Secure Aircraft Maintenance Records Using Blockchain (SAMR)

Ahrash Aleshi
SECURE AIRCRAFT MAINTENANCE RECORDS USING BLOCKCHAIN (SAMR)

Ahrash Aleshi

A Thesis in the Field of Cyber Security & Computer Science
for the Degree of Master of Science in Cybersecurity Engineering

Embry-Riddle Aeronautical University
May 2018
SECURE AIRCRAFT MAINTENANCE RECORDS USING BLOCKCHAIN (SAMR)

by Ahrash Aleshi

This thesis is prepared under the direction of the candidate’s committee chair, Dr. Remzi Seker, Department of Electrical, Computer, Software, and Systems Engineering, and has been approved by the members of her thesis committee. It is submitted to the Department of Electrical, Computer, Software, and Systems Engineering in partial fulfillment of the requirements for the degree of Master of Science in Cybersecurity Engineering.

Dr. Remzi Seker
Committee Chair

Dr. Radu F. Babiceanu
Committee Member

Dr. Jiawei Yuan
Committee Member

Dr. Timothy Wilson
Department Chair

Dr. Maj Mirmirani
Dean, College of Engineering

Dr. Christopher Grant
Vice Chancellor

2016-04-03
Date

4/16/2018
Date

4/3/18
Date
Acknowledgments

Special thanks to Embry-Riddle Aeronautical University for facilitating a positive and forward-thinking environment to conduct my research. I am also grateful for my committee members Dr. Remzi Seker, Dr. Radu F. Babiceanu, and Dr. Jiawei Yuan for their support and feedback on the paper.
Table of Contents

List of Figures ...................................................................................................................... v
List of Tables ....................................................................................................................... vi
List of Abbreviations ........................................................................................................ vii
Abstract .............................................................................................................................. 1
Chapter 1 ............................................................................................................................. 2
1.0 Introduction .................................................................................................................. 2
Chapter 2 ............................................................................................................................. 6
2.0 Literature Review ......................................................................................................... 6
  2.1 FAA Approval ............................................................................................................. 6
  2.2 Development Platform .............................................................................................. 8
  2.3 Current and Future Competitive Product Offerings ................................................... 8
  2.4 Literature Review Gaps ............................................................................................. 10
Chapter 3 ........................................................................................................................... 12
3.0 Methodology ................................................................................................................. 12
  3.1 Blockchain Background ............................................................................................ 12
  3.2 Alternative Blockchain Consensus Algorithms ......................................................... 16
  3.3 Blockchain Security .................................................................................................. 20
    3.3.1 Data Integrity .................................................................................................... 21
    3.3.2 Digital Signatures .............................................................................................. 23
    3.3.3 Access Control .................................................................................................. 24
    3.3.4 Threat Model .................................................................................................... 26
Chapter 4 ........................................................................................................................... 29
4.0 Development ................................................................................................................. 29
  4.1 Introductory Concepts ............................................................................................... 29
    4.1.1 Global State ..................................................................................................... 29
    4.1.2 Transactions ..................................................................................................... 31
    4.1.3 Validator .......................................................................................................... 33
    4.1.4 Journal .............................................................................................................. 36
    4.1.5 Proof of Elapsed Time Consensus Method ......................................................... 42
  4.2 Environment ............................................................................................................... 44
    4.2.1 Development process ....................................................................................... 44
4.2.2 Development .............................................................................................................. 45
4.2.3 Blockchain Code........................................................................................................ 47
4.3 Simulation ...................................................................................................................... 50
4.3.1 Scenario .................................................................................................................... 50
Chapter 5 ............................................................................................................................. 55
5.0 Results and Recommendations ................................................................................. 55
  5.1 Results .......................................................................................................................... 55
  5.2 Recommendations ....................................................................................................... 56
Bibliography ....................................................................................................................... 59
List of Figures

Figure 1: Transactions in the Bitcoin Network (From Bitcoin.org) ........................................... 15
Figure 2: Timestamp server hash example (From bitcoin.org) .................................................. 16
Figure 3: Merkle Hash Tree Example ............................................................................................ 30
Figure 4: Global State Change Process ........................................................................................ 31
Figure 5: Batches architecture (From Sawtooth.Hyperledger.org) ............................................... 32
Figure 6: Transaction Attribute Example ...................................................................................... 33
Figure 7: Validator Network Interacting with HTTP Client ............................................................ 34
Figure 8: Node Process .................................................................................................................. 36
Figure 9: Journal Pipeline (Figure modified from Sawtooth.Hyperledger.org) ...................... 37
Figure 10: BlockPublisher logic flow (From Hyperledger.org) .................................................. 39
Figure 11: ChainController block flow (From Hyperledger.org) ............................................... 40
Figure 12: Consensus Interface Classes ........................................................................................ 40
Figure 13: Blockchain Fork Example ............................................................................................ 42
Figure 14: High Level Blockchain Development Process .......................................................... 44
Figure 15: Docker .yaml functions .............................................................................................. 46
Figure 16: REST API in docker-compose.yaml ........................................................................... 46
Figure 17: Validator in docker-compose.yaml .............................................................................. 46
Figure 18: Blockchain settings in docker-compose.yaml .............................................................. 47
Figure 19: Sawtooth Logic UML Diagram ................................................................................... 47
Figure 21: Mike entering attributes into front end form ............................................................ 51
Figure 22: Successful entry of the block into the blockchain ...................................................... 52
Figure 23: List of block ID’s ......................................................................................................... 52
Figure 24: Lookup Block ID of completed transaction ............................................. 53
Figure 25: Deserialized data from the REST-API ...................................................... 54

List of Tables

Table 1: Hash Algorithm Properties ........................................................................ 22
Table 2: Actor Permissions ....................................................................................... 26
Table 3: Blockchain development time log .............................................................. 29
List of Abbreviations

AD – Airworthiness Directive
CFR – Code of Federal Regulations
CRS – Certified Repair Station
DSA – Digital Signature Algorithm
DSS – Digital Signature Standard
ECDSA – Elliptical Curve Digital Signature Algorithm
FAA – Federal Aviation Administration
FBO – Fixed Based Operator
HTTP – Hypertext Transfer Protocol
ICO – Initial Coin Offering
NBAA – National Business Aviation Association
NIST – National Institute of Standards and Technology
NTSB – National Transportation Board
P2P – Peer to Peer
Pk – Public Key
PoA – Proof of Activity
PoB – Proof of Burn
PoET – Proof of Elapsed Time
PoS – Proof of Stake
PoW – Proof of Work
REST API – Representational State Transfer Application Program Interface
SHA – Secured Hash Algorithms
SHS – Secure Hash Standard
Sk – Private Key
TEE – Trusted Execution Environment
UML – Unified Modeling Language
Var - Variable
Abstract

We propose to enhance the security and transparency of aircraft maintenance records in the aviation industry through the use of blockchain technology. A physical aircraft maintenance logbook is susceptible to being lost or destroyed. A nonexistent aircraft maintenance logbook hurts the confidence in integrity and reputation of the aircraft. Furthermore, fraud can occur through forgery of FAA personnel signatures and the installation of non-official aircraft parts. The scope of this work is to develop a secure blockchain that can store aircraft service records and information in a digital distributed ledger. By keeping the maintenance logbook on a digital ledger, records can be stored indefinitely in a trusted environment with the integrity of records guaranteed.

Additionally, to achieve being a distributed ledger, a consensus algorithm PoET is used to display the global state accurately to all users. The SAMR blockchain uses the Linux Foundations open sourced software “Hyperledger” to facilitate an environment that mimics a real-world implementation. The Python Programming Language was used for SAMR’s implementation of the blockchain logic through creation of a permission-based blockchain for holding the maintenance records.
Chapter 1

1.0 Introduction

Aircraft maintenance records are often an afterthought for many aircraft owners, however keeping accurate records is a critical procedure in maintaining an aircraft’s airworthiness. Currently, aircraft maintenance log books are stored in a physical ledger located in the aircraft or in the owner’s possession. A major risk associated with maintaining a physical logbook is to be lost or stolen, which will render the aircraft unairworthy (Federal Aviation Administration, 2017).

While the aircraft may be in perfect shape to fly, it is not airworthy under the FAA rules (CFR part 43) without the ledger showing that proper maintenance has been completed. If the log book is lost, the aircraft owner will have to reconstruct it from scratch, which increases in difficulty as the aircraft gets older. Reconstructing the maintenance history of an aircraft is extremely expensive and time consuming and often requires additional documents to satisfy the FAA requirements. To begin the reconstruction of maintenance logs, FAA rules (AC 43-9C) advises referencing other records that reflect the time in service, research records maintained by repair facilities, and reference records maintained by individual mechanics (Federal Aviation Administration, 2017). As the aircraft becomes older, repair facilities and mechanics may no longer be in business or hold the recorded log of events for the aircraft any more. If certain record parts are still not able to be reconstructed, an aircraft owner will need to make a notarized statement in the new record describing the loss and establishing the time in service based on the research and the best estimate of time in service (Federal Aviation Administration, 2017).
Additionally, at the time of sale of the aircraft, a lost maintenance log book can affect the resale value by as much as 50% as stated by various aviation insurance companies (H. Chappell, n.d.). Even if notarized statements have been made, potential customers are skeptical about buying an aircraft without the original documents. An aircraft buyer will typically look at the following items in a maintenance logbook to ensure compliance and aircraft continuity (A. Vasseur-Browne, 2018).

1. The aircraft’s total maintenance history.
2. Total airframe, engine, and propeller time.
3. Compliance with airworthiness directives (AD).

If the aircraft’s maintenance logbooks do not demonstrate compliance or continuity, the buyer of the aircraft would need to purchase it at a lower cost to cover the cost of reconstructing the records.

With a physical logbook, the falsification of maintenance records is also inherently possible through the forgery of signatures of an FAA inspection authority. While rare, the FAA takes a strict stance to ensure the integrity of aircraft maintenance records by prosecuting those responsible. The FAA is warranted to act in such a manner to avoid falsification of maintenance information for any reason, since it is pertinent to aviation safety. The FAA 14 CFR 43.12 title prohibits the falsification, reproduction, or alteration of maintenance records. Any operator found in violation of the title would have their certificate suspended or revoked.

The goal of this thesis is to address the flaws of using a physical aircraft maintenance log book by creating a secure and transparent blockchain ledger that will hold Aviation
Maintenance Records. The FAA has strict guidelines and regulations on when aircraft owners and operators should record any maintenance done on the aircraft but does not mandate the form of which they can be stored. The proposed method in this thesis is to develop a computationally secure blockchain to store aircraft records indefinitely, transparently, and securely.

The blockchain runs off a global collaboration opensource project hosted by the Linux Foundation named Hyperledger. Hyperledger, at the time of this writing, is currently working on eight different blockchain technology projects that differ in their intended uses ("Introduction - Sawtooth", 2018). Project Sawtooth, under the umbrella of the Hyperledger, was selected to be the development environment for the blockchain because of its ability to provide security and auditability. The blockchain developed in this thesis uses a distributed ledger data structure to allow all participants and authorized parties to view the same information which is important to promote reliability, transparency, flexibility, and FAA compliance, characteristics defined below:

- **Reliability**: Refers to the ability of the blockchain to sustain information in the event of a crash or potential attack by distributing its ledger on multiple nodes around the world.

- **Transparency**: The FAA, NTSB, and aircraft owners can benefit from being able to see the complete history of an aircraft with proper authorization and consensus among parties.

- **Flexibility**: The modular capabilities of the blockchain allow it to remain sustainable in the ever-evolving aviation and high technology industry.
• **Compliance**: Current FAA Requirements outlined in CFR part 43 and 91 are complied with, making blockchain a valid choice for replacing the physical maintenance record logbook.

To achieve these blockchain benefits, three basic components of a distributed ledger are used, namely a data model, language of transactions, and a protocol ("Introduction - Sawtooth", 2018). The data model used is a Radix Merkle Tree on an addressable 35-byte block used to store large amounts of records. The Python programming language is then used to change the state of the ledger under parameters set on transactions (new entries). Finally, Proof of Elapsed Time (PoET) is the chosen protocol used to provide consensus and ensure all data are presented correctly in the blockchain.

This thesis will focus on the physical recording of aircraft maintenance including parts, inspections, and services in the private aviation industry. The commercial industry is subject to additional regulations by the FAA, along with practices that differ between all flight operations. The blockchain will adhere to all current FAA regulations that outline retention requirements for maintenance record logbooks. In addition to the new blockchain, a simulation has been developed to highlight a real-world scenario.

With blockchain technology being used in the financial industry to record transaction data, there is an opportunity to store entire records instead. Moving aircraft maintenance records to a secure digital solution can help with many issues that aircraft owners face trying to preserve the physical logbook. Using the ideology of Haber and Stornetta, along with the proven track record of cryptocurrency such as Bitcoin, this thesis proposes a new solution for the aviation industry to continue its advancement in technology.
Chapter 2

2.0 Literature Review

2.1 FAA Approval

While the scope of this thesis does not involve getting FAA approval to use the blockchain, it is important to note current application and use cases. The FAA has outlined an entire section on the requirements of aircraft maintenance records in Advisory Circular 43-9C (Federal Aviation Administration, 2017). FAA uses Advisory Circulars (AC) to provide guidance on methods, procedures and practices that they have determined to meet their compliance and regulations. AC 43-9C specifically “shows acceptable means of compliance with the General Aviation (GA) maintenance record-making and recordkeeping requirements of Title 14 of the Code of Federal Regulations (14 CFR) parts 43 and 91” (Federal Aviation Administration, 2017). While AC 43-9C shows guidance on how to achieve proper practices, mandatory requirements are set forth in 14 CFR parts 43 and 91.

Title 14 CFR parts 43 and 91 of FAA regulations stipulate that several conditions must be met before a plane is eligible to fly again after service has been conducted. The most important regulation is that aircraft maintenance records must be updated and maintained to achieve airworthiness on the aircraft. When an aircraft is taken in for service, whether it be a routine inspection or major engine overhaul, all information must be logged by the mechanic for future reference (Federal Aviation Administration, 2017). Categories that must be filled out include a description of the duties conducted by the mechanic, the date in which maintenance was carried out, and the required certificate number and signature of the individuals responsible for the service.
Retention requirements as outlined by the FAA in CFR section 91.417(a) sets how aircraft maintenance records must be stored. “Maintenance records may be kept in any format that provides record continuity; includes required contents; lends itself to the addition of new entries; provides for signature entry; and, is intelligible” (Federal Aviation Administration, 2017). The blockchain technology solution approach proposed in this thesis conforms and meets all FAA requirements set forth in FAA’s publications.

Continuity of the records is achieved by using a distributed ledger where information is stored on nodes across the permission based network. Each node has access to the information on the permission based peer-to-peer system, which would be computationally impractical to bring down all at once. Nevertheless, backups of the blockchain states are routinely stored on servers in an unlikely scenario. Large amounts of data that come with aviation maintenance records can then be stored on the blockchain with all required contents. The data is serialized into a format which can be stored in a block and then able to be displayed again when the block is called. An identical digital formatted form is presented to the user to fill all attributes of a physical maintenance record which can then be uploaded for verification and entry into the blockchain. Entry of additional maintenance records and modification to any verified record can also be achieved through consensus on the blockchain. Multiple authorized users such as the aircraft owner, mechanic or aircraft operator can request and change information on the blockchain. Changes to the ledger are recorded and previous entries are indefinitely displayed and shown along with the new entry. Finally, the blockchain combined with a distributed ledger data format is intelligible enough to be used in a real-world setting.
With the requirements met, this blockchain could potentially be used in real world scenarios with proper approval.

2.2 Development Platform

The blockchain space is becoming more widely adopted with many development projects currently under way. This thesis uses a blockchain development platformed named Hyperledger to create a new method to store Aviation Maintenance records. Hyperledger has been in development since 2015 and was started by the Linux foundation to promote open source blockchain solutions ("Introduction - Sawtooth", 2018). In Hyperledger, a specific project was also chosen to start development on because of its enterprise focusing attributes. Named Sawtooth, this umbrella open source blockchain project will assist in the development of a blockchain that can meet the FAA requirements.

2.3 Current and Future Competitive Product Offerings

Existing market options include digital record storage, purchasing insurance, and one potential startup company. Multiple websites offer to digitize aircraft owners’ maintenance records and create backups on their internal servers including NBAA member Planelogix.com. The website offers a “premium” user experience and transparency in their payment process. Owners can either scan the pages themselves or ship the entire physical maintenance logbook to their processing facility. No information on how the maintenance records are stored is provided on the website. An attempt was made to reach out to the company but yielded no response. Software also exists, such as V-Log Aircraft Digital Records Management, which aircraft owners and operators can use to store in their cloud hosted digital portal. Once again, the company does not specify what measures or protocols are followed to achieve security for their customers.
Blockchain goes above and beyond digital traditional storage methods like Planelogix and V-Log by not having to rely on centralized servers to store data. Servers and cloud storage units can be hacked, and records stolen or lost, if not maintained properly. With a blockchain, all records are stored on nodes across the world and the state of the system will be retained in an unlikely event of a world crash. Furthermore, blockchain has been proven computationally secure and cannot be changed without the state of the system being approved and noticed (Nakamoto, 2009).

Insurance coverage can be purchased by aircraft owners to cover the reconstruction of the history of the aircraft at a premium price from insurance companies. Many aircraft owners forgo this offering because of its immense expense leaving them open to any cost and time associate with reconstruction. While the cost of reconstructing the records can be recovered, the time and depreciation on the aircraft from not having the original logbook will remain an issue. A blockchain could solve this issue by being indefinitely online and, in this case, easily accessible.

Aeron is a startup company that is creating blockchain technology for the aviation industry with a roadmap that completes its offerings in 2019. They conducted a successful Initial Coin Offering (ICO) worth $15 Million, in October of 2017, to fund their development and marketing ("Aeron - Blockchain for Aviation Safety", 2018). There is no other information available in the surveyed whitepaper about the technology created in their blockchain. The only technology information presented is that it will be created on the Ethereum network and no public code is available. The development of this thesis is done on the Hyperledger Sawtooth platform, which differs from Ethereum in
many ways. The protocols used, and code are different and will not interfere with each other’s works.

2.4 Literature Review Gaps

The new methods presented in this thesis defer from all available methods to store aviation maintenance records. A blockchain application for Aviation Maintenance Records created on the Hyperledger network has not been attempted before and Aeron is not seen as an immediate competitor due to their roadmap. Development on this blockchain includes coding the proper protocols and front-end form required to fulfill all FAA requirements on the storage and retention of Aviation Maintenance Records. Hyperledger provides a framework with potential protocols to follow, but the ultimate development of an application and its own blockchain will be covered in this thesis.

This thesis uses the algorithmic techniques that bitcoin originally brought together to transform a transaction-based ledger to a larger data structure. While distributed ledgers existed prior to bitcoin, the bitcoin blockchain marks the convergence of a host of technologies, including timestamping of transactions, Peer-to-Peer (P2P) networks, cryptography, and shared computational power, along with a new consensus algorithm ("Introduction - Sawtooth", 2018).

The differences between Bitcoin and the blockchain being developed in this thesis include data type, use cases, and consensus methods. Bitcoin stores financial transaction data that includes the addresses of the receiver and sender using a Merkle Tree. The blockchain in this thesis stores data in a ledger format in a Radix Merkle Tree that serializes the data. Storing data in the form of aviation maintenance records is more complex and different then storing transactional data. In addition to the data type, Bitcoin
was also designed to be decentralized currency used in direct competition with trust-based banking. Hyperledger on the other hand focuses on enterprise ready blockchains that can store various data types including Aviation Maintenance Records

Setting permissions in both blockchains are both different as Bitcoin is permission-less for anyone to use, while the blockchain in this thesis is permission-based. With Bitcoins blockchain, anyone and everyone has access to submit and view transactions. While the transaction addresses are hashed in such a way that the owners cannot be identified easily, it is still a security concern in an enterprise-based system. Being permission-based is essential in an enterprise system to guarantee the integrity and confidentiality of information which is where these two blockchains differ significantly. An aircraft hangar that stores its own blockchain would not want records open to the public due to liability issues. Use cases for both blockchains are different, with Bitcoin appealing to everyone, while the blockchain presented in this thesis is permission based and only viewable to relevant actors.

Finally, consensus methods of both blockchains differ. Bitcoin uses a Proof of Work algorithm, while the blockchain presented in this thesis uses Proof of Elapsed Time. At a high level, both blockchains use different algorithms to guarantee the information in the blockchain is accurate. A more technical explanation of both consensus methods is presented in the next section.
Chapter 3

3.0 Methodology

This section explains the technical requirements that must be achieved to conform to all FAA standards along with the blockchain technology used. The history of blockchain through its lifecycle will be examined along with a current use case (Bitcoin). The security of the blockchain will then be discussed which include data integrity, access control, and the threat model. Finally, the development section is broken down into introductory concepts, development lifecycle, and a simulation.

3.1 Blockchain Background

The idea of a blockchain has been around since 1991 when Stuart Haber and W. Scott Stornetta first used Merkle trees to be able to efficiently store several documents into one block (Haber & Stornetta, 1991). Their idea started with the problem of having to time-stamp scientific data in a notebook which would then have to be stamped by a notary and then verified by a company manager. If any research or experiment had to be defended, the notebook which is vulnerable in its physical form would then have to be presented with the proper documentation (Haber & Stornetta, 1991). This process was tiresome and presented a challenge for researches to keep their logs throughout their lifetime, if any part of their claim was disputed. At the time, the only methods to timestamp a document were to mail themselves a letter on an earlier date and leave it unopened or handling records with more than one person present to defend the idea.

Haber and Stornetta wanted to create a digital method to timestamp documents with two integrity promoting properties. The first property was “to time-stamp the data itself,
without any reliance on the characteristics of the medium on which the data appears, so that it is impossible to change even one bit of the document without the change being apparent” (Haber & Stornetta, 1991). In other words, the first property needs to ensure that the data cannot be changed on any device without it showing that it was tampered with. The second property states that “it should be impossible to stamp a document with a time and date different from the actual one” (Haber & Stornetta, 1991). This property ensures that the timestamp cannot be reversed and remain in place during the life of the block. The two researches then came up with an ingenious way to solve the timestamping issue by proposing a mathematically sound and computationally practical solution to the problem.

Their solution was to use blocks of data that are linked together by the previous blocks hash. The linked blocks make the previous and future blocks dependent on each other which guarantees integrity. To guarantee the integrity, one-way hash functions were used in place of a digital signature. This integrity solves the issue of tampering, as it will show if the previous block has been modified by comparing the previous hash value. The benefit of using such a system at the time was for documents to establish “precedence of intellectual property without disclosing its contents” (Haber & Stornetta, 1991). It allowed for someone to lay claim to their invention without having to keep a physical record book that was verified and stamped by a second person. At the time, they referred to their invention as linked digital time stamping. Developed out of Haber and Stornetta’s research, Bitcoin came out in 2008 to be the first to use the idea of linked digital time stamping in a ledger format.
The blockchain idea was then first used by an anonymous person or group that calls themselves Satoshi Nakamoto in a paper title “Bitcoin: A Peer-to-Peer Electronic Cash System (Nakamoto, 2009). The true identity of the creator of Bitcoin is still unknown with many speculating that it was a group of individuals rather than a single person (Nakamoto, 2009). The paper was published in 2008 launching www.bitcoin.org which laid out a plan to create a “purely peer-to-peer version of electronic cash to allow online payments to be sent directly from one party to another without going through a financial institution” (Nakamoto, 2009). Using the ideas and philosophy of Haber and Stornetta, Satoshi Nakamoto designed the peer-to-peer network around digital signatures and a linked hash chain. Satoshi argues in his paper that using a trust-based model suffers from weaknesses because transactions can be reversed through mediation. Mediation is a problem because “it increases transaction cost limiting the minimum practical transaction size and cutting off the possibility for small casual transactions, and there is a broader cost in the loss of ability to make non-reversible payments for non-reversible services” (Nakamoto, 2009). In other words, the argument is that a trust-based system gives uncertainties and requires more information to ensure that the payments are agreed upon by both parties.

To solve the inherent problem of a trust-based banking system, they designed an electronic payment system based on cryptographic proof instead of trust. The cryptographic proof “allows two willing parties to transact directly with each other without the need for a trusted third party” (Nakamoto, 2009). The resulting transactions would be protected from the weaknesses that trust banking has, such as fraud, and would protect sellers more than the traditional system. To develop a payment system based on
cryptographic proof, Bitcoin created an electronic coin that was transferred through the use of digital signatures. In Satoshi Nakamotos words “each owner transfers the coin to the next by digitally signing a hash of the previous transaction and the public key of the next owner and adding these to the end of the coin”. A visual representation of the transaction process is found below in Figure 1.

Another large problem that bitcoin had to overcome was the “double spending issue” where the same amount of money was spent more than once (Nakamoto, 2009). This issue stems from not waiting for confirmation from both parties and allowed fraud to pass through the system. To solve it, Bitcoin relies on the peer-to-peer distributed timestamp server to “generate computation proof of the chronological order of transactions (Nakamoto, 2009). The timestamp verifies that a transaction had to have had occurred because the data was hashed. A visual representation of the timestamp hash is shown in Figure 2 below.

Figure 1: Transactions in the Bitcoin Network (From Bitcoin.org)
The incentive to host nodes and complete transactions with a peer-to-peer system is a consensus method called Proof of Work. The Proof of Work system is based off Hashcash, which was developed first by Adam Back in 1997 (Back, 1997). Hashcash was originally developed for anti-denial of service and to stop email spam that came from network abuse (Back, 1997). By using Hashcash’s algorithm for the core of Bitcoins mining process, Proof of Work is easily verifiable but hard to produce by the miners. To produce a new block, miners must “mine” all the data in a block to create a new block, which then confirms all transactions in that block. Nodes then confirm transactions and compete against each other in a process that is called mining. Using the computational power of the nodes computer, mathematical formulas are solved to confirm the transaction and verify the new blocks. The blocks are then added to the blockchain and are irreversible because new transaction blocks are added using the hash value of the previous block. The cost in Proof of Work from confirming the transactions is the electricity used by the CPU of the miner.

3.2 Alternative Blockchain Consensus Algorithms

Consensus algorithms are necessary in a blockchain to ensure that all nodes have the same data of the global state at any given time. They must all be Byzantine Fault-Tolerant by design to ensure that no components of the system fail to correctly update all
nodes on the global state. They must also solve a computational scenario called the Byzantine Generals Problem. The Byzantine Generals problem was published in 1982 which used an abstract example to demonstrate how a “reliable computer system must handle malfunctioning components that give conflicting information to different parts of the system” (Lamport, Shostak & Pease, 1982). The abstract scenario presented in the paper is a group of generals that must agree on a battle plan but can only communicate by messenger. The catch is that one or more could potentially be traitors and ruin the plan. Related to computer science, the Byzantine Generals Problem means that the components of the system must not fail because of a lack of communication. To solve this dilemma, consensus algorithms were developed to ensure proper communication between the components.

Proof of Work was the first consensus algorithm to be implemented in a blockchain, but it does not come without its faults. Bitcoin suffers from being the first in the market and has inherent flaws from not being able to have predicted its rapid growth. Among those issues are the electricity that is needed to run the Proof of Work algorithms. Many miners sell their reward after completing confirmations which consequently drive the price of the cryptocurrency down. To solve this issue, alternative blockchain consensus algorithms have been developed over the years to solve various issues with Bitcoin.

Alternative consensus algorithms that have been developed include Proof of Stake, Proof of Activity, Proof of Burn, Proof of Capacity, and Proof of Elapsed Time. These algorithms ensure that each person on the blockchain is aware of the current global state of the chain and any updates that occur. Each consensus method has their own advantages and disadvantages but seek to fix issues that stem from Bitcoins blockchain.
Proof of Stake (PoS) consensus algorithm allows validators to validate blocks in accordance to how many coins they currently hold or “stake” ("Proof of Stake (PoS)", 2018). Proof of Stake does not contain miners but validators instead, because all the coins are already pre-mined with the reward being transaction fees ("Proof of Stake (PoS)", 2018). To bring down the cost of mining, PoS only allows validators to validate a portion of the block in relation to their stake in the blockchain. By only allowing validators to validate proportionality their stake, energy consumption decreases.

Proof of Activity (PoA) seeks to combine the above-mentioned Proof of Work and Proof of Stake to solve potential security weaknesses and flaws. One issue that Bitcoin might face in the future is that newly minted coins are capped at 21 million (Buchko, 2018). While not expected to run out until 2140, the approximately 4 million coins left will be the last minted (Buchko, 2018). After all the new coins are mined into circulation, miners will no longer receive the reward subsidy that they are currently collecting. Instead of the reward, miners will only be awarded transaction fees for verifying and confirming transactions. When left with no subsidy, an economic situation such as a tragedy of the commons may occur where individuals act in self-interest rather than for the good of all. Without subsidies to reward miners, transactions and the network will become less stable which impacts security (Buchko, 2018). PoA uses PoW initially by continuing to have miners solve mathematical formulas until a block is mined. The block will not contain any data in relations to transactions but will contain a header with the winning mining node (Bentov, Lee, Mizrahi & Rosenfeld, 2014). After the header is updated with the winning miners address, PoS chooses the validator to complete the block. With PoS, the
validator with the most coins has a better chance to sign and verify the block that came from the PoW miner.

Proof of Burn (PoB) is an interesting consensus algorithm that is not currently being used in any full-time project. In this method, miners “give” coins to the blockchain address to show that they have committed their coins to the system. Burning coins gives the “miner” the ability to participate in randomly selected to mining the next block (Castor, 2017). By using computational resources instead of computational power, energy use is decreased at the expense of wasting coins on being allowed to participate in the random block selection lottery.

Proof of Capacity (PoC) also known as Proof of Space, is a consensus algorithm designed for low energy consumption and is currently used in only one blockchain. PoC uses hard drive memory space instead of expensive equipment to store “plots” on the user’s system (Dziembowski, 2015). These Plots are generated as a nonce, which includes hashed data combined with the ID of the computer. The more plots or nonce a user has, the more likely they will get chosen to win the right to create the next block. Higher memory space in a computer also increases the chances of a user winning the right and claiming the reward for the next block. The main argument for this consensus method is that everyone has free space in their memory and by accessing and using it for plots, energy consumption would decrease dramatically (Dziembowski, 2015).

Proof of Elapsed Time (PoET) is the chosen consensus algorithm for this thesis as it is native to Hyperledger Sawtooth’s environment. Hyperledger Sawtooth uses a “trusted execution environment” that improves on current solutions to the Byzantine Generals problem discussed before. The benefits of using a PoET consensus method includes
fairness, investment, and verification ("PoET 1.0 Specification", 2018). Fairness is achieved by distributing the execution “election” process to all nodes to ensure each node gets a chance to participate. Investment refers to the total cost of execution proportional to the value gained from the execution. And finally, verification methods are in place to ensure all nodes can declare that the winner of the execution achieved it legitimately.

High electricity usage in PoET is therefore not an issue because of the Trusted Execution Environment (TEE). The TEE requires no extra work to be done by the computers hosting the nodes like Bitcoins PoW requires.

Each consensus algorithm brings something new to blockchain technology, but often while one issue is fixed, another one arises. For example, while PoS solves the electricity usage problem Bitcoin will face in the future, it brings an issue of security. A flaw in PoS is that if an adversary acquires 51% of the circulation of coins, the consensus algorithm will continue to reward them as the majority stakeholder. The algorithm works by giving the most reward to the highest coin holder and could easily be manipulated. While unlikely that a single individual could own 51% of the blockchain without any notice in the market, it is still a potential flaw that PoW and PoET don’t have. Furthermore, consensus methods such as Proof of Burn and Proof of Capacity waste unnecessary resources on validating blocks to solve the energy usage issues. PoET is therefore the perfect consensus algorithm for this thesis because the permission based environment filters out common issues with a public blockchain.

3.3 Blockchain Security

The security of using blockchain as a platform is important to consider when looking into real world implementation. Security benefits achieved with the blockchain developed in
this thesis include data integrity, signature verification, and a permission based network architecture. To highlight the benefits of the blockchain, Bitcoin is used as a comparison because of its history with being secure since its inception. A threat model is also presented.

### 3.3.1 Data Integrity

Data integrity is achieved with the use of hashing data in the blockchain. By using hashes, the blockchain can be verified to not have been tampered with or altered. Bitcoin and our blockchain use two similar but different methods of hashing data. The process of hashing refers to “any function that takes input of some length and compresses them into short, fixed length outputs” (Katz & Lindell, 2008). Each block in the blockchain computes a hash value into what is called a digest. The digest relays proof to the user that the data in the block is secure and the integrity remains valid. If unauthorized modification to the block occurred, the digest value would change producing an error on the blockchain.

There are numerous types of hashing algorithms but only a few are approved for use in secure cryptography because of their properties. Approved hash functions are required to be collision resistant, have pre-image resistance, and have a second pre-image resistance (U.S. Department of Commerce, 2015). Collision resistant refers to the ability of the algorithm to create different outputs from the same inputs of value \( m \) and \( m^1 \). If the outputs of \( m \) and \( m^1 \) were the same, an attacker could run the hash algorithm to find the value of the hashed message. Next, the pre-image resistance property ensures that the hash algorithm is a one-way function and an adversary could not reverse the hash digest. Lastly, a second pre-image resistance property is also required which makes “it
computationally infeasible to find a second input that has the same hash value as any other specified input” (U.S. Department of Commerce, 2015).

Hashing algorithms that implement the above properties were approved by the National Institute of Standards and Technology (NIST) called the Secure Hash Standard (SHS) to be used in secure cryptography (U.S. Department of Commerce, 2015). From the SHS, Secured Hash Algorithms (SHA) were specified which include SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224 and SHA-512/256 (U.S. Department of Commerce, 2015). While each algorithm achieves different objectives and are used in various use cases, they all achieve the same three properties outlined by the NIST. For comparison, Bitcoin uses SHA-256 while our blockchain uses SHA-512. The differences can be highlighted in the use cases that each platform uses hashing for. Bitcoin uses SHA-256 to create the Proof of Work algorithm and to create the users’ addresses while the SAMR blockchain uses SHA-512 to hash the contents in the batch to verify they have been unmodified. However, the structure of SHA-256 and SHA-512 are largely identical with differences in the computation of the message digest. Table 1 below shows the different properties of SHA-256 and SHA-512.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Message Size</th>
<th>Block Size</th>
<th>Word Size</th>
<th>Digest Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHA-256</td>
<td>$2^{64}$</td>
<td>512</td>
<td>32</td>
<td>256</td>
</tr>
<tr>
<td>SHA-512</td>
<td>$2^{128}$</td>
<td>1024</td>
<td>64</td>
<td>512</td>
</tr>
</tbody>
</table>

Table 1: Hash Algorithm Properties

As the above table shows, SHA-512 effectively doubles in size the parameters used to create the digest. A small benefit gained from using each algorithm is the speed of
processing with 32-bit and 64-bit software respectively (U.S. Department of Commerce, 2015).

### 3.3.2 Digital Signatures

Digital signatures are used in the blockchain by the actors to ensure newly created blocks are authorized and legitimate. By using digital signatures, an extra layer of security is achieved in addition to the hashes generated by the SHA-512 algorithm. Two techniques for creating digital signatures exist: symmetric and asymmetric encryption methods. In symmetric encryption, a single private-key is used to encrypt and decrypt messages. It is a fast way to communicate information but is susceptible to being broken if the single key used is leaked or acquired. In comparison, the blockchain uses asymmetric encryption techniques called public key encryption which is used to communicate information without the need to discuss secret information beforehand. The operation of generating both public and private key is slower but allows it to be used for digital signatures to provide authentication and integrity verification. Furthermore, only the private key needs to be kept secure for the scheme to work, as the public key can be known to all. Public key encryption works by “allowing a signer $S$ who has established a public key $pk$ to sign a message using the associated private key $sk$ in such a way that anyone who knows $pk$ (and knows that this public key was established by $S$) can verify that the message originated from $S$ and was not modified in transit” (Katz & Lindell, 2008). To create the public key pair, a variant of the Digital Signature Algorithm is used.

The blockchain uses an approved method to create the signatures as outlined in the Digital Signature Standard (DSS) published by the NIST. A variant of the Digital Signature Algorithm (DSA) mentioned in the DSS is used called the Elliptical Curve
Digital Signature Algorithm (ECDSA) (U.S. Department of Commerce, 2015). This variant of DSS uses elliptical curve cryptography along with secure parameters to generate secure public and private key pairs. The algorithm used are outlined in the Standards for Efficient Cryptography which provide elliptic curve domain parameters. The parameter used is called the secp256k1 which creates a 512-bit key pair for use in the blockchain. ECDSA and its parameters have been proven secure as long as the Random Number Generation (RNG) implementation has been followed correctly (U.S. Department of Commerce, 2015). In comparison, Bitcoin uses the same digital signature scheme that has proven secure from its launch in 2008.

The blockchain first digitally signs the block with sk on the transaction header block and then verifies that the public key associated with the actor is legitimate. Signatures are then used to authenticate the identity of the actors in the blockchain during the process of submitting a transaction for validation. Validators will only submit changes to the global state of the blockchain if the signatures are valid when transactions are submitted. By using digital signatures, the blockchain doubles its security layer in combination with hashes by authenticating the transactions from actors.

3.3.3 Access Control

Access control in blockchains vary depending on the use case and development process but must be considered during the development phase. The blockchain presented in this thesis uses a permission based network and system because it was designed to be implemented as an enterprise application. Blockchains such as Bitcoin are considered public blockchains because anyone can use their service to create and receive transactions of the cryptocurrency.
Being permission based allows the system administrator to establish roles with actors in the blockchain. The system administrator who globally sets up the blockchain for all parties is responsible for implementing access control in the system. For the scope of this thesis, the system administrator was run from the same virtual machine as the blockchain network and actors resided on. The limited actors in the blockchain enable it to keep it internally secure through authentication and authorization processes. Authentication layers previously discussed are used to ensure that only authorized actors are submitting transactions. Validators in the network are also permission based by the system administrator on a case by case basis. To be added to the network as a node the system administrator must do their due diligence to confirm that the repair station or government entity needs the required permissions. For example, the FAA and NTSB were added as actors because of their requirement to audit maintenance records at any given time. For the scope of this thesis, only a few nodes for the simulation were set up and established. Two certified repair stations were authorized for submitting transactions through the local network form with read and write permissions to create and view transactions. The FAA and NTSB were the government agencies chosen to perform audits on the blockchain with only read permission of the blocks. All actors except the aircraft owner or operator can be used by the consensus algorithm to execute and publish a blockchain. The aircraft owner or operator does not have a node in which to validate transactions. They just use a digital signature to sign off on the Certified Repair Station (CRS) work. A table of the actors’ permissions can be found in Table 2 below. The simulation section of the paper further elaborates on the scenario for the actors.
<table>
<thead>
<tr>
<th>Actor</th>
<th># of Nodes</th>
<th>Read permission</th>
<th>Write permission</th>
<th>Validate Permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRS</td>
<td>2</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Owner/Operator</td>
<td>1</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>FAA</td>
<td>1</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>NTSB</td>
<td>1</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
</tbody>
</table>

Table 2: Actor Permissions

3.3.4 Threat Model
Throughout the development of the blockchain, a threat model was used to discover potential weaknesses in the code and design. The threat model chosen is STRIDE which was developed by Microsoft in an effort for developers to start thinking about threats early in the application life cycle ("The STRIDE Threat Model", n.d.). STRIDE is an easy to remember acronym with every letter representing a different threat category. The definition, potential attacker vectors, and countermeasures are discussed below.

**Spoofing Identity**: The act of spoofing refers to an adversary using the identity of an authorized user to illegally access resources on a system where they would normally not have access to. In the blockchain, spoofing could occur during the transaction submission phase where an adversary could steal the login information of the front-end form. However, dual layer security in the form of digital signatures and hashing functions make it improbable that an adversary could submit transactions. The digital signatures verify that the user is who they say they are through authentication, and then authorizes them to use the submit button which would be the resource.
**Tampering with Data**: An adversary in this threat category would try to alter or modify any data either when it is stored or when it is in the process of transferring. The blockchain uses hashing algorithms to secure the payload during the transfer of transaction to the blockchain. The adversary would not be able to alter the data in the payload without the system knowing that the integrity of the file has been tampered with or modified. The Journal class would then give an error back to the submitter of the transaction with the proper response. Furthermore, the storage of the approved transaction in the blockchain cannot be altered as the original block is used to generate the hash of the future blocks that are added to the blockchain. If an adversary tried to alter a previous block the blockchain integrity would be compromised and the system administrator could revert the global state of the blockchain to before the attack. For a block to be changed, an authorized user would have to fill out a request form and submit it to the system administrator to change the block. The change would be noted on the previous block, but the old data would remain on the record and would contain a change link to the new data.

**Repudiation**: This category involves a user or attacker who leverages the inability of a system to track invalid actions and uses them to gain some advantage in the system. No threats are seen from this category in the blockchain because each section is as self-contained as possible. Proper error and response messages are also displayed for various failures.

**Information Disclosure**: Involves the leak or access of information with an individual who is not supposed to have access to it. The permission based system in the blockchain architecture does not allow for anyone in the public to view the blockchain. Digital
signatures are verified before the blockchain can be accessed and then double checked when a block ID is pulled. An adversary with no access to the system would not be able to pass through any layers. The information stored in the blockchain is also not seen as being valuable enough for this attack to occur in the general aviation industry.

**Denial of Service:** A Denial of Service (DoS) attack refers to an adversary flooding the network to disrupt the service of users. Protocols are built into Hyperledger Sawtooth's backend platform to prevent DoS attacks. Such protocols include limiting the number of incoming transactions (can be scaled to fit future needs) and by limiting the size of the payload to a reasonable number of bits.

**Elevation of privilege:** An adversary would want to gain access to systems without having the proper access through a compromised system. The only authority responsible for establishing access roles would be the system administrator responsible for adding actors to their roles. There is no other attack vector other than the system administrator assigning roles into the system for elevation of privilege.

By having used the STRIDE threat model to develop the blockchain, attacker vectors were limited as much as possible. Countermeasures such as dual layer authentication and data integrity were implemented based on the output of the threat model.
Chapter 4

4.0 Development

The development of the blockchain occurred over a twelve-week period and consisted of over 300 hours of work. Initial topic research was conducted to understand the Hyperledger Sawtooth platform and how it could be leveraged for this thesis. Development of the blockchain took the longest amount of time because of the interacting parts of the code. Python was used to connect the REST API, blockchain architecture, blockchain logic, and the environment. Further testing of the blockchain included the validator network, consensus algorithm and REST API. Finally, the simulation was created by adding actors and attributes to display a working blockchain. A further breakdown of the hours spent on the blockchain can be found in Table 3 below.

<table>
<thead>
<tr>
<th>Activity</th>
<th>Topic Research</th>
<th>Concepts</th>
<th>Development</th>
<th>Testing</th>
<th>Simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hours</td>
<td>55</td>
<td>30</td>
<td>150</td>
<td>40</td>
<td>25</td>
</tr>
</tbody>
</table>

Table 3: Blockchain development time log

4.1 Introductory Concepts

Knowledge of the architecture of Hyperledger Sawtooth’s environment is first required to understand how the blockchain was developed on its platform. These concepts include the global state, transactions, validator network, journal, and PoET.

4.1.1 Global State

A Global State is needed to begin the structure of the blockchain to ensure that all data is being displayed at the same time to each node. To do so, Radix Merkle Trees are used to store data in a single instance to ensure that each block has the same information for the
present and is updated in the future. By using the Merkle Trees, the blockchain ensures that the data from the maintenance records are stored securely and efficiently.

The Merkle Tree generates a single root hash which points to the current version (state) of the data. The root hash is then placed on the block header where consensus on the expected version of state along with the consensus of the chain of blocks is given ("PoET 1.0 Specification", 2018). The nodes or leaves of the Merkle Tree are considered one block in the blockchain which represents an addition or change to the maintenance records of an aircraft. Figure 3 below gives a visual representation of the Merkle Hash Tree process.

![Merkle Hash Tree Example](image)

Figure 3: Merkle Hash Tree Example

To change the state of the Merkle Tree, a five-step process must be completed presented in Figure 4. First, a transaction must be submitted via a client through the REST API into the blockchain. Next, the transaction is wrapped in a batch which passes the transaction batch to the validator network. The validators will then ensure that all attributes and
digital signatures passed security and serialization requirements. The transaction batch will then store the data in the leaf nodes to be serialized into a byte array defined as a 35-byte addresses. The new root hash will be recalculated, and the root hash value will update the global state.

Figure 4: Global State Change Process

4.1.2 Transactions

To change the global state of the blockchain, transactions are submitted and temporarily added to the Merkle Tree as a leaf. The transaction is the submitted maintenance record uploaded via the front-end form and submitted to the blockchain through the REST-API client. The transaction contains the digital signatures of the actors, along with the submitted maintenance record data. For a transaction to pass to the validators, it must be wrapped and submitted in a transaction batch.

The transaction batch is a combination of either multiple transactions or as few as one. Batches are treated equally and are submitted for validation in the order they are created. It is important to note that at this stage, transactions may be valid or invalid and could be
denied if all attributes are not completed. If one transaction in a batch fails to validate, the entire batch will fail to commit. The transactions would have to be submitted again after fixing the errors presented in the terminal. The leaf and subsequent change to the top hash is not committed until validation is successful. The below UML Diagram (Figure 5) shows the structure of the batches and transactions.

![UML Diagram of Batches Architecture](From Sawtooth.Hyperledger.org)

Figure 5: Batches architecture (From Sawtooth.Hyperledger.org)

The data from the batch and transactions are serialized into secure headers to sign the submitted batches. Using the digital signature security algorithms discussed in the security section of this paper, the private keys sign the non-serialized transaction and batches. The headers are then verified using the public keys of the actors to ensure the exact bytes match the transaction signature ("Journal – Sawtooth", 2018). The payloads
of the transactions are verified using the payload_512 field which contains a hashed value of the data ("Journal – Sawtooth", 2018). This process uses the SHA-512 hashing algorithm discussed in the security section of this thesis. Batches are an important part of transactions because without them the order of transactions could not be controlled. By using transaction_ids in the batch field, the blockchain can solve the problem by committing the transactions in the order they were submitted.

Transactions families may also be created which group together transactions that are alike in several aspects including attributes and identity. Transaction families can allow for easier use of uploading large batches of Maintenance Records from the same location. For example, a transaction family can be created for a certified repair station and all batches may be submitted at once at the end of a day cycle. By using repeated elements of the attribute list, transactions can be batched together. Figure 6 below shows an example attribute list that could create a transaction family. In this case, the family created would be for the Certified Repair Station ‘Embry-Riddle’.

```javascript
```

Figure 6: Transaction Attribute Example

**4.1.3 Validator**

Transaction validators apply blocks that allow the change of the Global State to occur. After a batch is submitted via a http web client or mobile app, the validators will check if the transactions in the batch are valid. The transaction validator will apply the change if all the attributes in the transaction batches are complete. If any attribute is not included,
the Global State and the Merkle Tree will not update. Figure 7 visualizes the interaction between the HTTP Client and the validators. A new transaction batch will be uploaded via HTTP / Mobile API Client and then validated by the validator to return either an updated global state or invalid transaction to the users’ client. The HTTP Client can be used to check ID’s of approved blocks to the blockchain when the global state updates.

![Diagram of Validator Network Interacting with HTTP Client](image)

**Figure 7: Validator Network Interacting with HTTP Client**

The logic that is used to run the validator in the blockchain are called transaction processors. Each node runs a transaction processor that is used to process incoming transactions submitted by the user clients ("Journal – Sawtooth", 2018). For our blockchain, the transaction processor was modified to take in the aircraft payload, state, and ID. For transactions to operate on the network, nodes are used to interact with the blockchain and update the network’s state. Each organization that is a part of the blockchain runs its own authorized node on their computer to contribute to a consensus or agreements on the blockchain. Communication is essential over the network between validators and incoming transactions which is why Sawtooth’s network was designed to

From Hyperledger.org, each node runs the following three processes:

- **The main validator process**: Works with the Journal to complete transaction handling, block management, consensus, state updates, and the P2P network.
- **The REST service**: Listens for requests on a dedicated and port for transactions from authorized HTTP clients.
- **Transaction processor(s)**: Runs one or more processor to enter logic into the validator and Journal.

Figure 8 below gives a visual representation of the three node processes running together.

A local front-end form will interact with the REST API to push a transaction batch to the validator. After other nodes compete for the process of execution on the P2P network, the winning node will complete the blockchain with the help of the transaction processor logic.
Figure 8: Node Process

After the node approves the transaction batch through the validators transaction processors, the block can now be added to the blockchain. Sawtooth uses a Journal comprised of multiple subprocesses to extend the blockchain.

4.1.4 Journal

The Journal class works with the validators to processes blocks and batches through different “pipelines”. Together, all the subprocesses form a system that can process batches and transactions which are verified through the different dependencies. While the Journal class is flexible enough to run a multitude of consensus algorithm, Proof of Elapsed Time is being used to allow it to be asynchronous. The asynchronous nature of the Journal “allows incoming blocks to be processed in parallel by the ChainController, as well as allowing the BlockPublisher to proceed with claiming blocks even when the incoming block rate is high” (“Journal – Sawtooth”, 2018). The benefits of PoET also
outweighed any other consensus algorithm for this thesis. A visual representation of the Journal pipeline and internal workings is shown below in Figure 9.

![Journal Pipeline Diagram](image)

**Figure 9: Journal Pipeline (Figure modified from Sawtooth.Hyperledger.org)**

The Journal begins with the BlockStore class which is responsible for storing the current blocks on the blockchain all the way to the first genesis block. Multiple classes including the validators rely on the BlockStore class to hold the current record of the blockchain. All blocks in the BlockStore are officially completed and are accessed by the Batch ID, Transaction ID, or by block number ("Journal – Sawtooth", 2018). Errors that can occur in this class have proven non-recoverable and fatal which is why it is imperative that each check class does the proper job to only ensure proper transactions are validated.
The BlockCache stores and keeps track of all validator blocks and their current states. The three states that a validator block can be in are valid, invalid, or unknown ("Journal – Sawtooth", 2018). Each time the blockchain is started, a new cache is created because it is a temporary in-memory construct. Blocks in the cache are not lost and only keep relevant in use blocks for ease of lookup in its memory. For example, a block that is looked up in the BlockStore will be entered into the BlockCache for the current session. After the session, the BlockCache will periodically purge the blocks to keep itself relevant and free of space.

The Completer class ensures that all batches and transactions are valid before they are sent further down the pipeline. It analyzes the block headers and attributes to ensure nothing is left blank that could cause an error in the subsequent block classes. Dependencies are also analyzed, and blocks can timeout if subsequent blocks are not found in the response window.

The BlockPublisher class is responsible for extending and adding to the blockchain after the proper checks have been conducted. The BlockPublisher follows the consensus algorithm that is used in the blockchain, in this case it will be using the PoET consensus algorithm. Each state of the logic flow allows for the validation of the block compared with the consensus algorithm ("PoET 1.0 Specification", 2018). The process begins with the class checking if it has a block that is a candidate for publishing to the blockchain. If it does have a block it will check if it is scheduled and authorized to push the block into the scheduler. The scheduler will then push the batch for verification and validation to be finally added to the head of the blockchain. If a block is not present it will attempt to create one that will later be added to the batch. A visual representation of the
BlockPublisher logic class is found below in Figure 10 taken from the open-sourced documentation of Hyperledger.org.

![Figure 10: BlockPublisher logic flow (From Hyperledger.org)](image)

The ChainController class is responsible for seeing which state the validator is on and applying the necessary state changes. When a block is pushed to the chain controller, it creates a BlockValidator which uses a thread pool for execution ("Journal – Sawtooth", 2018). The BlockValidator is only found in the ChainController class and is responsible for block validation and fork resolution ("Journal – Sawtooth", 2018). The BlockValidator will then determine if it is a valid or invalid block. If the block is valid, it will add it to the chain head and update using the BlockPublisher class in the consensus interface. If the block is invalid, it will not become the chain head and the state will not update. A visual representation of this process is found in Figure 11 taken from the open-sourced documentation of Hyperledger.org.
The final part of the Journal is the consensus class, which is comprised of three interfaces which are the Consensus.BlockVerifier, Consensus.ForkResolver, and Consensus.BlockPublisher. These three classes combined provide for the Proof of Elapsed Time consensus algorithm to run. While different consensus methods can be used, PoET is native to Hyperledger Sawtooth and was chosen for this implementation.

Figure 12 below shows the interface of the consensus class that runs PoET.
The Consensus.BlockPublisher class is only given access to read permission on the global state, what is currently on the BlockStore ("Journal – Sawtooth", 2018). Along with the view permissions, it has permission to push completed batches for publication on the blockchain. This class is also necessary for the creation of the genesis block when the blockchain is first created. The three events that occur Consensus.BlockPublisher are initialize_block, check_publish_block, and finalize_block.

Consensus.BlockVerifier provides Block verification services to the BlockValidator ("Journal – Sawtooth", 2018). This class provides an additional check to the blockchain by ensuring the proposed new block meets consensus publishing rules.

Finally, Consensus.ForkResolver handles any issues related to forks in the blockchain. A fork in any blockchain refers to the change in protocol that all nodes must change to still be valid ("Journal – Sawtooth", 2018). For example, if a critical error in the original blockchain or a substantial software upgrade is made to the system, the old blockchain will not accept block extensions. Instead of extending the older blockchain, a new blockchain is created parallel to the old one to follow the new rules and protocols set forth by the developer. While the old blockchain information will still be there, any transactions submitted to it will be considered invalid and must be published to the new blockchain. While forks occur regularly in a public non-permission based blockchain such as the Ethereum network, we do not see the blockchain presented in this thesis being forked at any time. Nevertheless, the Consensus.ForkResolver is responsible for carrying out the change to the blockchain in the rare case of a fork. A visual representation of a blockchain fork can be found below in Figure 13. The diagram demonstrates blocks that are on the old blockchain that have forked to the new forked protocol blockchain.
4.1.5 Proof of Elapsed Time Consensus Method

As discussed previously, Sawtooth uses a Proof of Elapsed Time (PoET) consensus method ("PoET 1.0 Specification", 2018) that solves the Byzantine Generals problem. The Trusted Execution Environment uses a random lottery mathematical function that chooses an individual peer to execute requests at a given rate ("PoET 1.0 Specification", 2018). The peer that “wins” the right to execute is chosen after individual peers each sample an exponentially distributed random variable and waits for a given amount of time given by the sample ("PoET 1.0 Specification", 2018). The peer with the smallest sample wins the execution. An example of the TEE flow process is shown below ("PoET 1.0 Specification", 2018).

1. Each node on the network requests a wait time from a trusted function which provides confidentiality and integrity.

2. The “leader” or execution validator (node) is chosen by the shortest wait time for a specific transaction block.
3. A function in this case “CreateTimer” issues a timer for a transaction block that is guaranteed to have been created in confidentiality and integrity.

4. Finally, a check function such as “CheckTimer” is used to verify the validator did indeed wait the time that it used to claim the leader role.

The introductory concepts discussed in this section were used to provide basic background on how the elements of the blockchain connected to each other. Aviation Maintenance Records have many requirements that must be achieved through the creation of a ledger based blockchain. The initial research done on the first linked blockchain by Haber and Stornetta led to the creation of Bitcoin which was the first documented paper that brought together multiple algorithmic ideas. From Bitcoin, other technology such as Hyperledger emerged to create a ledger based blockchain that could serve enterprise applications. Finally, the architecture discussed gives a high-level understanding of the elements of the ledger based blockchain. The next section goes more in detail on the technical side of the blockchain and its elements.
4.2 Environment

4.2.1 Development process
Figure 14 below gives a visual representation of the high level blockchain development process. First, we defined our actors and assets that would operate on the blockchain in order to assign roles. After the roles were established, a transaction template was created for the actors to enter and fulfill their roles in the blockchain to acquire all necessary data to be stored in the ledger. Then, after all information was acquired, the logic for how the blockchain would operate was developed parallel to the existing development environment of Hyperledger Sawtooth. A validator network was then created to accept or decline transactions based on developed logic. If the validator network approves the blocks, the global state will update and the blockchain will be extended.

Figure 14: High Level Blockchain Development Process

The required actors in the blockchain are the aircraft owner and aircraft mechanic. The official part supplier information can be omitted if no parts are changed on the aircraft. A scenario that this could occur in, would be scheduled required inspections on an aircraft
in good condition that needs no modifications. The auditing parties have read only
permission only after the Certified Repair Station has made their own audits on the
blockchain. The FAA periodically inspects CRS and Fixed Based Operators (FBO) to
audit their logbooks to search for any abnormalities. As discussed previously, the FAA
takes a serious stance on any fraud or misconducted when it comes to Aviation
Maintenance Records. Being able to provide audibility and transparency is a large part of
this thesis and is achieved in several ways by allowing audits to be made on the
blockchain.

4.2.2 Development

An open sourced copy of Ubuntu version 16.04 is used to install the Python SDK for
Hyperledger Sawtooth. Along with Ubuntu, Python version 3.5 was installed as the
development language for the blockchain. The blockchain and application development
process was completed using the Python programming language, as it is native to
Hyperledger Sawtooth. Other programming languages such as Go, Java, Javascript, and
C++ were considered but did not have the level of stability and support as Python did.

To contain the program and help package the different classes responsible for running the
blockchain the open sourced program Docker Engine and Docker Compose were also
used. The Docker Compose Tool is installed to run multiple applications within one
docker engine instance. An example of the multiple processes being run are the REST
API and Sawtooth blockchain interacting with each other when adding a block to the
blockchain.

To construct the Hyperledger blockchain environment in docker, a sample .yaml file was
modified. The modified file named docker-compose.yaml holds startup settings for the
blockchain which include a validator, REST API, and transactions processors. The interaction of these functions can be seen below in Figure 15.

![Diagram of Docker .yaml functions](image)

**Figure 15: Docker .yaml functions**

The REST API was established on TCP port 8008 which accepts submissions from the Front-End form created using Flask. The transactions are then submitted to the validator which actively listen for incoming messages on TPC port 4004. The validator will then work with the transaction processor logic to either approve or deny the transaction. The settings for the blockchain were left on default to achieve the benefits of the introductory concepts. Each process and the port they were established on can be found in the three figured below (Figures 16-18).

```
rest-api:
  image: hyperledger/sewtooth-rest-api:1.0
  container_name: sawtooth-rest-api-default
  ports:
    - "8008:8008"
  depends_on:
    - validator
```

**Figure 16: REST API in docker-compose.yaml**

```
validator:
  image: hyperledger/sewtooth-validator:1.0
  container_name: sawtooth-validator-default
  ports:
    - "4004:4004"
```

**Figure 17: Validator in docker-compose.yaml**
After the docker environment was set-up and verified to be running, the code for the blockchain was developed in the aircrafts-sawtooth container.

4.2.3 Blockchain Code

The logic behind the blockchain resides in the Aircraft package created in Hyperledger Sawtooth. The classes consist of AircraftPayload, AircraftState, AircraftsTransactionHandler and the main Transaction Handler. A UML diagram of the classes can be found below in Figure 19.

Figure 19: Sawtooth Logic UML Diagram
The logic classes roles and responsibilities are discussed below:

- **AircraftPayload**: The aircraft payload contains information regarding the family specific attributes of the maintenance record. For example, the location of the CRS and the aircraft ID will be included in the payload to show transaction processors that it belongs to a specific transaction family. The payload is hashed using SHA-512 and verified by comparing the payload header and the payload bytes. Deserialization of the payload will occur by the transaction processor that matches the transaction family. Classes that view the payload can only see the payload bytes and not the content inside.

- **AircraftState**: The aircraft state is responsible for storing the hashed headers of the submitted transactions. This class is used to verify the integrity of the files in the later processes. The headers can be used to get_aircraft if the user possesses an aircraft_id.

- **AircraftsTransactionHandler**: The main logic for the transaction processors resides in this class. It is used in conjunction with the main transaction handler class to verify transactions. The data from the front-end form is used to create transaction families. The attributes are then verified to be valid and the transaction is created. From here, the transaction is passed to the second handler class.

- **TransactionHandler**: The second part to the transaction processor, this class is responsible for connecting to the validators located on port 4004. The transaction from the AircraftsTransactionHandler is pushed to the validator which uses the journal subprocesses to add the block to the blockchain. An error will be returned if the validators fail to add the transaction.
In addition to the Aircraft package folder, a dedicated package was created for Flask which enables the front-end form to be created. Flask was chosen to host the local web page because for the scope of this thesis there is no need for a direct web hosted framework. Flask is considered a micro framework because of its limited abilities to not require libraries (Flask, 2018).

The classes of this folder are app.py and sawtooth_client.py. The data is parsed using a JSON data format which interacts with the REST API in the sawtooth_client.py class. The template for the web form is stored in the app.py folder also. Together, these two classes enable the user to enter in a transaction and submit it to the transaction. Due to permission requirements of the blockchain, the private key is unique for each web client and will sign the transaction when it is submitted. All classes and a copy of the virtual machine used to develop the blockchain can be found in the appendix section.
4.3 Simulation

4.3.1 Scenario

Our scenario involves Charlie who is the owner of a 2015 Cessna 172S with aircraft registration number N12345. Charlie operates his aircraft out of a small general aviation airport in San Jose, CA at Reid-Hillview airport. Charlie regularly flies his aircraft and gets inspection done at the onsite repair station of a local Fixed-Based Operator (FBO) named Aero Aviation. While the FBO has only done physical log book entries in the past, a new prototype blockchain application has begun testing at their CRS. Charlie has begun noticing that the door on his Cessna has started to vibrate midflight and wants his long time mechanic Mike at Aero Aviation to inspect the aircraft. Speaking to Mike, Charlie was interested in trying out the new blockchain to make his maintenance record log books more secure.

During the inspection, Mike finds that a bolt on the door needs to be replaced and needs to be ordered from a part supplier. Knowing that the blockchain records all order history, Mike goes with a slightly more expensive but official supplier for the Cessna 172 door bolt. Mike then enters the date, the aircraft make and model, Aircraft identification code, aircraft owner name, mechanic name, authorized repair station name/location, description of maintenance, and the part order history into his front-end form. A copy of the front end form is found below in Figure 21.
At this point, the repair is complete, and Charlie comes to inspect his aircraft. Mike will then click the create button which will digitally sign his mechanic signature to the transaction. After Mike digitally signs and clicks submit, the block transaction will be pushed to the blockchain for verification using the REST-API. Since all entry fields are filled out, and the integrity of the signatures has passed, the maintenance record for the repair has been added to the blockchain.

The events that take place in the blockchain are seen in the snapshot below (Figure 22).
Figure 21: Successful entry of the block into the blockchain

The validators listening on port 4004 picked up a push request from the REST-API. After verifying that the digital signatures are registered and approved to submit on the blockchain the block publisher subprocess in the Journal claims the block. The chain controller in the Journal class then receives the block and starts the block validation. The request is then sent to the handler where it is approved to be added to the blockchain. The chain head is then updated and the blockchain is extended. A block ID number is then presented to Mike and Charlie to easily look up and reference in the future (Figure 23).

Figure 22: List of block ID's
The FAA regularly audits Aero Aviation to ensure that they are compliant with recording aviation maintenance records. The FAA requests permission to view the block ID in which Aero Aviation gives them. The FAA can then view the block and the information that was entered by the mechanic. With all attributes and digital signatures verified, the FAA can give approval to the FBO to continue its operations. The lookup function returns the data in the block in its serialized form. From this block, the auditors will click the link to be directed to the REST-API which will desacralize the data and show the contents of the blockchain. The lookup function returns the information shown in Figure 24 below. JSON String texts are stored on the system administrators’ server which is linked directly to the block ID. When requesting information from a block, a link is provided which redirects the user to the local host page that shows the data in a readable format shown in Figure 25.

Figure 23: Lookup Block ID of completed transaction
### Registered Aircrafts

<table>
<thead>
<tr>
<th>Date</th>
<th>2018-04-03</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aircraft ID</td>
<td>N12345</td>
</tr>
<tr>
<td>Aircraft Make &amp; Model</td>
<td>2015 Cessna 172S</td>
</tr>
<tr>
<td>Aircraft Owner</td>
<td>Charlie</td>
</tr>
<tr>
<td>Mechanic</td>
<td>Mike</td>
</tr>
<tr>
<td>Station</td>
<td>Aero Aviation</td>
</tr>
<tr>
<td>Description</td>
<td>Owner reported aircraft door vibration. Upon inspection, door bolt needed to be replaced. Replace door bolt ordered from Textron. No more vibration.</td>
</tr>
<tr>
<td>Part 1 Supplier</td>
<td>Textron</td>
</tr>
<tr>
<td>Part 1 Tracking ID</td>
<td>A123456789</td>
</tr>
</tbody>
</table>

Figure 24: Deserialized data from the REST-API
Chapter 5

5.0 Results and Recommendations

5.1 Results
This thesis proposed and developed a secure and transparent blockchain that solves multiple issues currently plaguing the aviation industry. All security and transparency issues that this thesis set out to solve were completed along with proving that it is a viable solution for implementation.

Security was a high priority in the blockchain which was why SHA-512 and digital signatures were implemented to allow for multiple checks in the transaction flow. While SHA-512 might have been unnecessary because SHA-256 has yet to be broken, it demonstrates that the blockchain is ready to be scaled for future implementations. Digital signatures that verify the integrity of the submitters of transactions were also important to close any attack vectors found when implementing the STRIDE threat model.

Transparency was achieved by using retrieving blocks from the blockchain to conduct audits. CRS and FBO can audit their own blocks before telling the FAA that they can come and audit their maintenance record logbooks. FAA audits are made easier and are less time consuming when requesting transactions from one location. Part tracking will also cut down on any fraud occurring from the installation of non-official aircraft parts. By entering the order and/or tracking number into the blockchain, auditors can verify that the part was ordered as said by the mechanic.

In addition to security and transparency, all FAA requirements for a maintenance logbook were achieved by the below implementations:
• **Record Continuity**: The blockchain achieves continuity by having a reliable network of nodes that store the global state in Radix Merkle Trees. In the rare event of a complete network crash or attack, the nodes will continue to store the latest global state offline and will be restored when brought online again. Furthermore, the system administrator automatically keeps copies of all global states in case a global state rollback is required.

• **Includes Required Contents**: All entry fields on the digital form were taken directly from a physical maintenance record with additional fields including the part tracking and part supplier. The fields can be modified for future use.

• **Ability to Add New Content**: Being a digital ledger gives the blockchain the ability to continue growing and storing information indefinitely. The head of the blockchain is continuously updated as blocks are added to the system and the global state is updated.

• **Provides Entry for Signatures**: Digital signatures are substituted for physical signatures to provide an extra layer of security for the blockchain. The authentication and integrity of transactions was a high priority and was achieved.

By developing and verifying that all FAA requirements were met, the blockchain developed in this thesis is currently a viable method for storing aircraft maintenance records.

**5.2 Recommendations**

The blockchain developed in this thesis was a prototype to demonstrate that the use case was valid for the technology. The results concluded that blockchain technology is a viable solution to fixing current aircraft maintenance records. However, future
implementation will require the blockchain to be scaled to meet industry needs. Scalability can be achieved through the development of more nodes and a full application. Nodes can be added to the network to increase execution time and take on an increase in transaction network load. For a node to be authorized, the system administrator will need to create an application in which valid actors would apply and be authorized to use the system. A node program will then be installed on their computer to assist in creating a larger node network. A full application can also be developed for iOS and Android devices to facilitate an easier environment for mechanics and FAA officials to complete their responsibilities. For example, a retrieval tool could be created which simply sorts by maintenance facility to easily locate all transactions submitted by that location. The REST API used in this blockchain could also be used in the creation of the future application.

In addition to scalability, other use cases for blockchain technology exist including the below business sectors:

- **ADS-B**: The registration of aircrafts with ADS-B technology could be stored on a blockchain and be verified when needed without the use of servers.

- **UAV Registry**: The FAA could store UAV registration numbers and subsequent data in a blockchain as an alternative to currently storing the information on servers.

- **Airline Tickets**: Airlines could assign tickets a block on a blockchain which could then be resold by either travel agencies or individuals. The tickets would be a tradable asset such as crypto currency and could provide a new product offering for an airline.
- **Airline Rewards**: By developing a crypto currency to operate within the blockchain, airlines could reward their customers with coins instead of loyalty points. The coins could then either be sold on worldwide exchanges or used to trade goods without being constricted to one airline.

The development of this thesis combined technologies such as blockchain, cryptography, coding and networking. Together, these technologies make a Secure Aviation Maintenance Record Blockchain that is a viable alternative to the physical ledger. Blockchain has come a long way, and through the continuous development and use of the technology, it can move many industries into the 21st century.
Bibliography


*Sawtooth.hyperledger.org.* Retrieved 28 March 2018, from

https://sawtooth.hyperledger.org/docs/core/releases/latest/architecture/validator_network.html
