UiA University of Agder

Blockchain for Attribute-Based Access Control

A flexible access control scheme with Permission Delegations

VICTOR LARSEN VIGMOSTAD
TORBJØRN THORVALDSEN LAUEN

SUPERVISORS
Harsha Sandaruwan Gardiyawasam Pussewalage
Vladimir A. Oleshchuk

This Master's Thesis is carried out as a part of the education at the University of Agder and is therefore approved as a part of this education. However, this does not imply that the University answers for the methods that are used or the conclusions that are drawn.

Master's Thesis
University of Agder, 2021
Faculty of Engineering and Science
Department of Information and Communication Technology

Obligatorisk egenerklæring/gruppeerklæring

Den enkelte student er selv ansvarlig for å sette seg inn i hva som er lovlige hjelpemidler, retningslinjer for bruk av disse og regler om kildebruk. Erklæringen skal bevisstgjøre studentene på deres ansvar og hvilke konsekvenser fusk kan medføre. Manglende erklæring fritar ikke studentene fra sitt ansvar.

1.	**Jeg/vi erklærer herved at min/vår besvarelse er mitt/vårt eget arbeid, og at jeg/vi ikke har brukt andre kilder eller har mottatt annen hjelp enn det som er nevnt i besvarelsen.**	Ja
2.	**Jeg/vi erklærer videre at denne besvarelsen:** - ikke har vært brukt til annen eksamen ved annen avdeling/universitet/høgskole innenlands eller utenlands. - ikke refererer til andres arbeid uten at det er oppgitt. - ikke refererer til eget tidligere arbeid uten at det er oppgitt. - har alle referansene oppgitt i litteraturlisten. - ikke er en kopi, duplikat eller avskrift av andres arbeid eller besvarelse.	Ja
3.	**Jeg/vi er kjent med at brudd på ovennevnte er å betrakte som fusk og kan medføre annullering av eksamen og utestengelse fra universiteter og høgskoler i Norge, jf. Universitets- og høgskoleloven §§4-7 og 4-8 og Forskrift om eksamen §§ 31.**	Ja
4.	**Jeg/vi er kjent med at alle innleverte oppgaver kan bli plagiatkontrollert.**	Ja
5.	**Jeg/vi er kjent med at Universitetet i Agder vil behandle alle saker hvor det forligger mistanke om fusk etter høgskolens retningslinjer for behandling av saker om fusk.**	Ja
6.	**Jeg/vi har satt oss inn i regler og retningslinjer i bruk av kilder og referanser på biblioteket sine nettsider.**	Ja
7.	**Vi har i flertall blitt enige om at innsatsen innad i gruppen er merkbart forskjellig og ønsker dermed å vurderes individuelt.** Ordinært vurderes alle deltakere i prosjektet samlet.	Nei

Publiseringsavtale

Fullmakt til elektronisk publisering av oppgaven
Forfatter(ne) har opphavsrett til oppgaven. Det betyr blant annet enerett til å gjøre verket tilgjengelig for allmennheten (Åndsverkloven. §2).

Oppgaver som er unntatt offentlighet eller taushetsbelagt/konfidensiell vil ikke bli publisert.

Jeg/vi gir herved Universitetet i Agder en vederlagsfri rett til å gjøre oppgaven tilgjengelig for elektronisk publisering:	Ja
Er oppgaven båndlagt (konfidensiell)? (Båndleggingsavtale må fylles ut)	Nei
Er oppgaven unntatt offentlighet? (inneholder taushetsbelagt informasjon. Jfr. Offl. §13/Fvl. §13)	Nei

Acknowledgements

This report was written as part of a Master's thesis in IKT523-G at the University of Agder, Grimstad, Norway. This thesis represents the conclusion of the Master's program Cybersecurity at the faculty of Engineering and Science, Department of Information and Communication Technology.

We want to express our thanks to our supervisor Dr. Harsha Sandaruwan Gardiyawasam Pussewalage, and study coordinator Professor Vladimir A. Oleshchuk, whose insight, expertise, and guidance has pushed us to elevate our work to a higher level throughout the Master's program and the project period.

We would also like to acknowledge the various individuals who have taught or otherwise guided us through our educational journey. We would particularly like to single out university lecturer Christian Auby for his ability to motivate and inspire us to pursue an academic career within Computer Engineering and Cybersecurity.

Furthermore, Victor would like to thank Helena Nilsen for her patience, understanding, and always being a supportive second half and Maricken Maria Kittelsen for her wise counsel and motivation throughout the Master's program.

Victor Larsen Vigmostad &
Torbjørn Thorvaldsen Lauen
Grimstad, June 4th 2021

Abstract

Access Control (AC) is a fundamental aspect of modern technology infrastructure. While most primitive versions of AC use an Access Control List (ACL) to represent access, a more versatile variant called Attribute-Based Access Control (ABAC) has quickly become a popular approach to implementing AC due to its flexibility and scalability and enabling cross-domain access.

This thesis proposes a Blockchain-based ABAC scheme with delegatable permission tokens through the use of Hyperledger Fabric and Java Web Tokens (JWT). Fabric is used to store various attribute and revocation transactions necessary in an ABAC system, while JWT's purpose is to delegate access to entities outside the organization domain. Both aspects of the implementation show consistent results while staying within the acceptable performance parameters and reveals the proposed model is a suitable and scalable solution for AC.

Keywords: ***Blockchain, Access Control, ABAC, Hyperledger Fabric, Java Web Token***

List of Figures

List of Tables

List of Listings

Glossary

AA: Attribute Authority
ABAC: Attribute-Based Access Control
AC: Access Control
AP: Access Policy
BCC: Blockchain Cloud
CA: Certificate Authority
ECDSA: Elliptic Curve Digital Signature Algorithm
HMAC: Hash-based Message Authentication Code
JSON: JavaScript Object Notation
JWT: Java Web Token
MSP: Membership Service Provider
NIST: National Institute of Standards and Technology
PAP: Policy Access Point
PDP: Polocy Decision Point
PEP: Policy Enforcement Point
PIP: Policy Information Point
PK: Public Key
RBAC: Role-Based Access Control
RSA: Rivest-Shamir-Adleman
SHA: Secure Hash Algorithm
SK: Secret Key
SSO: Single Sign-On
TAP: Token Administration Point

Contents

Chapter 1

Introduction

Access control (AC) is arguably a fundamental aspect of modern information technology infrastructure, as it allows control over who has access to what. Access control fundamentally consists of two primary components, authentication, and authorization. Authentication is the process of verifying the user and confirming the claimed identity, whereas authorization is the process of granting the authenticated user access with desired privileges.

The most primitive access control method uses an Access Control List (ACL) to represent who has access to what through a table of data. The downside of a standard ACL is that it must be updated for each user's access to a specific object; thereby, the list itself becomes increasingly large with a wide range of users.

Another approach is to assign a role to an individual, such as an admin, and then configuring resources and access based on those roles. This type of AC is known as Role-Based Access Control (RBAC). RBAC is usually the conventional AC method in modern IT solutions because of its ease of use compared to ACLs. However, a downside of RBAC is that despite its ease of use for a single business, sharing resources across domains is difficult as the roles are restricted to a given organization.

A newer type of AC known as Attribute-Based Access Control (ABAC) is quickly becoming the most commonly used AC among businesses today[1]. RBAC and ABAC's primary difference is the utilization of attributes instead of a purely role-based model. In an ABAC model, it is possible to assign attributes to entities (subjects and objects) and use these in a logic-based fashion to allow or deny access. This approach allows for a more varied AC and a much higher amount of potential combinations to manage AC policies. ABAC being a logic-based model dictates that it primarily operates on IF and THEN operators, checking the subject attributes versus any stored Access Policy (AP) on the system.

Another upside of ABAC is that enabling cross-domain access is easier to facilitate in comparison to an RBAC system. An organization could utilize already verified attributes of a user from a sister organization to authorize the user on their systems instead of assigning a permanent role.

In the last few years, there has been significant interest in blockchain and its various corresponding cryptocurrencies, as introduced by Satoshi Nakamoto in his original whitepaper [2]. The blockchain's underlying technology has various other uses than cryptocurrency, such as securely storing and sending data or decentralized marketplaces[3], among other uses. Utilizing a blockchain to store AC information would allow the data to benefit from blockchains underlying properties of tamper-resistance and transparency and be distributed, thus minimizing the risk of data loss from a single compromised server.

Sometimes it is desirable to grant a user access to a system without making permanent additions. This form of granting temporary access is called access delegation, or just delegation, and can take many forms. An example of access delegation is the Linux command "Sudo." A person first must log in using their credentials and then use a delegated system administrator password to be granted temporary access to administrator commands.

1.1 Motivation

One of the main problems with standard RBAC is the multi-domain usage, where each user needs to be assigned a role per domain. ABAC solves this by introducing attributes that can be utilized in multiple domains, allowing for a more flexible access control system.

Blockchain is a technology that is based on immutability, transparency, and distribution. A blockchain is a list of digital transactions stored in blocks distributed in a peer-to-peer network of users and nodes. Each block also contains the hash of the previous block, thereby assuring immutability and tamper-resistance. Finally, because each node contains a copy of the entire ledger, all transactions are permanent and fully transparent for everyone. By taking advantage of a blockchain's underlying properties, as previously mentioned, it is possible to store the various ABAC data as transactions on the blockchain and benefit from the tamper-resistance, transparency, and distribution of a blockchain ledger. Tamper resistance would ensure no non-authorized party can change or otherwise remove essential data of the system, which is often possible in a typical system by attaining administration privileges. Employing blockchain technologies with ABAC would also allow for a higher degree of scalability. The stored attributes and transaction data could be distributed throughout the system, potentially reducing authentication and authorization times.

When operating an access control system, it is not always desirable to give out the exact information required to authenticate. Therefore, many modern systems utilize a form of tokenization, which is substituting sensitive access data with a simple access token. This process creates an additional layer of security for the system to work against potential adversaries. When it comes to access control systems, and more specifically ABAC systems, it is sometimes beneficial to grant temporary access to a client without permanently assigning them attributes or roles in the system.

This process could be done through tokenization, which is why this thesis looks into the use of tokens to delegate access to users who do not own any permanent attributes.

1.2 Thesis Definition

ABAC is based on the premise that a user has a set of stored attributes that can be utilized to authenticate and authorize access, should the attributes satisfy a governing access policy set by the system administrators. As blockchain provides a tamper-resistant chain of data, it could store the attribute and revocation transaction data from an ABAC system. Employing blockchain technologies in such a fashion is very convenient, as anyone can easily verify the blockchain transactions to see if a user satisfies the access policy and has the corresponding attributes.

Most state of the art regarding blockchain-based ABAC systems does not support delegation of access to clients outside the domain [4][5]. The schemes that do propose access delegation lack scalability in their solutions because everything is based on the blockchain [6], which means the verification time of attributes and delegations will increase as the ledger grows, which this thesis attempts to reduce.

The proposed scheme in this thesis will allow for standard authentication and authorization of attributes stored on the blockchain to remain relatively small, and utilizing temporary token access for outside clients, thus not needing additional attributes to be added to the blockchain. The proposed delegatable blockchain-based ABAC system must therefore support;

- A blockchain to store attribute assignment and revocation transaction data from an ABAC system.
- A token scheme to delegate access to users outside the organization's domain.

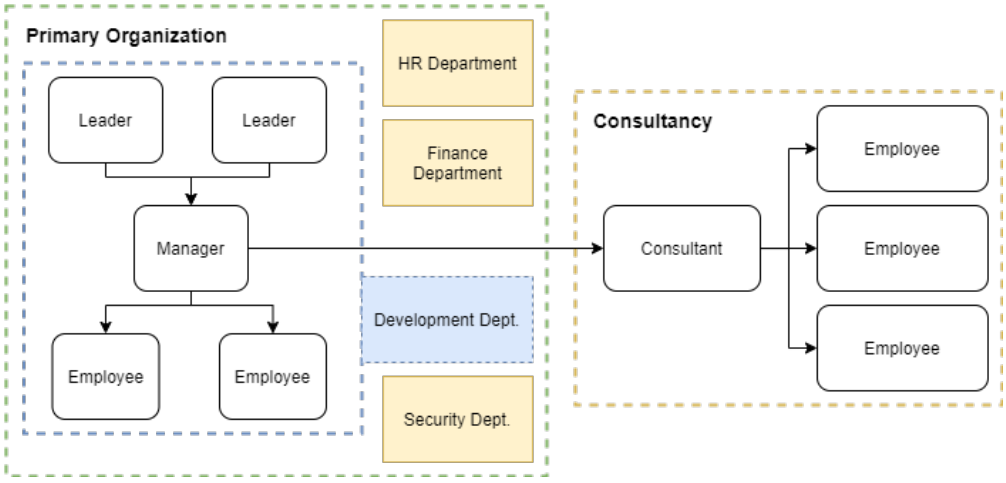


Figure 1.1: Application scenario model.

1.2.1 Case Description

The suggested application scenario for the thesis' proposed scheme is a standardized business setting with various users and roles such as leaders, managers, and employees, as seen in Figure 1.1. The proposed scenario includes anything a regular business would require, such as various departments, people, or data. In order to illustrate the motivation and use of delegatability, the application scenario includes the use of outsourced consultants and their employees. The application scenario is primarily limited to a single organization, even though the proposed scheme could be utilized in a multi-domain scenario.

1.2.2 Research Objectives (RO)

- **RO 1:** *Design a model of the proposed Blockchain-based ABAC scheme with delegatable permission tokens to satisfy the access control requirements specified in the thesis definition.*
- **RO 2:** *Implement the proposed solution through proof-of-concept applications utilizing the designed model.*
- **RO 3:** *Identify the important performance parameters associated with the application*
- **RO 4:** *Obtain the results for the identified performance parameters and thereby provide evidence on the practicality of using Blockchains and delegatable tokens for access control purposes.*

1.2.3 Research Questions (RQ)

- **RQ 1:** *How can we implement a prototype for the Blockchain-based ABAC scheme with permission delegations as proposed in 1.2 ?*
- **RQ 2:** *How can we optimize the implementation so that it is possible to achieve higher levels of performance ?*
- **RQ 3:** *Is Blockchain-based access control with permission delegation suitable and realistic for the application scenario specified in the thesis definition?*

1.3 Scope

The proposed scheme will be developed through a prototype utilizing the pre-existing blockchain test network Hyperledger Fabric. The prototype will be populated with various data such as attribute assignments and attribute revocation transactions, to accurately analyze the proposed scheme's performance. The scheme will also support one-time-use delegatable access tokens that the client can generate locally in batches without centralized systems. As the tokens will be single-use only, the proposed scheme will not support token revocation. Instead, it will limit their use

based on time as a specific token is only valid through a time duration that the client specifies when generating the token.

As the thesis focuses on the proposed scheme's potential scalability and performance aspects, an ABAC system will not be implemented. Instead, a blockchain test network will be utilized to analyze the performance variables for adding, checking, and verifying data on the blockchain. Similarly, a simple token prototype will be implemented to check its potential performance impact when generating and verifying access. The token side of the prototype will not be directly linked or otherwise connected to the blockchain. However, it will instead serve to analyze the performance impact of a delegatable token system. This token performance will be combined with the performance data gathered from the prototype's blockchain part into one complete performance impact analysis.

1.3.1 Assumptions

When designing the proposed Blockchain-based ABAC scheme, a primary assumption is that the delegation tokens will have an expiration time lower than 24 hours. This low expiration time is to combat potential misuse of delegated tokens, as, with our proposed scheme, there is no way of recalling or marking a token as invalid. There are, however, ways of denying access to a delegated token, which will be further discussed later in the report.

1.3.2 Limitations

The global COVID-19 pandemic put some limitations on the project's collaborative aspect as it was not possible to have access to the university campus for most of the project period. Therefore, most communications, development, and other implementation have been limited to the available local hardware at home and various internet-based communications platforms such as Discord and Zoom.

When working with potentially big datasets, there is a need for a large amount of computational power. As a result of the various lack of resources for the project due to Limitation 1, the implementation of the prototype has been limited in various ways;

1. The proposed scheme and implementation will be limited to a single attribute authority, which a single local machine could represent.

2. The proposed scheme will also be primarily limited for use in a single organization instead of a multi-organizational collaborative system.

3. With the delegation tokens being single-use only with a low expiration time, a user may need to store significant amounts of tokens locally, depending on the need.

1.4 Thesis Outline

After Chapter 1, the thesis is presented in the following way, each chapter detailing an aspect of the project, and building on previous related sections:

- **Chapter 2 Theoretical Background:** Relevant background knowledge required to understand the proposed design, implementation, and results.
- **Chapter 3 State of the Art:** Goes through the various related work and previous contributions to the related fields.
- **Chapter 4 Requirements & Design:** Detailed analysis of the proposed scheme and its requirements.
- **Chapter 5 Solution & Implementation:** In-depth walkthrough of the implemented prototype and its various functions.
- **Chapter 6 Testing & Results:** States various performance parameters the implementation must uphold, showcases the results and security analysis of the proposed implementation, and answer the thesis's research questions.
- **Chapter 7 Conclusion & Future Work:** Concludes the findings and details several potential future work additions, changes, or implementations to the proposed scheme.

Chapter 2

Theoretical Background

Understanding how ABAC functions and how blockchain operates is vital to understanding the blockchain-based ABAC scheme proposed in this thesis. This chapter conveys the required background knowledge in the three fields of ABAC, blockchain, and tokenization to understand the proposed scheme and its implementation. The chapter starts with an overview of AC and ABAC, followed by a deeper dive into blockchain and its various subchapters, describing some underlying technologies that the thesis will utilize. Finally, the chapter finishes with tokenization, how tokens usually function, and how they are typically used.

2.1 Attribute Based Access Control

Access Control is primarily used for authentication and authorization of users and systems to verify who they are and whether the system should grant access. An RBAC system was the most common form of AC in the past. A new form of AC has emerged in recent years, utilizing attributes instead of authentication and authorization of roles.

Standard role-based access control, or RBAC, works by verifying the user's role and comparing it to any access policies for the object the user is attempting to access. Figure 2.1 shows a simplified version of RBAC where the user gets its role, sends it to the policy checkpoint to verify access, and is granted or denied access based on the policy decision. The process would be similar in an ABAC system, but instead of assigning a single role, the user would be granted multiple relevant attributes for their position that the system could use, resulting in a more flexible approach than what RBAC would allow.

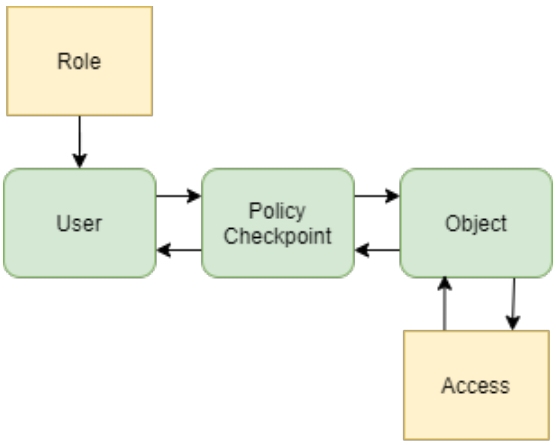


Figure 2.1: Basic Access Control

ABAC is a form of AC where each user (subject) is assigned a set of attributes that can later be verified by an embedded device or system (object) to grant access. This use of attributes opens up a plethora of potential combinations for verification. An example would be instead of only authorizing a user because they have the correct role, you could use ABAC to authorize them based on their attributes such as; position, length of tenure, security clearance, or other outside (environment) attributes such as a building lockdown or working hours.

An example would be an employee (subject) who has been given some attributes upon being hired, either manually by an administrator or automatically by a system. Whenever the employee wants to access a resource such as a software project (object), various attributes from both the subject and potential environmental attributes such as time and location could be utilized for authorization purposes within an ABAC model. Another example directly comparing the role-based model to the attribute-based model could be where a company employee with a particular clearance level would always be allowed access to a specific resource in an RBAC model. In contrast, in ABAC, you could pose further restrictions such that the employee could only be authorized during the day or other such factors. ABAC also makes it possible to perform business-wide rules at a moment's notice if necessary, such as giving all sensitive information new attributes that only allow high-level clearance or locking down a section of the build by merely updating the access control attributes for the entrance points.

2.1.1 The ABAC Model

While the fundamental idea has been established, there are many more aspects to an ABAC model to consider. Of course, an ABAC model's essential building block is the attributes and its various logical policies. Each access policy consists of various IF-THEN statements that stipulate that if a user has attributes X, Y, and Z, give access to the requested resource. It is also possible to require that a user has a set of attributes, but missing others, such that only if the user has A and B but lacks C authorize the access. When discussing the ABAC model as shown in Figure 2.2, a user is often replaced by "subject," and the requested resource is denoted by the term "object." The following explanation of the model utilizes the word "point" to represent various aspects of a system, such as a server or a specific application.

While ABAC systems' implementation may vary, the underlying scheme is a standardized model proposed by the National Institute for Standards and Technology (NIST) [7], from which most of the descriptions and explanations in this chapter originates.

Authorization Services

The first part of the model to interact with the users' access request is the Policy Enforcement Point (PEP). The PEP is where the decision is enforced, meaning that this is where the subject is

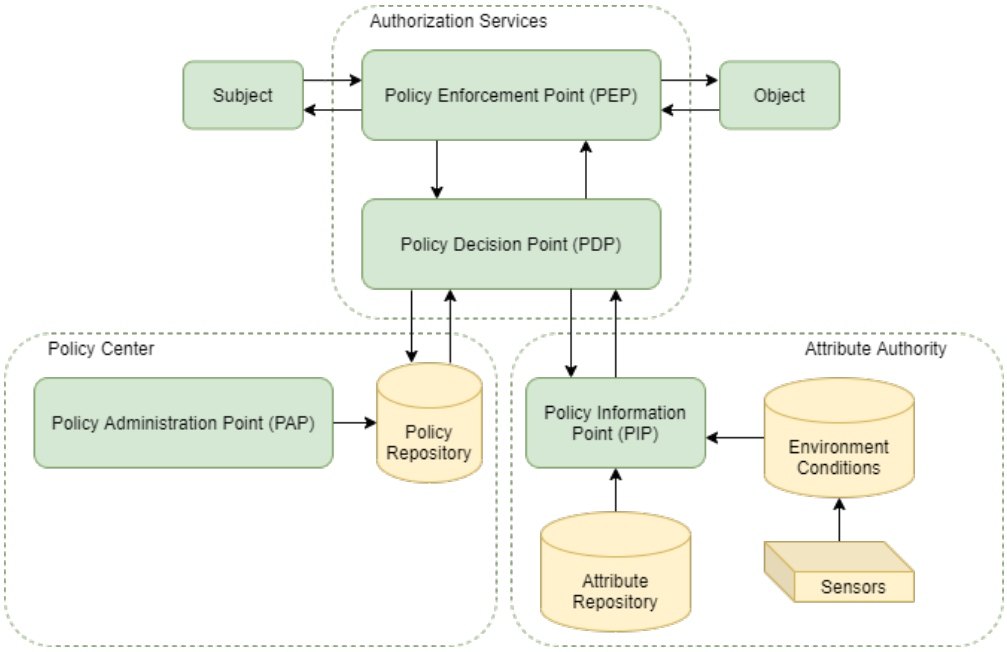


Figure 2.2: The ABAC Framework

granted or denied access. Before making a decision, PEP forwards the access request to the Policy Decision Point (PDP). The PDP can be considered as the core of the ABAC model, as it is the point that communicates with the rest of the components to gather information and ultimately make the decision. When PDP receives an access request from PEP, it will identify which object the user attempts to access and forward a request to the Policy Center for the appropriate access policies.

Policy Center

An AC underlying ruleset for granting access comes in the form of a set of access policies. Within an ABAC model, these policies are often stored in a Policy Repository, that depending on the implementation, the PDP may access directly. Adding, modifying, or otherwise editing existing access policies goes through the Policy Administration Point (PAP) within the policy center, which provides a user interface for managing policies.

Attribute Authority

The main component of the ABAC model is the Attribute Authority (AA). This point is where the various attributes available to an organization are decided upon, granted to users, or otherwise handled. The AA primarily consists of the Policy Information Point (PIP), an Attribute Repository, and various outside Environment Conditions and sensors.

The Policy Information Point (PIP) serves as the retrieval point for attributes stored within the system and other relevant data the Policy Decision Point (PDP) requires to make decisions. Within a standard ABAC model, the attributes are also stored within an Attribute Repository that the PIP will use to store, modify or add new attributes.

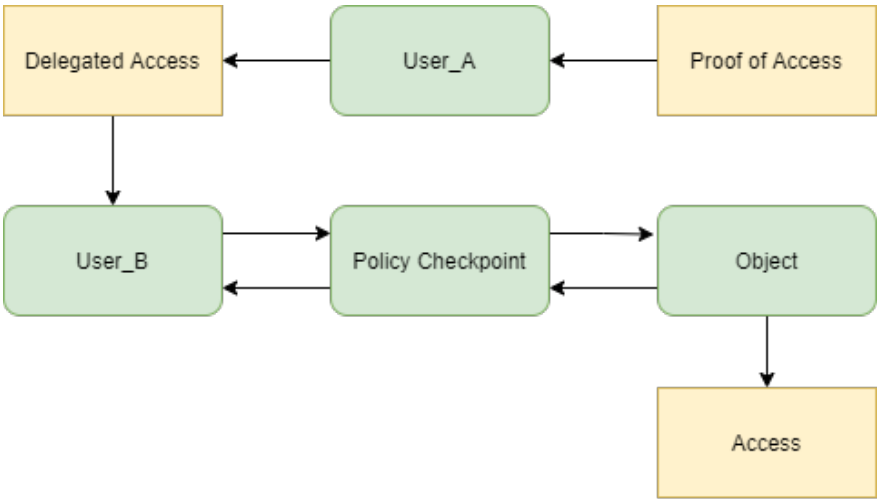


Figure 2.3: Basic Delegation of Access

Within an ABAC model, there is also the possibility to utilize external factors as attributes. An example of such environmental conditions could be the time of day or day of the week. These environmental attributes are often dynamic within a system, as they can either be obtained by sensors or manual admittance from an administrator.

2.1.2 ABAC With Delegatability

Oftentimes it is desirable to grant access to a user without permanently adding them to the system via roles or attributes. An already authenticated user can share or delegate their access to another user in such a circumstance, thereby granting them temporary access to the system. Figure 2.3 shows the basic idea of delegation where User_A delegates access to User_B, to which User_B can authenticate with the system as usual. This delegation method is the premise of the model scheme proposed in this thesis but was first introduced in [6].

2.2 Blockchain Technologies

Blockchain is a technology that allows for a digital decentralized distributed ledger of data stored in a list of records or blocks transmitted through a peer-to-peer (P2P) network of nodes. Each block contains various information such as the previous block's hash, a timestamp, and the specific data stored within the block, typically digital transaction data. At its core, blockchain technology is based upon three idea pillars; immutability or tamper resistance, decentralization, and anonymity.

The first pillar of blockchain is immutability. Because each block in the chain contains the previous block's hash, if any of the earlier blocks were to be tampered with or changed, it would have a ripple effect through the chain and not add up when verifying each block of the chain. This way of constructing the block means that once something has entered the chain, it cannot be changed, altered, or tampered with, resulting in an immutable or tamper-proof chain of blocks.

The subsequent basic idea of blockchain is anonymity. To achieve the anonymity of blockchain, cryptographic public keys are most commonly utilized as identifiers for performing various transactions and operations on the blockchain. Public key cryptography is based on the idea that a user computes two keys known as a key-pair, which are mathematically linked to each other, of which the private key could be used to sign various transactions. The private key could be used for authenticating the corresponding public key with the private key.

Furthermore is the idea of the decentralization of the blockchain. There is usually a trusted authority (TA) in most information technology systems, such as a company server, that performs various tasks such as authorization and authentication. The idea behind blockchain is that each ledger, or blockchain, is distributed in a decentralized peer-to-peer (P2P) network. Each user or node in the network contains a copy of the ledger, thus removing the need for a trusted centralized authority.

However, decentralization is not required for a blockchain. A private, centralized version would still use the same basic building blocks while maintaining immutability and transparency at its core. Utilizing the userbase workforce helps divert resources but does lead to problems with power balance in the system allowing for 51% attacks. In a scenario where users have the power, one user or group of users could seize the majority of the market and monopolize the benefits.

With these three pillars implemented in a blockchain, we are left with a fully transparent, immutable, distributed ledger that can be implemented and utilized in various technologies, not just cryptocurrency.

2.2.1 Smart Contracts

Smart contracts were first introduced in [8] to present a digital form of a legal agreement to hold participants accountable for their mutually agreed-upon obligations. However, due to computing limitations, the technology was not properly utilized and implemented until the emergence of blockchain technologies. With smart contracts often considered a crucial part of a blockchain [9] it has developed since its original implementation with Satoshis Bitcoin, where it served as a simplistic way of performing basic arithmetic, encryption, and logic operations [2]. Smart contracts have since been improved upon with their implementation in various other cryptocurrencies. Ethereum is one of the most significant contributors to its development, creating a Turing-complete scripting language for smart contracts called Solidity [10].

Generally, a blockchain smart contract can be described as a set of processes executed according to rules written in a specific programming language or script [9]. Figure 2.4 shows the smart contract model and how it may exist on the blockchain, consisting of various states, rules, and triggers for executing the instructions. Smart contracts are the basis of adding or modifying

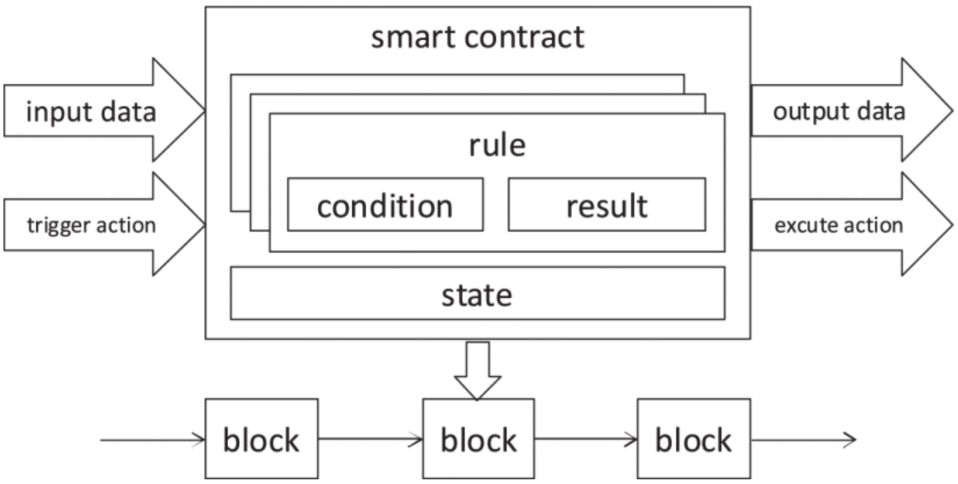


Figure 2.4: Smart Contracts on the blockchain [11]

transactions in the proposed blockchain model in the thesis, of which the implementation will be further explained in Section 2.3.

2.2.2 Blockchain with ABAC

One of the challenges in AC schemes is storing data related to access policies and users, and how to avoid data tampering. In an ABAC system, this issue arises in how to store attribute ownership; simply creating a database has its issues with tampering and could become a single point of failure, and keeping attributes locally on the users' computer has problems with falsifying the attributes. However, storing attribute ownership on a blockchain would utilize the immutability of the blockchain to its advantage, creating a system that is much more resistant to tampering.

2.3 Hyperledger Fabric

Creating a private blockchain is simple and can be done by anyone. However, there are dangers with self-made blockchain solutions, such as utilizing a secure hashing algorithm or spending unnecessary time trying to fix minor bugs. A library helps streamline the work and reduce the chance for mistakes throughout the process of creating a blockchain.

Hyperledger Fabric is a well-documented, open-source library that allows its userbase to create blockchains, peers, and smart contracts and publish them into production. Hyperledger Fabric is made more accessible through its extensive tutorial that shows the ropes of how to set up and create each component, explaining the steps along the way. One of the key differences in Fabric from prominent blockchains is its ledger system.

2.3.1 Ledger System

Blockchains are ledger systems that store various transactions and changes. Hyperledger Fabric uses a ledger system that comprises two parts; the blockchain and the World State, as shown in Figure 2.5.

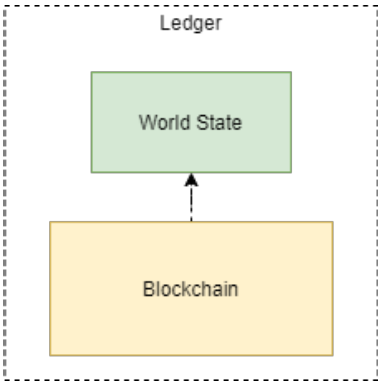


Figure 2.5: Ledger-system for Fabric

The blockchain is where we put all our transactions like normal, but since we cannot remove a block when we undo the transaction it contains, we end up with a long chain that contains invalid information. The World State takes all the transactions on the blockchain and sums their effects into one readable database. For example, a user who has an attribute granted and revoked repeatedly would appear many times in a blockchain but would only have a single entry in the World State saying whether they have the attribute or not.

2.3.2 Network

The system that Hyperledger Fabric employs builds upon a peer network, where each peer is a connection point with a copy of the ledger that can be accessed and maintained by the users. Peers are connected through channels that allow them to communicate openly such that transaction data is not hidden from view by other parts of the network. In this way, the consensus regarding transactions is kept from being falsified by having all communication public. The ledger is deployed on a channel, not on the peers themselves, which allows a peer to keep multiple different ledgers for different channels and participate in each of them without compromising the integrity of a distributed ledger. Figure 2.6 shows an example of a set of channels that share some peers.

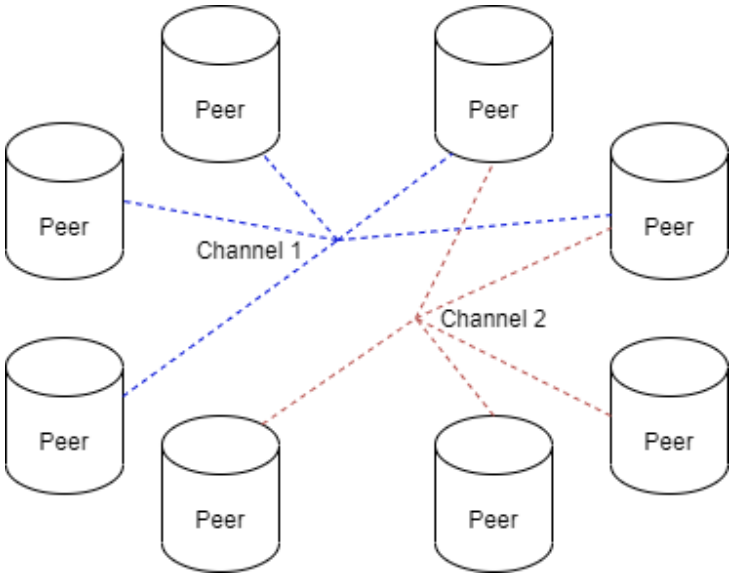


Figure 2.6: Channels and peers

2.3.3 Chaincode

A smart contract is a type of code utilized often in blockchain technologies today that allows an automated process for fulfilling deals and transactions within the confines of immutability. Two companies that want to trade assets, for example, can utilize a smart contract to make sure the other company does not deviate from the agreement.

Transactions are stored in a channel-specific ledger, but they are created through smart contracts. Smart Contracts within Hyperledger Fabric are referred to as chaincode in the documentation and are a set of functions installed on peers and deployed on a channel. Chaincode is written in one of the three coding languages: Go, Java, or JavaScript (using Node). Chaincode is used to create assets, delete them, edit, and query the world state for existing assets. Creating, deleting, or editing an asset through chaincode will generate a transaction to be put on the blockchain. A successful validation from the system is required to commit a transaction, which is done through endorsement policies. Endorsement policies are rules for what endorsements a transaction needs to be accepted; they can take the form of needing a majority of peers or only one specific peer to validate and allow a transaction to be added to the blockchain.

2.3.4 Membership Service Provider

Attributes are assigned by using the chaincode, but a random user should not be allowed to assign attributes to users, which is why invoking chaincode can only be done by those who have a certificate from a Certificate Authority (CA). The Membership Service Provider (MSP) is a component within Hyperledger Fabric that determines the validity of certificates as a proof of identity within the system. The type of certificates that the MSP accepts can be customized; the default types in Fabric are clients, admins, peers, and orderers. A client may be denied in an attempt to delete an asset on the world state but is perfectly capable of querying the world state for existing ones.

2.3.5 Transactions

Creating an asset using the chaincode will generate a transaction for the blockchain detailing the change it will make on the world state. By default, the transactions contain various information, such as a header, signature, proposal, response, and endorsements, shown in Figure 2.7. The header contains metadata like the name of the chaincode used and the version of the world state. Version is an incremental value of how many changes have been made to the world state and used internally to ensure version history is correct. The proposal is the action that the client wants to take, along with the specific information related to it, for example, create an asset belonging to Tom, which will automatically generate a header. The client signs the proposal and shares it on the channel so that the peers receive it. Each peer then validates the information, creates their response and endorsement, and returns them to the user. The response is a "write-set," a functional change on the world state that directs the peer to resolve the new information. 'Endorsements' is a list of signed transaction responses from various peers needed to fulfill the endorsement policy mentioned earlier. Finally, the header is created in conjunction with the block itself by the ordering service.

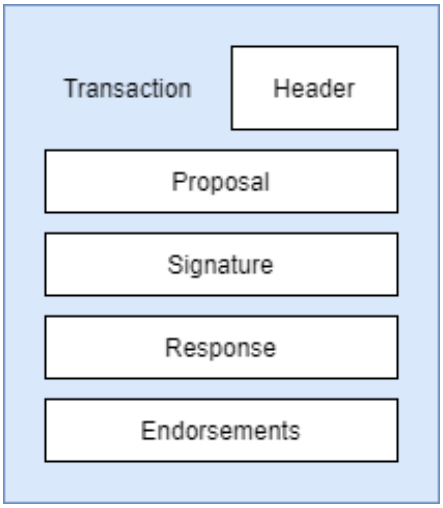


Figure 2.7: Fabric Transaction

2.3.6 Ordering Service

Hyperledgert Fabric uses an ordering service component to handle the block creation and appending it to the blockchain, which receives all the transactions that the peers have processed and gathers them into a block. Fabric does this differently from well-known blockchains in that it does not generate blocks continuously; instead, it generates blocks only when transactions are submitted. Then, since Fabric does not generate a block every 10 seconds, or even 10 minutes, a block needs to end at some other point. Fabric uses transaction count, byte size, and a timer to determine when a block is finalized.

Starting when the ordering service receives a transaction, the timer determines how long it will wait for additional transactions to add to the next block. Should the number of transactions exceed the transaction count, it will preemptively stop the timer and finish creating and commit the block to the blockchain. Like with transaction count, byte size will also stop the timer and create the block early if the size of the block becomes too large. Finally, only when the block is finished and appended to the blockchain is the world state updated with the proposed additions, removals, or edits of assets.

2.3.7 Block

A finalized block is made up of a few different sections; first, it has a block header. The block header contains three parameters: Block number, Current block hash, and previous block header hash. Block number is simply an integer that incrementally increases upward according to how many blocks have been created and indicates where in the blockchain the block is located. "Current block hash" is the hash of this block's transactions, and "previous block header hash" is the "current block hash" from the previous block in the chain. These block hashes are what make the blockchain immutable.

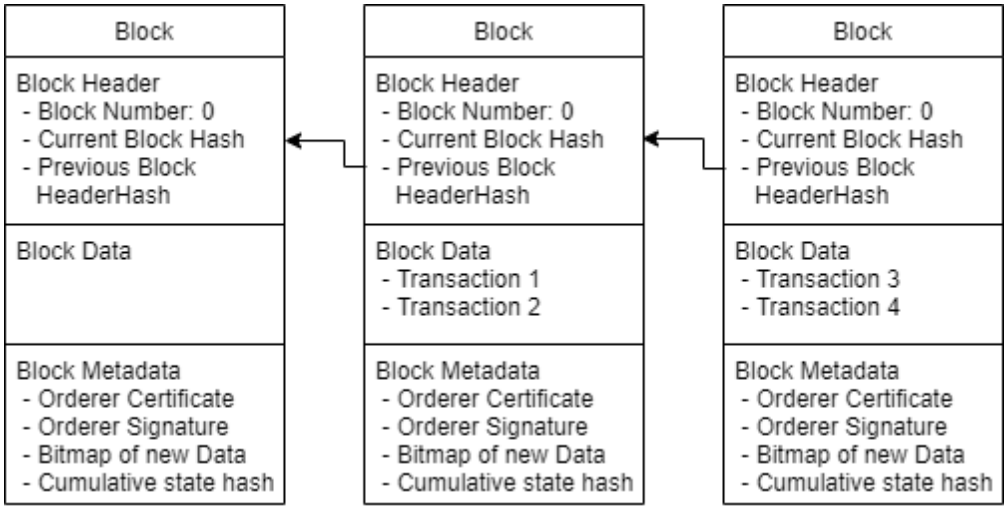


Figure 2.8: Simple Blockchain in Hyperledger Fabric

Secondly, a block contains block data, the transactions themselves. As previously mentioned, a block can only have so many transactions in it, and here is where that takes its secondary effect of limiting the size of the block itself.

The last part is the block metadata, which contains information about the ordering service in the form of its certificate and signature. It also contains a bitmap that indicates whether each transaction is valid or invalid, and a hash of the cumulative state updates for all blocks. The cumulative state update hash provides a way to detect state forks, where there exist multiple world states that are different from each other. Figure 2.8 displays a small example of a blockchain with the specified parameters.

2.3.8 Docker

Docker is a tool used to create virtual containers that are segregated from each other, allowing testing between separate systems to simulate a real system with multiple computers and servers. Hyperledger Fabric's tutorial network uses Docker to run on entire blockchain clouds on local machines, allowing the user to experiment outside of public test networks.

2.4 Tokenization

Tokens are not exclusive to information technology and have often been used to refer to various objects that serve different purposes. A simple version of a token may be a medallion someone can provide to prove the person in charge has authorized them for a set of actions, or a more modern version could be a bus ticket proving you have paid for the fare. In information technology, tokens see several different uses, all from token rings to session tokens that serve various purposes within a system. These multiple uses have in common that they represent some evidence of rights or authorized access to actions or systems.

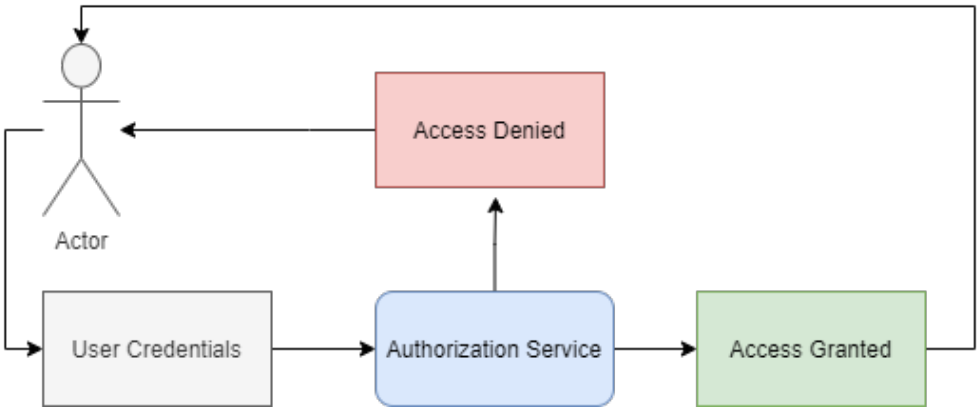


Figure 2.9: Simplified standard access model

Within data security, substituting sensitive data with different data is often a utilized tactic to reduce the impact of data leakage. The act of replacing data is similar to how tokenization works and is what this thesis suggests as a potential solution to the proposed problem. Instead of always using users' login credentials, a system could generate an access token based on successful authentication and distribute the access token for further authorization needs for a limited time. Suppose the issuing token authority is part of a more extensive system. In that case, the user could potentially utilize the access token in various other linked systems within the business instead of their login credentials, often called a Single Sign-On (SSO).

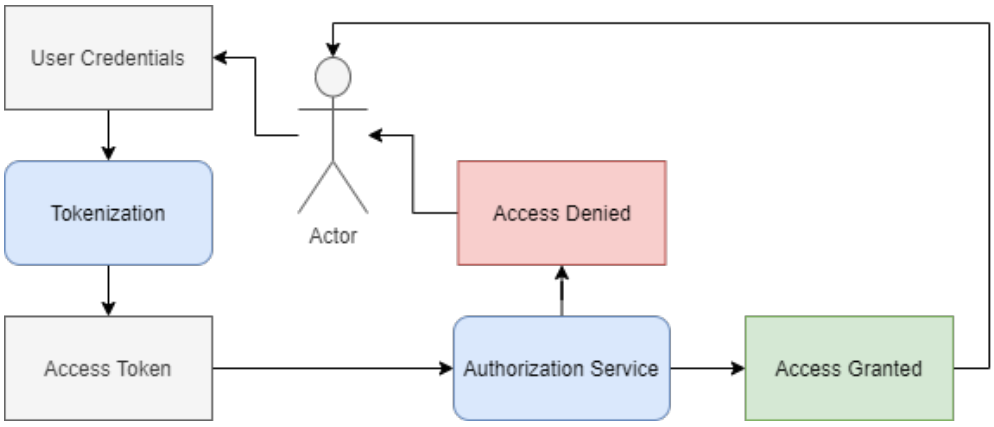


Figure 2.10: Simplified tokenization model

Figure 2.9 shows a simple access model where users send their credentials to an authorization service and are either granted or denied access to the resource. This way of operation is the standard for most login functionality and is also the first part of a tokenization process. In order to utilize an access token, the user must first go through the standard authentication procedure depicted in Figure 2.9, but instead of being granted direct access to a resource, is given an access token representing the user's rights towards that specific resource. The token's validity is often based on the issuing authority's signature and includes a timestamp to combat replay attacks. The user can then apply the newly received access token in place of standard credentials for authenticating, as shown in Figure 2.10.

2.4.1 Java Web Token

The proposed solution presented in this thesis utilizes tokenization and tokens in the form of Java Web Tokens, using various libraries and the JavaScript programming language to implement the token prototype. A Java Web Token (JWT) is a compact way of transferring data between two parties, as its encoded form is simply a string of variable length based on the contents of the JWT[12]. The most common usage of a JWT comes in the form of the previously described access token. When a user authenticates with a website, it is typical for each subsequent request within the session window to be through a JWT. This form of token usage is often called SSO and is a widely used feature today. Another use case for the JWT is a simple data transfer between two parties. It can be securely transmitted in its encoded form and natively supports integrity checks through HMAC.

The JWT was first introduced in 2010 as a simple signing mechanism for JavaScript Object Notation (JSON) [13] but has since been revised, resulting in the proposed standard RFC 7519 [12]. The token is represented as a sequence of URL-safe data parameters separated by a period character, where each section of data contains base64 encoded values. The token consists of a header holding the signing algorithm and type of token, a payload with the JSON object, and a hash-based message authentication code (HMAC) for integrity.

The payload of a JWT holds various custom data and commonly contains predefined parameters to provide valuable information. Some of these predefined parameters are issuer (iss), expiration time (exp), and subject (sub). Listing 1 shows an example payload where each parameter is specified in standard JSON format. Once the data has been structured according to the JWT model format, the header and payload are encoded using the Base64Url format to form the first and second part of the JWT.

The last part of the Java Web Token holds the HMAC signature that serves to ensure message integrity. The HMAC is calculated based on the algorithm specified in the header, usually Secure Hash Algorithm 256-bit (SHA256) or SHA512, the two Base64Url encoded header and payload parts, and a private key. Listing 1 shows a simple code example of the HMAC signature of a JWT.

```
{
    "alg": "HS256",
    "typ": "JWT"
}
{
    "sub": "1234567890",
    "name": "John Doe",
    "admin": true
}
HMACSHA256(
    base64UrlEncode(header) + "." +
    base64UrlEncode(payload),
    secret)
```

Listing 1: Example Java Web Token (JWT)

The algorithm specified in the header example in Listing 1, "HS256" means the HMAC should be calculated using SHA-256. Other supported algorithms include the Rivest–Shamir–Adleman algorithm (RSA) and the Elliptic Curve Digital Signature Algorithm (ECDSA).

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.
dyt0CoTl4WoVjAHI9Q_CwSKhl6d_9rhM3NrXuJttkao
```

Listing 2: Encoded JWT from Listing 1

Finally, the encoded header, payload, and signature are put together into a string separated by a period character to form the encoded JWT, as shown in Listing 2. The token is designed to be lightweight and secure and can then be quickly passed in an HTTP environment.

Chapter 3

State of the Art

Briefly summarized, the thesis definition defines several requirements of a proposed scheme that combines the utility of an ABAC system, the immutability, transparency, and tamper-resistance of the blockchain, and the flexibility of a delegatable permission-based token system. This chapter showcases previous work in the various fields and how they relate to the thesis definition and the proposed scheme, as much of the work in this thesis has been inspired by previous work in the field.

One of the primary technology schemes used in this thesis is ABAC. It makes sense to mention and introduce the NIST special publication 800-162[14]. This special publication details various definitions and methodologies for implementing and using an ABAC system, many of which have been detailed in Section 2.1.1. Papers such as [6][15] make use of the ABAC model in their research to create a modern AC system.

A problem stated in this thesis' definition is to design a system to allow for access delegation for its users. While simultaneously utilizing ABAC and implementing cryptographic multi-authority delegatable attribute tokens [15], manages to elegantly design a proposed solution that proves attribute tokens are unforgeable under chosen-attribute and chosen-nonce attacks. This thesis, however, utilizes a version of permission tokens closer to that of [11], in which they store the access tokens on the blockchain by designing a generalized token structure with a standard AC scheme.

One of the essential parts of the thesis' related work comes in the form of blockchain technologies. Satoshi Nakamoto first introduced blockchain in his white paper [2], presenting his version of a peer-to-peer electronic currency system. Nakamoto's white paper has since been the foundation for many new technologies and research on utilizing this unique and fascinating data storing method.

One of the many uses of blockchain technologies has come in addition to pre-existing technologies such as access control. One of the problems posed by this thesis is how to combine ABAC and blockchain. It serves well to look at previous work of research that has integrated access control and blockchain successfully.

One such contribution is [4], where the authors propose a solution for storing the access policies on the blockchain, thereby making them visible and accessible to all users. This solution allows for a distributed auditability and transparency, which this thesis uses as a fundamental idea for designing the blockchain aspect of the proposed scheme. In addition to the previously mentioned paper, there is [16] that combines blockchain and access control that specifies a more enterprise-focused application, making it a highly relevant paper for this thesis, as it closely follows the scenario in which this thesis' proposed scheme will be designed. Lastly, two potential papers utilize the standardized tool Hyperledger Fabric to implement their blockchain solutions for AC [17][5], focusing on using the blockchain to serve as a transparent way of storing access policies.

In [18], the authors introduce a way to delegate access through an AC system utilizing the blockchain. Their way of delegating access goes through either events or queries, in which case a third party could request access and be granted. This approach is similar to how this thesis plans on implementing delegations but goes against one of the design requirements; a user should be capable of delegating access without accessing a centralized system.

The authors of [6] propose a different way of delegating access in their paper. They suggest using the blockchain to store delegated access and attributes used in an ABAC scheme on the blockchain. This approach has been the baseline for implementing and designing this thesis' proposed scheme for assigning access to users, namely by storing their attributes on the blockchain. The downside of storing both attribute assignments and delegation transactions on the blockchain this way is the increasing performance requirements as the blockchain gets longer. The principle of a delegation chain and further delegating access are also introduced in [6], which can be a source of performance difficulty when stored on the blockchain, as the application must check each delegation transaction on the chain. This problem is what this thesis attempts to solve with delegated permission tokens.

Chapter 4

Requirements & Design

The thesis definition details various requirements the proposed scheme must fulfill. The following chapter will detail the multiple models and approaches the thesis has taken to answer Research Question 1 (RQ1) and fulfill the Research Objective 1 (RO1), which are repeated below:

RO 1: *Design a model of the proposed Blockchain-based ABAC scheme with delegatable permission tokens to satisfy the access control requirements specified in the thesis definition.*

RQ 1: *How can we implement a prototype for the Blockchain-based ABAC scheme with permission delegations proposed in 1.2 ?*

The first part of the chapter goes into the blockchain model and proposes storing, utilizing, and otherwise managing attributes on the blockchain. While the second part details the proposed tokenization model and its various verification and generation aspects and how it fits into the larger proposed scheme.

Since the scope of the thesis encompasses two proposed prototypes and models, the attribute storing blockchain and the delegation tokens, there is not a proposed ABAC model scheme as this is outside the scope of the thesis. Instead, the standard ABAC model will be referenced throughout the chapter and built upon to add the blockchain and token functionality to the standard model.

4.1 Blockchain

The upcoming proposed blockchain-based ABAC scheme will primarily be used to store transactions related to the attributes used in an ABAC scheme. Detailed in Section 1.2 Thesis Definition, there are requirements for the blockchain model and its prototype to fulfill Research Objective 1 (RO1), as shown in Section 1.2.2:

"A blockchain to store attribute assignment and revocation transaction data from an ABAC system."

From this thesis requirement, we can extrapolate some more specific requirements to explain the implemented solution. Thus the blockchain implementation must support the following:

1. A blockchain to store attribute-related transactions.
2. Attribute assignment transactions to assign one or more attributes to a user.
3. Revocation transaction to remove or otherwise flag one or more transactions as no longer valid.
4. The scheme should accurately verify whether someone has the attributes required when granting access.

The proposed blockchain-based ABAC system takes inspiration from the previous work done by Pussewalage and Oleschuk [6] and thereby carries similarities to their scheme. The following few parts will detail and explain the motivation and proposed solution to each of these requirements and how they fit into a grander blockchain-based ABAC solution with delegatable permission tokens.

As with any blockchain application, there are several potential choices for which type of blockchain to utilize, each with its advantages and disadvantages. For the proposed prototype, it would be possible to utilize a pre-existing blockchain such as the Ethereum or Bitcoin network, which would reduce the necessary costs of developing your own application, but would come with problems such as limited performance and cost of operation. Hyperledger Fabric was chosen for its well-documented and open-source approach and how it is highly customizable and could be made to fit any business environment necessary.

4.1.1 Blockchain Model

As per the first extrapolated requirement:

1. A blockchain to store attribute-related transactions.

This thesis proposes an approach in which attributes are stored on the blockchain. This blockchain approach utilizes the immutability and transparency aspect to allow guaranteed verification from any system observer. The ABAC system, as shown in Figure 4.1, will interact with the blockchain through the blockchain cloud (BCC) whenever it assigns, revokes, and verifies a user's attributes but otherwise functions as a normal ABAC scheme when handling non-delegated access.

An AA in Fabric could be either a peer or a specially implemented "AA" identity accepted by the MSP. The AA would be uniquely required to endorse any transaction related to the attributes it has control over, being directly stated as such in the endorsement policy, as discussed in Section 2.3.3. While the assignments and revocations are stored on the blockchain, the AA's attribute repository only contains a list of available attributes that it controls. Each peer, AA, and user would require their own key pair and certificate issued by a trusted CA to create their identity. These keys are used to ensure the integrity and origin of messages and transactions within the system through signing utilizing ECDSA.

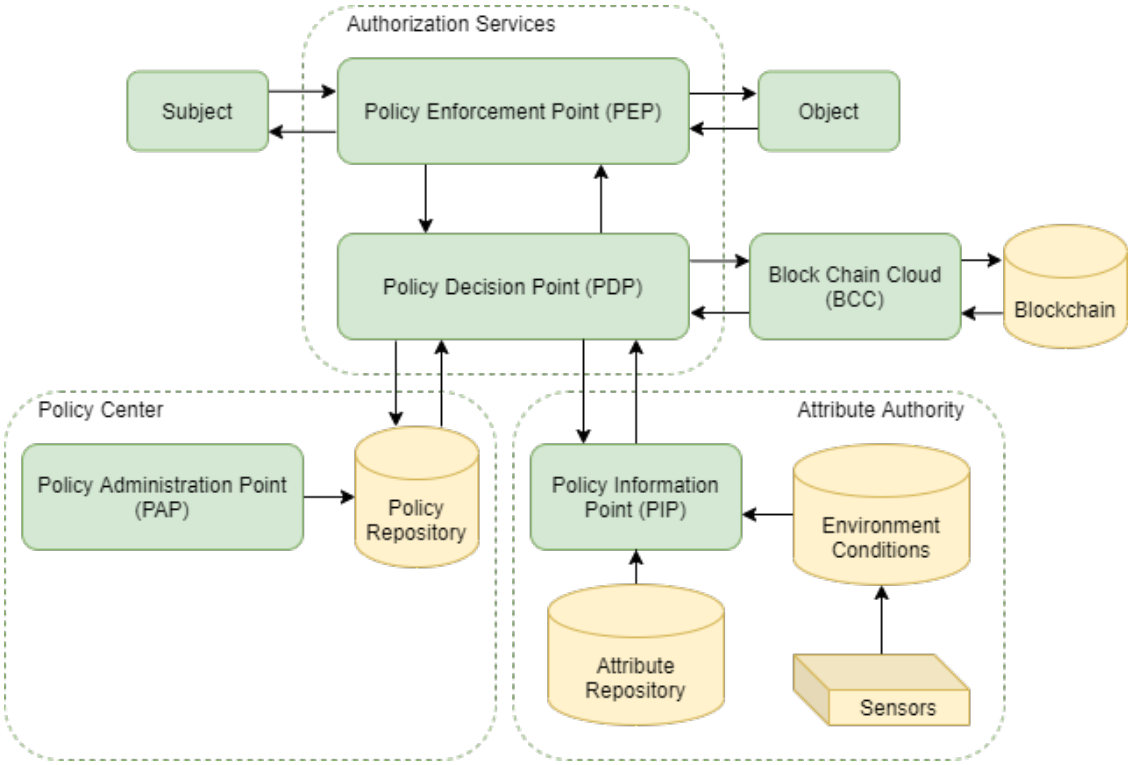


Figure 4.1: Illustrating blockchain-based ABAC

This blockchain usage takes root in Pussewalage and Oleschuk's research paper [6], in which they use the blockchain to store assignments and revocations, but also delegations. Delegations in their system could get cumbersome as their delegation chains would increase the blockchain's length and required a long process of searching the blockchain for revocations of every single delegation in the delegation chain. Therefore, moving delegation away from the blockchain was one of the first steps taken to increase the system's performance.

The blockchain transactions should include two different transactions; assignment and revocation. With only two types of transactions differentiating between them becomes easy; it is one or zero, true or false. These two transactions being the opposite of each other means that a transaction on the blockchain only needs to either contain information about a new assignment or point to a previous assignment for the system to work.

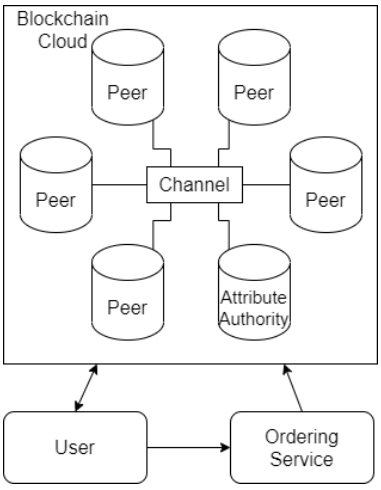


Figure 4.2: Simple model of the assignment transaction process

4.1.2 Assignment Transaction

One of the most important requirements, as extrapolated from the thesis definition regarding the blockchain prototype:

2. Attribute assignment transactions to assign one or more attributes to a user.

Attributes are controlled by the AA and can only be assigned by the AA; doing so creates a transaction that will be put on the blockchain. Assignment transactions include the receiver's public key, the attribute in question, and a signature from the AA.

So when a new employee, Charlie, is hired, he needs a set of keys and a certificate provided by a CA and various attributes that relate to the work he will be doing at his new job. Charlie's boss, Bob, creates a new user U_c with a private (SK_{Uc}) and public (PK_{Uc}) key pair and a certificate based on the public key. Bob is the head of the development on one of the company's newest projects, "Orion." Orion is split into three different parts, and Charlie only needs access to one of them, so Bob requests the "Orion" and "Orion-UI" attributes through Charlie's user. The AA that contains the attributes for Orion is called AA_o and, like the user, has a key pair denoted SK_{AAo} and PK_{AAo}. Likewise, the ordering service has the keys SK_O and PK_O. This process goes like so:

1. U_c creates a proposal (as explained in Section 2.3.7) in which they are to have the attributes "Orion" and "Orion-UI" assigned to themselves. U_c makes a header, signs the proposal with SK_{Uc}, and then broadcasts the information to the system.

2. Peers in the system verify the signature with PK_{Uc} and the contents of the proposal and header, ensuring everything is in order. Should nothing be amiss, the peers will endorse the proposal, sign it with their private key, and return their response to U_c. One of these peers is AA_o which controls the attributes in question; it creates an endorsement and signs with SK_{AAo}, which are required for the transaction to be valid.

3. U_c can then check the responses, endorsements, and signatures from the peers on the system but is not required to do so before broadcasting the transaction to the ordering service.

4. The ordering service receives the transaction from U_c and verifies its signatures using related public keys, information, and endorsements, and any faults end the process. If the endorsements do not validate the transaction according to the endorsement policy, the orderer stops. On the other hand, if everything has passed the control checks, the orderer creates a block structure and awaits further transactions. In this case, the orderer receives two transactions before finishing the block creation process. Then orderer generates a hash of the block data, retrieves the hash of the previous block's data, makes a bitmap for each transaction, creates an updated hash of the cumulative state updates, and finally provides the orderer's certificate and signs using SK_O.

5. Once the block is finished, it is broadcast to all peers, who will append the new block to their local blockchain.

6. Finally, the world state is updated with the new assignments.

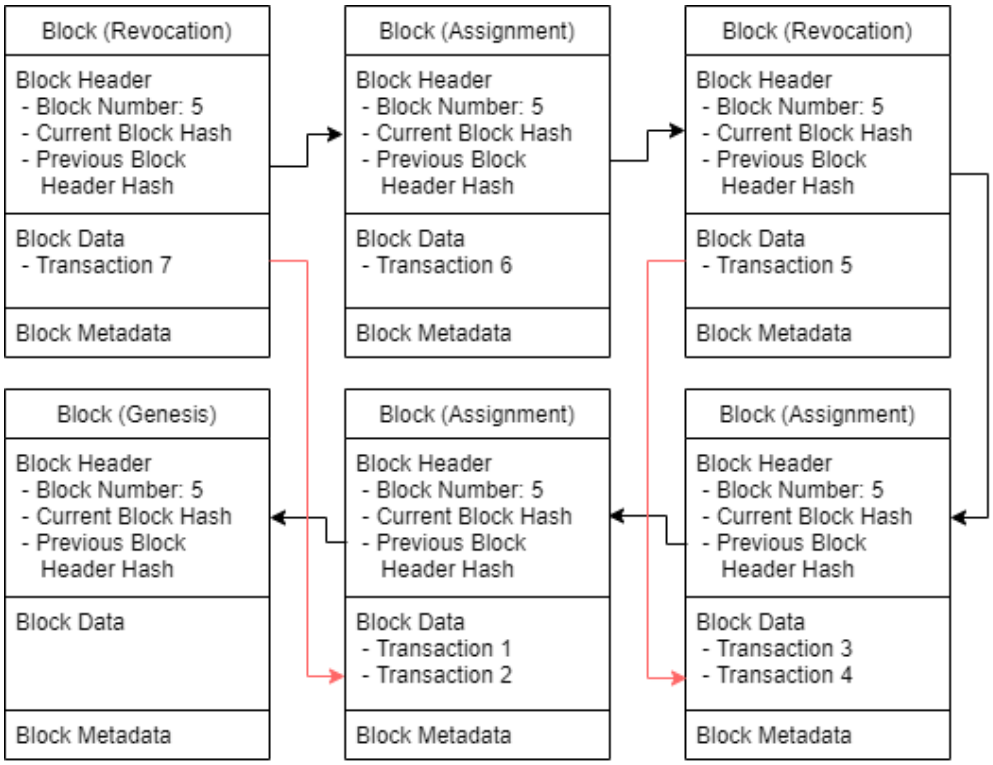


Figure 4.3: Example Blockchain with assignments and revocations

4.1.3 Revocation Transaction

For the third requirement listed at the start of the chapter:

3. Revocation transaction to remove or otherwise flag one or more transactions as no longer valid.

Then, for the second type of transaction that goes on the blockchain: Revocation. As per the requirement, the revocation transaction functions like a flag, marking a previous assignment transaction as invalid. Since the AA assigns attributes, it only makes sense that the AA is responsible for revoking them as well. The transaction data only contains a reference to the block and transaction, but otherwise, this is quite similar to the assignment process:

1. AA_o makes a proposal to revoke an existing transaction, as well as a header and a signature using SK_{AAo}. The information is broadcast on the channel to all the peers.

2. The peers check the new message's signature and content to verify its origin, returning a response and an endorsement to AA_o with a signature.

3. AA_o can check the responses from the peers and forward the data to the orderer. Even if the endorsements did not fulfill the endorsement policy, AA_o could still transmit the transaction onwards to the ordering service.

4. The orderer verifies the endorsements, the responses, the data, and the signatures using their corresponding public keys. If successful, the orderer starts creating a block including the

transaction in its data, then waits for further transactions. Once it has waited for a set amount of time, it will finish creating the block with the transactions it received. The orderer generates a hash of the block data, retrieves the hash of the previous block's data, makes a bitmap for each transaction, creates an updated hash of the cumulative state updates, and finally provides the orderer's certificate and signs using SK_O.

5. The block is finished, and the orderer broadcasts it on the channel, letting all the peers append the new block to their blockchains.

6. The world state is updated for each peer.

Thus, a blockchain fulfilling the requirements can be created using only assignment and revocation transactions. Figure 4.3 shows an example blockchain that features a few blocks with both revocations and assignments present.

4.1.4 Attribute Verification

The requirement for verification;

4. The scheme should accurately verify whether someone has the attributes required when granting access.

The verification process should be guaranteed to get the correct result, some of which is handled internally in Fabric by having version history in the form of the cumulative state hash, integrity checks in signatures, and the immutability of the blockchain itself. That leaves only a process to check the specifics of a user's assigned attributes.

When a user wants to read or write to a file, they will first have to mutually authenticate with the PEP and request access using their public key. The PEP will then send the information to the PDP, which will gather the necessary access policies and environment conditions, which could stop the system without further checks. If the environment conditions do not prevent access, then the PDP asks the BCC if the required attributes are assigned to the public key.

The BCC starts searching for valid assignments of the attribute to the public key, which in Hyperledger Fabric is done on the world state. The world state summarizes the transactions on the blockchain, which reduces search time considerably from checking multitudes of blocks on the blockchain. Once PDP receives the result from BCC, it will decide to grant or deny access based on the access policy and forward it to the PEP, which enforces the decision. This process can be seen in Figure 4.1.

4.2 Delegation Token

As part of the initial thesis definition in Section 1.2, several requirements were stated that the proposed scheme must support. Included below are the token-specific requirement mentioned in the thesis definition and Research Objective 1 (RO1) shown in Section 1.2.2:

"A token scheme to delegate access to users outside of the system."

Here we can once again extrapolate additional requirements to aid in the understanding of the proposed token solution. Therefore the proposed token scheme must support the following:

1. Easily verifiable delegatable tokens that utilize ABAC for authentication and authorization by checking the blockchain.
2. It should be possible to further delegate a received token to an additional third party, when necessary.
3. Clients can quickly generate tokens without the need to go through a centralized system
4. The tokens must be one-time use tokens and made in batches with a limited use time.

With these design parameters in mind, the following chapter will detail each proposed requirement for the delegatable permission token of the thesis.

4.2.1 Token Model

Before detailing the various ways in which the delegated permission token can be verified, authorized, and otherwise used, it is essential to understand the proposed token model and which data parameters need to be included and why. The following chapter explains the token structure and what each parameter is used for, and how each part of the token fits into the broader design.

```
{
    "DP"
    "ObjectId"
    "Action"
    "Receiver"
    "RS"
    "Previous"
    "Issuer"
},
PrivateKey,
{
    "algorithm"
    "expiresIn"
}
```

Listing 3: Delegatable Java Web Token Structure

Listing 3 shows the basic token structure of the proposed token scheme. Each data parameter is either predefined by the generating system or is a custom parameter that the generating user can specify. The first four parameters are custom specified by the issuing user, where the first parameter, "DP," stands for Delegation Permission and is the deciding factor of whether a user can further delegate a token or not. "ObjectID" contains the specific identification number of a requested resource or object, and the "Action" parameter has which action the user wishes to perform on the said object. It can either be Read (R), Write (W), or Read/Write (RW) for both. The "Reciever" parameter contains the public key of the receiving party. It is used in the verification process to verify that the intended party is trying to use the token, which should match the public key in the initial mutual authentication handshake with the Access Control system.

A Random Seed (RS) is included in the token to lessen the chance of a collision. The RS is generated based on the current timestamp of the generated token, combined with a randomly generated nonce to ensure a higher degree of randomization of the seed. The "Issuer" parameter contains the issuing user's Public Key and is used for token verification and corresponds to the inputted Private Key, as shown in Listing 3, which is used to sign the token contents and ensure authenticity.

Finally, in the last section of the token structure, are two predefined parameters set by the token generating system. The "Algorithm" parameter details the specific algorithm used to sign the token and is hard-coded into the token generation and chosen based on various security parameters. The system uses the "expiresIn" parameter to ensure the token is still within its allocated validity time slot.

4.2.2 Token Verification & Authorization

The essential requirement for the delegatable permission tokens proposed in this thesis is that they should be:

"1. Easily verifiable delegatable tokens that utilize ABAC for authentication and authorization by checking the blockchain."

As detailed in Section 2.1.2, the ABAC model primarily works through attributes and checking whether or not a user has valid attributes to access a resource based on pre-determined access policies. This thesis proposes storing those attribute transactions on a blockchain, as previously mentioned, so the actual delegatable tokens would only need to prove the token's Issuer has verifiable access to the system resource, for which the access is required.

The fundamental idea behind the delegatable permission tokens is to allow a user to delegate access to a resource to an outside party without going through a centralized system to authenticate. Any user in the proposed scheme can generate permission tokens by specifying a range of infor-

mation in the token payload, signing it, and sending it off to a potential third party. The verification of the token then comes down to primarily two things:

1. First, the token's signature is checked with the help of the Issuer's public key to ensure the authenticity of the token message between transfers.

2. Second, the tokens' expiration time must be within acceptable parameters. As the tokens are designed to have a limited use time of fewer than 24 hours, the token will be invalid if it is outside its validity period.

These two steps are purely to authenticate the validity of the token itself. Still, they do not include the crucial access control authentication aspect, which happens when a third party attempts to use a delegated token. When a token is used to gain access to a resource, the receiving system will first go through the two simple steps of verifying the token's validity before checking its payload. In the payload of the token, there will be three crucial parameters; "Issuer" containing the public key of the issuer, the "ObjectID," and the "Action".

The Issuers public key is used for two things: verification of token authenticity and checking the blockchain. Once an access request comes in through the use of an access token, the receiving system will first validate the token and then use the public key, object ID, and action to see whether or not the issuing party has access to the object with said actions through the standard ABAC access policy model detailed in Section 2.1.2. Should the initial issuer's public key be found on the blockchain with the corresponding attributes needed to perform the requested action on the requested object, the system should grant access to the requested third party.

4.2.3 Nested Delegation Tokens

One of the design aspects outlined in the Section 4.2 is:

"2. It should be possible to further delegate a received token to an additional third party, when necessary."

What this means in practice is that if a third party has been given a delegated permission token by a user with attributes on the Blockchain, that third party user should be capable of repeating the process the initial Issuer performed to generate a new batch of tokens and thereby further delegate the access to another party.

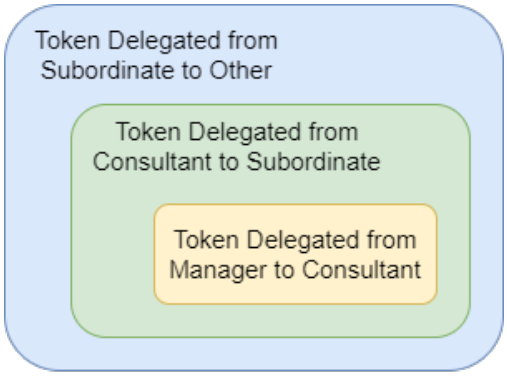


Figure 4.4: Nested Token Model

The idea behind this design goal is better illustrated with a small example scenario. Imagine a project manager has their attributes on the blockchain and has full access to the project resources. The project manager wishes to hire some consultants

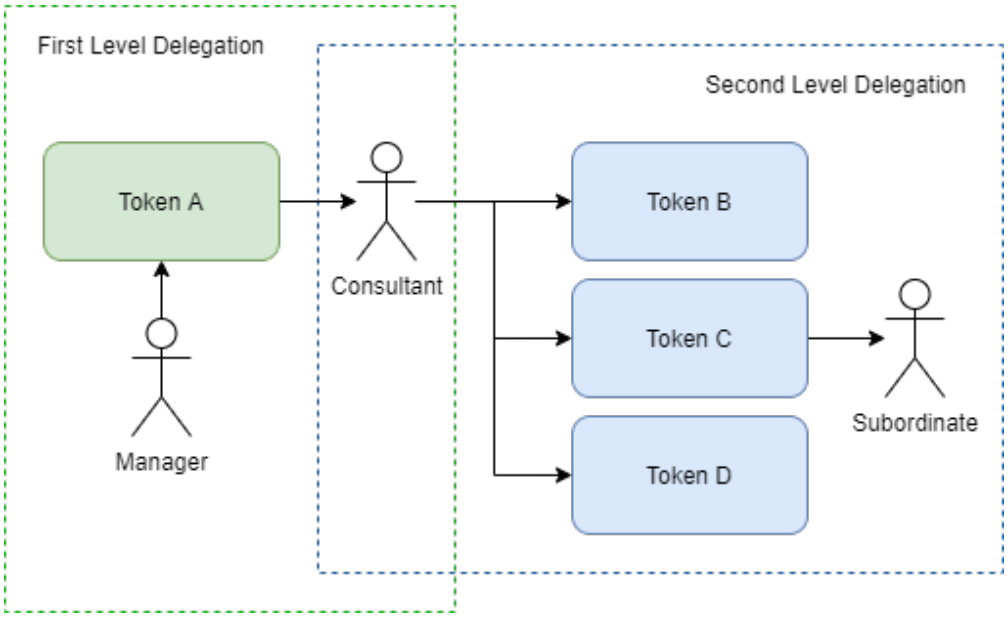


Figure 4.5: Illustrating first and second level delegations.

to work on the project but does not wish to add them permanently to the system. The project manager can generate a batch of permission tokens and hand them out to the various consultants. If a consultant then also has people working under them that need access to the resources, it should be possible for the contractor to further delegate access to their subordinates without going through either the manager or the initial AA of the system. These levels of delegation can be seen as illustrated in Figure 4.5. They could potentially continue indefinitely, although, at some point, the performance impact of a nested token outweighs its uses, which will be further investigated in Chapter 6.

This delegation chain is derived from two parts of the token model: the Delegation Permission (DP) parameter and the "Previous" parameter. The DP parameter serves as a way for our project manager to deny further delegation of their issued tokens, such that the receiving consultant is unable to delegate further. If the DP parameter allows for more delegations, the consultant can generate a new token. They put the previously issued token from the manager inside the "Previous" parameter of their newly made token and then delegate it further. This type of nested token is illustrated in Figure 4.4, which shows how each token down the chain contains the previous. Otherwise, the token generation and delegation work as usual.

Nested Verification

As a token has the potential to be a nested token, the verification and authorization process will have a couple of additional steps than what was described previously:

- The system needs to iterate through each nested token's Previous parameter and verify each nested token based on that token issuer's public key and expiration time.

- The system must check if any of the nested tokens in the Previous parameters have been added to the blacklist.
- When a token is successfully verified and access is granted, the initial token and any nested tokens will need to be added to the blacklist.

As each issued token will contain the Issuer's Public Key and is signed by the Issuer's Private Key, the system can verify each nested token's authenticity and ensure no tampering has occurred by decoding the token and using the stored Public Key. The first issued token that no longer holds any data in its Previous parameter is the original first-level token, which will have its corresponding public key on the Blockchain. Each token contained within the "Previous" parameter must also pass the standard token verification check to ensure the token has not expired and is in the expected format. Because each token is nested within each other, all delegations are dependant on the first level tokens' expiration time. If the first level token has expired, none of the further delegated tokens from that first level will pass the verification check, and thereby access will be barred.

When it comes to blacklist verification, it is essential to check each nested token on the blacklist. Figure 4.5 shows an illustration of first and second-level token delegation, in which case tokens B, C, and D, should not be valid if token A has already been used. Take the previously used example of the manager and the consultant, assuming that the manager has delegated the first-level token to the consultant. Should the consultant further delegate a second-level token to a subordinate, that subordinate should not be capable of using their second delegated token to get access if the consultant has already used the issued first-level token. This problem means that the system must check each nested token on the blacklist to ensure no higher-level nested token has already been used when verifying a token.

Lastly, when a token has been successfully verified, and the user has been granted access, the system must add the spent token and any nested tokens in the Previous parameter to the blacklist. The reason each nested token also needs to be blacklisted is to combat free token generation. Suppose the scheme did not add each nested token to the blacklist when it was used. In that case, the consultant in Figure 4.2 could indefinitely generate new tokens and effectively gain permanent access to the system, using only a single, one-time-use token.

4.2.4 Token Generation

One of the core ideas behind the delegatable permission tokens were:

"3. Clients can quickly generate tokens without the need to go through a centralized system"

The motivation behind this design parameter comes from the underlying idea of utilizing blockchain technologies and potentially keeping the technologies as distributed as possible while still supporting regular centralized authentication. Therefore, the proposed ABAC scheme would include a

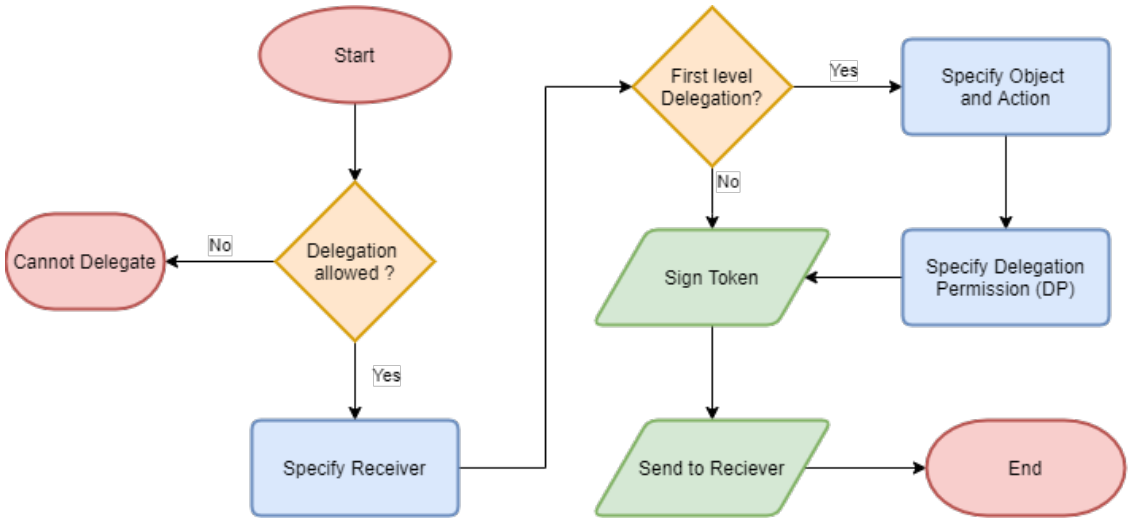


Figure 4.6: Token Generation User Flowchart

local application that could generate tokens by the user specifying various data parameters, most likely through a user interface.

Figure 4.6 shows a flowchart from the user's perspective when generating a token. Generating a token would be as simple as specifying who the recipient would be, what sort of access should be granted, and whether or not the token should have the capability of being further delegated, as detailed previously in Section 4.2.1. The recipient's public key would either be obtained through a trusted issued certificate or a locally stored list of public keys linked to their respective identities.

When specifying what access the token gives, the issuing user will specify an action such as read or write and which identification number the object has in the system. This object ID could easily be stored locally in the users' token generation application. The user specifies what the object is from a list of potential object resources, and the system takes the object ID and embedding it in the token. Finally, the issuing user signs the token using the Elliptic Curve Digital Signature Algorithm (ECDSA) with varying byte sizes (256 to 512), which will be further detailed in Chapter 6.

4.2.5 Single Use Design

According to one of the specified design requirements at the start of this chapter, the tokens must be a one-time use.

"4. The tokens must be one-time use tokens and made in batches with a limited use time"

The reasoning behind this is to ensure a degree of control over issued tokens. The problem arises when a user has been given a token, but you no longer wish for the issued token to be valid. Revocation can solve this problem similar to how attributes can be revoked on the blockchain, as detailed in Section 4.1. The primary issue with revoking a token is the simple fact that the system

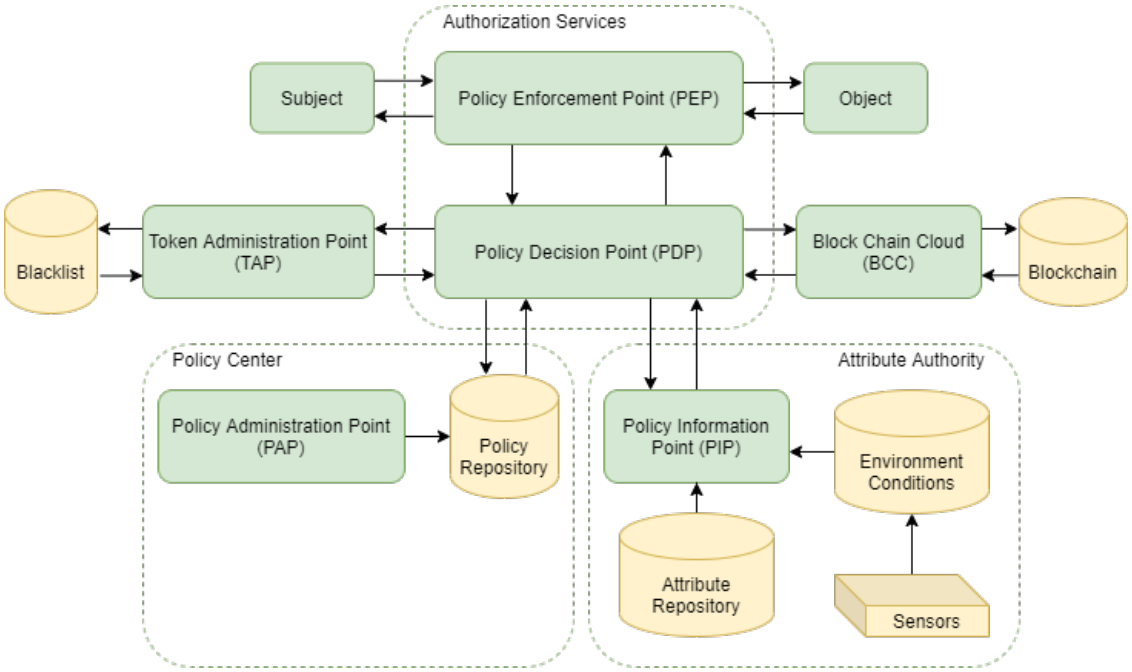


Figure 4.7: Standard ABAC framework with BCC and TAP.

can't know how many or which tokens have been issued by its clients, as one of the design requirements is for the users to generate their tokens without going through a centralized system.

The revocation issue with delegated tokens has been solved in this thesis by adding a simple centralized blacklist database to the proposed ABAC scheme that stores used tokens as they have been successfully verified. This addition requires the inclusion of an additional point in the standard ABAC framework previously detailed in the thesis called the Token Administration Point (TAP). Figure 4.7 shows how TAP and the blacklist fit into the broader ABAC model, with all requests to and from the blacklist database goes through the TAP.

This blacklist database adds an extra step to verification and may impact the overall system's performance, but it is a necessary part of the proposed scheme to ensure a token is not used multiple times. This solution ensures consistency throughout the system, as each time a token is attempted verified, the system must check the blacklist database for previous utilization of the token.

Another possible variant of the blacklist database is to store a local list of already-used tokens on the endpoint system and replicate it throughout the system with specific time intervals. However, such a locally stored replicating list could potentially cause issues by using an access token multiple times in a short window of time before all endpoints receive the updated used token list. The choice between a blacklist database and locally stored endpoint lists is mainly between consistency at the cost of performance or scalability of a potential implemented scheme. Since the scope of the thesis only includes implementing a blockchain and token prototype and not an actual ABAC system, the implementation and possible performance parameters of a blacklist implementation are outside the scope of the thesis, but is something to be considered for future work.

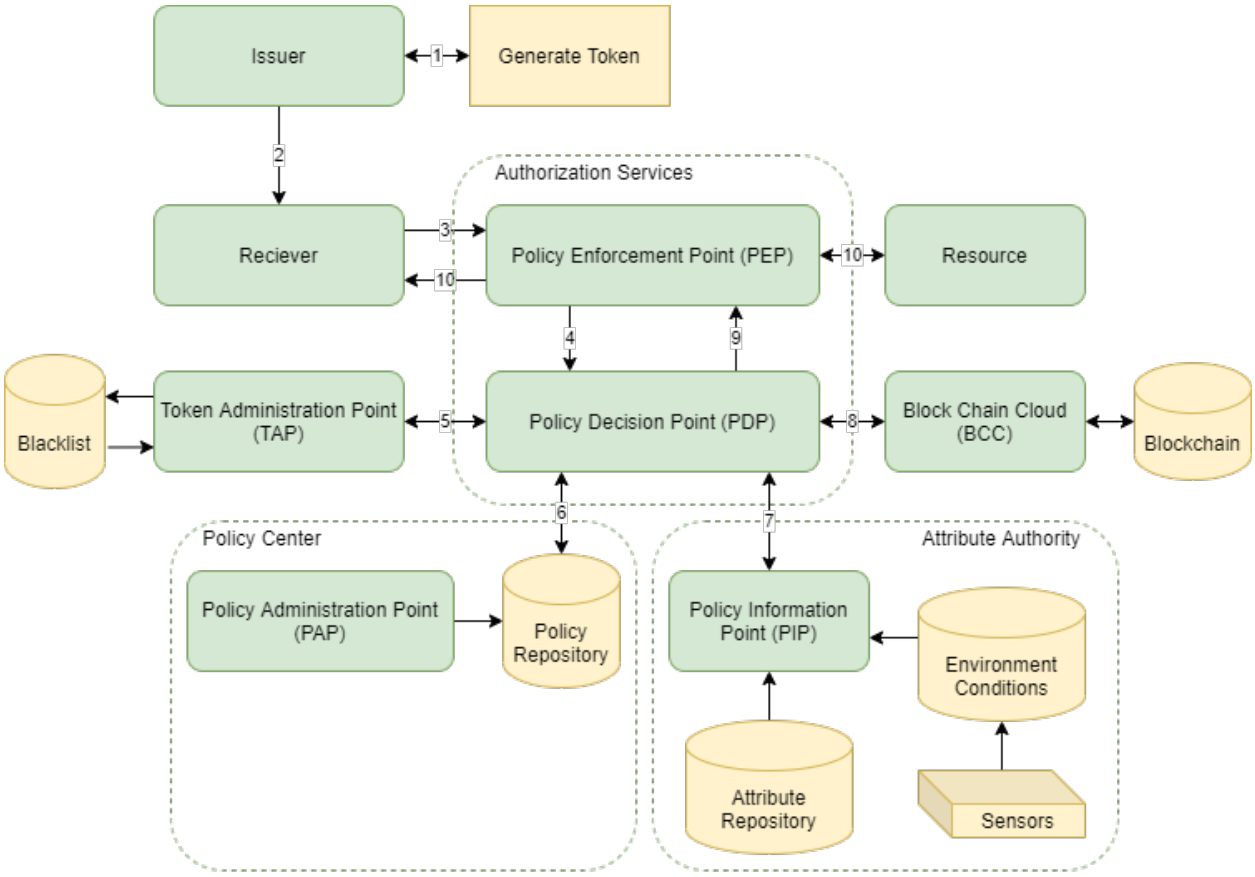


Figure 4.8: Modified ABAC Model showing delegation process

4.2.6 Detailed Delegation Procedure

To better understand the design philosophy behind the delegatable permission tokens, Figure 4.8 shows the delegation process of the proposed scheme. The following part of the section will walk through each step of the process and explain each section illustrated in Figure 4.8.

1. The first step is for the Issuing user to generate the token. The Issuer specifies the following information: DP level, object ID, action, and Reciever. The rest of the parameters are pre-defined by the generation application, as detailed in Section 4.2.4.

2. The Issuer will mutually authenticate with the Reciever, potentially utilize a Diffie Hellman key exchange to ensure secure communication. In the process, the Issuer will send the token to the Receiver.

3. The Receiver mutually authenticates with the PEP and requests access to a resource through a token.

4. PEP takes the initiated access request and forwards it to the PDP to gather data and make a decision.

5. PDP will request the TAP to verify the token's integrity through its signature and expiration time and whether or not the token itself or any nested ones are on the blacklist. TAP then returns its valid or invalid decision to PDP.

6. If the token is valid, PDP will query the Policy Repository for any Access Policies (AP) regarding the object.

7. Once the PDP receives any Access Policies, it will query the Policy Information Point (PIP) for any external environmental attributes that may impact the decision.

8. Then, the PDP will query the BCC containing the Blockchain for whether or not the public key of the first level tokens issuer has attributes on the Blockchain that would satisfy the retrieved access policy.

9. Finally, PDP makes a decision based on all the gathered information and forwards it to the PEP.

10. Lastly, access is either granted or denied to the access requested user.

Chapter 5

Solution & Implementation

This chapter conveys the solution to the proposed design scheme of the blockchain-based ABAC scheme with delegatable permission tokens and fulfills Research Objective 2 of the thesis:

RO 2: *Implement the proposed solution through proof-of-concept applications utilizing the designed model.*

This prototype's development and consequent implementation and testing process have primarily been split into three parts, as shown in Figure 5.1. The first part has two separate implementations; a blockchain prototype to evaluate the useability of the Blockchain to store attribute transactions of the model and the token application for access delegation, each of which will be showcased in this chapter. The other two parts take up the performance and testing analysis of the thesis and will be detailed later in Chapter 6.

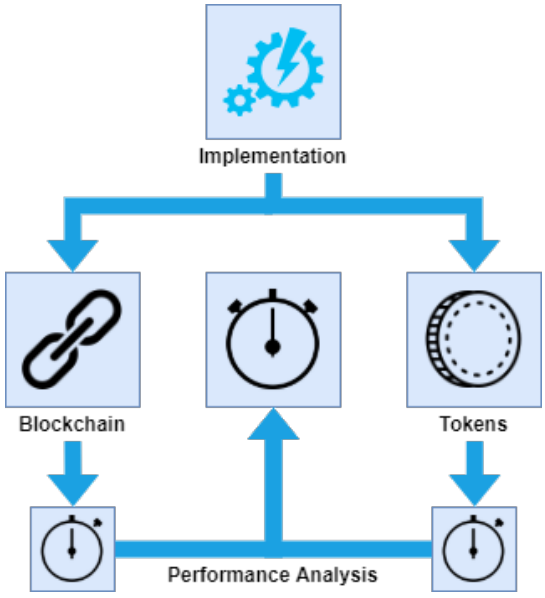


Figure 5.1: Implementation Process of the proposed scheme.

We start by providing the details of the blockchain prototype, how it was implemented and developed, along with how it performs with actual data. Then we describe the proposed token prototype, including its various scripts to generate and verify tokens, and an overview of how the various technologies were utilized in its implementation.

5.1 Hyperledger Fabric Test Network

Following the ideas in the proposed scheme, Hyperledger Fabric was used to create a system that could function as a prototype for the testing phase. This section will detail how the ledger and the chaincode were implemented.

To start with, Hyperledger Fabric has a well-documented tutorial, which includes a test network with various examples of chaincode and configurations. This test network was deemed suitable for prototype testing and modified to allow for the proposed system's use case. Hyperledger Fabric's world state is by default stored with a database library called LevelDB but was changed to CouchDB due to limitations in query possibilities. A rich query system is required in order to search the world state for a combination of values, which cannot be done in Fabric's default LevelDB. Therefore, CouchDB is used so that verification can be done as proposed: With a search of public keys and attributes.

Furthermore, the chaincode that is deployed on the channel has been modified to support the transactions and functions required to create attribute assignments and revocations. The functions are recreated from existing ones that are used for a different transaction structure to minimize the internal errors in Fabric.

5.1.1 Transaction Data

The data structure, commonly called a struct, from the implementation can be seen in Listing 4 and features five strings. The first one, "ID," is a unique identifier for the transaction used on the world state when deleting (revoking) and querying. Next, "PublicKey" is the parameter that holds the user's public key, which is used in the verification process along with the following parameter: "Attribute," containing the user's attribute. "TimeStamp" is a representation of when the transaction proposal was created and can be used to determine a possible expiration time of assignments. Lastly, "Seed" is a random string that serves as an enrichment factor for the hashing algorithm to lower the chance of attackers finding collisions. TimeStamp and Seed could very well be integers or other numeric variables, which would make for a more straightforward operation when determining the expiration time and a faster generation of a random seed.

```
type Assignment struct {
    ID          string `json:"assignmentID"`
    PublicKey   string `json:"pk"`
    Attribute   string `json:"attribute"`
    TimeStamp   string `json:"timestamp"`
    Seed        string `json:"seed"`
}
```

Listing 4: Assignment Data

5.1.2 Assignment of Attributes

The struct is explicitly used for creating an assignment on the blockchain, which is done through the CreateAssignment function. Thus, the function for assigning an attribute to a user is displayed in Listing 5.

The function header first declares an object "t" out of a struct made from a "contract" structure in Fabric's contract API. Second, it declares the function's name and the parameters it requires; the first, "ctx," is made by calling an interface type from the contract API, while the rest are specified in the function call. Third, the header contains the return type, which defines what type of variable the function returns.

The first step of the assignment function declares two variables to receive the return output of the "AssignmentExists" function, which is called immediately after. As can be seen in Listing 6, the function takes the unique identifier of the assignment and searches the world state for any existing entries with that ID. Then, it checks whether there were any errors in the search, and if there were none, it returns a boolean value to indicate whether the assignment exists already along with the error value.

```
func (t *SimpleChaincode) CreateAssignment(
ctx contractapi.TransactionContextInterface, assignmentID,
pk string, attribute string, timestamp string, seed string) error
{
    exists, err := t.AssignmentExists(ctx, assignmentID)
    if err != nil {
        return fmt.Errorf("failed to get assignment: %v", err)
    }
    if exists {
        return fmt.Errorf("assignment already exists: %s", assignmentID)
    }

    assignment := &Assignment{
        ID:         assignmentID,
        PublicKey:      pk,
        Attribute:      attribute,
        TimeStamp:     timestamp,
        Seed:          seed,
    }
    assignmentBytes, err := json.Marshal(assignment)
    if err != nil {
        return err
    }

    return ctx.GetStub().PutState(assignmentID, assignmentBytes)
}
```

Listing 5: The CreateAssignment function used for assigning attributes to a user

```
func (t *SimpleChaincode) AssignmentExists(
ctx contractapi.TransactionContextInterface, assignmentID string)
(bool, error)
{
    assignmentBytes, err := ctx.GetStub().GetState(assignmentID)
    if err != nil {
        return false, fmt.Errorf("failed to read assignment
        %s from world state. %v", assignmentID, err)
    }

    return assignmentBytes != nil, nil
}
```

Listing 6: AssignmentExists checks for existing assignments with the given unique identifier

Returning to the AssignmentCreation function, it will check the values it received. If "error" is anything but nil, then there was an error, and the function returns a failure statement. Then, "exists" is checked and, if true, returns that the assignment already exists on the world state.

Next comes the declaration of the object itself, created from the previously discussed struct, and assigns the appropriate values from the function call to the fields within the struct. Two new variables are declared and equals to the function call "JSON.Marshal." The object is passed through a JSON encoding function and returned along with an error value. The error value is once again checked before the process calls return with a function that starts creating a transaction for the blockchain using the encoded JSON object and its unique identifier. Once the transaction has been added to the next block, the user receives a confirmation of its success.

5.1.3 Revocation of Attributes

Revoking an attribute assignment on the blockchain can be done by calling the function "Delete-Assignment," named for its effect on the world state. The code snippet for revocation is shown in Listing 7; it shows the same basic outline as the "CreateAssignment" function for the header, only changing the input parameters to a single string: "AssignmentID."

Executing the revocation process, it steps into the "ReadAssignment" function with the "AssignmentID" as input. This function, as shown in Listing **??**, calls "GetState" to search the world state for an existing entry using that ID, which returns the encoded assignment and an error code. It checks the two return values for errors and whether the assignment exists, returning an error message if either fails. Then, the process creates an assignment object out of the struct and starts the "JSON.Unmarshal" function, passing the confirmed existing assignment and a pointer to the empty assignment object. The "Unmarshal" decodes the encoded "AssignmentBytes" and checks that it contains data, returning an error if it does not and nil if it does.

```
func (t *SimpleChaincode) DeleteAssignment(
ctx contractapi.TransactionContextInterface, assignmentID string) error
{
    err := t.ReadAssignment(ctx, assignmentID)
    if err != nil {
        return err
    }

    err2 := ctx.GetStub().DelState(assignmentID)
    if err2 != nil {
        return fmt.Errorf("failed to delete assignment
        %s: %v", assignmentID, err)
    }

    return err2
}
```

Listing 7: Revocation function

Once the return value comes through to the original function, it is tested for any errors. Now that the assignment has been successfully confirmed to exist, it is deleted from the world state by "DelState" using its unique key. When deleting the assignment from the world state, it will generate a proposal and follow the standard Hyperledger Fabric procedure, creating a transaction that flags the assignment transaction and deletes the unique identifier's entry from the ledger. Any possible errors are returned and examined for faults, and the process finishes.

```
func (t *SimpleChaincode) ReadAssignment(
ctx contractapi.TransactionContextInterface, assignmentID string) error
{
    assignmentBytes, err := ctx.GetStub().GetState(assignmentID)
    if err != nil {
        return fmt.Errorf("failed to get assignment %s: %v", assignmentID, err)
    }
    if assignmentBytes == nil {
        return fmt.Errorf("assignment %s does not exist", assignmentID)
    }

    var assignment Assignment
    err = json.Unmarshal(assignmentBytes, &assignment)
    if err != nil {
        return err
    }

    return nil
}
```

Listing 8: ReadAssignment verifies that assignments have data and exist on the world state

5.1.4 Verification of Attributes

Lastly, the verification process searches the world state and returns results regarding a user's assigned attributes, the code for which can be found in Listing 9. This function also has a similar header to the previous ones, notable differences being the "QueryString" parameter and the additional bool return value. The input parameter consists of all variables stated in the function call, allowing the client to specify attributes, public keys, timestamps, and such information fields from the structs. The input parameter is used to search the world state and can include any variety of fields, but for verification, only attribute and public key are necessary.

```
func (t *SimpleChaincode) Verification(
ctx contractapi.TransactionContextInterface, QueryString string)
(bool, error)
{
    assignmentBytes, err := ctx.GetStub().GetQueryResult(QueryString)
    if err != nil {
        return false, fmt.Errorf("failed to read assignment with
        %s from world state. %v", QueryString, err)
    }
    assignmentBytes.Close()

    if !assignmentBytes.HasNext() {
        return false, nil
    }
    ejson, err := assignmentBytes.Next()
    if ejson.Value == nil {
        return false, err
    }

    return assignmentBytes != nil, nil
}
```

Listing 9: The Verification function checks the world state for existing assignments based on input

1. The first step taken is to call the function "GetQueryResult()" with the "QueryString," which is why CouchDB is used over LevelDB; since querying with struct fields is impossible in the base version of LevelDB. The function "GetQueryResult()" returns an iterator object that contains any encoded assignments found on the world state and an error value, which are stored in "AssignmentBytes" and "err." However, if there were no assignments found on the world state, it will still return an object, but it won't have any values in the list. The error value is checked, returning an error message on failure, and the stream is closed to prevent memory issues, then the object's content needs to be checked since it could be empty.

2. Whether the list has any entries can be found with "HasNext()," which returns a bool value that can be used to determine if there are any entries in the list. If "GetQueryResult()" found nothing, then "HasNext()" returns false and ends the process with a rejection.

3. Thereafter, the process checks whether the first assignment contains information by assigning it to a newly-declared object using the "Next()" function and verifying its value parameter. Value is a JSON parameter that stores the data of the encoded JSON struct, so if it is empty, the process once again denies the user's claim of the attribute.

4. Then, the verification process returns a boolean value of whether the "AssignmentBytes" object has data to represent either rejection or successful verification, along with the error value nil.

Therefore, when a user or a system component wishes to verify attributes on the blockchain, they will input relevant information such as public key, block ID, or attributes, which will go through the process detailed above. Should the queried information exist, the process will return true and confirm to the user or system that the requested data exists on the blockchain and is verified.

5.2 Token Implementation

Access tokens are a widely used technology, as are the various libraries, frameworks, and modules that allow developers to create applications to utilize these technologies. The proposed solution for the token framework detailed in the previous chapter uses Java Web Token (JWT) to generate and validate tokens. As previously explained in Chapter 2, JWT are a versatile JSON-based token structure primarily comprised of a header, payload, and signature.

In the following section, we discuss the various ways the proposed token design has been implemented through pre-existing frameworks and custom design. The following scripts related to JWT have been developed using the JavaScript programming language and the Node.js runtime environment to execute the script code. All development and testing of the token prototype have been done in a Linux environment running on local hardware.

5.2.1 Key Generation

Instead of using a pair of test keys for the token implementation, a set of JavaScript programs was developed to generate a fresh key pair each time a token generation script is used. These scripts

NIST Recommended Key Sizes		
Symmetric Key Size (bits)	RSA and Diffie-Hellman Key Size (bits)	Elliptic Curve Key Size (bits)
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	521

Table 5.1: NIST recommendations for key size in bits [7]

were created to ensure the result is closer to a real-world scenario. The type of keys chosen for implementing this prototype was Elliptic Curve keypairs using the P-384 curve for a higher degree of security, thereby generating 384 bit EC keys. As EC keys are inherently smaller than RSA keys, it is possible to get similar levels of security with a 384 bit Elliptic Curve key, like that of a 7680 bit RSA or Diffie-Hellman key showcased in Table 5.1.

The keys are generated using a Node.js module called Crypto [19], which is a native module to Node.JS, supporting various cryptographic functionality such as key generation, issuing certificates, and performing symmetrical cipher and decipher operations. The code shown in Listing 10 is the primary function used to generate the keys used in the token generation software. The function takes various parameters by first detailing the type of key, where Crypto supports RSA, DSA, EC, DH, and various combinations of those. Furthermore, the type of public key generated is "spki," which stands for Simple Public-Key Infrastructure, specified in RFC 2692/2693. The type utilized for the private keys is that of "pkcs8", which is a standard for storing private key information specified in RFC5958. Both keys are in Privacy-Enhanced Mail (PEM) format, which is the de facto file format for storing cryptographic keys and certificates, as specified in RFC7468.

A second JavaScript program was also created as part of this thesis to generate vast numbers of public keys for utilization in the previously detailed blockchain part of the prototype. This program utilizes the same Crypto module and code as shown in Listing 10 but only generates public keys and stores them in a standard newline-separated text file for testing purposes.

```
generateKeyPair(
  "ec", //Set key type
  {
    //modulusLength: 2048,
    namedCurve: "secp384r1", //secp384r1 = P-384
    publicKeyEncoding: {
      type: "spki", //Note the type is spki not pkcs1
      format: "pem",
    },
    privateKeyEncoding: {
      type: "pkcs8", //For RSA use pkcs1 format, for EC use pkcs8 or sec1
      format: "pem",
      //cipher: "aes-256-cbc", //Optional
      //passphrase: "", //Optional
    },
  },
```

Listing 10: JavaScript function to generate Elliptic Curve keys

```
var tokenStruct = { //Object used to pass to functions.
    DP: "0",
    ID: "1",
    Algorithm: "ES512",
    ExpirationTime: 3600,
    Action: "RW",
    ObjectID: 12345678,
    Previous: "NULL",
    Reciever: "Error: No reciever set",
    Issuer: "Error: No issuer set",
    PrivateKey: "Error: No key set",
}
```

Listing 11: JavaScript object structure detailing the token

5.2.2 Token Generation

Before generating a token, an object structure detailing the various information the token will contained was created. As shown in Listing 11, the token struct shows the various data parameters sent to the token generation function in Listing 12. As previously detailed in Section 4.2.4, the user only specifies a handful of information depending on whether it is the first token in the chain or not. Regardless, the token structure is used to pass to the token generation function instead of passing individual variable parameters.

```
function GenerateToken(tokenStruct){
    var time = new Date().valueOf();
    var Seed = Math.floor(Math.random() * time);

    var generatedToken = jwt.sign(
    {
        DP: tokenStruct.DP,
        ObjectId: tokenStruct.ObjectID,
        Action: tokenStruct.Action,
        Receiver: tokenStruct.Reciever,
        RS: Seed,
        Previous: tokenStruct.Previous,
        Issuer: tokenStruct.Issuer,
    },
    tokenStruct.PrivateKey,
    {
        algorithm: tokenStruct.Algorithm,
        expiresIn: tokenStruct.ExpirationTime,
    });
    return generatedToken;
};
```

Listing 12: JavaScript function to generate delegation token using Node.js module jsonwebtoken

All JWT token generation and verification comes from the standalone Node.js library aptly named "node-jsonwebtoken [20]" used by almost a million developers. As shown in Listing 12, the first parameter takes in the token structure object and generates a Seed based on the current time. Furthermore, the actual signing function from the external library takes three parameters; any specified custom data, a private key for signing purposes, and finally, any JWT specific options such as the signing algorithm or expiration time. The function itself merely uses the information passed to it through the object, and the user would update the struct to include its requested information before passing it to the token generation function.

The rest of the token generation script code specifies information such as; issuer, receiver, object ID, and action to make the generated tokens dynamically different each time they are generated for more accurate testing. Finally, the signed and encoded token is written to a text file for further use in the verification script.

```
victor@victor-ubuntu:~/tokens/jwt$ node generateToken.js
Generated 1 token(s) at level 5 in 0 s and 28 ms.
The token was written to filepath:  ./tokens/testToken.txt
victor@victor-ubuntu:~/tokens/jwt$ node generateToken.js
Generated 10 token(s) at level 7 in 0 s and 93 ms.
The token was written to filepath:  ./tokens/testToken.txt
victor@victor-ubuntu:~/tokens/jwt$ node generateToken.js
Generated 100 token(s) at level 3 in 0 s and 314 ms.
The token was written to filepath:  ./tokens/testToken.txt
victor@victor-ubuntu:~/tokens/jwt$
```

Figure 5.2: Console output of the token generation script

The token generation script that serves as a proof-of-concept for the proposed token design of the thesis has varying outputs depending on the preferred outcome. In order to replicate a fully developed application, it is possible to specify the number of tokens that the user wishes to generate and the nested level of that batch of tokens. Figure 5.2 shows the console output of three different script executions, with varying levels of nested and amount of tokens.

The time it takes to generate tokens will be investigated and detailed in Chapter 6. The purpose of the manual setting of the token amount and nested level is for the verification script to run through the token regardless of what level or amount of tokens it receives. Therefore it is possible to generate higher levels of nested tokens that would otherwise be realistic to test the prototype.

The signing algorithm used in the token generation function as specified in the object struct listed in Listing 11 is "ES512," which, according to the Node-Jsonwebtoken documentation, uses ECDSA signing with the P-521 curve and SHA-512 hash algorithm. As previously specified in Section 5.2.1 Key Generation, the EC keys were chosen for their smaller size and comparable security to higher RSA keys. As these tokens are designed to be lightweight and easily transported, the ECDSA signing algorithm was chosen to support high-security levels and keep the token size low. Different key sizes and signing algorithms will be explored in Chapter 6 when testing the prototype.

5.2.3 Token Verification

Once a token has been generated, the "tokenVerification.js" script is used to verify the token. The script works by iterating through the tokens nested parameter, verifying the signature of each nested token before continuing to the next, before ending up with the first-level token of the original issuer. Once the first level token has been located and verified from the nested token, the public key of the first level token issuer can be forwarded to the Blockchain to verify whether or not the user in question has the requested access detailed in the token.

```
function VerifyToken(input, cert){
    var isValid = false;
    jwt.verify(input, cert, { complete: true }, function(err, decoded){
        if(err){
            console.log("Error: ", err.message);
            return err;
        }else{
            var truebool = true;
            isValid = true;
            var timeNow = Date.now() - t1;
            console.log("Verify success at time: ", timeNow);
        }
    });
    return isValid;
};
```

Listing 13: Function to verify the encoded JWT

Listing 13 shows the primary verification code used in the prototype and takes the encoded token and public key (or certificate) as input. Then the code calls the "Node-Jsonwebtoken" module to perform the verify operation on the encoded token with the signature, and then finally returns a boolean to indicate whether or not the input token was successfully verified. The "VerifyToken" function will return a false, if the used certificate does not match the one used in signing the token.

```
//Decodes the encoded JWT
function SimpleDecode(token){
    var token_decoded = jwt.decode(token, {complete: true});
    return token_decoded;
}
```

Listing 14: Decode function to extract JSON object from encoded JWT strings

```
//Locates the first level token
function FindFirstToken(input){
    var temp_token = input; //Encoded
    var temp_decode = SimpleDecode(temp_token); //Holds decoded JSON object
    var temp_verify = VerifyToken(temp_token, temp_decode.payload.Issuer);

    loop();
    function loop(){
        if((temp_verify) && (temp_decode.payload.Previous != "NULL")){
            temp_token = temp_decode.payload.Previous;
            temp_decode = SimpleDecode(temp_token);
            temp_verify = VerifyToken(temp_token, temp_decode.payload.Issuer);
            if(temp_decode.payload.Previous != "NULL" ){
                loop();}
        }else{
            console.log("Error: Token is already first token!");}
    }
    var returnVar = [temp_decode, temp_verify];
    return returnVar;
}
```

Listing 15: Code to locate and verify the first level token in a nested JWT

As the purpose of the token scheme is to fetch the issuers public key, requested action, and forward it to the ABAC system, there needs to be a function to iterate through the received nested token and verify each nested token with the issuers of the previous token public key. Listing 15 shows the code used in the prototype for this exact purpose, where it takes in an encoded token and returns the first level token nested within.

As JWTs are not inherently encrypted and thus do not require a key to decode, the SimpleDecode function from Listing 14 takestakes the encoded JWT and returns it in a decoded JSON format to access the information within. To ensure a token's integrity, however, it is not enough to decode the token and verify it at each step; hence, the FindFirstToken function calls the previously

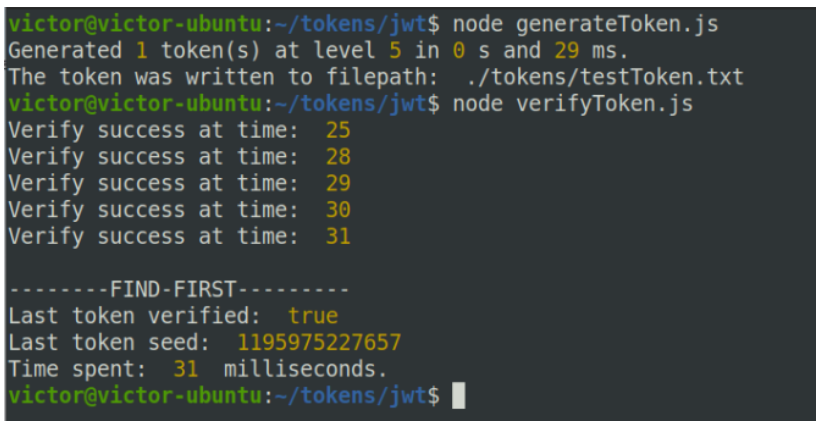


Figure 5.3: Console output of the tokenVerification.js script

described Verify function from Listing 13 as it iterates through each nested token. Once the function has reached the token with its previous parameter equal "NULL," it has located the first token and will return it.

The output of the token verification script can be seen in Figure 5.3, where the token generation script was first used to make a level five token and then verified through the verification script. Each time the script iterates through and successfully verifies a nested token, the timestamp is outputted to show progress. Finally, the first token is returned, and parts of its data are outputted to the console for manual verification, while the time it took to verify the token will be utilized more in Chapter 6 when analyzing different test cases.

Chapter 6

Testing & Results

This chapter illustrates how we have achieved Research Objectives 3 and 4 by showing various tests and results from the developed prototypes.

- ***RO 3:*** *Identify the important performance parameters associated with the application*
- ***RO 4:*** *Obtain the results for the identified performance parameters and thereby provide evidence on the practicality of using Blockchains and delegatable tokens for access control purposes.*

The chapter is structured first to introduce the potentially identified performance parameters related to the implementation and test those parameters alongside any results from the prototypes. Then, following the implementation results, a security analysis of the solution will detail various common attack vectors and whether or not the proposed model is vulnerable to them. The chapter will also discuss how well the implementation meets the original requirements stated in the thesis definition and attempts to answer the posed research questions in the introduction chapter.

6.1 Hardware & Project Configuration

Before detailing test cases and results of the designed prototypes, it is essential to showcase the hardware and project environment the prototypes have been implemented to give a broader understanding of its later results.

Token Implementation & Testing Hardware	
Identifier	Hardware Name
OS	Ubuntu 20.04.2 LTS
Motherboard	ASUSTek B85-PRO GAMER
RAM	16GB DDR3 @ 1600MHz
CPU	Intel(R) Core(TM) i7-4790 Quadcore @ 3.60GHz
GPU	NVIDIA GeForce GTX 1070 Ti @ 1750MHz 8GB GDDR
Storage	480GB SATA 6Gb/s SSD

Table 6.1: Table showing Token Prototype Hardware

Due to the limitations underlined in Section 1.3.2, only personal hardware was available to develop and test the prototypes. Table 6.1 & 6.2 give an overview of the hardware on which the development and testing of the various prototypes have been performed.

Blockchain Implementation & Testing Hardware	
Identifier	Hardware Name
OS	Debian 4.19.0-14-amd64
Motherboard	HP 85C6
RAM	5660MB DDR3 @ 1000MHz
CPU	Intel Core i7-8565U Quadcore @ 1.60GHz using 2 cores
Storage	20GB SATA 1.6Gb/s SSD

Table 6.2: Table showing Blockchain Prototype Hardware

All token-related scripts and applications have been developed on the hardware and operating system detailed in Table 6.1, using Visual Studio Code and Node.js. As this is a proof-of-concept, the project structure has been purposefully kept to a minimum. Any generated keypairs or tokens used in the generation and verification of the token scheme have been stored in their separate directories. This structure is done to ensure an easily readable overview for testing and development purposes.

6.2 Performance Parameters & Testing

As defined in the thesis definition, fulfilling Research Objective 3 (RO3) requires various performance parameters to be investigated and concluded based on the implementation.

RO 3: *Identify the important performance parameters associated with the application*

Furthermore, several tests need to be carried out to measure the solution's viability to determine the overall performance of the system. Specifically, there are primarily three use-cases of the implementation that need to be investigated to measure the performance impact on the system:

- Token generation and verification, and verifying the corresponding attributes on the blockchain,
- Adding new attribute and revocation transactions to the blockchain through Hyperledger Fabric, and
- Verifying a user's attributes on the blockchain.

The combined performance of the two prototype implementations will give a complete picture of the overall system performance to use in a final product. Some of the performance tests need to consider the various states the system may be in and the edge-cases that define the upper and lower boundaries to portray and convey the implementation's performance accurately.

6.2.1 Blockchain Parameters

The blockchain component of the system has three major elements whose performance should be tested in their viability: The ledger, transaction creation, and attribute verification. These three parts will determine whether the proposed blockchain usage has any future in access control systems.

The Ledger

Due to the immutability of blockchain making it impossible to delete blocks, as defined in Section 2.2, the ledger will only grow in size. Over time, a blockchain will thus take up too much physical storage for its capabilities to be practical. However, the degree to which a blockchain grows is dependent on how much use it sees. In the proposed scheme, it is unlikely that the blockchain sees daily transaction additions as security changes are relatively infrequent, but a new employee or a new attribute addition would require further assignment transactions, and an employee leaving the company would require the revocation of the user's attributes.

A company with one hundred employees with an average of thirty attributes for each user initially has a blockchain with 3000 assignment transactions. It has been found that the average yearly turnover rate is 10% [**AverageTurnover**] which means that ten employees with three hundred attributes will need to be revoked and reassigned to new employees every year. Furthermore, attributes and their policies may be refactored, causing even more transactions to be created. In five years, 3000 transactions will be created and added to the blockchain due to employee turnover. Changes to the attributes could cause another 1500 transactions, for a total of 7500 potential transactions on the blockchain in five years.

Based on the example above, with organization sizes varying from ten to one thousand employees having an average of thirty attributes each, the data sets used for testing the blockchain implementation will be the following:

- 750 transactions for a company with 10 employees
- 7500 transactions for a company with 100 employees
- 75000 transactions for a company with 1000 employees

While the most significant amount of data lies within the transactions, there is also the encapsulating block's header and metadata to consider. The number of transactions that can fit within one block is limited by the "batchcount" and "wait" parameters, meaning that the size of the blockchain can be reduced by increasing these parameters, thus including more transactions per block. In the previously mentioned example with 7500 potential transactions, a block configured to store a maximum of fifty transactions would result in a minimum of 150 blocks and even more if the transactions are distributed over time. Thus, the blockchain's size will vary based on how many transactions can be in each block. Testing for blockchain size reduction with transaction count

should be carried out in an identical environment; thus, the following parameters were chosen for testing purposes to ensure a degree of variability:

- 1000 transactions with a transaction count per block of 10
- 1000 transactions with a transaction count per block of 25
- 1000 transactions with a transaction count per block of 50
- 1000 transactions with a transaction count per block of 100

The time it takes to create a transaction should influence how long the orderer waits for further transactions in the block creation process. The reasoning for this is because it should be possible to create the transactions within the time frame specified in block creation to benefit from the expanded transaction count of the block.

Transaction Creation

Adding a new user or attribute to the system requires a batch of attribute assignments to give access rights; a quick and efficient transaction creation process would allow users to work seamlessly without delay. However, a lower block creation time would mean increased storage requirements due to fewer transactions per block. Thus, the block timer and batch count should be adjusted to the specific organization's requirements, balancing speed and storage.

Efficiently assigning a user's attributes may improve workflow, but revocation relies more on speed to stop an attacker from abusing access rights. Revocation also influences the debate for speed over storage but adds a slight favoring for security to speed. Even ten seconds could be enough time for an attacker to retrieve critical information, especially if the intrusion was discovered late. If we consider security and efficiency, the block creation timer should be low, and batch size should be large; that way, the block could be filled with hundreds of transactions in a short time. Thus, the speed of creating a transaction will influence the amount of transaction data in the block, reducing the overhead of the block data.

Attribute Verification

The last part of the blockchain implementation is the verification procedure. Attribute verification is the component that determines whether or not someone has the attributes required to access the system and its data. When a user requests access to a resource, the system will check the Access Policy and the blockchain for assignment transactions for the necessary attributes. This process may take some time and varies depending on how many transactions the system must query to satisfy the access policy. Thus, the impact the amount of transactions on the world state has on performance must be measured. Testing the verification process will be done using the previously stated parameters of 750, 7500, and 75000 transactions, of which revocations account for 20% through employee turnover, leaving only 60% valid attribute assignments for each parameter.

The verification process should be as fast as possible while still maintaining the necessary level of security, but any AC system should take the human component into consideration. In leisure, users have been found to tolerate a waiting time of up to two seconds [21], hence the processing time to satisfy an access policy through attribute queries should preferably be less than two seconds for the proposed model to be considered a success.

6.2.2 Token Parameteres

When it comes to permission token performance, multiple factors weigh into any potential parameters or results. These factors can be split into three parts; generation, verification, and transfer time. The generation and verification parameters would have to be coupled with the data transfer rate of the potential implemented ABAC solution and combined after that with the blockchain solution to form a complete picture of the system's performance. In order to set some preliminary performance parameters to investigate, some initial results from the system and any system limiters are necessary to be investigated.

Data Transfer Rate

The first potential performance parameter of the system would be the data transfer rate of which the token itself can be transported between users and the system. If we look at various technologies that could be in use in modern businesses with the expectation that they use somewhat outdated protocols, their data transfer speeds set the lower bound, and we get the following:

- The Wi-Fi 802.11b standard released in early 2000 has a data transfer rate of 11Mbps.
- The USB2.0 standard was released in April 2000 and supported a 480Mbps transfer rate.
- A LAN environment with a standard Ethernet connection supports up to 100Mbps speeds.

Which shows that the 802.11b Wi-Fi standard is the lower bound for data transfer speeds for identifying performance parameters related to the implementation.

Size Limitations

The JWT standard, as described in RFC 7519, has no standard upper limit on the size of the token itself. However, because a JWT is typically included in an HTTP header, the various web server technologies set upper limits to the size of their header fields:

- Apache 2.0 and 2.2 have an upper limit of 8KB[22].
- Nginx specifies 4-8KB[23].
- The Internet Information Services (IIS), depending on the version, has a limit of 16KB[24].
- Lastly, Tomcat specifies a header limit of 8 - 48KB[25].

That leaves us with a lower bound of 8KB and an upper bound of 48KB for a standard HTML header, in which the JWT token is typically included, and a transfer rate of 11Mbps for the implementations' lower performance bound.

Key size & Hashing algorithm

Moreover, the key and hash could impact the performance of the implemented solution. The delegation token key size will depend on the hashing algorithm used for the token signature, which means that the size of the key will always have to be lower than the hash size. As the prototype is designed to utilize ECDSA to sign and corresponding EC keys, we get the following potential performance parameters that need to be investigated:

- ECDSA256 can only use 256-bit keys.
- ECDSA384 can utilize 256 and 384-bit keys.
- ECDSA512 can use 256, 384, and 521-bit keys.

If we assume maximum security for the upper bound of performance testing, we are left with ECDSA512 and a key size of 521 bits; however, all six combinations of supported key and hash sizes should be investigated for potential performance impacts.

6.3 Results

In the following section, we have illustrated the obtained results based on the previously detailed performance parameters concerning the implemented blockchain-based ABAC with a delegatable permission token solution, thus fulfilling Research Objective 4 (RO4):

RO 4: *Obtain the results for the identified performance parameters and thereby provide evidence on the practicality of using Blockchain and delegatable tokens for access control purposes.*

6.3.1 Blockchain Results

The blockchain prototype was tested using the parameters from Section 6.2.1, and the results will be displayed in graphs in this section. It should be noted that these results are influenced by the hardware used, detailed in Table 6.2, utilizing more powerful hardware would reduce the processing time considerably.

Before we get into the results, some of the terminology used must be clarified. The time a task takes to complete from the user's perspective is referred to as "real time" and includes the waiting time for other processes running on the operating system. In contrast, "process time" is the time spent by the system executing the task without waiting for other processes. While "assignment number" and similar variants are the number of times a task has been run. Next, the results use a standard block creation timer of two seconds and a block batch size of ten unless otherwise stated.

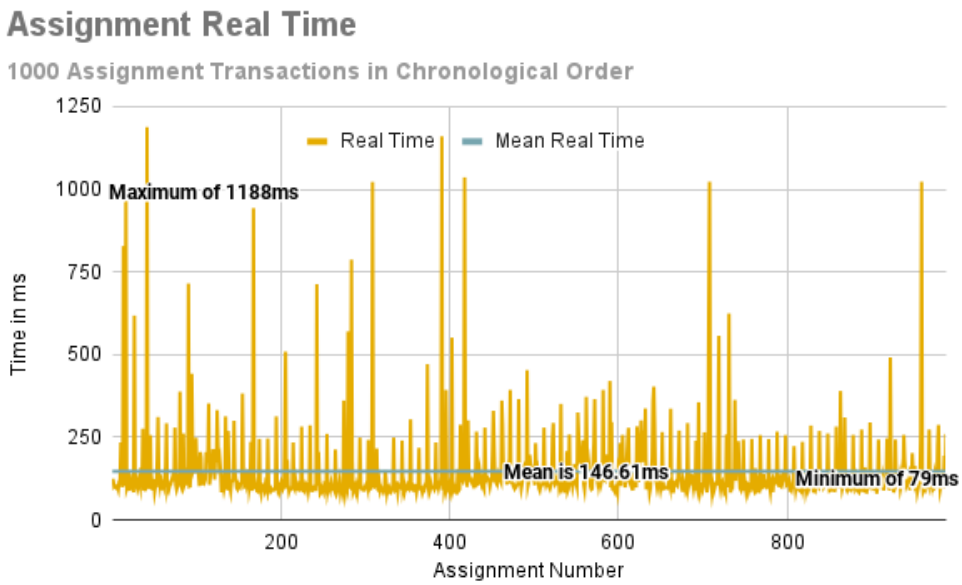


Figure 6.1: Time passed for the user in Assignment testing.

Assignment Creation

The results in Figure 6.1 show the time used by the assignment task in real-time. The peaks on the graph are likely due to block creation and distribution slowing down the transaction process but could also be related to scheduling process time on the operating system. The maximum value of 1188ms creates an upper bound for how long the user would have to wait, meaning only a single transaction would be included in the block when using a two-second block timer. However, it is more realistic to use the mean to determine how many transactions are in a block, which would allow thirteen consecutive assignments within two seconds. The minimum time of 79ms is likely due to an almost perfect scenario of uninterrupted process scheduling and would almost double the number of transactions created compared to when using the mean.

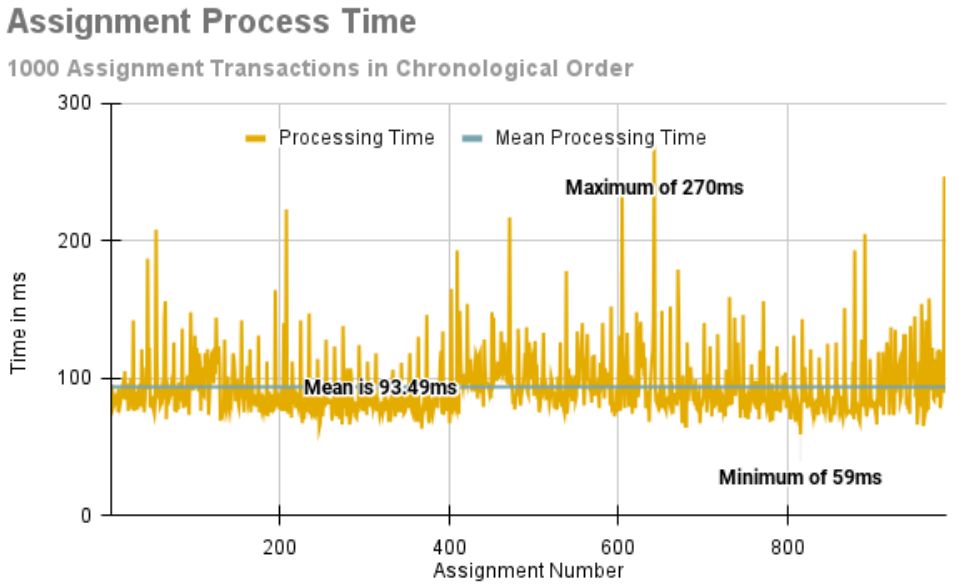


Figure 6.2: Processing time of Assignment function.

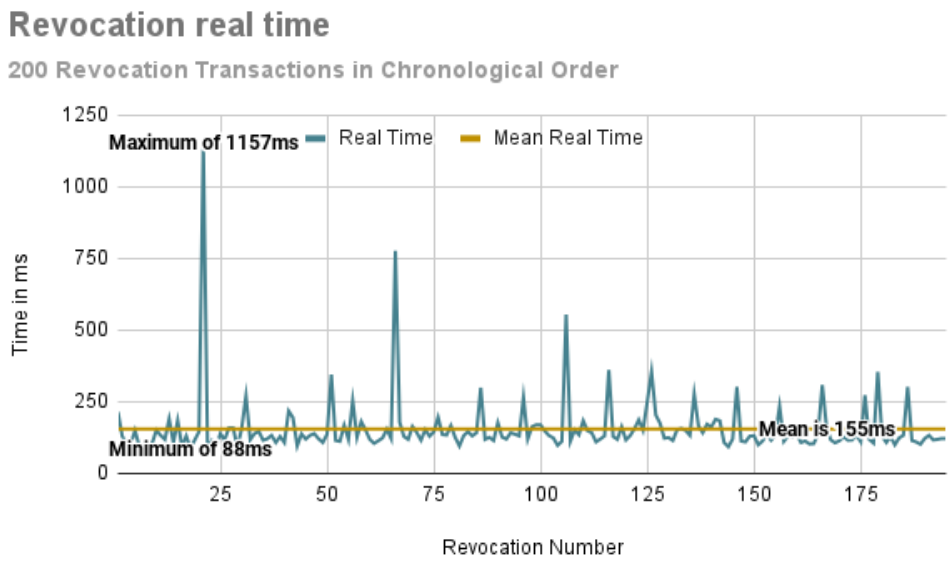


Figure 6.3: Time passed for the user in Revocation testing.

Next is Figure 6.2 which provides some insight into how long the process has to wait for scheduling when compared to Figure 6.1. Results show that peaks in processing time are spaced out roughly every ten transactions, aligning with the batch size limitation on block creation, concluding that the orderer's block creation process is included in the overall processing time. Various anomalies related to the peaks in processing time can be attributed to blocks that met the time limit before the batch size limit. The disparity between processing time shown in Figure 6.2 and real time shown in Figure 6.1 means that using a specifically designed system architecture prioritizing blockchain tasks could reduce the waiting time of a task significantly.

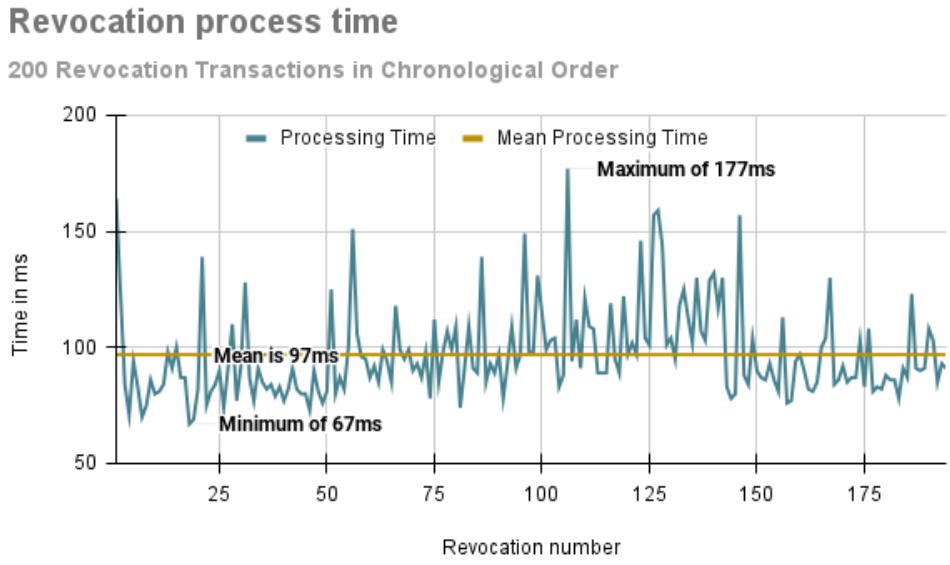


Figure 6.4: Processing time of Revocation function.

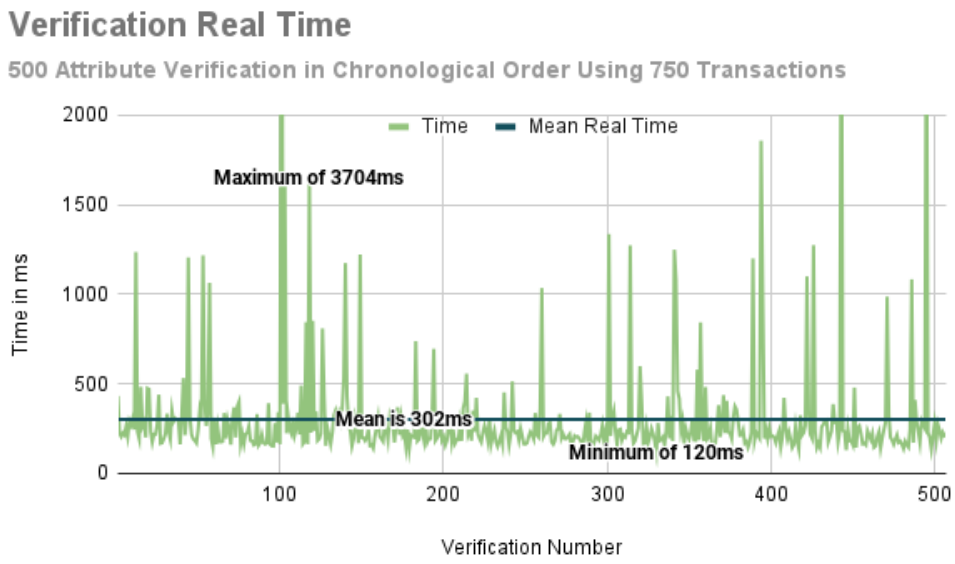


Figure 6.5: Time passed for the user in Verification testing.

Revocation Transaction

Revoking a compromised user's attributes should be a quick process and requires a low block creation timer, as discussed in Section 6.2.1. The results in Figure 6.3 show that a revocation task may take as long as 1157ms or as little as 88ms. Considering the average attributes assigned to a user being thirty, even the minimum real-time for a revocation transaction creation would not include all the transactions in one block given the two-second limit. However, using the mean time and multiple blocks would still revoke up to 64 attribute assignments within ten seconds, being well within the criterium. Furthermore, Figure 6.4 shows that the time it takes to fulfill a revocation task could be lowered with the use of a specific system architecture prioritizing blockchain operations, like with attribute assignments.

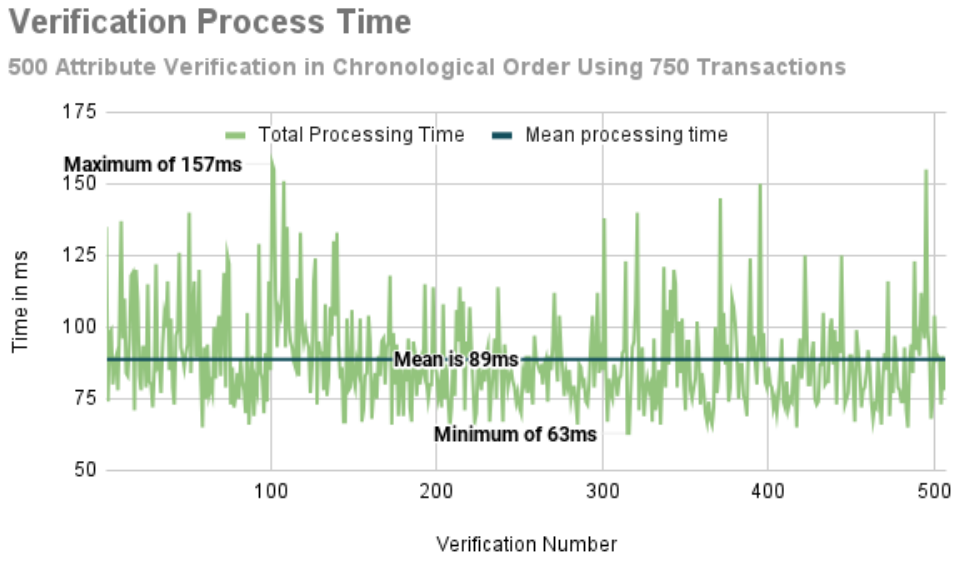


Figure 6.6: Processing time of Verification function.

Blockchain Verification

Verification time is an important factor in the blockchain-based ABAC system since it would be the most impactful element on most users. Results from Figure 6.5 show a maximum of 3704ms when searching for a single attribute on the world state, which is higher than the two-second verification time needed for a successful implementation, defined in Section 6.2.1. However, verification times longer than a few hundred milliseconds is the exception, and using the mean or the minimum verification time is a success.

Once again, there is a discrepancy in the real-time and the processing time shown in Figure 6.6, but this time there is no block creation taking place, only a world state search that is not included in process time. While it is probable that an optimized system to impact the verification time, it is not concluded so because the searching time is part of the verification proper, but not the processing time.

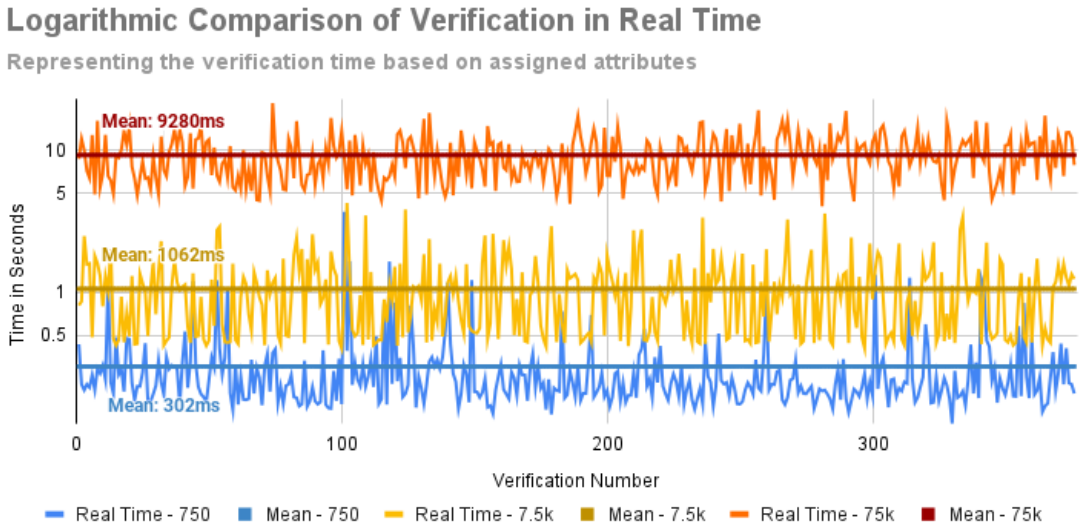


Figure 6.7: Logarithmic comparison of Verification function.

Verification sees a fluctuation in time to complete based on assignment count; the more attribute assignments there are on the world state, the longer it takes to verify one. The impact of transaction count on verification time can be seen in Figure 6.7, in which the three size parameters stated in Section 6.2.1 are used. The world state in the 750 transaction example has 450 valid attribute assignments and takes an average of 302ms to verify one single attribute, and for every step up in transaction count, the verification time increases as well. Waiting for three seconds once in one thousand times is reasonable, but an average waiting time of nine seconds is unacceptable. Thus the verification component is successful in smaller world states but fails when there are too many attribute assignments to search through.

Blockchain Size

The blockchain size results, tested with the parameters from Section 6.2.1 are shown in Table 6.3. While there is a definite decrease in size depending on how many transactions are contained in each block, the block's header and metadata do not have a significant enough impact to warrant further investigation.

Blockchain size by batch size				
Batch size	10	25	50	100
Blockchain size	6.169MB	6.168MB	6.167MB	6.167MB

Table 6.3: Blockchain size by batch size using 1000 transactions

6.3.2 Access Token Results

In this section, we have provided various charts and figures to showcase the results of the delegatable token implementation. As one of the defining performance parameters of the token comes in the form of its various supported key and hash size combinations, as previously explained, each graph will have the data set for each combination to showcase the overall performance of the implementation while simultaneously drawing conclusions based on them.

Each data parameter has been named in the following manner: "Type_KeySizewHashSize," where the first part identifies the type of data (verification, generation, or size), the second part shows the key size in bits (256, 384, or 521), and the last part shows the size of the SHA hash of the ECDSA signing algorithm (SHA256, SHA384, or SHA512). An example would be the size graph of a 384-bit token using the SHA512 hash algorithm with a naming identifier of "Size_384w512".

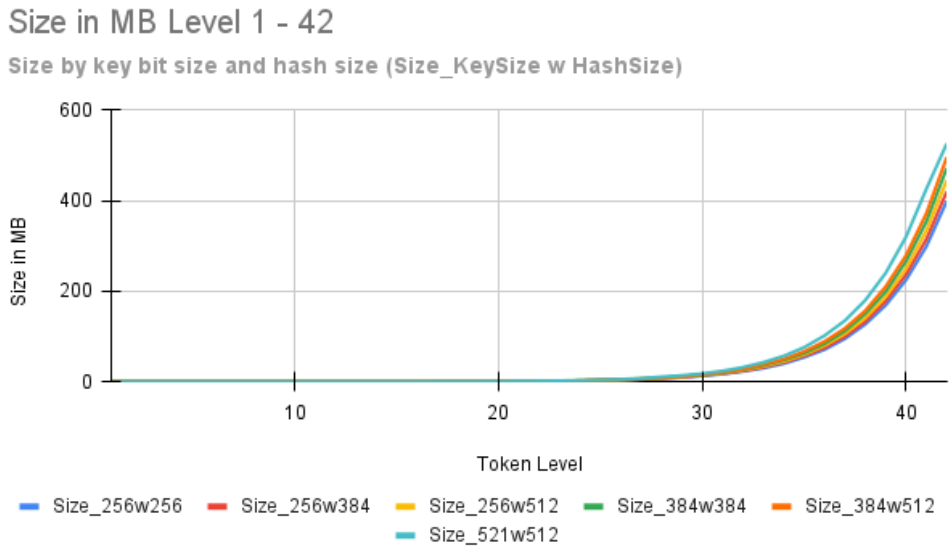


Figure 6.8: Delegation Level 1 - 42, shown in MB

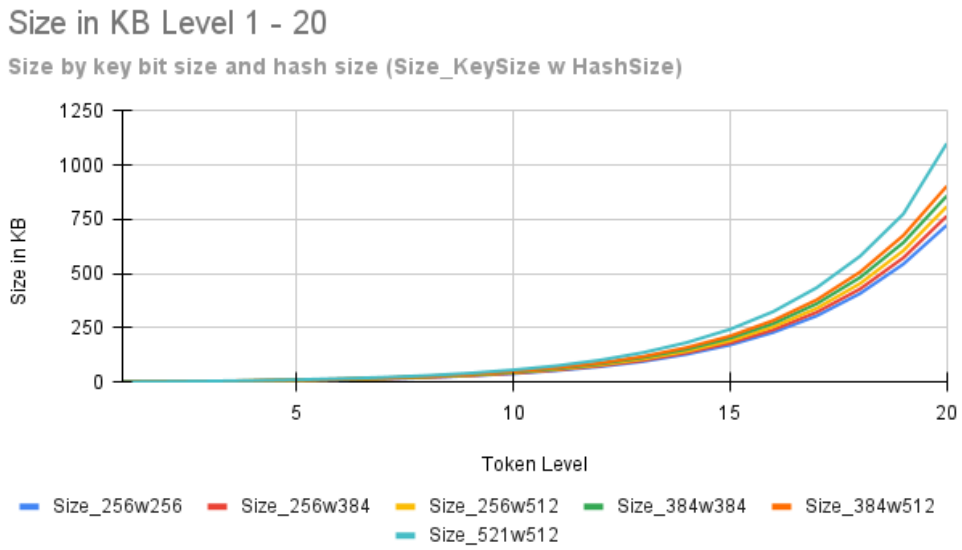


Figure 6.9: Delegation Level 1 - 20, shown in KB

As the highest possible token to be generated using the implemented system is a token of 42nd degree, the dataset used to generate the tokens go from 1 through 42 to showcase the performance results of the entire implementation, even if the higher levels of nested tokens are not realistic for deployment.

Size Limitations

The first and arguably the most crucial performance parameter in relation to the delegatable token implementation is the upper and lower bounds of supported sizes. As previously mentioned, the various web server applications have default or maximum allowable HTML headers for their solutions, which means that the proposed token must not be larger than the specified size limitations in their respective documentations.

As detailed in Chapter 4, each nested token contains all lower-level variants, which means that the size of the implemented token increases exponentially up to the performance ceiling of 42. Figure 6.8 shows the increase in token size as measured in MegaBytes and shows that the last few generated delegation levels can reach sizes of 500MB. This increase in size can already be seen in the lower token levers, as shown in Figure 6.9, where the graph incline begins building quite rapidly at delegation level 15.

However, as the previously stated performance requirements for size were 8, 16, and 48KB, respectively, we need to look at those lower and upper bounds in more detail. Figure 6.10 details the Apache and Nginx lower bound limit of an 8KB HTML header. The graph shows that only two combinations can reach the 5th level while still keeping under the lower bound of 8KB: 256-bit keys using SHA256 and 256-bit keys using SHA384. The hash size contributes significantly to the overall token size, as the 256-bit key with SHA512 is too large to fit within the lower bound. These

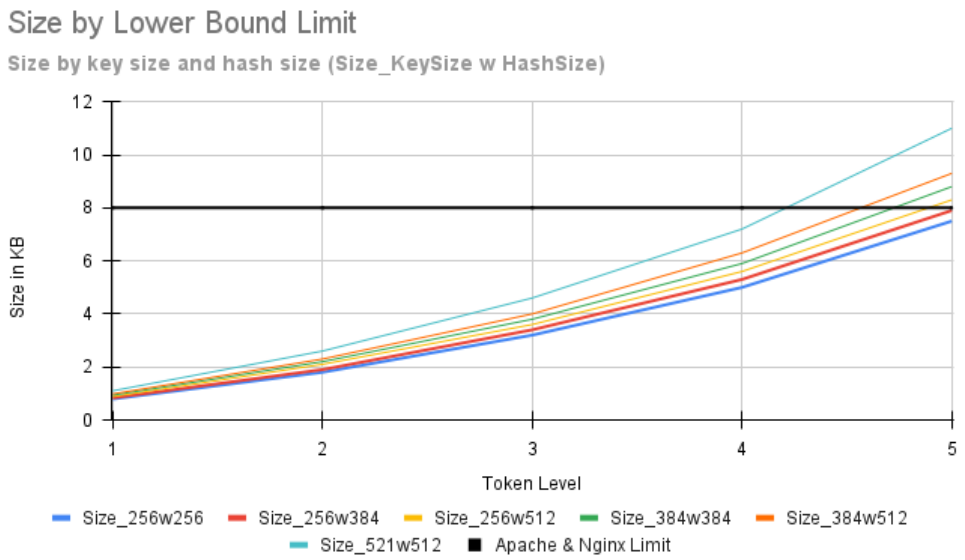


Figure 6.10: Lower Bound of 8KB

results show that all six combinations can reach the 4th level, meaning a token can be delegated four times without encountering size issues.

The upper bound of 48KB can be seen in Figure 6.11, where the only two combinations that exceed the upper bound were 384bit and 521-bit keys using SHA512. This result shows us that all token variations except for the two largest (384bit and 521-bit) keys using SHA512 can use a token of 10th degree; otherwise, all token variations support tokens of the 9th degree.

Based on these results, we can conclude that to satisfy the lower size bound of 8KB, thus ensuring a token fits within those transport protocols as previously mentioned, the token delegation level must be limited to the 5th degree, as shown in Figure 6.10. Should the implemented

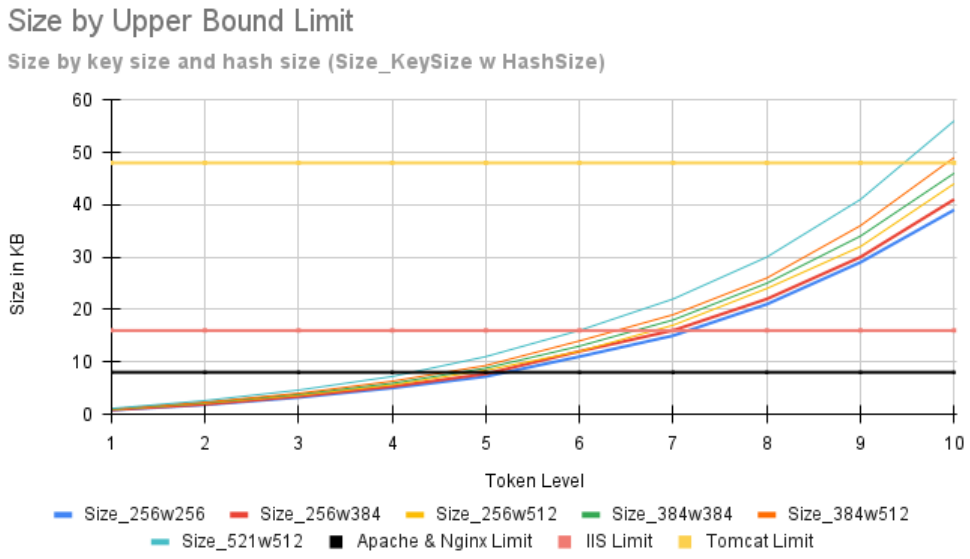
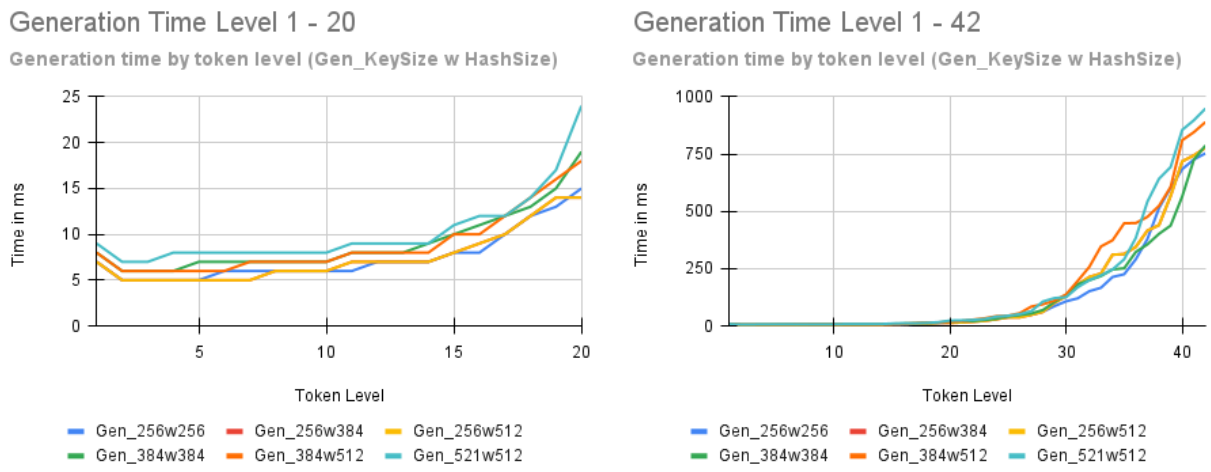


Figure 6.11: Upper Bound of 48KB



(a) Delegation Level 1 - 20

(b) Delegation Level 1 - 42

Figure 6.12: Graph showing the generation time of the token implementation.

transport protocol be Tomcat with its header limit of 48KB instead, the token application could at most support delegations up to the 10th degree.

Assuming, as previously stated in Section 5.2.1, we follow the NIST recommendation for a key size of 2048bit RSA keys or 224bit EC keys, we can safely say that 256bit keys with SHA512 for signing for the highest degree of security can support a delegated level of 4 to satisfy the lower bound, or delegation-level 10 for the upper bound.

Generation Performance

As previously specified in Section 5.2.2, the actual procedure to generate a token is two-fold. First, the user will specify various information, then the application will generate the token and sign it

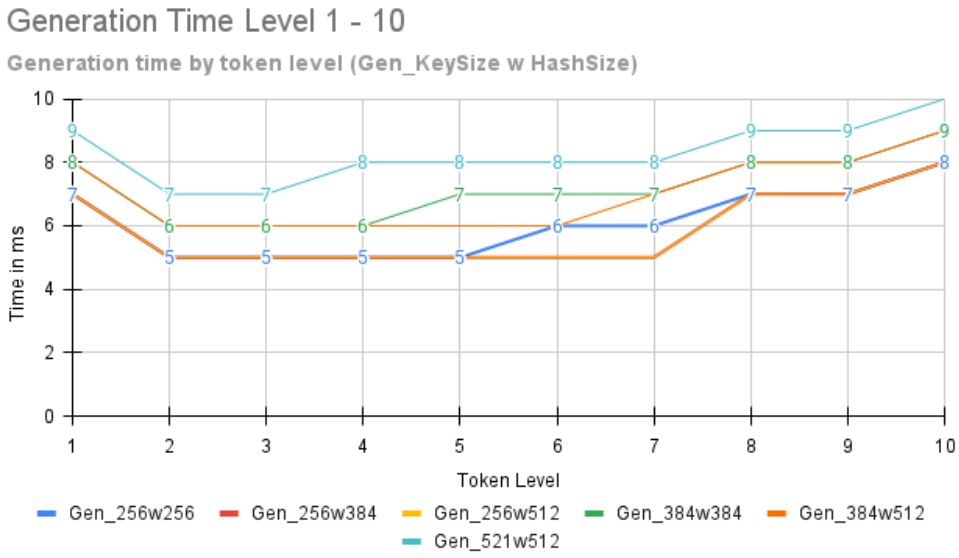


Figure 6.13: Token generation time level 1 through 10.

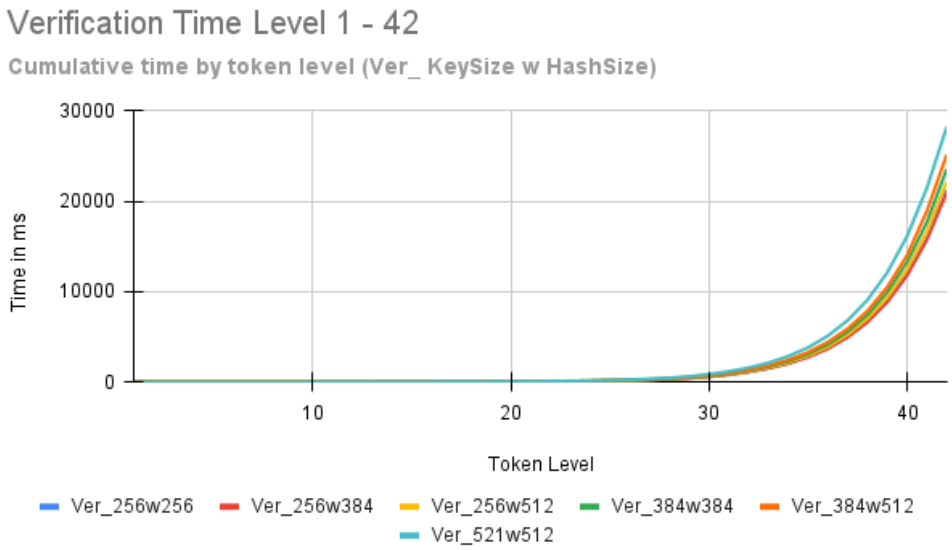


Figure 6.14: Delegation Level 1 - 42

using the ECDSA with the specified hash size (SHA256, SHA384, or SHA512). That means that the most significant performance impact on the token generation time will be the key size and hash algorithm, as previously stated in the performance requirements.

Figure 6.12 (b) shows the overall time it takes to generate the various delegatable tokens showing levels 1 through 42. As shown in the graph, the results are consistent until the 20th-degree mark, when the different combinations of keys and hashing algorithms start to differ in time. Figure 6.12 (a) shows the various times it takes to generate tokens level 1 through 20, where the largest supported key size of 521-bits and SHA512 takes the longest, while the others are more evenly distributed.

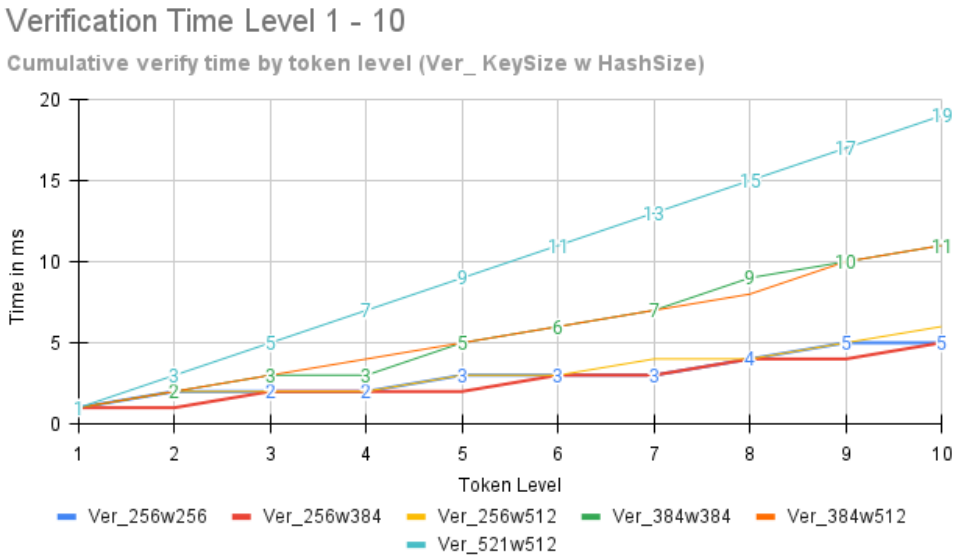


Figure 6.15: Token verification time delegation level 1 through 10.

However, as previously concluded with sizing restrictions in the last part, the highest supported token needs to be limited to a 10th level token. Figure 6.13 shows the performance data related to token generation for tokens of 1st through 10th degree. The early drop in timing can be attributed to the fact that more information needs to be specified when generating a first-level token than for a second-level token, which merely takes the first and generates a second. The overall performance that Figure 6.13 shows a maximum generation time of 10ms overall regardless of which key size or hash algorithm is used. However, this can be halved to 5ms for tokens utilizing the recommended 256bit key size with SHA256 or SHA384 as the hashing algorithm.

Verification Performance

As Section 4.2.3 mentions, once the ABAC system receives a permission token, it is forwarded to the Token Administration Point (TAP) for verification. The verification process essentially iterates through all potentially nested tokens, ensures it is within its expiration time and validates any signature within the token to verify authenticity.

As a 42nd-degree permission token is the highest supported delegation level of the implemented system, Figure 6.14 shows the cumulative verification time of a 42nd level token, where it exponentially increases as the delegation level goes up. Because of the extreme ramp-up in verification time after delegation level 30, Figure 6.16 shows the cumulative verification time of tokens 1 through 20, where the increase in time is somewhat more gradual, having only about a four times increase in verification times between level 10 and 20, versus some 30-time increase from level 30 to 40.

Furthermore, Figure 6.16 quite elegantly shows how as the key size increases, so does the

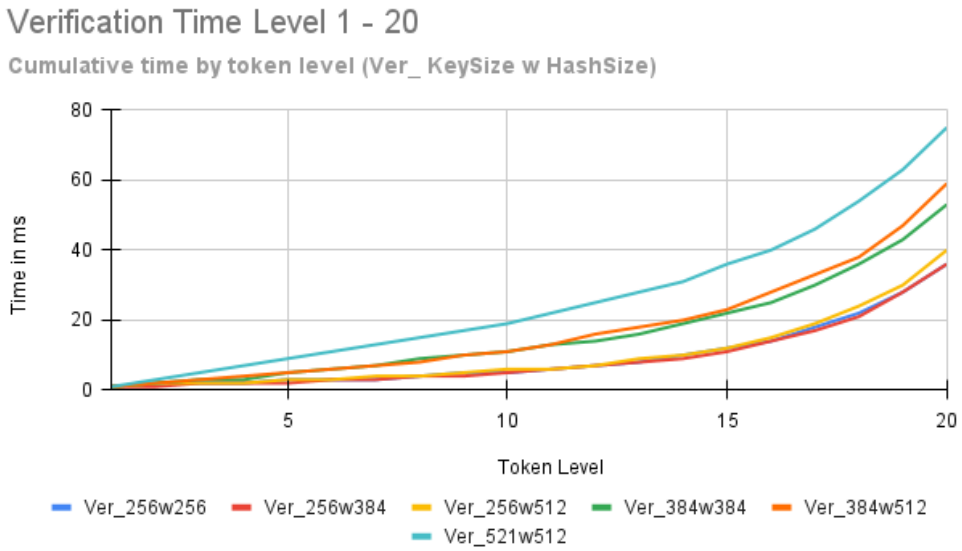


Figure 6.16: Delegation Level 1 - 20

verification time, as all 256-bit keys are clumped together and have about a 20ms verification time at 20th level, the 384-bit keys conform around 55-60ms, and the 521-bit key surpasses all previous key sizes with a verification time of almost 80ms.

The conclusion regarding the size requirements of the HTML headers' upper and lower bounds, shows that the implementation will potentially be limited to a 10th-degree level token. Figure 6.15 shows the verification times in detail for delegation levels 1 through 10, where each matching key size and the hash algorithm is highlighted with their respective performance numbers. If we take the previously concluded result based on NIST recommendations and focus on the 256bit key using the SHA512 hashing algorithm, we can analyze its performance from Figure 6.15. For the lower bound of an 8KB header, the token would be limited to a 4th-degree delegation, in which case the verification time would only be a measly 2-3ms. If we focus on the upper bound of 48KB and its corresponding 10th level token, we get the verification performance of around 5ms when using a 256bit key.

Token Performance Analysis

This section combines the previous size, generation, and verification results, with the various transfer rates concluded previously to illustrate the overall performance of the token implementation. To accurately analyze the application's performance, the lowest transfer rates of 11Mbps will be combined with the upper bounds of the transport protocols of 48KB. Due to limitations in token size from the upper bound of 48KB, delegation-level is also limited to the 10th degree. Therefore, delegating access from a 9th-degree access token to a 10th is the slowest process the system must accommodate. Using these numbers, we can calculate the expected slowest transfer speed of the highest delegation-level the system can accommodate:

1. 11Mbps is equivalent to 1.375MBps and 1375KBps.
2. Transferring a 9th-degree token of 32KB with a transfer rate of 1375KBps would then take 24ms.
3. Transferring a 10th-degree token of 44KB with the lower bound transfer rate would then take 32ms.

If we go back to the thesis scenario and assume a Manager wants to generate a 9th-degree token, as shown in Figure 6.7, that would take 8ms. If we combine the generation time with the transfer time, it will take (8ms + 24ms) 32ms for the system to generate the Managers token and transfer it to his Employee. If the Employee wishes to delegate access to an outside consultant further, he will generate the maximum supported 10th-degree token taking 9ms and transfer it to the Consultant for a total of (9ms + 32ms) 41ms to re-delegate his access.

When it comes to verification, the results can be seen in Figure 6.10, where our Consultant has just received a 10th level access token and attempts to gain access to the system. Using the

previously calculated transfer rate of 32ms for a 10th-degree token, the graph in Figure 6.10 shows us that the verification process takes 5ms to validate the token authenticity, totaling 37ms.

If we combine the results and assume no delay between the parties, we can see that the Manager takes 32ms to generate and transfer to his Employee, which takes an additional 41ms to re-delegate to the Consultant, which finally takes 37ms to transfer the token and get it verified by the system. Thus, the results combined are 110ms from the Manager generating the delegation token to the system verifying the Consultants' token authenticity.

An important thing to note regarding the concluded result is that this is the approximate time it takes for the various system components of the proposed model to generate, transfer, and verify the access token. A fully implemented Blockchain-Based ABAC system would potentially have a higher performance impact than the results given, as it would include mutual authentication between users and various other performance aspects of the ABAC system. However, these parameters are outside the thesis scope and are a performance analysis left for future work.

The next section of the thesis will combine these acquired token results with the performance data from the blockchain implementation to give a complete view of the proposed system performance.

6.3.3 Prototype Performance Analysis

Looking at the combined results of the token and blockchain performance analysis, we can investigate whether or not the two parts of the proposed model are a viable solution for deployment, and can be seen in Figure 6.17. As mentioned in Section 6.2.1, users can tolerate a waiting time of up to two seconds, meaning the combined performance of the delegated access token and blockchain prototypes must fit within these parameters to be a viable solution.

Section 6.3.2 uses the highest delegation-level the token prototype supports, combined with the lowest transfer speeds, to conclude that the three-step process between a Manager, their Em-

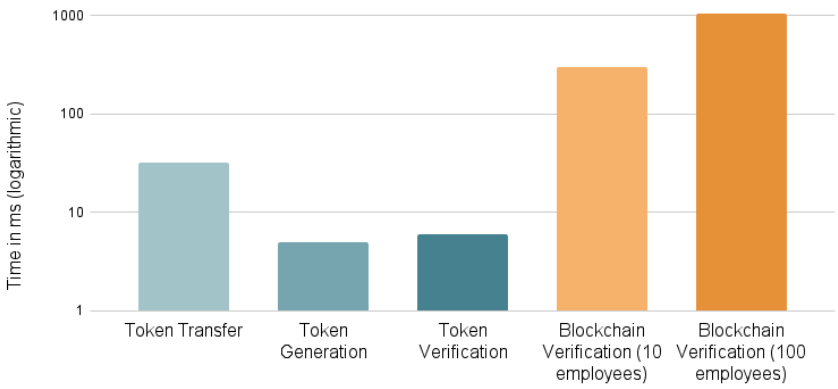


Figure 6.17: Combined performance results.

ployee, and a Consultant takes 110ms to complete. As detailed in Section 6.3.1 and shown in Figure 6.7, utilizing the blockchain implementation with ten employees, takes an average of 302ms to verify a single attribute on the world state, and 100 employees give an average of 1062ms. The same blockchain results also conclude that the upper parameter of testing performed with 1000 employees takes an average of 9.3 seconds to verify a single attribute, thereby not within the viable limits of two seconds. If we take the two potential blockchain performance parameters of 10 and 100 employees, we can combine the verification times with the delegated token performance to show the system's overall performance.

Seeing as the previously concluded example in Section 6.3.2 only took 110ms, and verifying an attribute in a system with ten employees takes 302ms, it is safe to conclude that at this employee level, the performance of the proposed solution is viable with a total time of 412ms. This result is not including the out-of-scope parameters of an ABAC system. Given the example in Section 6.3.2 with the Manager that delegates access to an Employee, which further re-delegates to a Consultant, we can see that the delegation verification time of 110ms only accounts for about 10% of the total performance if paired with the highest supported blockchain parameter of 100 employees taking 1062ms.

With the token generation and verification time being low relative to the blockchain processing time, it is more or less possible to base the overall system viability primarily on the blockchain performance. Thus, the second blockchain performance parameter of 100 employees is also a viable solution, while the third tested parameter of 1000 employees is outside the acceptable performance window using the currently available testing hardware. However, this also means the proposed delegation model is a viable solution, regardless of the number of transactions on the blockchain, since it constitutes such a small percentage of the overall performance using the lowest supported transfer speeds and the highest supported delegation levels of the system.

6.4 Security Analysis

Multiple potential attack vectors towards both the blockchain and token implementation and the model itself need to be analyzed. Table 6.4 shows an overview of the analyzed attack vectors and whether or not the various attacks are possible on the proposed model. The following part will list the various potential attack vectors and discuss whether or not the proposed model could be vulnerable to such attacks in detail. Before the attack vector analysis, there are many integral components to the proposed model that need to be discussed, namely attributes and the signing algorithms.

In the proposed model and implementation, trusted attributes are utilized through a blockchain implementation to authenticate and authorize users based on access policies in an Attribute-Based

Security analysis of various attack vectors	
Attack Vector	**Vulnerable**
Forging attributes	Not possible
DoS / DDoS	Possible (at model level, not prototype)
Sybil attacks	Not possible
51% attack	Not possible
Double-Spending	Possible (under some circumstances)
Man-in-the-Middle	Not possible
Insider Delegation	Possible
Routing attack	Not possible
Race attack	Not possible
Finney attacks	Not possible

Table 6.4: Security analysis of various attack vectors

Access Control (ABAC) scheme. The following scenarios detail potential attack vectors towards the trusted attributes of the system.

Can a user collude or otherwise forge attributes?

All potential attributes within the current proposed model are stored within a trusted Attribute Authorities' Attribute Repository. What this means, in essence, is that it should not be possible for a malicious third party to be capable of generating attributes or otherwise modify the Access Policies to be allowed access based on their already owned attributes.

A user cannot attempt to forge an attribute and append it to the blockchain because each attribute assignment must be signed by the Attribute Authority, which means that a potential malicious third party would have to forge the attribute and the signature of the AA. The signature scheme utilized by the token and blockchain implementation both use the ECDSA signing algorithm, which according to NIST [26], is unforgeable if the hash algorithm used is that of SHA2, which is equivalent to the hashing algorithm used by the proposed scheme of SHA256. This unforgeability also extends to the delegatable token model as it uses the same signing algorithm as the blockchain implementation. Therefore a potential malicious third party can forge neither the tokens nor the attributes under the proposed model.

Denial of Service (DoS) attacks

A malicious actor could overwhelm a node with requests in a DoS attack, thereby preventing legitimate users or systems from utilizing the node. Within the current proposed model, whenever a user wishes to access a resource, either through their attributes or a delegated token, the requests come in through the Policy Enforcement Point (PEP), that further queries parts of the system, thereby also the blockchain, for whether or not the user should be authorized access. Therefore, a malicious actor could overwhelm the first point of access, namely the PEP, meaning the proposed Blockchain-based ABAC system as a whole would be vulnerable to this attack vector. However, the

blockchain implementation itself and its nodes would not be vulnerable, as only an administrator or trusted system component could query the nodes on the blockchain.

Sybil and 51% attacks

A Sybil attack would be where a node uses multiple different identities to get control over a large portion of the network in a standard blockchain model. Sybil attacks are not a vulnerability in the proposed model, as each node is a trusted part of the system and not controlled by third-party systems. By extension, that also means that "51% attacks", where an actor gets control of more than 50% of the computing power of a network, is also not possible in the proposed solution.

Double-Spending attacks

In a traditional cryptocurrency blockchain model, a double-spending attack is where an actor performs multiple transactions with the same currency, which does not apply to the proposed model. However, a malicious actor could attempt a double-spending attack using the delegated tokens and attempt to access the system on two different access points using the same delegated token. As this is outside the scope of the project, it is not possible to conclude for sure whether or not a double-spending attack with tokens would be possible or not, as it would most likely create a race condition on whether or not two tokens can be processed simultaneously and therefore both be granted access before the token is added to the blacklist of the system.

Man in the Middle attacks

A Man in the Middle (MitM) attack is where an attacker places themselves between the communication of two parties and can modify or block certain communications. When two parties mutually authenticate with each other to exchange tokens, there is a possibility that the legitimate user is communicating with a false identity of an attacker using a forged certificate. As the proposed system utilizes a trusted Certificate Authority through the Attribute Authority and employs ECDSA signing algorithms, it is impossible to forge the system's signature, so it is impossible to forge an identification certificate to use in a MitM attack. If, however, that was possible, the token's issuer still specifies the receiver in the token generation before signing it. If the user requesting access through a token does not match the "receiver" parameter of the issued token, the authorization process is terminated. Hence, it is impossible to perform a man-in-the-middle attack through the proposed permission token model.

Insider Delegation attack

Assuming a user has verified attributes in the system and wants to generate a delegated permission token to a third party, there is no verification check on the receiver's credentials within the current model. The model assumes that it would only be possible to delegate to trusted users within a pre-defined domain, but no restrictions have been put in place as the model is currently proposed.

Therefore it could be possible for a verified insider to delegate access to a malicious third party and gain access to the system.

Routing, Race, and Finney attacks

A malicious actor could perform various other common attacks on a blockchain, namely Routing, Race, and Finney attacks. Routing attacks are where an attacker intercepts and modifies a message on the blockchain; Race attacks are where a user generates two transactions and uses the first transaction legitimately, and broadcasts the second transaction before the first one is verified. Lastly, Finney attacks are where an attacker creates two transactions where one targets the legitimate user and one target himself. The common denominator of all these attacks is the requirement of direct user input on the blockchain, which from the proposed model is not possible for outside third parties. All updating, modifying, or querying the blockchain happens strictly through a trusted channel of system components such as the Attribute Authority, Policy Enforcement Point, or a systems administrator. At no point does a client or user directly communicate with the blockchain. Therefore these attacks are not possible with the proposed model.

6.5 Meeting requirements

This section discussed how well the implementation meets the original requirements based on the results detailed in Chapter 6 and attempt to answer the various Research Questions stated in the introduction of the thesis.

The informal requirements of the proposed implementation were that it had to support a blockchain to store various transactions and a token scheme to delegate access to users outside the organization domain. The implemented prototype fulfills these requirements by utilizing Hyperledger Fabric and a world state database to represent the blockchain aspect of requirements, which the results show is a success given the current hardware limitations. Furthermore, the proposed token scheme can generate tokens within a few ms, and the system can verify the token's authenticity within a short period; thus, the system fulfills the requirements, and delegation of access is possible within the proposed model.

The four Research Objectives stated in Section 1.2.2 have been fulfilled through the various relevant parts of the thesis and are mentioned at the beginning of the appropriate chapters. Section 1.2.3 poses three Research Questions, of which the following parts will directly answer:

RQ 1: *How can we implement a prototype for the Blockchain-based ABAC scheme with permission delegations proposed in 1.2?*

The process of implementing the proposed model is showcased in Chapter 4, where Hyperledger Fabric is utilized for the blockchain aspect and Java Web Tokens for the delegatable access part. The different results shown in this chapter serve as proof that the proposed implementation is a success and that both the token and blockchain features of the prototype work and provide concrete results about the system performance.

RQ 2: *How can we optimize the implementation so that it is possible to achieve higher levels of performance?*

Various optimization aspects have been identified through the testing phase of the thesis. When it comes to the token implementation, the primary limiter is the size increase of the token as external factors limit it through the HTML header in the transport protocols. A solution to this problem was located during testing, where it was concluded that if the design of the token model were changed to only include the requested "action" and "objectID" in the first token, it would overall reduce the size of the consecutive levels. The rationale behind this conclusion is that only first-level tokens are checked for requested object and action parameters, and so including these in all tokens is a waste of resources.

Another area of access delegation optimization is reducing the key size used for signing the token, as a lower bit-sized key would reduce generation and verification time and reduce storage requirements. However, this change must be based on necessary levels of security, as the smaller the key size, the weaker the security.

The blockchain verification process searches the entire world state for all possible assignments that fulfill the query because there could be more than one instance of a user having an attribute. While the user should not have multiple copies of the same attribute assigned to them, it is also unnecessary to keep searching once one assignment has been found.

The verification process is slow, and one way to increase its performance is to adopt the searching function used in revocation. The revocation function, as covered in Section 5.1, searches the world state for existing assignments at a much faster rate than the verification process. The difference in speed could be due to the rich query utilized in verification searches, which could be mitigated by creating a composite key from the attribute and public key. A composite key would also remove the possibility of having multiple assignments of the same attribute to a user.

RQ 3: *Is Blockchain-based access control with permission delegation suitable and realistic for the application scenario specified in the thesis definition?*

Based on the results concluded in this chapter, the proposed model is realistic and suitable for deployment for the specified application scenario in the thesis definition. The delegatable token aspect of the implementation is a success even when using the upper limits of supported delegation levels taking a combined time of less than 100ms to generate and verify through the system. As

the blockchain performance increases with higher levels of transactions stored on the ledger, the token performance remains consistent regardless of the parameters used, meaning it is suitable for the proposed application scenario.

The blockchain implementation also shows that it can perform the necessary operations to store the various transactions necessary to operate an ABAC system. The proposed model supports up to one hundred employees with thirty attributes each through various revocation and assignment transaction combinations shown in the results using the current hardware available. As the hardware utilizing in testing is far from a realistic deployment scenario, the proposed model could likely support many attributes and transactions while still being within acceptable performance parameters, as concluded in Section 6.3.3.

Thus, the proposed Blockchain-Based ABAC scheme with delegatable permission tokens is suitable for the proposed scenario and potentially other similar scenarios if the proposed performance optimizations were implemented as well as upgraded hardware.

Chapter 7

Conclusion & Future Work

The thesis proposes a Blockchain-based ABAC scheme with delegatable permission tokens through the use of standardized technologies such as Hyperledger Fabric and Java Web Tokens. The project consisted of developing two separate prototypes to showcase the proposed model utilizing the aforementioned technologies.

Both visual and quantitative results show that the proposed model fits the application scenario, with delegatable token performance staying well under 50ms to generate, verify, or transfer the token. At the same time, the blockchain performance stays under the acceptable performance ceiling of two seconds as long as the application scenario is limited to 100 employees or less. Utilizing more powerful hardware would increase the number of supported employees and transactions for the blockchain. At the same time, the delegatable tokens show consistent performance results even for the upper bounds of performance analysis, meaning it is a suitable solution to delegating access in an AC system.

Furthermore, the results show that the model is scalable through the use of the proposed blockchain and delegatable tokens, thus solving some of the issues posed by related work mentioned in state-of-the-art. Testing the blockchain prototype shows it can support upwards of one hundred employees before reaching the acceptable performance ceiling. At the same time, the token prototype supports upwards of ten delegation levels before reaching the size limit of the transport protocols.

The security analysis reveals that the proposed model is resistant to most of the standard blockchain-based attack vectors investigated and many other typical attack vectors. The primary concern regarding storing the attribute transactions on the blockchain is whether or not a user can forge attributes and thereby gain access. The security analysis demonstrates this is not possible as the AA signs each attribute transaction through the ECDSA, which is unforgeable using the applied hashing algorithm of SHA2 in the model.

7.1 Future Work

Several potential improvements to the model and implementation have been identified throughout the project period. Most notably are the ones ascertained in Section 6.5 when answering Research Question 2 regarding prototype optimization:

- Redesign token model so delegated tokens after the first does not include unnecessary data parameters such as "ObjectID" and "Action," as the verification process only verifies the authenticity of each nested token and merely checks the first token for the requested access parameters.
- Modify the transaction implementation to utilize a composite key made from the public key and attribute as the unique identifier.

Furthermore, several other ideas and additional development aspects have been identified through the implementation and testing of the model. The most apparent work would be to combine the two prototypes by implementing an ABAC system and the various system components that would require. This development would allow for a complete analysis of the system's usability and provide more accurate performance data than is currently available through the implemented prototypes. Additional potential future work on the model or developed implementation are:

- Investigate the performance impact and usability of a locally stored blacklist versus the proposed centralized blacklist database, which is used to ensure access tokens are not re-used.
- The endorsement list in each transaction grows larger for every peer in the system, which should be tested and examined for problems.
- Public keys are long and could be replaced with pseudo-identities, which could decrease the size of both blockchain and tokens.
- The system has not been stress-tested if a significant amount of users and peers make assignments, revocations, and verification requests simultaneously, which could have unintended consequences and slow down the system.

Appendices

A Git Repository

Due to the nature of the project and the number of project files, they have been uploaded to a Git repository for further review rather than included in the appendix. Anyone can find the Git repository with the project files at the following link GitHub Repository. The repository is structured in two parts to contain the two proof-of-concept prototypes detailed in this report.

Bibliography

[1] *Gartner IAM 2020 Predictions*. https://www.avatier.com/products/identity-management/resources/gartner-iam-2020-predictions/. Accessed: 2021-1-20.

[2] Satoshi Nakamoto and Www bitcoin.org. *Bitcoin: A peer-to-peer electronic cash system*. https://bitcoin.org/bitcoin.pdf. Accessed: 2021-4-8.

[3] *25 blockchain applications & real-world use cases disrupting the status quo*. https://builtin.com/blockchain/blockchain-applications. Accessed: 2021-4-8.

[4] Damiano Di Francesco Maesa, Paolo Mori, and Laura Ricci. "Blockchain Based Access Control." In: *Distributed Applications and Interoperable Systems*. Ed. by Lydia Y. Chen and Hans P. Reiser. Cham: Springer International Publishing, 2017, pp. 206–220. ISBN: 978-3-319-59665-5. DOI: 10.1007/978-3-319-59665-5_15.

[5] H. Liu, D. Han, and D. Li. "Fabric-iot: A Blockchain-Based Access Control System in IoT." In: *IEEE Access* 8 (2020), pp. 18207–18218. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.2968492.

[6] H. S. Gardiyawasam Pussewalage and V. A. Oleshchuk. "Blockchain Based Delegatable Access Control Scheme for a Collaborative E-Health Environment." In: *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. July 2018, pp. 1204–1211. DOI: 10.1109/Cybermatics_2018.2018.00214.

[7] Elaine B Barker and Quynh H Dang. *Recommendation for key management part 3: Application-specific key management guidance*. Tech. rep. Jan. 2015.

[8] Nick Szabo. "Formalizing and Securing Relationships on Public Networks." In: *First Monday* 2.9 (Sept. 1997). DOI: 10.5210/fm.v2i9.548. URL: https://journals.uic.edu/ojs/index.php/fm/article/view/548.

[9] M A Chunguang et al. "Smart Contract in Blockchain." en. In: *Netinfo Security* 18.11 (Nov. 2018), p. 8. DOI: 10.3969/j.issn.1671-1122.2018.11.002. URL: %5Curl%7Bhttp://netinfo-security.org/EN/abstract/article_6708.shtml%7D.

[10] M. Wohrer and U. Zdun. "Smart contracts: security patterns in the ethereum ecosystem and solidity." In: *2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*. Mar. 2018, pp. 2–8. DOI: 10.1109/IWBOSE.2018.8327565.

[11] G. Gan et al. “Token-Based Access Control.” In: *IEEE Access* 8 (2020), pp. 54189–54199. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.2979746.

[12] John Bradley, Nat Sakimura, and Michael B Jones. “JSON Web Token (JWT).” In: *Proposed Standard RFC7519* (May 2015).

[13] *JSON Simple Sign 1.0 draft 01*. https://jsonenc.info/jss/1.0/. Accessed: 2021-4-20.

[14] Vincent C Hu et al. *Guide to attribute based access control (ABAC) definition and considerations*. Tech. rep. Jan. 2014. DOI: 10.6028/NIST.SP.800-162.

[15] Y. Zhu et al. “Cryptographic Attribute-Based Access Control (ABAC) for Secure Decision Making of Dynamic Policy With Multiauthority Attribute Tokens.” In: *IEEE Transactions on Reliability* 68.4 (Dec. 2019), pp. 1330–1346. ISSN: 1558-1721. DOI: 10.1109/TR.2019.2948713.

[16] Lei Xu et al. “Blockchain-based access control for enterprise blockchain applications.” In: *International Journal of Network Management* 30.5 (2020). e2089 nem.2089, e2089. DOI: https://doi.org/10.1002/nem.2089. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/nem.2089. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/nem.2089.

[17] H. Guo et al. “Access Control for Electronic Health Records with Hybrid Blockchain-Edge Architecture.” In: *2019 IEEE International Conference on Blockchain (Blockchain)*. July 2019, pp. 44–51. DOI: 10.1109/Blockchain.2019.00015.

[18] Gauhar Ali et al. “Blockchain based permission delegation and access control in Internet of Things (BACI).” eng. In: *Computers & security* 86 (2019), pp. 318–334. ISSN: 0167-4048. DOI: 10.1016/j.cose.2019.06.010.

[19] *Crypto*. https://nodejs.org/api/crypto.html.

[20] *Node-JsonWebToken*. https://github.com/auth0/node-jsonwebtoken.

[21] Fiona Fui-Hoon Nah. “A study on tolerable waiting time: how long are Web users willing to wait?” In: *Behav. Inf. Technol.* 23.3 (May 2004), pp. 153–163.

[22] *Apache HTTP Server Version 2.2 Documentation*. http://httpd.apache.org/docs/2.2/mod/core.html. Accessed: 2021-5-5.

[23] *Nginx Module Documentation*. http://nginx.org/en/docs/http/ngx_http_core_module.html. Accessed: 2021-5-5.

[24] Ramakoni. *[Microsoft Internet Information Services (IIS) Documentation]*. https://docs.microsoft.com/en-US/troubleshoot/iis/httpsys-registry-windows. Accessed: 2021-5-5.

[25] Craig R McClanahan and Yoav Shapira. *[Apache Tomcat Documentation]*. https://tomcat.apache.org/tomcat-8.0-doc/config/http.html. Accessed: 2021-5-5.

[26] Information Technology Laboratory. *Digital Signature Standard (DSS)*. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf. July 2013.