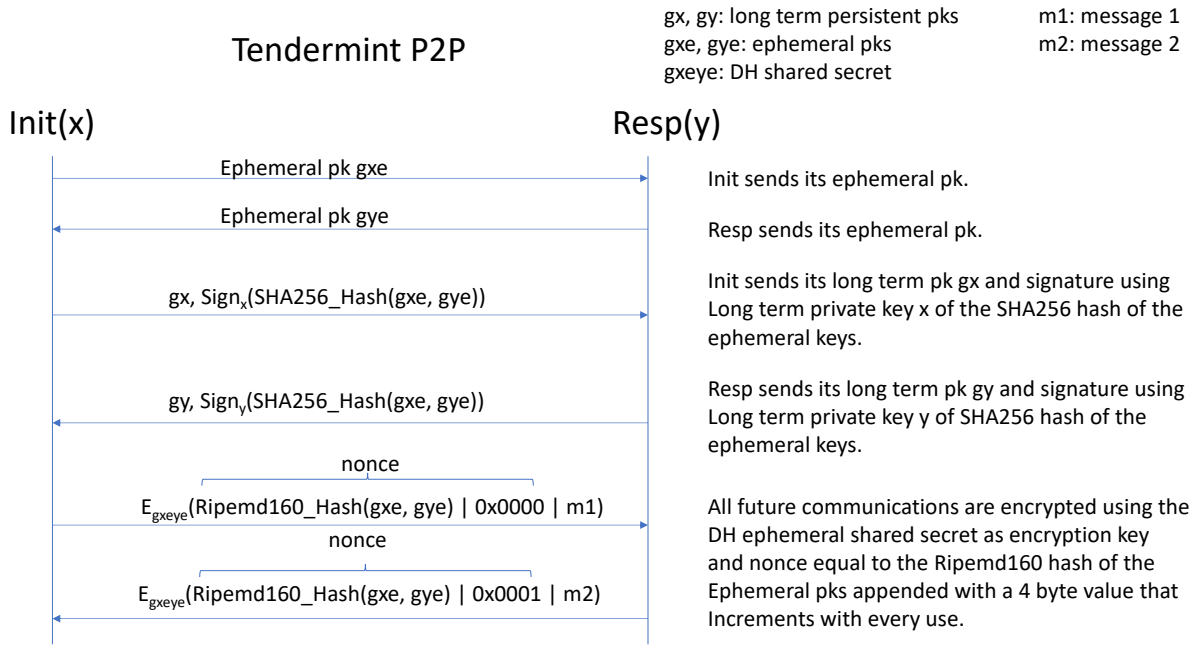


**Figure D-1 Tendermint P2P protocol message exchange**

## D.2 Some Examples of CPSA Analysis Graphical Output

A protocol may achieve (or fail to achieve) different security properties when taken from different points of view. For example, a protocol may achieve server authentication but not client authentication. For this purpose, we analyze the protocol from both the initiator's and responder's point of view.

### D.2.1 Protocol from the Initiator's Point of View

From the initiator's point of view of the protocol, CPSA produces two valid graphs in the output tree which is omitted since the idea was sufficiently introduced in Appendix C-2. The valid graphs from the tree are shown in Figure D-3. The first graph (number 1) shows the case where the initiator is interacting with itself, a case we can ignore. This is demonstrated in the gray message box that appears above the graph showing that the initiator identifies the supposed responder public key gy as gx, its own public key. The second graph (number 5) shows the protocol executing almost as expected with the difference from expected being that the third message, from the initiator to the responder, does not have to be delivered as expected. Examining the gray message box we see that the initiator sends the message containing its public key gx and a signature using its associated private key. However, the responder can receive a message from an adversary using the adversary's public key gx-0 and signature using the

adversary's associated private key and the responder will continue the protocol. This indicates the possibility of a man-in-the-middle attack where the responder thinks it is interacting with one actor but is really interacting with a different actor.

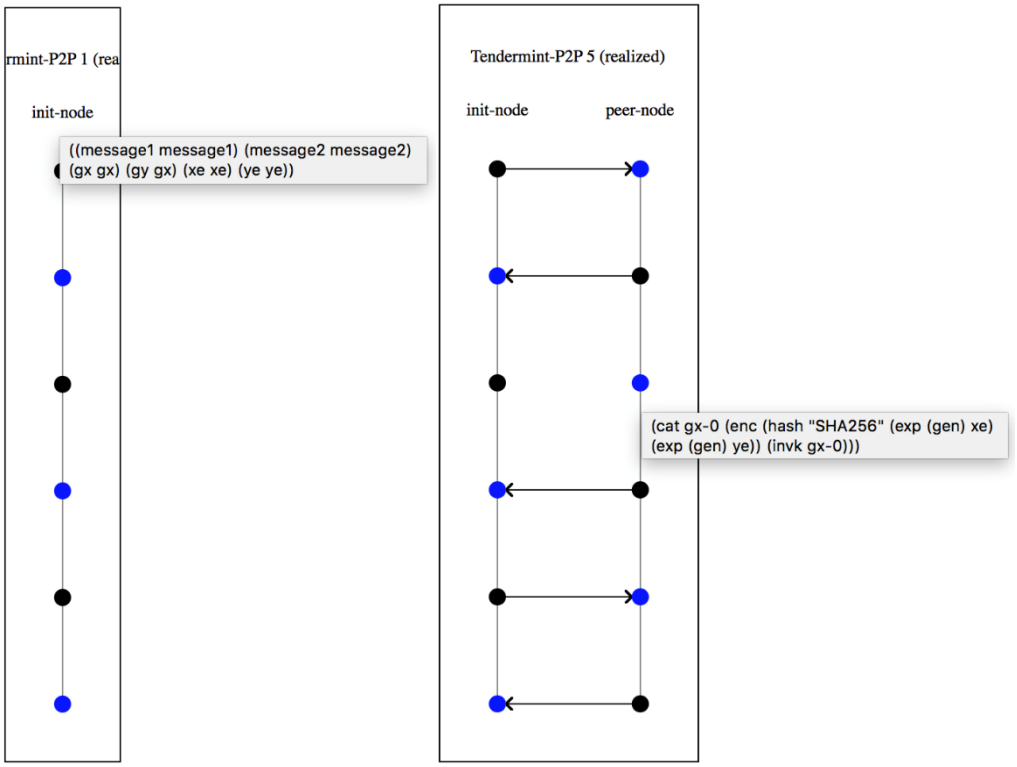


**Figure D-2 CPSA output from the initiator point of view**

### D.2.2 Protocol from the Responder's Point of View

From the responder's point of view of the protocol, CPSA produces only one valid graph in the output tree. The graph demonstrates the same man-in-the-middle attack as was shown in the initiator's point of view. In this case, the initiator thinks it is interacting with an actor with public key gy-0 different from the initiator public key gy, possibly an adversary.

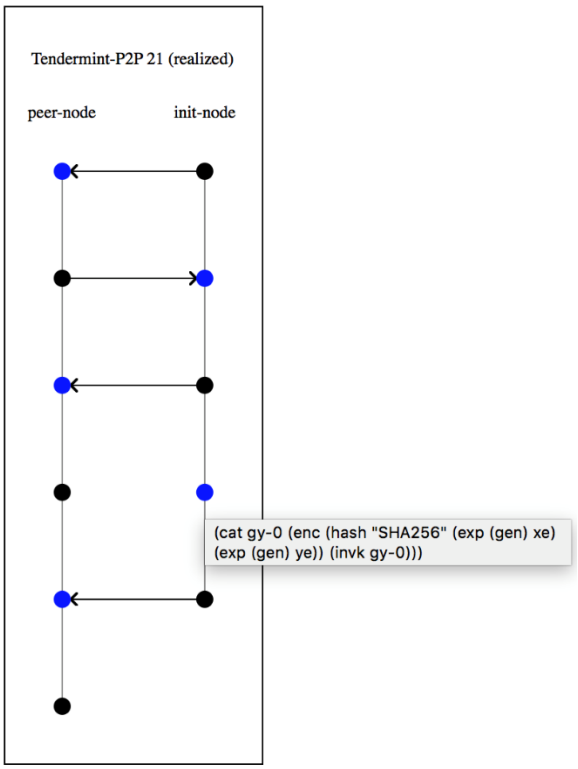


**Figure D-3 CPSA output from the responder point of view**

## D.3 Non-Cryptographic Issues

A post to the Tendermint github site [24] indicates a possible attack on the current Tendermint implementation as follows. Tendermint nodes gossip known peers between each other with an exchange of [peer id, peer ip] pairs. Honest peers will exchange [true peer id, true peer ip] pairs, whereas a malicious peer may send a [false peer id, true peer ip] pair to its peers. An examination of the implementation code shows that an attempted connection to [false peer id, true peer id] will result in no connection. This means that a malicious node can control attempted connections to peers. This potentially allows a malicious node to prevent new nodes from joining the network. Or, it could force all new nodes to connect to itself.

## D.4 The Effects of a Quantum Computer for Tendermint

The first two messages of the protocol are the initiator and responder sending their respective ephemeral public keys to the other party. Using Shor's algorithm, an adversary can recover the corresponding ephemeral private keys (although the adversary only needs one) and therefore can compute the ephemeral shared secret which is used to encrypt all future messages. The adversary can then either read all messages between the initiator and responder or forge messages to either party. However, the nonce for encryption includes a four-byte counter that is incremented with each encryption, so if the adversary chooses to forge messages to either the initiator or responder, the initiator and responder will become out of sync with respect to the counter unless the adversary is careful to forge a message to each party every time a message is forged.