Dancing with Fischer, Lynch & Paterson's Impossibility Result

by

Håvard Nordlie Mathisen

Thesis of 60 ECTS credits submitted to the School of Technology,
Department of Computer Science
at Reykjavík University in partial fulfillment
of the requirements for the degree of
Master of Science (M.Sc.) in Computer Science

December 2024

Examining Committee:

Jacqueline Clare Mallett, Supervisor
Assistant Professor, Reykjavík University, Iceland

Michael J. Fischer, Examiner
Professor, Yale University, USA

Marcel Kyas, Examiner
Assistant Professor, Reykjavik University, Iceland

Dancing with Fischer, Lynch & Paterson's Impossibility Result

Håvard Nordlie Mathisen

December 2024

Abstract

This thesis presents Mimir, an asynchronously distributed Single Sign-On (SSO) service that uses biometric scanners on physical devices to reach states of authorized and authenticated. As an asynchronously distributed system, Mimir requires guaranteed consensus to achieve authorised and authenticated states, an impossibility to guarantee proved by FLP's (Fischer, Lynch, Paterson) Impossibility Result. A practical approach was taken to implement Mimir as an asynchronously distributed system, and the implementation was used to study the navigation, mitigation, and practicalities of situations where practical systems encounter hindrances in reaching consensus. This included mechanisms for data synchronisation to enforce an orderly sequence of messages, timers for partial synchrony with the expectation of eventual synchronisation, and fault prevention. Experiments on fault prevention and orderly sequence were conducted to assess the constructed environment, and a standard approach to probability calculations was made. Despite the measures taken, our findings show that limitations, such as the FLP's Impossibility Result, are more than theoretical boundaries; instead, they are shapers of practical systems and how they are built.

Keywords: FLP Impossibility Result, Consensus, Single Sign-On Service

Titll verkefnis með þöæéíó

Håvard Nordlie Mathisen

desember 2024

Útdráttur

Hér kynnum við Mími, sem er ósamþætt og dreifð Single Sign-On (SSO) þjónusta sem nýtir sér handhægan tækjabúnað sem numið geta merki sem eru líffræðilegs eðlis til þess að staðfesta auðkenni. Þar sem Mímir er bæði ósamþætt og dreift kerfi, krefst það þess að samþykki sé tryggt til þess að afla auðkennis, en það getur reynst ógerningur að staðfesta slíkt eins og sannað hefur verið með FLP (Fischer, Lynch, Paterson) ómöguleikaniðurstöðunni. Hagnýtum aðferðum var beitt til þess að þróa Mími sem bæði ósamþætt og dreift kerfi og sú hönnun nýtt til rannsóknar á boðleiðum, mótvægisaðgerum og hagkvæmni í þeim aðstæðum þar slík kerfi geta mætt hindrunum við öflun á samþykki. Þar með talið eru ferlar fyrir samþættingu gagna til að tryggja skipulega röðun boða, tímamælingar á samþættingu að hluta þar sem búist er við að hún verði á endanum endanleg, ásamt bilanavörnum. Tilraunir með bilanavarnir og skipulega röðun boða voru gerðir til að þess að meta umhverfi kerfisins og voru staðlaðar aðferðir nýttar við líkindareikninga. Þrátt fyrir þessar aðgerðir sýna niðurstöður okkar að takmarkanir, eins og FLP ómöguleikaniðurstaðan, eru meira en bara fræðilegir jaðrar og hafa því mótandi áhrif á hvernig megi bæði þróa og hanna hagnýt kerfi.

Efnisorð: FLP Impossibility Result, Samstaða, Single Sign-On þjónustu

Dancing with Fischer, Lynch & Paterson's Impossibility Result

Håvard Nordlie Mathisen

Thesis of 60 ECTS credits submitted to the School of Technology,
Department of Computer Science
at Reykjavík University in partial fulfillment of
the requirements for the degree of
Master of Science (M.Sc.) in Computer Science

December 2024

Student:

.....
Håvard Nordlie Mathisen

Examining Committee:

.....
Jacqueline Clare Mallett

.....
Michael J. Fischer

.....
Marcel Kyas

.....
date

.....
Håvard Nordlie Mathisen
Master of Science

I dedicate this work to my family and those that were told they couldn't.

Acknowledgements

I want to thank my advisor for this thesis, Dr. Jacqueline Clare Mallett, for the many meetings, the unyielding guidance offered to steer me through my self-doubts and uncertainties, and for putting yourself, your knowledge, and endless experience and expertise available to me and all my strange ideas. Never pay the Danegeld.

To my parents and sister, for always sticking by my side, for offering your undying support and love, for all the laughter, the memories and moments, and for helping to shape the way I am today. This achievement would not have been possible without you.

A special thanks go to Dr. Michael J. Fischer from Yale University and Dr. Alice Fischer from the University of New Haven for insightful discussions about the thesis topic and everything else during the summer of 2024.

Thank you to PhD students Vasiliki Kyriakou for checking and verifying formalizations, Benedikt Hólm Þórðarson for general input on writing, and postdoc Gabriel Marc Marie Jouan for verifying the probability calculations.

Thanks also to postdoc Shalini Chakraborty and MSc. student Guðmundur Óli Halldórsson for reading the thesis.

And to my wonderful niece; May all your dreams come true and be as boundless as your imagination and endless as the joy you give me. Never stop exploring, discovering what magic the world has to offer, or let anyone tell you that you can't.

Contents

List of Figures

Chapter 1

Introduction

This thesis presents Mimir, an asynchronously distributed Single Sign-On (SSO) service where a user's identity is verified through their phone's biometric scanners, such as facial recognition. A Single Sign-On is a service that allows users to sign in to multiple websites or applications with a single set of credentials. Consider a user, Alice, who wants to access her online banking service. Instead of remembering and typing a complex password, she responds to a phone notification and verifies her identity using the phone's facial recognition. Once verified through Mimir, Alice can access her bank and other linked services, such as her email account, without needing to sign in again.

This contrasts with standard, centralised solutions using a username and password combination verified by individual hosts, which also serves as a single point of failure [1], and the commonly used alternative to Single Sign-On where an identity provider, such as an organisation, validates Alice's identity across multiple different hosts and applications. When users have tens to hundreds of different accounts online [2], [3], it becomes impossible to maintain different passwords for these accounts separately, and 52% of users are reported to reuse their password across websites and applications [4].

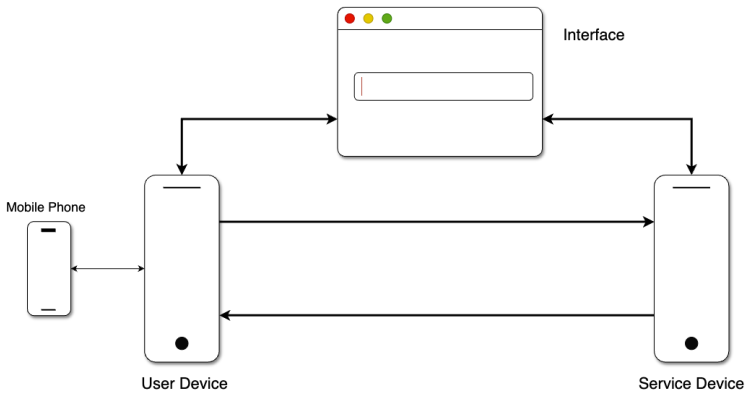

Figure 1.1: Mimir's architecture.

To illustrate this issue using an example of how Mimir works as a Single Sign-On: Let us say Alice wants to sign into her bank. She needs two components:

- A User Device running on a computer represents her identity in the procedures and stores information about services, such as the bank, to which she has given access.
- A phone with a biometric scanner to verify her identity, notify her about ongoing procedures and permit her to grant or deny the bank access to her data.

On the bank's side, it also needs two components:

- A Service Device component that represents the bank in procedures and stores information about users who have granted the bank access to the user's data.
- An Interface, which is the website where Alice will browse her bank accounts.

While these components are part of Mimir's system, each party is responsible for their own components: Alice manages her User Device and phone, while the bank manages its Service Device and Interface.

Two key concepts are considered in Mimir:

- **Authorisation**: The initial registration procedure where Alice grants a service (like her bank) permission to access her identity information. This only happens once when first connecting with a new service.
- **Authentication**: The subsequence procedure where Alice verifies her identity through her phone's biometric scanner when she wants to access the service.

Let's follow Alice through both of them: In Mimir, the first time Alice wants to log in to her bank, she initiates a registration procedure, known as authorisation, by clicking a button on the bank's Interface. The Interface sends a request to Alice's User Device. The User Device communicates with Alice's mobile phone, notifying her about the authorisation attempt and allowing her to decide whether to grant the bank access. Alice verifies with facial recognition on her phone, and the phone sends a response back to the User Device, where tokens are created for the bank to use when Alice wants to sign in again. The tokens are stored on the User Device and then communicated to the bank's Service Device. The bank's Service Device stores the tokens on its side. A confirmation is sent back to Alice's User Device, notifying the Interface and Alice that the procedure has been completed. This permits the bank to access Alice's data and identify her when she signs in again, and the bank is authorised.

Later, when Alice wants to sign in to the bank again, she logs in, also known as an authentication procedure, to verify her identity through the same Interface and another button. This time, the tokens generated during authorisation are used. The bank's Service Device checks the tokens to verify that Alice's User Device exists in their records, before sending a message to her User Device to notify it about the sign-in request. Alice's User Device, in turn, communicates with her mobile phone, allowing Alice to verify her identity and the request. Alice's mobile phone then responds to the User Device, which confirms the verification to the bank's Service Device. At this point, the Service Device releases the tokens generated during authorisation. These are then sent back to Alice's User Device, which verifies the tokens and responds to the bank's Service Device again. Finally, the bank's Service Device sends a message to the Interface to notify that the procedure has been completed in an authenticated state.

This may seem straightforward to a user, but as we can see from this example, it is anything but. These interactions place Mimir at the heart of a fundamental challenge in computer science: When N ($N \geq 2$) components in distributed systems – like the User- and Service Device – need to agree on whether the user is authorised or authenticated, they face what is known as the Fischer, Lynch, and Paterson's Impossibility Result. Fischer, Lynch, and Paterson (FLP) proved in their 1985 paper that no polynomial-time algorithm or protocol can guarantee consensus or decision-making in

any asynchronously distributed system if one process might fail [5]. An asynchronous system is one that is not synchronous. From here on, we refer to this as the consensus problem.

In Mimir's context, the consensus problem manifests in practical scenarios: during the authentication procedure, Alice's phone might become unavailable, for example, by losing service. A message may be sent from Alice's User Device but never be delivered at the bank's Service Device, or either of the components might fail after partially processing a sent message, never responding to the other component. These are not merely implementation considerations for Mimir and other distributed systems but represent practical failures that can critically hinder the components from reaching a consensus about authorised and authenticated states.

In the rest of this thesis, we will use our experience developing Mimir to examine situations that can hinder consensus around authentication. To our knowledge, this is the first academic contribution to bridge the theoretical findings of Fischer, Lynch, and Paterson's Impossibility Result with practical implementation.

Chapter 2

Background

One of the earliest uses of passwords dates back to the Zhou dynasty in China [6].[1] The dynasty used tallies as a symbol of authorisation for troop movement and restricted passage to cities or the Emperor's palace. Polybius' *The Histories* [8] describes how the Romans employed *"watchwords"* to corroborate identities within the military. Both practices offered translucent, simple, but sophisticated ways to permit or refuse access to confined areas restricted to authorised personnel only, documenting an extended and honourable history for passwords.

The creation of the first computer password is accredited to Fernando J. Corbató, responsible for the Compatible Time-Sharing System in 1961 at the Massachusetts Institute of Technology [9]. Corbató implemented a `"LOGIN"` command into the system, which prompted the user to follow up with a `"PASSWORD"` command that shut off the system's printing mechanism to allow the user privacy to type their password [10]. With the extended history of passwords in mind, one could argue that this virtual implementation of physical keys was a justified step in the evolution of computer systems.

Previous studies have revealed several vulnerabilities in web-based Single Sign-On services [2] as well as uncertainties in the precise implementation of the technology [11]. Despite that, Scott et al. found that the benefits of using an SSO to authenticate far outweigh the risks associated with the technology. The study discussed significant vulnerabilities and attacks to which an SSO could be exposed, such as Unified Resource Locator (URL) Redirect[2] and phishing[3]. The same study discovered that the latter attack could prove successful, even when performed on individuals aware of the concept [2]. The authors of the paper neglected to mention the concerns regarding the vulnerability of an SSO as a single point of failure and never discussed the concerns of the consensus problem or reaching a consensus. In the case of the former, the SSO might be rendered unusable if an attacker could compromise or halt operations within the system, making it impossible to reach any form of consensus. Uptime and availability are crucial factors for an SSO, as information needs to be accessible at all times. Designing a fault-tolerant system is paramount to mitigating issues such as the consensus problem.

[1] Part of this literature review is extracted and rewritten from a report by the same author, Håvard Nordlie Mathisen, published on ResearchGate [7]

[2] The process of redirecting a URL to a nefarious or malicious address.

[3] A process of tricking a user to provide sensitive information, such as passwords or credit card information, without knowledge of disclosing it.

FLP's Impossibility Result [5] is a fundamental proof within asynchronously distributed systems. The proof demonstrates that it is impossible to find any message exchange protocol that can guarantee consensus in any asynchronously distributed system, for even a single bit value, where faults can occur [5]. Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson first published a mathematical proof to document the impossibility in 1985. The paper defined an asynchronously distributed system as one in which no guarantees are given on the timing of message delivery or relative speed of processes. In such a system, where nodes, or computers, are operating entirely independently, messages can experience arbitrary delays, and there is no guaranteed way to assert whether one of the nodes has crashed or is slow to respond [5]. The proof profoundly impacts the understanding, implementation, design and management of distributed systems, such as an SSO. The result presents an unsolvable problem for guaranteed consensus of asynchronously connected systems, and so it must be navigated in all systems rather than attempted to be solved. Breaking the asynchrony assumption and turning the system synchronous is a way to solve the consensus problem, but this solution runs into scaling issues, which can easily be overlooked by system architects. Navigating the consensus problem becomes crucial in Mimir, given that an SSO needs to perform reliable authorisation once and then trust the authorised state later.

The National Institute of Standards and Technology (NIST) defines authorisation as the procedure to determine the right or permission granted to a system entity to access a resource [12]. NIST defines authentication as the procedure to verify the identity of a user, process or device in an information system [13]. When the authorisation or authentication is successful, each procedure grants permission where said entity is considered authorised or authenticated. Conversely, the entity will remain unauthorised or unauthenticated if the relevant procedure fails.

Since Corbató implemented the first computer password, technology has evolved significantly. Despite the evolution, passwords continue to be the most common form of authentication as of 2023 [14], among public key cryptography and two-factor authentication/One-Time Passwords [4] (OTP) through phones. FLP's consensus problem also remain a challenge within distributed computing. With the advancement of technology, newer and improved authentication methods, such as biometrics, hardware tokens, and passwordless authentication, have evolved.

2.1 Consensus

Consensus in distributed systems with asynchronous properties is a long and well-documented field of study in computer science. Lamport's foundational work led to the 1978 paper on event ordering [15], along with Dijkstra's publication on self-stabilising systems [16], laid some of the groundwork for modernised distributed systems. Many contributions have been made as solutions to the consensus problem in various contexts [17]–[20]. Perhaps the most widely acknowledged attempts to solve consensus are the algorithms Paxos [21], also by Lamport, and Raft by Ongaro and Ousterhout [22]. Both algorithms are designed to ensure consensus in distributed systems by proposals or leaders, which are voted upon or elected through a majority by the other nodes in the system, essentially providing mechanisms to create temporary points of synchroni-

[4] A one-time password is generated through a physical device or a mobile application to authenticate users.

sation which can be relied upon. However, we consider them partial solutions because both algorithms rely on an optimal network connection and that all nodes operate without misconfiguration and bugs while relaxing the asynchronous requirements described by FLP to function reliably. These are part of the asynchronous model defined in the consensus problem [5]. [23] modified Paxos, inspired by Dutta et al. [24], by constructing a Solicit phase where other nodes gathered proposals. The former reduced the processing time needed to establish synchronised communication in the latter and provided mathematical proof of their findings [23].

Examination of the details and verification of the correctness of FLP's impossibility result has also been performed by Bisping et al. [25]. The researchers performed a mechanical verification study of the consensus problem through the automated theorem prover Isabelle, using Higher-Order Logic [25]. The study concluded the correctness of the consensus problem, serving as a starting point to conduct similar proofs.

Lu and Yang discussed the challenges of consensus in cyber-physical, multi-agent systems experiencing denial-of-service (DoS) attacks [26]. The environment described assumed that each node in the distributed system would be DoS'd independently of one another to render certain parts or the entire system inaccessible over time. The paper concluded that, despite the ongoing independent attack, it was possible to achieve consensus in the given environment through a state-feedback controller which only used information from available channels, and observer-based controllers that estimated the state of attacked channels combined with states from available channels [26].

Dwork, Lynch, and Stockmeyer [27] introduced the concept of partial synchrony as an intermediate state between synchronous and asynchronous operations. In the case of partial synchrony, the theoretical model described a scenario where it ran asynchronously initially with the expectation that the system would eventually reach synchrony while imposing bounds on time and speed, unknown a priori [27]. The authors constructed protocols which allowed a set of processes to reach an agreement of values despite the pressing challenge of faults occurring and varying delays in message passing [27]. The intermediate state between synchronous and asynchronous represents a more standard approach to systems development but meets severe scaling issues as more nodes are added to the network [28], [29] and more load is added to the CPU that deals with the synchronisation to wait. Through deploying several protocols and clock distribution mechanisms, Dwork et al. concluded that achieving consensus is possible through partial synchrony [27].

Li et al. conducted a study to discover the minimum bits needed to orchestrate consensus in a data rate-limited or energy-constrained network [30]. The authors proposed a protocol based on exchanging encodings of symbolic data between adjacent neighbors [30]. A network model was designed, and the protocol was constructed and customised to fit the network. A theoretical analysis was also conducted before introducing a performance test to describe the energy required to achieve average consensus in the network. Further, asymptotic properties were discussed and analysed as the number of connected nodes approached infinity [30]. The results concluded that as little as one bit was necessary to establish consensus between neighbouring nodes. With the nodes approaching infinity, the rate depended on connectivity and the ability to synchronize [30]. Finally, the study did not discuss or consider the consensus problem.

The authors of [31] constructed a theoretical system where intentional failures could occur to study and try to solve consensus. Chandra and Toueg modelled the

system in tangent with those discussed by FLP with no constraints on time, clock drift, or message delays to execute the model [31]. The paper defined a specific term of *"Unreliable Failure Detectors"*, which consisted of modules that could monitor the status of other processes with weak detectors that could intentionally make mistakes. The conclusion was that consensus was possible to obtain, provided that a weak failure detector is used, leading to each process eventually being considered a subject of a crash [31]. A limitation of this model is that it cannot provide strong guarantees about the monitored process' current state, such that a detector might consider operational processes as crashed or vice versa.

Despite the extensive documentation of solutions to the consensus problem in various contexts, it remains unsolvable unless certain conditions are met and restrictions are set. Therefore, distributed systems with asynchronous properties still lack a generalised consensus algorithm and protocol, and the consensus problem should be considered an unsolvable computational limit as it continues to be a fundamental limitation in the field. As a result, it is essential to examine how consensus requirements and the imposed limitations affect standard systems and their ability to operate.

2.2 Passwordless Authentication

Passwordless authentication is accessing secured resources without needing a traditional password and instead using biometrics.

It is an expanding research area within Computer Science and Computer Security [32]–[34]. It has gained recognition due to the surge of vulnerabilities in conventional authentication techniques like passwords. These are susceptible to many forms of attacks, such as phishing and credential stuffing[5]. Furthermore, passwords and confidential data can fall prey to brute-force attacks [35], either due to data breaches, hacking, or acquiring information on the dark web.

Wang et al.'s [4] extensive empirical analysis of leaked passwords went through significant data breaches that happened over eight years. They found that many users reuse passwords across different websites and platforms [4]. The analysis detailed several approaches to password cracking/guessing, reuse, and modification. From the study, it became clear that as many as 52% of the analysed users had either reused their password or slightly modified it to reuse it across services or websites [4]. This statistic suggests that continuously remembering these passwords is challenging, and the number of systems needing passwords was not foreseen by the original implementers.

As technology and computer performance have increased, limitations in traditional password authentication have become more apparent. This has led to the development of alternative authentication methods, such as passwordless authentication. Various methods exist for passwordless authentication, including adding a second layer of security through one-time passwords. Several attractive solutions, such as Universal Authentication Framework (UAF) [36] and Web Authentication (WebAuthn) [37], developed by the Fast IDentity Online Alliance (FIDO) and the World Wide Web Consortium (W3C), serve as notable alternatives.

According to NIST [38], [39], authentication should contain at least one of the three principles:

[5] A process where an attacker uses information exposed in one data breach to attempt to gain access to another system.

- **Something you know**: Such as a password, an answer to a personal question, or a Personal Identification Number (PIN).
- **Something you have**: A device that contains a token or cryptographically signed keys.
- **Something you are**: Biometrics of any sort or kind. Facial recognition, retina scan, fingerprint, voice recognition, or similar.

The advantages and drawbacks of the most common passwordless authentication methods were detailed and described by Parmar et al. [40]. The study revealed that users associated passwordless solutions with the least friction when authenticating through user interfaces [40]. It further mentioned biometrics as attempts to capture the supposed uniqueness of human traits, such as fingerprinting, facial recognition, or retina scans, as approaches to passwordless authentication [40].

A promising strategy for passwordless authentication involves leveraging the joint effort of the FIDO Alliance and W3C to develop WebAuthn. WebAuthn offers a solution where the user utilises two of the principles of NIST's recommendation for authentication, namely *something you are* and *something you have*. WebAuthn utilises the user's biometric data stored on a device they possess, such as a phone, a computer, or a physical security key (like YubiKey, NitroKey, etc.) to authenticate [37]. This approach uses a physical device as a unique synchronisation point, where the device is subject to being stolen or lost, making consensus impossible.

Virtual Private Networks (VPN) are often used to access restricted networks, and Huseynov built a web interface using WebAuthn to authenticate users to a legacy VPN [41]. The paper highlighted that WebAuthn is the only known solution to problems like phishing [41]. On a successful request with WebAuthn, the legacy system would create a temporary username and password, which would be sent back to the user in response, allowing them to copy and paste the credentials After a specific time, the password would become invalidated, and the user would have to authenticate again [41].

Gordin et al. implemented passwordless authentication in a real-world scenario. Throughout the study, they used fingerprints as the primary form of biometrics, citing cost and that some users struggle with their device's biometric scanners as the primary discouragement for adapting FIDO2. Their findings showed fewer attacks towards their server due to the lack of secret authentication information stored there [32].

A report published by Statista in 2020 [42] estimated that between 77–86% of mobile devices globally contain some biometric scanner, suggesting that the cost of adapting FIDO2 is not as significant as Gordin et al. thought.

[4] suggests that users find the application of passwords as a burden, and Lyastani et al. uncovered that users found passwordless solutions more manageable and secure [34]. The research discovered that users associated losing their authentication device with losing access to their accounts [34], highlighting another aspect of the guarantees of consensus, where a lost device can be considered a fault that renders guaranteed consensus impossible with the phone acting as a point of synchronisation. This imposes significant challenges for adapting an entirely passwordless solution, as users may require – and opt for – more familiar alternatives due to the single point of failure. This challenge suggests two scenarios:

- What is the backup solution in case the authentication device is lost?

- NIST's principles of *Something you have* and *Something you are* might not be sufficient to replace the first principle of *Something you know*.

With FIDO2 first being released in 2015, Morii et al. conducted a study in 2017 to investigate the potential of FIDO as a viable PA solution. At the time, the authors noted that the successor, FIDO2, was not widely adopted, and Microsoft Edge was the only browser that had implemented support for the technology [33]. Despite the lack of browser support, the researchers successfully implemented PA into Shibboleth, a widely deployed, open source-identity management solution [43], demonstrating the viability of integrating PA into existing systems. Since then, the browser environment has drastically evolved. As of this writing, Vivaldi, Google Chrome, Mozilla Firefox, and Apple Safari all support these standards alongside Microsoft Edge. With further adaptation by large corporations, the technology has become more accessible to the general public across services, facilitating widespread deployment that enhances security and improves user experiences as people become more familiar with and learn to use the technology.

Authentication can also be based on Certificate-Based Authentication (CBA). CBA is a process where a successful authentication procedure generates a set of key pairs on a device in the possession of the applier[6]. This certificate is then verified by a form of Certificate Authority (CA). The use of certificates in computer security is widespread, and it is used in Transport Layer Security (TLS) to secure communication over the internet. TLS was built on the foundation of the now-deprecated Secure Sockets Layer (SSL) protocol. Despite TLS's widespread use, it is not a fault-free solution and is subject to vulnerabilities such as certificate expiration [7], misconfiguration [8], and certificates issued by untrusted authorities [9]. As of February 2024, the company IdenTrust is responsible for over 50% of the world's issued certificates [44]. This makes IdenTrust a desirable target for attackers who wish to compromise a significant portion of the internet's security infrastructure through IdenTrust's supply chain.

Despite the promising solution with WebAuthn, the challenges surrounding the consensus problem still need to be considered. A lost or stolen phone is a practical example of one aspect of the consensus problem. Neither of the papers went into detail about how this can be solved or considered the consensus problem.

2.3 Single Sign-On

The topic of users being able to access multiple resources with a single set of credentials is a large research field within Computer Science, Information and Computer Security. Shaikh et al. reviewed the many ways this has been done [45]. The paper discusses three common protocols for this practice: Security Assertion Markup Language (SAML, RFC7522 [46]), OpenID (based on RFC6749 [47], RFC6750 [48]), and OAuth (RFC6749 [47], RFC6750 [48]). Microsoft's SQL Server's usage of a single

[6] We specify applier, seeing that a user is not required to be part of the transaction scheme to perform the authentication.

[7] A situation where the certificate expires due to age and needs renewal to work again renders a service insecure despite the usage of a certificate.

[8] A practice where the certificate is incorrectly configured leads to potential security vulnerabilities through examples such as weak cyphers or protocols.

[9] A certificate may be issued on behalf of an entity it does not belong to, which can happen when a certificate authority has been compromised.

identity provider for access details was explained in short detail; but again, there was no mention or description of the application's implementation in practice or relation to the consensus problem.

SPRESSO (Secure and Privacy-Enhancing Single Sign-On) formalised implementing a Single Sign On service in 2015 [49]. The paper outlined an alternative to more familiar systems, such as OAuth and OpenID; a formalisation and security analysis was provided and implemented in JavaScript. A decentralised solution was discussed, where the user would provide their email, and the email's domain would be used to identify the correct authority to handle the authentication using a forwarding system, avoiding tracking. The email also verified the user's authenticity, delegating this responsibility to the authentication authority. It emphasised that such a forwarding system was necessary to avoid tracking and profiling users through the authentication provider, noting that there was no other alternative at the time [49]. The paper did not discuss how SPRESSO dealt with the consensus problem or fault mechanisms. With passwords being the primary form of authentication in most systems [14], it further raises the question of how different the situation would be if solutions like passwordless authentication were applied.

An approach to rethinking the way of single set credentials for multiply linked systems was introduced in 2018 when Johnson et al. suggested utilising smart contracts and verifiable representation [50]. When the user signed up for the system, it created a verifiable contract for their future authentication attempts through blockchain-enabled decentralised identifiers [50], and a representation of the identifier could be generated from the original credential.

A Single Sign-On service can make the user experience more convenient and remove some cost burden of support staff [45], where the latter serves as a point to establish synchronised communication with the authorisation provider. A product manager of one of Europe's digital ID providers disclosed in an interview that the provider's most significant cost is phone support to reset users' passwords and lost or renewed phones for authorisations. This indicates that establishing synchronised communication is not free. Researchers at the University of Chicago described a solution where they migrated their existing user management system over to a solution with one credential for all their services [51]. The researchers used the open-source system Keycloak as a backbone for the implementation and developed a custom authentication module to support their requirements. They chose Keycloak, arguing that it could support and integrate with many identity providers and protocols. The solution facilitated support for the existing user base, permitting the users to use their credentials to connect through services like Google and access GitHub's API. In addition, it provided the ability to support access through the Texas Advanced Computing Center, where the solution was also deployed [51].

A few alternatives to open-source Single Sign-On exist, and, as previously mentioned, Keycloak, developed by Red Hat Inc., is one of them. In 2014, they published it as an open-source, all-in-one solution for companies needing single-set credential management. Its primary language is Java, and Stian Thorgersen and Pedro Silva wrote a book describing how to use it [52]. In addition to Keycloak, there is a more recent solution, Zitadel, written in Go and had its first public release in 2020. Both alternatives offer a complex, out-of-the-box solution for companies and organisations, containing user management, identity brokering, and social login features.

As part of the authorisation and authentication procedure, signing out is just as necessary to protect user data [53] and reflect the system's state. Ramamoorthi and

Sarkar showed this by revealing that in some cases, a user that used a Single Sign-On service could be compromised by not terminating a session[10] properly [53]. Highlighted in the study was the case where the user signed out of the Service Provider[11] but was still signed in to the Identity Provider[12]. The complexity of this practical example of consensus issues emphasises how essential it is to navigate according to the consensus problem. As previously discussed, concerning the implications of Fischer et al.'s work, it is crucial to note how the limitation dictates challenges in managing distributed systems. However, the paper never noted that this is the underlying issue. They proposed two alternatives for an effective sign-out procedure: Educating users or integrating the process seamlessly into the service. Research documents educating users as nearly impossible [54].

The researchers implemented a solution where the sign-out procedure would happen automatically [53] through Shibboleth. The user could also see their current sessions through an extension[13] running in the browser. Despite offering the solution to sign out automatically, a user who frequently accesses a service like Google (i.e. using Google as an authentication authority) might not want to sign out entirely.

This situation brings us to the challenges of distributed systems and the constraints enforced upon them due to the consensus problem. As previously mentioned, the paper proved a boundary in the capabilities of distributed systems, particularly concerning faults like network delays, partitions, or message loss [55]. This crucial boundary presents a hidden threat with the implemented solution. For instance, while being automatic, the sign-out solution might still encounter periods of inconsistency. An unsuspecting user might believe they have signed out of all applications due to the presented information. Still, the system might not have updated the state correctly within a particular time or phase. Thus, considering the consensus problem becomes a pivotal focus when evaluating these solutions for their real-world robustness and security.

With mobile devices becoming central to our daily lives, their computing capabilities are increasing. Equally, managing a single set of credentials for multiple services has evolved in the same direction, and Yuan Zhang et al. introduced PROTECT, a scheme including multiple identity servers sharing parts of a secret and a mobile phone [56]. PROTECT introduced a renewal algorithm to protect from online dictionary attacks, which renewed the secrets on the identity servers at a periodic interval in a distributed sharing algorithm. The system necessitated a degree of synchronicity with N identity servers during the initiation of the authentication process. The specific mechanisms to achieve this synchronisation, however, were not outlined, and this solution was also prone to the consensus problem since there is no way to guarantee that the initiated synchronicity contains no faulty processes, where a faulty process would corrupt the synchronisation.

[10] A session is a form of stateful communication between two or more computers.

[11] An entity providing a service or application a user wants to access.

[12] A system whose job it is to authenticate a user, issuing user information tokens.

[13] A form of software that extends the functionality of a browser, often not provided by the browser itself.

2.4 OAuth

OAuth is a protocol developed by the Internet Engineering Task Force (IETF) that serves as an open standard for authentication. The research community has contributed to the body of material on the protocol [57] and made suggestions on its issues [58]. There have also been controversies surrounding the protocol [59]. Hossain et al. stated that the protocol had the potential to pose security risks when handled by inexperienced developers [57] due to its many edge cases. The researchers suggested a set of building blocks to serve as a checklist, improving security when using the protocol [57], such as whether techniques to prevent Cross-Site Request Forgery (CSRF) and HyperText Transfer Protocol Secure (HTTPS) are deployed.

In [58], the authors used Laravel to implement a role-based system, examining its performance implications within microservice architectures compared to traditional monolithic systems [58]. They concluded that while performance impacts were negligible, the approach significantly simplified the authentication process in microservices [58].

2.5 Password Manager

A password manager is a system or service that manages and keeps track of high-entropy passwords for other systems or services. In a typical scenario, a user would be unaware of the passwords for each service. This procedure allows for integrating highly complex strings – which often present a challenge for human memory – as passwords, enhancing overall security. However, a password manager can also serve as a tempting target for attackers, as it houses all the user's passwords in one place. In the context of consensus, password managers serve as a point of synchronisation and a single point of failure, where knowing that a password manager has been compromised can also be considered a consensus issue.

Previous leakages have been reported where password managers have been compromised [60], [61]. We typically expect a master password [62] to secure the passwords inside the manager, requiring the user to remember it. Imposing such a requirement may cause the master password to become a low-entropy phrase, protecting high-entropy values due to the vital necessity of remembering it to maintain access. Many password managers, therefore, offer a second step to verifying the authenticity of the user, which is called multifactor authentication.

Dhanalakshmi et al. described an implementation of a password manager with a different way of safe-keeping the passwords [62]. The paper suggested using a four-word phrase or a Personal Identification Number (PIN), including the master password [62]. This phrase would authenticate the master password in a shuffle phase based on permutations of the selected PIN. While different, this complicates the procedure and only serves as a way to obscure information, and given enough attempts, since this pin is subject to the user remembering it, such a phrase could be guessed.

A recent study on users' perception of password managers provided insights into the current practices [63]. The survey, which included 300 responses, found that most participants relied on other means than remembering passwords. As many as 67% disclosed that they used a password manager, while 18% said they wrote down passwords [63]. It became evident in the study that 67% of the asked participants used the password generation functionality in the password manager they were using. While

this suggests a significant portion of users, many still rely on passwords they created themselves, which can potentially be low-entropy.

Pandare et al. wrote another suggested password manager implementation [64], encrypting the passwords with AES-256 and then hashed them with the Secure Hash Algorithm (SHA-256) in a hybrid approach [64].

Researchers introduced a solution to the compromised manager issue where the encrypted passwords are never stored. Shirvanian et al. studied and built a password manager that utilised a domain name, master password, and an Oblivious Pseudo-Random Function (OPRF) to compute a hash and, in such a way, verify the entered password [65]. This way, the password was not stored but computed as part of the sign-in process. They also built an Android app to support the system, where the user would need to consent to the sign-in that was happening. The work in the paper is based on Bellovin and Merritt's 1992 paper on Password Authenticated Key Exchange (PAKE) to defend against dictionary attacks [66], and the IETF has since created a standard on the PAKE protocol [67]. Despite using a phone to authenticate, the paper did not give details about fault tolerance, consensus, or mechanisms to operate within the constraints of the consensus problem effectively. OPAQUE, an in-draft standard, proposes a similar modernised scheme where the server doesn't see the password [68].

Detailed breaches and a thorough investigation of password manager implementations show the relevance of generating high-entropy strings to represent unique values, avoid collision, and evade guessing commonly used phrases to prevent faults in an SSO.

Chapter 3

System Design

This chapter outlines the methodology used to implement Mimir, a service in which a user can use a single set of credentials to access multiple websites or applications, whose asynchronous architecture puts it at the heart of several situations that might hinder consensus about authorised and authenticated states. The chapter begins by describing Mimir's architecture, followed by its authorisation and authentication procedure flow. Subsequent sections detail the specific architectural and implementation decisions necessary for Mimir to operate and connect these to the underlying foundational consensus problem. They highlight how these theoretical limitations, such as the consensus problem, manifest in practical systems and shape them.

3.1 Architecture

Mimir's architecture is designed to manage distributed authentication and authorisation procedures throughout its components. For the system's operational ability and functionality, it is crucial that it can reach a guaranteed consensus about the states of authorised and authenticated users and services. This section provides an overview of these components and their responsibilities. Communication between the components is conducted with sockets over Transmission Control Protocol/Internet Protocol (TCP/IP). The data format is Protocol Buffers (Protobufs), a language-agnostic serialisation format for interchanging data. Incoming requests to the components can be received by all parties at any arbitrary moment, making the system asynchronous.

3.1.1 Interface

The Interface is the initiating component of the authorisation (Figure 3.1) and authentication (Figure 3.2) procedures. Requests are triggered by end-user interactions and sent to the User Device or service-controlled server, where completion or termination of the procedures will communicate a result back to the interface. Sockets facilitate this communication, but any method capable of network communication could have initiated these tasks.

3.1.2 Mobile Phone

The mobile phone is a method to establish synchronous communication with the system, enabling the user to approve or deny authorisation and authentication procedures

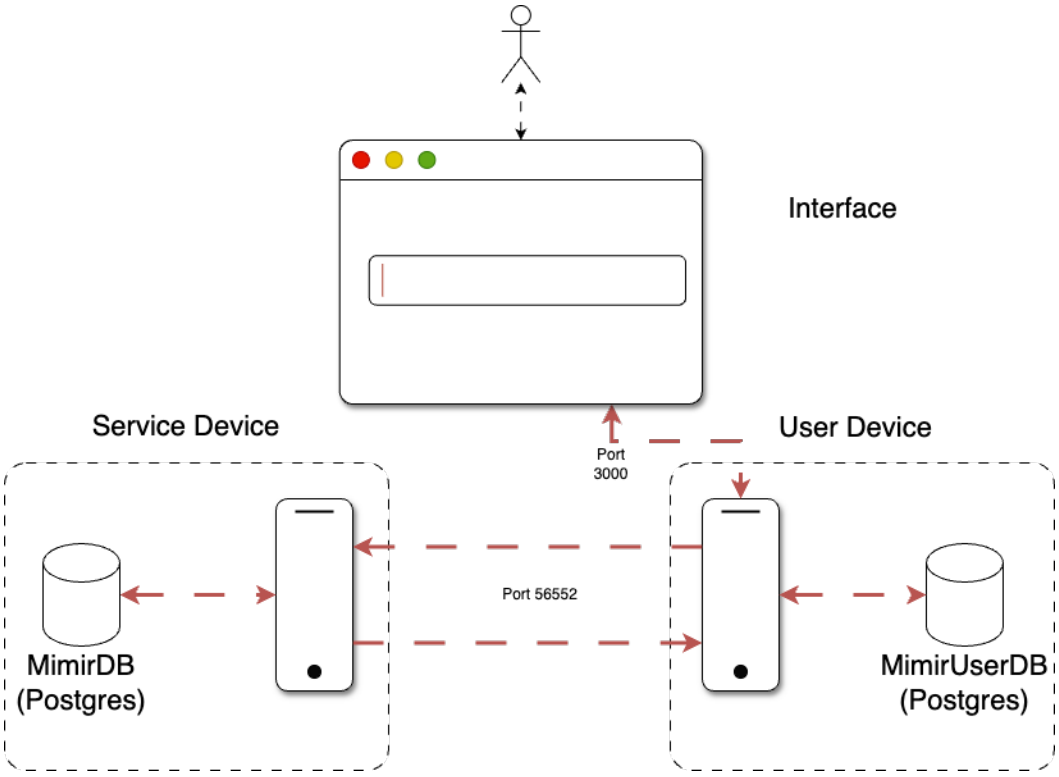

Figure 3.1: The authorisation architecture. Operations are performed clockwise. Dashed lines indicate operations prone to the consensus problem.

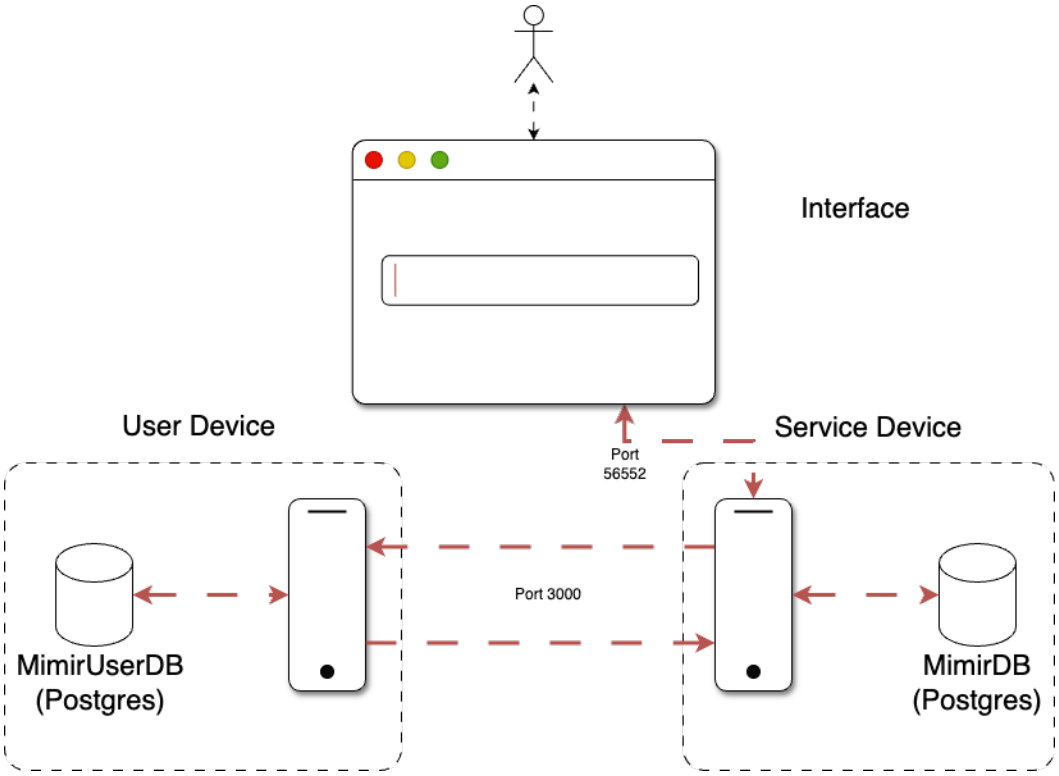

Figure 3.2: The authentication architecture. Operations are performed clockwise. Dashed lines indicate operations prone to the consensus problem.

through biometric means via the User Device. Alerts are provided regarding any authorisation and authentication procedures to offer insights about unauthorised attempts, and the application monitors the activity of the User Device instance for faults.

3.1.3 User Device

The User Device coordinates the authorisation procedure in collaboration with the Service Device. This approach is chosen to give users control over their data, and no information is disclosed from the User Device without the user's explicit knowledge and consent through the mobile phone. A connection to a Postgres database limits procedures to one at a time and maintains a registry of authorised services. Each service is associated with a client identifier and secret, a securely generated service identifier to distinguish the services, and a JavaScript Object Notation (JSON) Web Token (JWT)[1], which can be used to enforce reauthorisation upon expiration. The User Device verifies these tokens during authentication; port 3000 is the default for incoming connections. The decentralised nature of the component provides sufficient scalability, as each user manages their instance. Attempts to authenticate are logged in the database for monitoring purposes, and the mobile phone is notified upon attempts made to authorise or authenticate. The architecture can be seen in Figure 3.1.

3.1.4 Service Device

The authentication procedure is coordinated with the User Device by the Service Device to verify the user's existence. A registry of authenticated users is maintained and populated with the service-related information generated during authorisation through a Postgres database. The database is further used to limit authentication procedures to one per connecting User Device, and a secure identifier is generated to track the procedure. Port 56552 listens for connections from the User Device and Interface. The architecture can be seen in Figure 3.2.

[1] A JWT is a token constructed from some data and then signed with a private key for verification.

3.2 Message Flow

In this section, we describe the message flow and how the components handle these incoming messages. Many operations in Mimir's message flow are affected by the consensus problem because it is impossible to guarantee that a message will be sent and arrive at the receiving component, that the receiving component will manage to complete a procedure without fault, and that the components will reach the same agreement upon even a single-bit value. Similar operations are marked with red dots in Figure 3.3 and Figure 3.5 while vertical bars signal synchronisation points or checkmarks for a procedure to permit faults to occur, which might be rectified within an acceptable time.

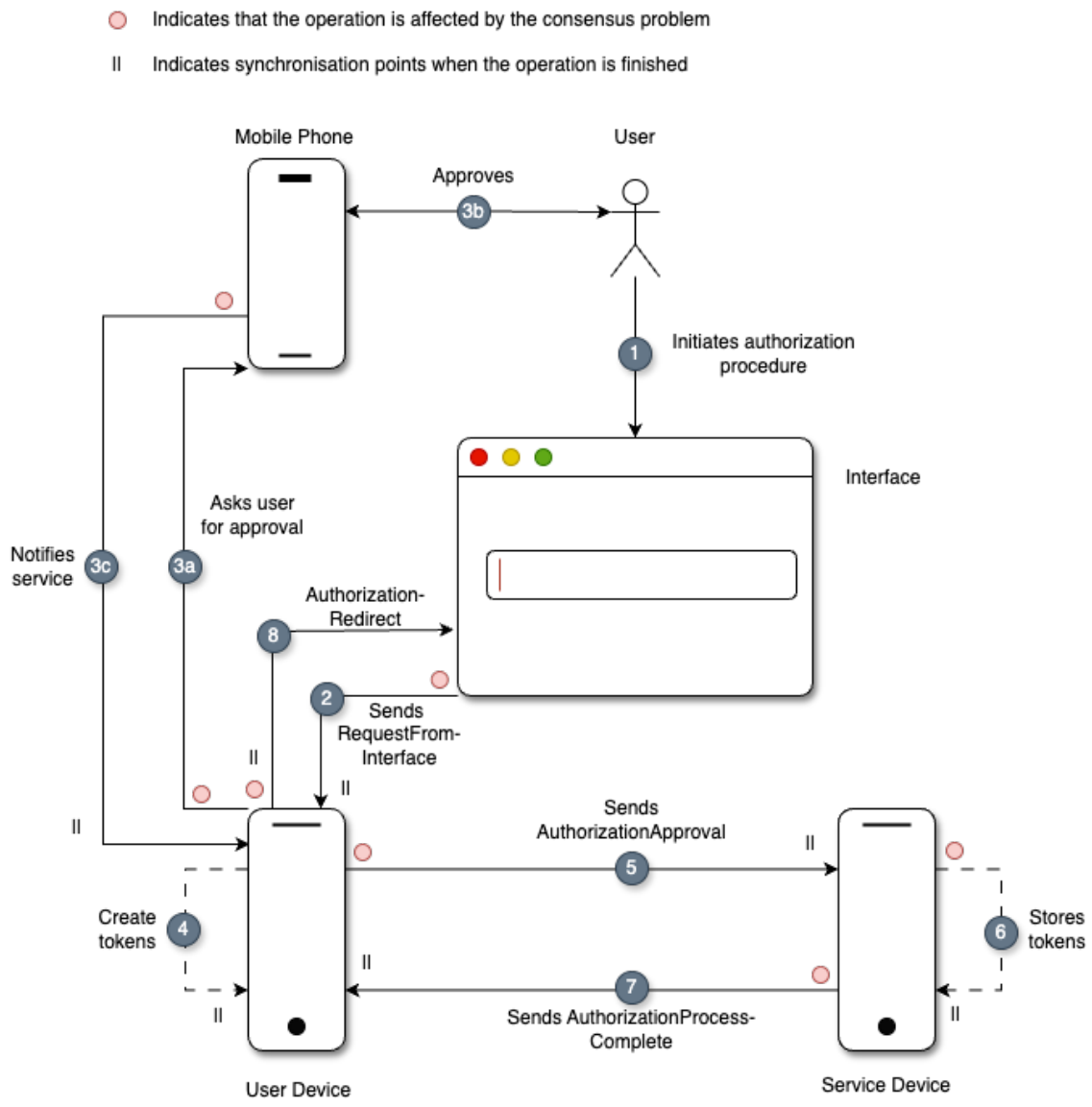

Figure 3.3: The authorisation procedure in Mimir.

3.2.1 Authorisation

The user, Alice, starts the authorisation procedure by interacting with the Interface, as seen in step 1 in Figure 3.3, and it is steps 4-7 that are the most crucial for the

consensus problem. The Interface constructs a `RequestFromInterface` message and is wrapped in a `MimirMessage` sent to the User Device, as seen in step 2. All subsequent messages are wrapped similarly before sending.

The Service Device's IP address and a human-readable service name are included in this message. When the message arrives, the IP address is parsed, and the Service Device's default port is appended if none is present. A verification check is then performed to ensure no ongoing authorisation procedures exist for the human-readable service name. An entry is created in the database about the authorisation attempt to log the request in case of faults and for monitoring. The entry contains a uniquely generated state identifier and an enum used to track the procedure's progress, modelled after the protocol, seen in Figure 3.4. Subsequently, the mobile phone is prompted, and synchronous communication is established with the user to authorise through biometrics, as seen in steps 3a-c. After a successful sending from the User Device, the enum stored in the database is updated to reflect the preceding step, which is applied for all subsequent successful sendings. A two-minute timer is constructed to allow the user sufficient time to authorise the request.

If no authorisation is given by the user, then the procedure fails, and by convention, it is up to the user to retry authorisation. In practice, the application can attempt to discover whether the communicating component is overloaded when no response is received. However, it would then create scaling issues where the component, if it is overloaded, would need to handle even more load. No method can be provided to determine the reason for lack of response, where the application or user can be slow to respond, or the application can have crashed [2], as per [5]. The request is also invalidated if the user denies it.

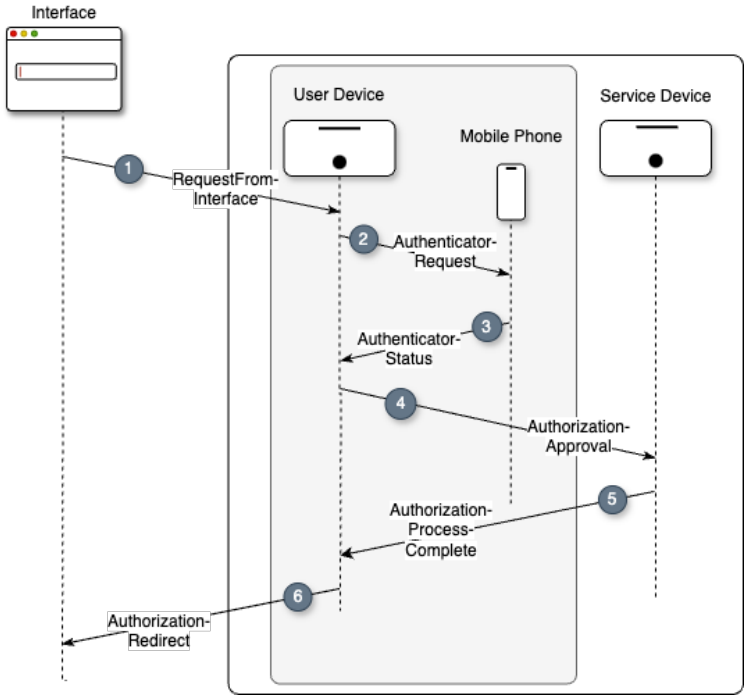

Figure 3.4: The authorisation procedure.

[2] A crashed application entails a lost connection with TCP, but a lost connection can have many causes, not just a crash.

Authentication tokens, which include a client identifier and secret, a JWT (called the auth token) constructed from the human-readable service name, and a unique service identifier, are then securely generated by the operating system, as seen in step 4. The auth token's expiration time is set to 60 days and signed with a probabilistic signature scheme using RSA and SHA512 as the hashing algorithm.

The client identifier and secret are utilised to distinguish the service upon reauthorisation. At the same time, the JWT is used to determine when a reauthorisation is needed and to validate the authenticity of the Service Device. Finally, the authorised service is inserted into the database; an `AuthorizationApproval` is generated, containing the secure service identifier, unique state identifier, client identifier and secret, and the generated auth token. At this point, a separate TCP connection is established to the Service Device, and an `AuthorizationApproval` is sent to the Service Device, as seen in step 5. Another timer is imposed to limit the component's response time before the procedure is invalidated. The time is set to five seconds to account for network congestion, routing, or other network-related issues, as well as the receiver's expected processing time. A similar timer is imposed for all messages that expect a more critical response than an acknowledgement from TCP.

Step 5 is affected by the consensus problem. We consider our user, Alice, whose User Device is about to link the bank's Service Device to itself. Alice's User Device stored the tokens in the previous step and sent a message with them to the bank's Service Device. But if the bank's Service Device does not acknowledge or respond to the message from Alice's User Device, the User Device can not determine whether:

- The message arrived, but the Service Device crashed before processing it.
- The message arrived and was processed, but the acknowledgement was lost.
- The Service Device is taking an unusually long to process the message.
- The message was lost in transit due to faulty hardware between the two components, i.e. the Service Device is operational, but the network is unreliable.

In the scenario where Alice's User Device has stored the tokens but does not receive a response from the bank's Service Device, it is up to the User Device to resolve the consensus conflict because the User and Service Device will have divergent information.

By convention, it is up to Alice to retry the attempt with the bank. Still, the User Device must communicate to the Interface that the operation did not succeed, where the same uncertainty is met: it is impossible to guarantee that this communication will reach the Interface where Alice initiated the procedure, similar to the case described in the Two Generals problem.

From the Service Device's perspective, as the receiver, it faces the consensus problem in a similarly problematic way. When a message is lost in transit, the Service Device:

- Does not know that a message was ever sent.
- Cannot request retransmission of a message it doesn't know existed.
- May receive duplicate messages if the User Devices decides to retry.
- Has no way to distinguish the concepts of a 'no message sent' and 'message lost in transit'.

This is a consensus problem where neither party can reach a definitive conclusion about the larger system state. These failure scenarios clearly illustrate the practical consequences of the consensus problem for application and system developers. The inability of both the sender (User Device) and receiver (Service Device) to agree on whether a message was successfully transmitted and processed reflects the fundamental challenge of achieving consensus in a distributed system.

As the sender, the User Device cannot safely retry without risking duplicate processing, and the Service Device, as the receiver, cannot proactively request missing messages. This uncertainty persists even with acknowledgement mechanisms provided by TCP, as acknowledgements are subject to the same uncertainties: they can be lost or delayed.

In the event of a successful send and receive, the `AuthorizationApproval` is received by the Service Device, and the secure service identifier, auth token, client identifier, and secret are stored in another Postgres database associated with the Service Device, as seen in step 6. Contrary to the User Device, the authorisation procedure step is not stored on the Service Device as it only needs to handle one message.

Step 6 is also affected by the consensus issue because even if the two separate processes, the Service Device and database, are observed by the same CPU, they are unbound by time. The bank's Service Device process tries to store the tokens by communicating with the Postgres database. Still, the database process might fail arbitrarily, either by crashing or by the CPU being occupied with other tasks and, therefore, slow. This creates uncertainty, as the bank's Service Device process cannot know whether the tokens are successfully stored before the database process becomes slow or unresponsive, crashed before completing the writing operation, is just taking a long time to respond, or is experiencing other problems, such as internal queuing in the database process. Either reason can be equally valid, but it is impossible to know which it might be [5]. Even if the Service Device process receives no response and assumes failure, the database might have succeeded in storing the tokens but failed before acknowledging the operation. This fundamental uncertainty means the Service Device process cannot reach a definitive consensus about the database operation's actual state. When these uncertainties occur, the component experiencing them must decide on an appropriate resolution, as it is the synchronisation point.

As in the first example, this scenario exemplifies consensus issues by demonstrating how uncertain states can persist in the system, highlighting the practical implications of the consensus problem. In distributed systems like Mimir, it is impossible to determine whether operations have succeeded or failed under the assumptions of asynchrony set forth by the consensus problem.

Upon success, a message indicating `AuthorizationProcessComplete` is constructed and returned to the User Device with the received state and service identifier, a success value, and the destination where the Interface should be directed, as seen in step 7. When the User Device receives the `AuthorizationProcessComplete`, it checks the stored enum through the unique state and service identifier to confirm the current step of the ongoing procedure. Upon a successful check, the destination to send the interface is used to construct a `AuthorizationRedirect`, and the enum is updated accordingly. The incoming messages to the procedure are read and handled through a loop, and recursion is utilised to ensure that the resulting `AuthorizationRedirect` message can be passed back to the interface through the originating TCP stream. A final update to the enum is first performed to reflect the completed procedure. In this case, the update is performed before the message is returned to the initiating Interface,

as seen in step 8. The interface is then considered authorised. If any authorisation steps go wrong, an error message is sent to the Service Device and interface, and the mobile phone is notified. The message flow can be seen in Figure 3.3, and the message protocol can be seen in Figure 3.4.

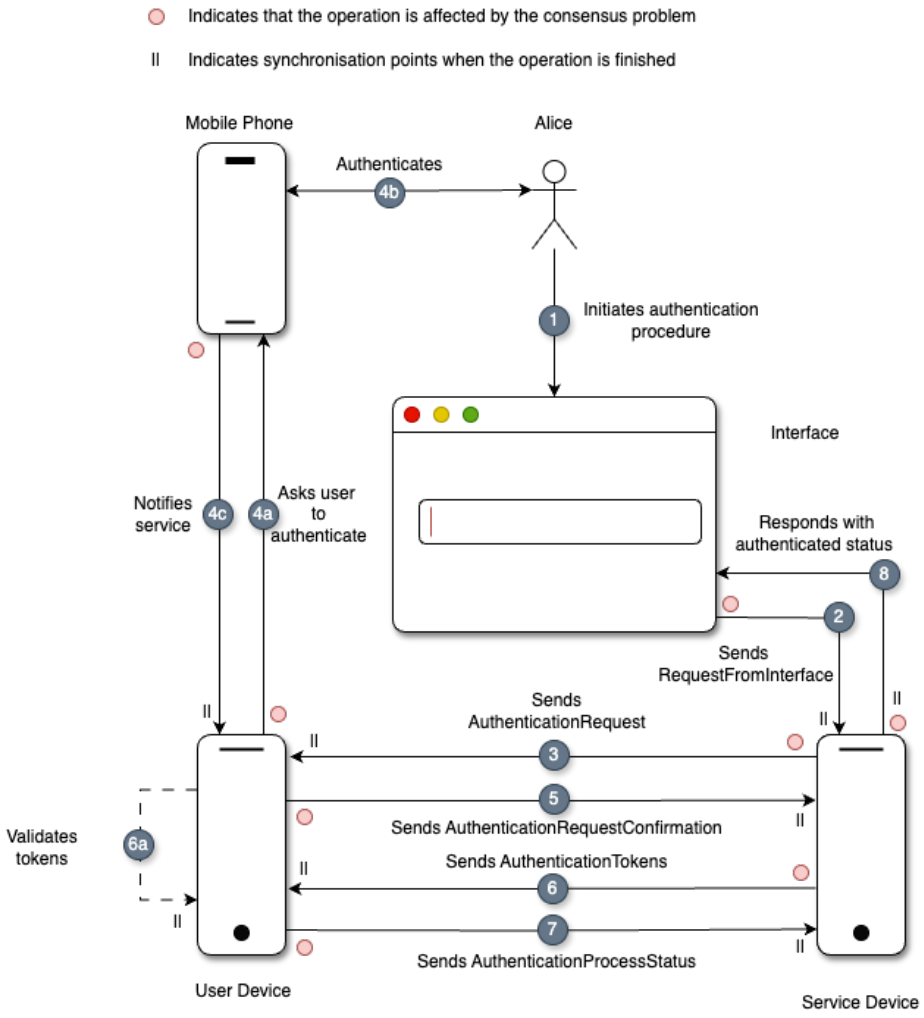

Figure 3.5: The authentication procedure in Mimir.

3.2.2 Authentication

Alice starts the authentication procedure by interacting with the bank's Interface, as seen in step 1 of Figure 3.5. A `RequestFromInterface` message is sent from the Interface to the Service Device, which is step 2. The service identifier generated during authorisation is used for this procedure instead of the human-readable name. When the Service Device receives the message, two checks are performed. First, it is verified that no other authentication procedure with the specific identifier exists. Second, the existence of the user is confirmed. Similarly to the authorisation procedure, the `RequestFromInterface` contains the IP address and equal steps are taken to parse it, append the default port if missing, log the authentication attempt with a generated state, the service identifier and an enum to track the procedure before establishing a separate TCP connection to the User Device. The enum is modelled after the message protocol, seen in Figure 3.6. With the connection established, an

`AuthenticationRequest` message is constructed with the service identifier and unique state and sent to the User Device, as seen in step 3. All messages are wrapped in a `MimirMessage` before being sent, and a five-second timer is similarly imposed to wait for a response. Recursion and a loop are again utilised to ensure the resulting message can be passed back to the initiating Interface.

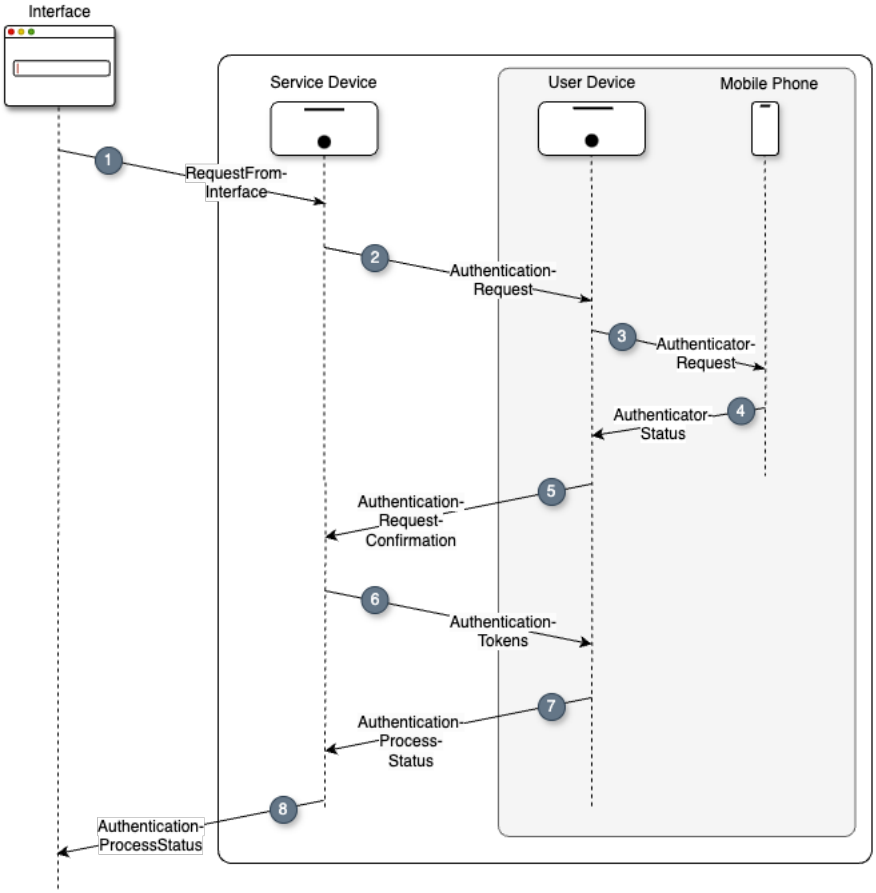

Figure 3.6: The authentication protocol diagram.

When the `AuthenticationRequest` is received by the User Device, a check is performed to verify that the service identifier exists as authorised. The authentication attempt is logged with a separate enum due to multiple messages being passed between the components, as seen in Figure 3.6, and to track the progression on both sides of the connection. The User Device sends a request to the mobile phone, establishing synchronous communication with the user to approve or deny the request. A two-minute timer accounts for user-related delays, as seen in steps 4a-c. A timed-out request or denied response will result in an invalidated procedure, and by convention, it is up to Alice to restart the authentication procedure if that happens. With the request logged and the mobile phone's blessing obtained, an `AuthenticationRequestConfirmation` is constructed with the unique state and service identifier and sent back to the Service Device, shown in step 5.

The tracked procedure state is verified against the stored enum on the Service Device when the `AuthenticationRequestConfirmation` is received using the state and service identifier contained in the message before confirming that the authenticated user exists again. The latter is required as a second layer of verification, given that the succeeding action will expose the authentication tokens. If the checks are success-

ful, the enum on the Service Device is further updated to the next step before the authentication tokens associated with the service identifier are retrieved. The service identifier, along with the JWT-based authentication token, the client identifier and the state, are then used to construct an `AuthenticationTokens` message, which is sent back to the User Device and the enum updated, depicted in step 6.

With the incoming message, the state and service identifier are used to confirm that the procedure is at the correct step before the enum on the User Device is updated again. Following the update, the service identifier is used to verify that the `AuthenticationTokens` belongs to the service. The JWT verified the Service Device's authenticity using the public key from the authorisation procedure, ensuring that the value matched the one used during authorisation, as seen in step 6a. An `AuthenticationProcessStatus` is finally constructed, containing the state and service identifier, with an optional info value containing status information about the procedure or reporting about failures. Upon success, the info value is not used. Finally, before the `AuthenticationProcessStatus` is sent back to the Service Device, step 7, the enum on the User Device is updated again. The
`AuthenticationProcessStatus` is returned to the Interface in step 8, completing the process and considering the interface authenticated.

If any steps during authentication fail, an error message is displayed on the Service Device and sent to the Interface and User Device, which notifies the mobile phone. The message protocol can be seen in Figure 3.6, and the message flow can be seen in Figure 3.5.

Domain names were considered for use instead of IP addresses, as IP addresses could change. However, domain names would need to be resolved to be associated with an IP address, and the resolver could be poisoned, which was considered an extra complication.

3.3 Procedure Limitations

The consensus problem proves that even the simplest form of guaranteed consensus, where components must agree on a single bit value, is impossible to achieve [5]. In Mimir, authorisation and authentication are restricted to one ongoing procedure at a time, constructing a more controlled environment where users can grant access to their data. By reducing the procedures to one ongoing procedure, a system acknowledges that adding more ongoing procedures will not help overcome the consensus problem's fundamental limitations and will only append load to the CPU. The approach reduces complexity by recognising that users' cognitive behaviour naturally limits them to focusing on one procedure at a time. The procedure restriction prevents the mobile phone from being flooded with requests while making procedure-specific faults more manageable.

Restrictions on ongoing procedures are performed by Mimir as follows: The User Device, which represents the user, is restricted to a single ongoing authorisation procedure using the name of the service that seeks authorisation. The authorisation procedure is shown in Figure 3.3. In contrast, the Service Device enforces the restriction for authentication using the service identifier generated during authorisation. While multiple authentication procedures can be executed simultaneously, only one is permitted for each authenticated User Device. The authentication procedure is shown in Figure 3.5.

3.3.1 Partial Synchrony

An upper time limit is set for each step in the procedures to establish a maximum time-bound, and this time limit then acts as a synchronisation point to conclude the procedures at both components. In distributed systems, partial synchrony entails a theoretical model where the system might experience initial asynchronous behaviour, hoping that synchrony will eventually be reached [27]. Therefore, a maximum processing time between sending a message and an acceptable limit for a response to arrive is set to account for initial asynchrony. The procedure is invalidated locally by the timer to ensure that the component can agree with itself in the event of faults. A more deterministic environment is created by imposing the time restriction, where an unreceived message is communicated to the component that a fault is ongoing, and a conclusion can then be reached by invalidating the procedure.

Formalization

The timeout on all messages is formally expressed in the following way:

$$\forall m \in M, \quad T(m) \leq T_{\max}$$

Where M represents the set of all messages sent through the system, $T(m)$ denoted the time taken to process a message $m \in M$, and $T_{\max}$ described the maximum time allowed for the receiver to process the message.

In cases where a message's response is not received in time, the procedure is considered invalidated and abandoned for a later attempt. It is then unknown whether the receiver had been slow in responding or had crashed [5], shown in Conjecture 1.

Conjecture 1. *Message Response Uncertainty*
Let m be a message, R the expected response to m, t_s the time at which m is sent, t_r the current time, and τ an upper timeout bound. Then,

$$\mathit{send}(m, t_s) \land \neg \mathit{receive}(R, t_s + \tau) \implies (\mathit{slow}(R) \lor \mathit{crash}(\mathit{receiver})),$$

where:

- *$\mathit{send}(m, t_s)$: denotes that message m is sent at time t_s.*
- *$\mathit{receive}(R, t)$: denotes that the response R is received by time t.*
- *$\mathit{slow}(R)$: denotes that the response R is delayed beyond the timeout τ (i.e., $t_r > t_s + \tau$).*
- *$\mathit{crash}(\mathit{receiver})$: denotes that the receiving system has failed.*

If the response R is not received by $(t_s + \tau)$, it is uncertain whether the receiver is slow to respond or crashed.

We further denote that if a message was not sendable, then it cannot be receivable, and as such, a procedure remained incomplete as the message was not deliverable, as seen in Conjecture 2.

Conjecture 2. *Message Delivery Impossibility*
Let $m \in M$, and S be a computer system. Then:

$$\neg sendable(m) \implies \neg receivable(m, S) \implies \neg deliverable(m)$$

where:

- *sendable(m) denotes that message m can be sent*
- *receivable(m, S) denotes that computer S can receive message m*
- *deliverable(m) denotes that message m can be delivered*

These conjectures signify the importance of making the components conclude with themselves deterministically by underlining the uncertainties described by FLP and acknowledging that multiplying the procedure will only add more load and not make consensus more achievable. Since it is impossible to guarantee the receiving component's response and success, the sender has a resolution.

Limiting Mimir to one active procedure at a time removes the opportunity for an attacker to perform process flooding with repeated requests. However, other problems with scaling and distributed denial of service attacks come into play.

3.4 Heartbeat Synchronisation

The operational state of the User Device is continuously monitored through a sequence of heartbeats exchanged at ten-second intervals with the mobile phone, which observes and reports faults and inactivity related to the User Device. The mobile phone considers the User Device inactive and non-operational if a heartbeat is not responded to within five seconds, making ongoing faults detectable in the asynchronous system. A phone notification is sent to alert about a fault when the non-operational state is discovered. The synchrony between the user and the mobile phone is used to explain whether the fault is related to the Authenticator, for example, lost connection [3], or the User Device. If the fault is at the User Device, it is impossible to remotely distinguish between a crash and a slow, overloaded process [5] without synchronised communication with the component through physical access.

During authorisation and authentication procedures, the User Device notifies the mobile phone about the ongoing procedure to establish synchronised communication with the user. Consequentially, no data can leave the User Device without the user's explicit permission through biometrics on the mobile phone and synchronised communication. Biometric approval is prompted by the mobile phone, requiring synchronous interaction within two minutes for the procedure to continue. The two-minute timer limits the duration of procedures at the User Device and Service Device before concluding that the procedures are invalid if the timer runs out, Mimir requires a new procedure in such a case.

The mobile phone made two attempts every minute to re-establish a lost connection with the User Device, provided that mobile service or Wi-Fi is present. If the connection is not re-established during the attempts, synchronised communication will need to be established physically with the User Device, as a more serious fault is likely

[3]The phone might have lost connection due to unforeseen circumstances, i.e. travelling through a tunnel, entering flight mode, poor reception, or running out of battery.

ongoing at the component. Through this approach, Mimir attempts to work within the limitations imposed by FLP's consensus problem on asynchronously distributed systems, where guaranteed failure detection and consensus are impossible, using the timer as a practical compromise to invalidate procedures at the components locally, in case of unresponsiveness or delays among the other components. The consensus problem in asynchronously distributed systems necessitates synchronisation on some level, physical or otherwise.

The primary biometric interaction throughout the development was Face ID authentication on an iPhone 13 Pro Max.

3.5 Data Synchronisation

A solution to the consensus problem is to break the asynchronous assumptions and opt for synchronisation in the system. Two separate enums are constructed in Mimir to track the authorisation and authentication procedures, enforcing a specific sequence for message processing. Significant steps in the procedures are represented through the enums, which reflects steps in the protocol, illustrated in Figure 3.4 and Figure 3.6. Updates to the enum are made at relevant points in the procedures, such as between sent and received messages and after calls to the mobile phone. Since a solution to the consensus problem is to synchronise and break the assumptions of asynchrony, it is impossible to guarantee agreement for even a single bit value without synchronisation, and the enums served as minor synchronisation marks, or checkpoints, for the procedures, individual to the component. These checkpoints established an orderly sequence for interactions among the distributed components of Mimir, in line with Lamport's principles on maintaining order in distributed systems [15]. It is impossible to guarantee that the Service- and User Device will reach the same conclusion on a procedure. The checkpoints instead allows the components to synchronise the procedure locally and provide an ability to know what message to expect next.

If the steps in the procedure are not performed as expected, the sequence of events would be broken, and the procedure would then know that a problem is ongoing, but the system can not know why a problem has occurred under the consensus problem [5]. Although nothing stops Mimir from trying to share instances of the enum, this set of virtual synchronisation across devices does not scale as more nodes are added to the procedure, as shown by [28], [29], [69][4]. A shared database could have been used for this purpose, but it would have caused a major security risk, as the User Device was a separate entity with an individual user as its owner. The Service Device belonged to a service, and the "sharedness" would result in a coupled process between the components at the expense of independence and separation of concerns.

The enum validates the relevance of the incoming message to ensure it is processed only at the right moment during the procedure. Divergent states among the components would have resulted from wrongful processing, posing a security risk: A service or user could have been authorised by an attacker without fully performing the required procedure. The individual states are stored separately on the Service Device and User Device. Each instance is updated by its respective component as the procedure progresses. An ongoing procedure without the expected response will eventually reach the maximum allowed time and invalidate the entire procedure, requiring a restart to become authorised or authenticated. This structured synchronisation is

[4]Gupta and Kumar first published the proof, but Scaglione simplified it significantly.

constructed to offer minor checkpoints individual to each component for the procedure, track procedure progress, and prevent the components from diverging states through an orderly message flow sequence. Reaching the maximum allowed time forces the User Device and Service Device to conclude even if the other component does not respond or responds with the wrong message.

Communication between the components is conducted through sockets over TCP, guaranteeing the same order for sent and received messages. Delivery of the messages cannot be guaranteed, however, as proved by the consensus problem [5], so a built-in retransmission timeout mechanism is included in TCP with a delay that increases with each new attempt until the maximum number of attempts is reached. The delay, along with the timer imposed on each message, is utilised to invalidate the procedures at the components, as indefinitely delayed messages or maximum attempts of retransmission entail a terminated connection. Extra CPU power is required by this retransmission and delay.

The checkpoints used by Mimir enforce an expectation on the order in which messages are to be received, similar to Lamport's "happen before" [15], which creates local synchronisation points to navigate the uncertainties in asynchronous communication, aligned with FLP's theoretical constraints.

3.6 Fault Tolerance

Several situations were identified that are not directly related to the consensus problem but might have amplified consensus issues, inconsistencies, and fault triggering. For example, data formats that lack type safety, buffer overflow, dangling pointers, or memory overwrites, and mistakenly identifying Service Devices. The specifics of fault tolerance can also be related to attack resilience, where these faults open up opportunities for triggering attacks towards the system and render Mimir non-operational. Requirements of authentication and authorisation are aligned with consensus, as both the User Device and Service Device must agree on the states for the user to be considered authenticated or authorised, and in situations where faults might occur, these faults might impact the reliability of the states. In this section, we elaborate upon their relevance to the consensus problem.

3.6.1 Data Communication

A specific protocol is defined for data communication between the components in Mimir. It is observed that the specifics of data communication are not directly related to the consensus problem. Nevertheless, a system with proper data handling will reduce the likelihood of run-time faults through increased robustness, making the possibility of encountering consensus problems less frequent. While true in Mimir, it also holds for any data communication between computers. Two data serialisation formats were considered for Mimir's communication: JSON and Protocol Buffers (Protobufs). Although other serialisation formats exist, such as XML and YAML, they were ruled out due to the need to write scaffolding code and the indentation requirements. JSON is a widely adapted, lightweight, language-independent data serialisation format that uses name-value pairs to represent and interchange data between web applications and servers [70]. The data format is flexible and can accommodate rapid changes to schemas, and data can take many forms. JSON imposes syntax requirements but

does not do any schema-type enforcement. The definition for JSON does not offer any language-specifics on types or parsers, and implementation and support are left to developers [70], which then varies across languages. The inherent lack of type-checking can lead to scenarios in which an attacker triggers internal faults in one of the components by passing malformed, unexpected data that crashes the component due to improper error handling or storing inaccurate information about services and users.

Protobufs is a compiled binary data serialisation format that is not readable in transfer. It enforces a strict process for defining language-agnostic data structures through the requirement of types, offering reusability. Data structures in Protobufs, called messages, are compiled into language-specific structs during the project build process before use, and the expected types are well-defined for several languages [71]. Because of the compilation requirement, changes to a defined schema require consideration for backward compatibility.

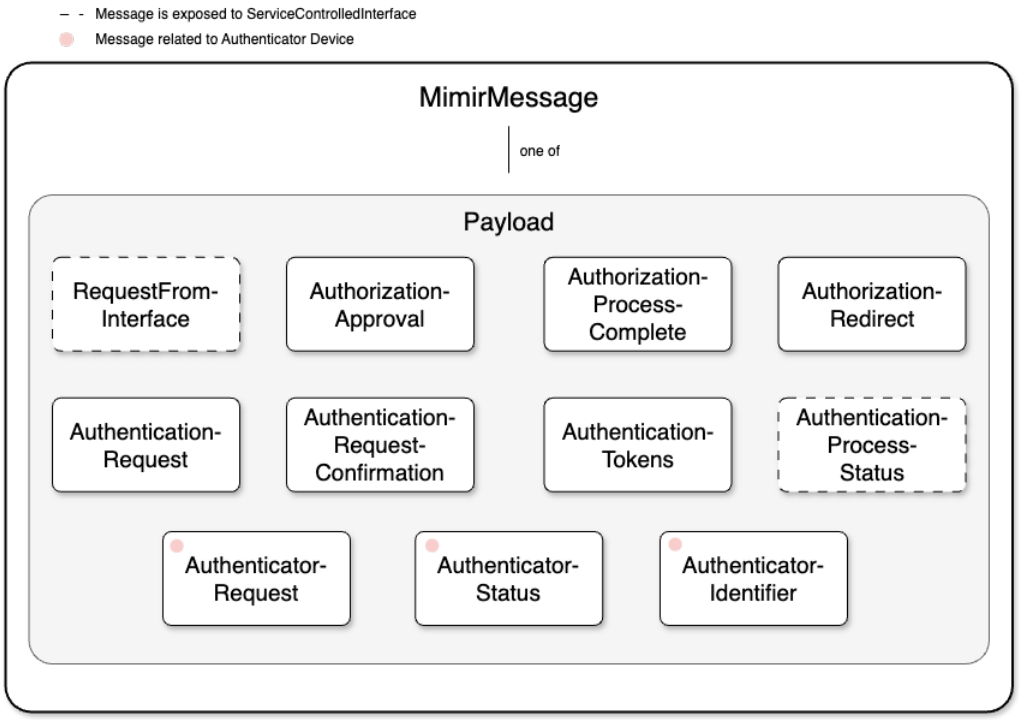

Figure 3.7: MimirMessage with Payload options.

Protobufs are elected for Mimir because of backwards compatibility and strict typing definitions across languages. Mimir's messages are shown in Table 3.1 and are defined to facilitate communication between components. Each message is designed to share the least amount of data while attempting to maintain the procedure state between the components. A unique, securely generated state identifier is attached to each message to track which procedure it belonged to. Before being sent, messages are encapsulated in an enum within a `MimirMessage` Protobuf. The approach is selected to improve Mimir's ability to accept or reject unintentional data, preventing fault occurrences. The messages and enum encapsulation can be seen in Figure 3.7. Due to Rust's type safety, empty messages could be sent, but the User Device and Service Device are designed to disregard messages not intended for them.

Regarding distributed computing and the consensus problem, these messages ensured consistency in message formats across Mimir and facilitated serialisation for network communication. These strongly typed data structures provided a standard-

ised way to represent data, reducing the risk of ambiguity in the communication among the distributed components. They also enforce careful consideration when evolving the schema and warn about changes whenever the Protobufs are used.

MimirMessage	**Description**
RequestFromInterface	Incoming message from Service Device, triggering the authorisation or authentication procedures.
AuthorizationApproval	Authorisation is approved by the mobile phone, constructed by the User Device.
AuthorizationProcessComplete	The Authorisation procedure is complete, constructed by the Service Device.
AuthorizationRedirect	Returned to the connecting Interface upon completion, which contains where to send the Service Device.
AuthenticationRequest	Initiated the authentication procedure from the Service Device and sent it to the User Device. Used to verify that the unique service identifier existed on both sides.
AuthenticationRequestConfirmation	Indicated that the unique identifier existed at the User Device.
AuthenticationTokens	It represents tokens stored on the Service Device and used to verify their authenticity during authentication.
AuthenticationProcessStatus	Contains information about the authentication procedure. Success, failure, and other relevant information.
AuthenticatorIdentifier	Used to distinguish the mobile phone from other incoming connections to the User Device.
AuthenticatorRequest	A message is sent to the mobile phone from the User Device to synchronise with the user.
AuthenticatorStatus	Response from the mobile phone, whether the user approved or denied the request.

Table 3.1: The messages and their intentions in Mimir.

3.6.2 Cryptographically Pseudorandom Identifiers

Cryptographically secure pseudorandom identifiers are constructed by the operating system where Mimir is running. These identifiers distinguish unique authorisation and authentication procedures from one another and make out the authentication tokens during authorisation. Upon initiation, each authorisation and authentication procedure is assigned an identifier, which is used as a synchronisation point between the User Device and Service Device. A 32-byte identifier is constructed to ensure that services are distinctly identified, thereby minimising the risk of inaccurate tracking or incorrect procedure execution due to misidentification. The 32 bytes are intended to help minimise identifier conflicts in an environment where many procedures can happen, leading to consensus issues amongst the procedures and components, materialised as faults. The selection pool used to construct identifiers should be significant enough to make a guessing or brute-force process redundant for an attacker through probabilistic uniqueness. Identifiers are generated from the set of capital and lowercase alphanumeric characters in English, unionized with the set of `{!, $, %, ^, &, *, @, #}`.

The combined domain space of characters creates a set of 70 elements, providing probabilistic uniqueness, where 32 samples are picked, permitting repetition. The result represents all possible elements within the domain space, and the option calculation can be seen in Figure 3.8.

$$70^{32} \approx 1.10442 \times 10^{59}$$

Figure 3.8: Opportunities for unique identifiers.

3.6.3 Programming Language

Rust is used to develop Mimir, and it is increasingly recommended as a language by NIST [72] due to its strict and robust type safety, performant execution of low-level code, and emphasis on constructing a secure environment where most common memory mistakes are eliminated [5] where the consensus problem could be explored. Rust enforces strict compile-time checks, and most common types of memory safety errors are prevented unless they are explicitly permitted through an unsafe block of code. No such block was used during development.

[5] A library exists which takes advantage of a bug in Rust and shows how to provoke some common memory errors [73]. Language creators are familiar with the issue [74].

Chapter 4

Discussion

For almost four decades, FLP's Impossibility Result has been a problem studied in computer science. In this section, we discuss the broader implications of our investigation and the practical steps used to attempt to operate effectively within the constraints of the consensus problem. From the start, it was acknowledged that the consensus problem is unsolvable in an asynchronously coupled system, so the focus was instead on creating an application where the implementation could be connected and attached to the consensus problem. Many attempted solutions to the consensus problem have been introduced [21], [22], [75] through creating nodes serving as leaders but without maintaining the asynchronous nature by using novel approaches to synchronise. Still, while these solutions address aspects of the consensus problem, none of them can guarantee unanimous consensus for even a single bit value in the presence of a faulty process, despite a high probability [75]. Regardless, the solutions remain relevant as consensus algorithms in the broader context. In contrast, our approach considered the impossibility result as a fundamental limit, which constraints should be navigated in practical systems. The methodologies used to navigate the consensus problem in a practical environment were: data synchronisation, procedure limitations, partial synchrony, and fault tolerance mechanisms.

4.1 Procedure Limitation

Imposing a procedure limit of one ongoing authorisation at a time for the User Device and one authentication per User Device connecting to the Service Device increased the system's security. Users naturally focus on registering or signing in with one application at a time, as human attention and cognitive load limit parallel task processing. This human behaviour aligned with Mimir's single-procedure restriction, preventing opportunities that could lead to unintentional authorisation of services. The system reduced the complexity of simultaneous failures by limiting Mimir to a single ongoing authorisation procedure and controlling the authentication procedures per service. Although the simpler environment and restriction can be seen as limitations, they represent architectural choices aimed at reducing complexity and acknowledging another aspect of the consensus problem: even the simplest form of consensus, agreeing on a single bit value, is impossible. Dumping more procedures on the User Device and Service Device would not have helped solve the consensus problem. It would have only increased the load across components and confused the user if the mobile phone had been called with multiple authentication and authorisation procedures. Mimir uses

TCP's built-in retransmission functionality to attempt to rectify lost messages. More requests related to the same procedure would have backed up the task log of operations to be performed on what might already be suboptimal conditions as faults occurred. The simplified environment improved the ability to trace faults where sensitive data is shared. This trade-off of procedure limitation acknowledged that achieving perfect solutions to unanimous system consensus is impossible. In this approach, the uncertainties surrounding a guaranteed consensus on the procedures were minimised by using partial synchrony with timers and reducing the space of the consensus problem but never eliminating it. Either the system would manage to agree on authorised and authenticated states, or it would invalidate the procedure, regardless of knowing with absolute certainty whether the other components were slow or crashed.

Restricting Mimir to one concurrent procedure reduces potential race conditions, where two procedures could be initiated simultaneously for the same service, which can lead to potential conflict regarding the procedure and the status of authorised and authenticated.

Mimir's attack surface was reduced by limiting it to one concurrent procedure at a time. This limitation prevented users from being overloaded with requests sent to the mobile phone and confusing them with the difference between a fake and genuine request. An attacker would need to access the unique procedure identifier generated when each procedure was initiated to gain insight into it.

4.2 Partial Synchrony

The implementation of an upper-bound time where initial asynchrony could occur, wagering that the procedure would reach synchrony before the timer ran out, was a compromise between fully synchronous and asynchronous systems, as detailed by Dwork et al. [27]. Mimir imposed a time limit for each step, creating a timed environment where components could acknowledge guaranteed consensus as impossible due to potential faults, and the mobile phone could prompt users to synchronise remotely with the device or physically with the User Device in cases of faults. Timer-based invalidation converted the potential of unpredictable external failures into predictable outcomes by ending the component's procedures at its time limit, even if it cannot distinguish between a slow response and a crash [5]. Despite establishing partial synchrony through timers, Mimir needed to find a conclusion for the authorisation and authentication procedures if consensus could not be reached within the maximum bound time limit. The procedure still operated within a context where message delays and timing uncertainties could occur. Therefore, invalidating the procedures when the timer expired created a fallback consensus local to the components, and this operation did not require the components to reach the decision collectively. It would instead happen individually on both components when the timer expired. The timer limited how long a procedure was valid, gambling that there was an initial asynchrony before the system could reach a synchronous state. It's impossible, though, to guarantee that the timer would be significant enough to outlive the asynchronous uncertainties. In a single procedure, several parties communicate multiple messages and each time a component sends a message, a new timer is needed because of the uncertainties described by Fischer et al. [5].

Considering the security of the timers, the most significant is the two-minute operation in which the user attempts to authorise or authenticate through the mobile

phone. The two minutes were used to allow sufficient time for the user to discover the device's location, locate it, and perform authentication while balancing a window sufficiently large to perform said actions and avoid an attacker's ability to hijack the procedure. The five-second timers attempted to balance the expectation of common delays and congestion on the network, where the average round-trip time between New York and Sydney is 200ms when measured with the `Ping` command. Five seconds was then considered sufficient time for a message to pass over the wire, be processed, and for a response to arrive under most circumstances. Mimir did not implement any specifics on encryption or verifying the device's authenticity, but a Public-Key Infrastructure under an encrypted connection would have more thoroughly integrated the User Device with the mobile phone.

4.3 Data Synchronisation

An important part of the asynchronous properties described in the consensus problem is that messages, through network communication, can experience arbitrary delays [5]. Separate, individual synchronisation points, called checkpoints, were created in Mimir through enums. They tracked ongoing procedures and prevented the components from proceeding into divergent states due to delays or miscommunication. Mimir used these checkpoints, combined with timers, to let the procedure wait for a few seconds, accounting for insignificant delays and waiting for the arrival of an incoming message or aborting the whole procedure if the timer ran out, separately reaching a conclusion for the procedures on a local level. This created an orderly sequence in which messages must be processed, in line with [15]'s "happen before".

We can observe a scenario where the User Device receives `AuthorizationProcessComplete`. Without the enum safeguarding the procedure, the User Device could receive the message and consider an arbitrary service as authorised even though there were no ongoing, associated authorisation procedures at the necessary step, leading to security vulnerabilities and consensus failures amongst the components. The enum ensured that the procedure was at the correct step before progressing, and inconsistency could be detected if the current step did not match the incoming message.

If the `AuthorizationProcessComplete` was received in an unexpected state, Mimir could handle the situation gracefully by sending a response back to the sender or ignoring it rather than allowing potentially harmful and inconsistent data to cause and propagate divergencies amongst the components. As messages can experience arbitrary delays or arrive out-of-order, checking the enum allowed us to detect and minimise these situations, a requirement imposed by potential network congestion, traffic and noise, leading to inconsistencies.

The checkpoints reflected the procedure's state and allowed the tracing of failures back to the exact location where they happened. While these checkpoints did not eliminate the fundamental challenges of consensus in asynchronously distributed systems, they created a more controlled, deterministic environment, working within the boundaries set by the consensus problem, minimising the occurrence of inconsistent states and reducing consensus problems through explicit state handling. It permits systems to know what message to expect next but does not offer a way to detect that a message has been lost in transit. Lost messages would break the orderly sequence in which Mimir expected messages to arrive. An unexpected message could be inter-

preted by the system as a hostile response from the communicating component and as a Byzantine General.

The challenges of maintaining accurate state reflection through the enums persisted across components, as updates needed to be coordinated with message transmission status and coordinated between the components. When messages failed to transmit due to faults, the system could end up in a divergent state where other operations had been completed successfully, but the final state update was reflected as incomplete, leaving the procedure in an uncertain limbo. Although TCP provides order delivery guarantees, it does not guarantee connection health. Therefore, it was necessary to confirm a successful transmission before updating the enum to accurately reflect the component's state. Moreover, the sender could not guarantee the success of the operations at the receiving component, which would introduce additional uncertainties and inconsistencies surrounding consensus and, therefore, require a fallback to messages without a response. Although the checkpoints indicated what message to expect during the procedure, they did not inform Mimir if a message had vanished or which one.

4.4 Consensus Challenges in Practice

Consider a traditional scenario for computer security, a malicious actor attempts to compromise a user account on an arbitrary service, where the account owner would be notified via email about the suspicious login attempt. If the account owner notices this email, a race condition occurs between the account owner and the attacker. This race condition is more than just a security concern. This is the direct consequence of the consensus challenges described by FLP, where timing and message delivery uncertainties directly impact the system outcome. The mobile phone, in this case, a phone, is an attempt to throw the consensus problem up to the final line of defence, namely the user, and provide a synchronisation point through the user with a different timing concern in a more direct, prioritised line to the user. In turn, the attack surface is moved from the mailing service to the mobile phone, and concerning the consensus problem, the issues are just moved to a separate device, instead of serving as a solution.

When faults or hard crashes occur, synchronous communication, often human interaction, is necessary to correct the mistake. A European electronic ID provider reported the same issue. The provider's biggest expense is the manpower and technical support needed for password resets over the phone, highlighting that establishing these synchronization points comes at a significant cost.

This scenario plays out regularly in distributed systems we interact with daily. When an HTTP connection terminates after a timeout period, or TCP takes advantage of using SYN timeout mechanisms, the same fundamental challenges are being grappled with - that of the impossibility of distinguishing between crashed components and a delayed response due to a slow process somewhere in the network model. A browser's "Connection timed out" message represents the same uncertainty as an undelivered or unseen authentication notification email.

Moving the notification from a typical email to a dedicated application creates a more deterministic communication channel aligned with the partial synchrony assumptions described by Dwork et al. [27]. Apache Kafka, a distributed event streaming application, is another external tool which performs similar measures in timeouts and leader election processes. When a timeout for a heartbeat is passed in Kafka, it can

not determine whether the node is slow in responding or crashed, facing the same uncertainty as Mimir determining whether an authentication attempt has failed or is delayed.

The parallel extends to other major distributed systems. DynamoDB's implementation of distributed locks uses lease-based mechanisms with timeouts, acknowledging that perfect consensus is impossible. Similarly, Mimir's requirement for a dedicated device introduced a security barrier through biometric authentication and a consensus boundary in the sense that the phone became a participant in the distributed procedure, where the lack of the phone would make the procedure impossible. It then becomes paramount to consider scenarios where a phone, an expensive device, can be lost or stolen, ending up in China [1]. As such, the need for fallback mechanisms to re-establish synchronised communication is crucial in those events. The use of phones makes operating the consensus problem even more necessary. Phones can experience the same delays as computers and drop connections with the network provider, serving as a second front to tackle.

An additional layer is introduced by requiring physical devices. While it wasn't the primary consideration, it creates an extra barrier against spammers by imposing a physical cost per authentication instance - similar to rate limiting, and resource allocation serves as a practical constraint in other distributed systems, like Application Programming Interfaces (API).

These practical demonstrations of the impossibility result contribute an important insight: Practical systems and consensus considerations demonstrate how theoretical constraints shape practical system design. Whether it is Mimir deciding to abandon an authentication attempt after a timeout, Kafka determining node health, or a web browser giving up on establishing a connection, these systems are all governed by the consensus problem through establishing synchrony. The key is recognising that these aren't just engineering compromises but necessary responses to the fundamental computational limitation of distributed system constraints.

4.5 Scaling

Scaling introduces new complexities for achieving consensus across the distributed system, with increased node count and network complexities, making practical approaches more resource-heavy. In the context of Mimir, scaling became less relevant as each User Device belonged to one user, and it only connected to Service Devices during authorisation and authentication. On the other hand, scaling for Service Devices is still relevant, given that multiple authentication procedures could happen simultaneously. In Mimir, these must be coordinated through centralised means, such as a database and a single processor on a component. Relying on a centralised synchronisation point does not scale because there is a limit to how much a node can process [29] where waiting times like partial synchrony increase the workload for a CPU as more nodes are added to the network. The throughput potential of networks is severely constrained by N nodes within a network [28], and Gupta and Kumar established that with increased network sizes, available bandwidth (and processing power) decreases due to routing overhead at a rate of $O(1/\sqrt{N})$ [29]. Then, the question becomes how long a component should wait before abandoning a procedure. The goal is to find a balance that

[1] From an article in The Times, it was reported that stolen devices in London end up in a specific street market in China, where iPhone owners track their devices [76].

doesn't degrade component performance when faults occur and as more procedures are ongoing. Algorithms like Raft [22] and Paxos [21] serve as partial solutions to the consensus problem because of the network limitations described in [28], [29], [69]. The algorithms relax the assumption of guaranteed consensus to circumvent the consensus problem, and as a network scales, communication overhead increases [29] along with the potential for node failures, and nodes can split into smaller subnetworks where they become unreachable. There is no way to guarantee that these faults do not occur.

Modern distributed systems must balance theoretical limitations with practical requirements for scalability, which leads to a probabilistically high likelihood of success through lowered expectations on the grounds of operational feasibility, compared to the properties set forth by FLP [5]. The consensus problem highlighted by Fischer et al. is more relevant in large-scale system deployment, where network asynchrony, partial failures, and workload for synchronisation points are not just theoretical concerns but operational challenges.

4.6 Fault Tolerance

A fundamental property of the consensus problem is a faulty process in asynchronously distributed systems. Therefore, preventing faults which can be controlled became a critical aspect of working with the consensus problem. In Mimir, fault tolerance was handled using three strategies: data communication, selection of programming language, and cryptographically secure pseudorandom identifiers. By preventing simple errors and limiting escalating faults, the likelihood of meeting the consensus problems could be minimised, and a more robust framework could be provided where fewer faults would occur. Fault tolerance can also be considered attack resilience, which improves security and is essential for applications such as Single Sign-On services.

4.6.1 Data Communication

As data communication is the source of system failures that can escalate into consensus problems, the type-safety offered by Protobufs was seen as an advantage in making it more applicable in distributed computing compared to other data formats, such as JSON. JSON's lack of default type-checking can lead to various issues, such as runtime parsing errors, silent data corruption exemplified by precision loss with large numbers, and challenges related to schema evolution. While these issues are not directly linked to consensus problems, they can exacerbate problems in asynchronously distributed systems by introducing additional points of failure and inconsistencies, ultimately hindering consensus.

To illustrate the importance of type safety, we observe a scenario with type-less data serialisation, where the absence of strict typing can lead to faults through improper distribution of malformed packets to other components. A trivial example is a node that inadvertently distributes an unparsable string insteamd of an integer, causing errors in the receiving component without type-specific parsing. These errors can cause system crashes, halt operations, and prevent the system from reaching consensus.

In Mimir, type constraints were enforced by Protobufs during serialisation and deserialisation, acting as a bouncer to prevent type-related faults from reaching the system's core functionality. This enforcement, combined with Rust's compile-time checks, collectively reduced runtime faults by ensuring that types were consistently

specified and maintained throughout the system, thereby limiting the effects risks of errors during data transmission and processing. The strict and shared communication prevented faults from spreading to other nodes, even when the received data was not associated with any authorised parts or ongoing procedures in Mimir. Malformed data could cause system failures or crashes, requiring synchronised intervention, often through humans, to correct. The specifics of types between languages are well-defined by the maintainers of Protobufs [71], which decreases ambiguity among developers across systems where different languages are used. Figure 4.1 compares malformed JSON and its counterpmwart in Mimir's Protobuf implementation.

Mimir's AuthorizationProcessComplete

```
1  message
2    AuthorizationProcessComplete
3    {
4    string state = 2;
5    bool success = 3;
6    optional string where_to = 4;
7    string secure_service_id = 5;
8  }
```

Malformed JSON

```
1  {
2    state: 456467457432654.45454,
3    where_to: [127, 0, 0, 1],
4    success: "false",
5    secure_service_id: 55,
6  }
```

Figure 4.1: Comparison between Protobuf and JSON.

Despite the shared message structure in Mimir and Protobufs' ability to enforce type validation, this approach does not directly solve any aspects of the consensus problem. Instead, it highlighted the opposite: the potential for more crucial consensus and security issues arose when loosely typed data serialisation formats were used. Messages travelling over the Internet encounter noise and faulty hardware, which can cause data corruption in transit, and strict types serve as bouncers during serialisation to these errors.

In examining these network issues, we find that they make it impossible to determine whether the communicating processes are simply slow or have crashed, as described by Fischer et al. [5]. The issues may be more relevant for the consensus problem today due to network complexity with more devices added daily. They align with the consensus problem and can escalate into faulty processes.

Timing synchronisations were not a result of this shared communication. Instead, standardisation in communication created a synchronisation point that reduced ambiguity in message interpretation and offered practical improvements. This was possible despite the theoretical impossibility of guaranteed consensus, and large-scale systems still need to be built. In asynchronous systems, where the consensus problem made unanimous guarantees impossible, the solution is to synchronise, which does not scale with the introduction of more nodes, complexity increases, and the need for expensive human interaction to recover situations where faults occur, acting as a final point of synchronisation.

The requirement to compile messages provided a way to prevent divergent schemas without careful consideration and warning across the entire system.

Performance

While the primary motivation for using Protobufs was type safety and fault tolerance, the performance benefits, seen in Table 4.1 [77], provided a minor addition to our

time-bound implementation. Although this is unrelated to the impossibility result, it added an almost insignificant reduction in the likelihood of procedures being incorrectly considered invalid by Mimir due to performance-related delays as a compliment to the fault tolerance.

Metric	**JSON**	**Protobufs**
Binary size (bytes)	781	687
Avg Serialization (ms)	4.177	2.339
Avg Deserialization (ms)	1.199	0.298

Table 4.1: Comparing data formats through books [77].

4.6.2 Programming Language

Rust was chosen as the primary language for implementation due to considerations regarding fault prevention and guarantees against common system-level programming errors, such as dangling references, uninitialised variables, null pointer dereferencing, memory overwrites, and buffer overflow, and it is one of NIST's recommended languages [72]. Rust's focus on memory management with ownership and borrow-checking during compilation offered guarantees to prevent these implementation-specific errors from interfering with faults outside our control.

Unlike established, older languages like C, which operates without built-in safety measures, Rust requires developers to mark unsafe code explicitly. This approach gives programmers more control over potentially dangerous operations while maintaining system safety elsewhere. Rust's memory management and static, declarative typing help prevent common memory errors from spreading throughout distributed systems. By isolating unsafe code, developers can identify and manage potential problems before they can affect system-wide operations and interact with practical consensus-related challenges.

Rust's ownership model, however, presents two significant challenges. First, lifetime management of variables creates additional complexities, often resulting in more verbose code, which can be problematic in distributed systems, where code clarity is essential. Second, the ownership rules make it difficult to share data between different parts of the program, such as threads. While solutions exist through channels, implementing these can lead to complex code that is harder to maintain and follow. These limitations make Rust less suitable for complex, distributed systems.

4.6.3 Standard Calculation on Cryptographic Security

Cryptographically secure pseudorandom identifiers were essential to operate the constraints of the consensus problem by distinguishing various aspects of Mimir, and a collision between these identifiers would lead to consensus problems where procedures could wrongfully progress.

A standard approach to illustrate the mathematics and verify the collision rate and security of the 32-byte identifiers follows: By declaring X the random variable as the number of collisions following the Poisson distribution, a probability calculation was performed to determine the chance of the first collision ($X = 1$) between them. The birthday paradox, which was used to approximate the probability of a collision occurring and to calculate the expected number of generated strings before the collision

occurred, where λ was derived as $\lambda = \frac{n^2}{2\times N}$. n represented the number of generated identifiers, and N the total number of possible unique identifiers, reflecting the expected number of collisions. The calculation was estimated based on the generation of 10^9 strings, where the probability of a collision is 1 subtracted from the probability of no collision occurring, i.e. all strings were unique, seen in Equation (4.2) and Equation (4.3). The probability of no collision can be expressed as a product series:

$$P(\text{no collision}) = \prod_{k=0}^{n-1} \frac{70^{32} - k}{70^{32}}$$

For an n small relative to possible outcomes, this product can be approximated using the exponential function:

$$P(\text{no collision}) \approx \exp\left(-\frac{n^2}{2 \times 70^{32}}\right) \tag{4.1}$$

$$\begin{aligned} P(X = k) &= \frac{\lambda^k e^{-\lambda}}{k!} && \text{(Poisson PMF)} \\ P(\mathrm{X} = 1) &= 1 - \exp\left(-\frac{n^2}{2 \times 70^{32}}\right) && \text{(General case)} \end{aligned} \tag{4.2}$$

$$P(\text{collision}) \approx 1 - \exp\left(-\frac{(10^9)^2}{2 \times 1.19 \times 10^{60}}\right) \approx 4.2016 \times 10^{-43} \quad \text{for } n = 10^9 \tag{4.3}$$

Calculating string collisions through the Birthday paradox.

The Poisson approximation was used to determine the timing of a collision, based on an estimated constant string generation of 10^4 per second, seen in Equation (4.4).

$$\begin{aligned} T &= \frac{n}{r} = \frac{1.54 \times 10^{30}}{10^4} = 1.54 \times 10^{26} \text{ seconds} \\ T &= \frac{1.54 \times 10^{26}}{3.154 \times 10^7} \approx 4.88 \times 10^{18} \text{ years} \end{aligned} \tag{4.4}$$

Using Poisson to approximate years before identifiers would crash.

Both calculations showed that the identifiers at a length of 32 bytes would have an almost negligible probability of colliding at a billion generations, and only after 10^{18} years would the identifiers collide at a rate of 10,000 generations per second. This showed that the identifiers could represent ongoing procedures and the other aspects of Mimir that needed to be unique, while the chance of meeting the consensus problem due to a crash was near zero.

The probabilistically unique domain space for the identifiers minimized scenarios like a replay attack with the identifiers attached to the procedure, as each procedure contained an identifier uniquely generated upon each initiated procedure.

4.7 Experiments

A script was written to test Mimir's ability to withstand completely random messages sent to it. The script iterated on various types of Protobuf messages sent in a series of events with randomised data, where the intention was to test whether either of the messages, regardless of data, would cause Mimir to crash. The script established a TCP connection, encoding a Protobuf to bytes and sending the message before closing the connection again with a slight delay of one millisecond between each message, and performed this task 10^4 times. Each message consisted of randomised data generated through the Rust `Fake` crate, and none of the messages made the User Device or Service Device crash.

A second script was written to test Mimir's ability to handle incoming messages that did not belong to any procedure. Similarly to the first script, a connection was established, and a 32-byte identifier was generated and sent in a Protobuf. The identifier did not belong to an initiated authorisation or authentication procedure but a later part of both. The script iterated 10^5 times and generated messages before sending them to the User Device and Service Device through separate iterations, which tested Mimir's ability to handle the orderly sequence, as per [15]. An error was displayed as intended, but the messages did not crash the instances and weren't processed.

Chapter 5

Conclusion

In this thesis, we presented Mimir, an asynchronously distributed Single Sign-On service that leverages mobile phone biometric scanners to verify the user's identity. A Single Sign-On service allows users to use the same credentials to sign into multiple websites and applications linked to the service.

Mimir's architecture consists of User Devices that store information about linked services and represent a user, Alice, in the sign-in procedure, and mobile phones in the user's possession with biometric capabilities. The user is responsible for these components. The architecture also consists of Service Devices that store information about Alice and other associated users, as well as Interfaces – websites or applications – that are the responsibility of the service where Alice wants to sign in, such as a bank.

Mimir's approach contrasts traditional username/password systems where credentials are verified by individual services (which serves as a single point of failure [1]) and conventional identity management systems where centralised identity providers validate users' identities. By addressing these limitations, Mimir provides a solution to the critical challenge of modern digital identity management, where users often resort to reusing passwords across services [4].

This work surrounded two fundamental challenges: The practical need for better authentication systems in an era where users manage dozens to hundreds of accounts online [2], [3], and the theoretical challenge of consensus in distributed systems as defined by Fischer, Lynch, and Paterson's Impossibility Result [5].

Most significantly, this work contributed to the field by providing practical insights into navigating the consensus problem in distributed systems, as defined by Fischer, Lynch, and Paterson's Impossibility Result [5]. Mimir's distributed architecture places it at the heart of the consensus problem because the components must reach an agreement on the state of authorised and authenticated.

In our implementation, we identified key aspects to address consensus challenges while working within the boundaries of the consensus problem: procedure limitation, partial synchrony, data synchronisation, and fault prevention mechanisms. For procedure limitation, we limited the ongoing authorisation and authentication procedure to one at a time, acknowledging that the simplest form of consensus – agreeing on a single bit value – is impossible to guarantee and that appending more procedures would only increase the load on the CPU. In partial synchrony, we enforced timers on all messages being passed back and forth in Mimir's architecture to prevent network delays and unacknowledged messages from spinning out of control and preventing consensus. The timers allowed the components to reach a conclusion independent of the other components' responses. For data synchronisation, we placed small checkpoints

throughout the procedures to prevent logical errors and potentially delayed messages from interfering with an ongoing procedure. The checkpoints tracked the procedure's state and progress locally at the component. Fault prevention mechanisms involved defining a strict protocol and types in the data exchange format, Protobufs, selecting a programming language that prevented common memory errors, and unique identifiers to distinguish ongoing procedures and services from each other.

Connecting Mimir's implementation details to the consensus problem gives a more deterministic environment. However, our work demonstrated that the consensus problem remains fundamental: it is an unsolvable problem to guarantee consensus in asynchronously distributed systems, which must be navigated in all systems rather than solved. Consensus problems in large-scale systems materialise in significant and practical ways, and it is possible to distinguish codebases that either recognise the problem (implementing timers and synchronisation points to account for Fischer, Lynch, and Paterson's asynchronous assumptions) or escalate it to users and system providers across network levels with an expectation of recovery at other layers. Working within the boundaries of the consensus problem requires accepting that no perfect solution maintains an asynchronous nature; trade-offs must be made to operate within its constraints. Despite these types of theoretical boundaries, large-scale solutions still need to be built, and navigating theoretical boundaries shapes systems and their development as fundamental problems.

The research goal was never to solve the consensus problem, which remains unsolvable, but rather to demonstrate practical steps for operating within its constraints. It is the worst-case scenario for any asynchronously distributed system and the default case under which all systems are developed unless steps are taken to manage it. Through Mimir, we showed how a distributed authentication system can navigate these theoretical boundaries while addressing real-world authentication challenges. This intersection of theoretical limitations and practical needs shapes how we build and understand distributed systems.

Building on this research, several promising areas for future exploration include enhancing the understanding of distributed systems, authorisation and authentication, and consensus. Best practices and guidelines in relation to FLP's Impossibility Result should be developed by the scientific community. Future studies could investigate effective fallback mechanisms for lost or stolen devices. For example, a device associated with a user should be replaceable, or a second associated device, such as a hardware key, should be used as a backup solution. Another aspect is how to implement or carefully handle the replacement of either of the devices, given the same context as a lost or stolen device. More deep, specific research can be performed on the relationship between phones and their ability to report faults and non-operational states. Other forms of authentication mechanisms could further be explored, such as behaviour biometrics like walking patterns and context for the associated device. Methods for establishing synchronous communication in emergencies, where the device is unavailable or a system has been compromised, and methods for preventing unauthorised access could be considered. An interesting aspect would be to research protocols such as PAKE and OPAQUE, where the secrets are never shared with the server and instead computed locally on a device, before integrating these as opportunities to authenticate in Single Sign-On service, combined with hardware devices and biometrics. Further investigations could be done into consensus and asynchronously distributed systems. Additionally, research into app attestation, maintaining exclusivity to authenticator devices through private key infrastructure, and constructing

untraceable hashes without human-readable names would be valuable for sharing sensitive information. For Single Sign-On services, investigating the cost of operation to drive digital identification online, along with complexity barriers and scaling for these services, which require consensus, is a field that requires investigation. The specifics of the CAP theorem and trade-offs to guarantees in distributed authentication systems when dealing with delays and device disconnections.

Long-established research, such as Turing's Halting Problem, FLP's Impossibility Result, Rice's Theorem, and similar, can be defined as "Laws of Computers" and can improve our understanding of computers and what is possible to perform with them.

Bibliography

[1] M. Bakhtina, R. Matulevičius, A. Awad, and P. Kivimäki, "On the shift to decentralised identity management in distributed data exchange systems", en, in *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*, Tallinn Estonia: ACM, Mar. 2023, pp. 864–873, ISBN: 978-1-4503-9517-5. DOI: 10.1145/3555776.3577678. [Online]. Available: https://dl.acm.org/doi/10.1145/3555776.3577678.

[2] C. Scott and et al., "Examining the privacy of login credentials using web-based single sign-on - are we giving up security and privacy for convenience?", *2015 Cybersecurity Symposium*, 2016. DOI: 10.1109/cybersec.2016.019.

[3] A. Blanco-Justicia and J. Domingo-Ferrer, "Efficient privacy-preserving implicit authentication", *Computer Communications*, vol. 125, pp. 13–23, Jul. 2018, ISSN: 0140-3664. DOI: 10.1016/j.comcom.2018.04.011.

[4] C. Wang, S. T. Jan, H. Hu, D. Bossart, and G. Wang, "The next domino to fall: Empirical analysis of user passwords across online services", Codaspy '18, pp. 196–203, 2018. DOI: 10.1145/3176258.3176332. [Online]. Available: https://doi.org/10.1145/3176258.3176332.

[5] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process", *Journal of the ACM (JACM)*, vol. 32, no. 2, pp. 374–382, 1985.

[6] *Text and Ritual in Early China*. University of Washington Press, 2005, ISBN: 9780295985626. [Online]. Available: http://www.jstor.org/stable/j.ctvcwn98n (visited on 12/16/2024).

[7] H. N. Mathisen, *Cryptochronicles - lesspm's quest to a passwordless utopian ecosystem*, https://www.researchgate.net/publication/372477823_CryptoChronicles_-_LessPM's_Quest_to_a_Passwordless_Utopian_Ecosystem, Preprint, 2023.

[8] Polybius, *The histories*, Accessed on: March 23, 2023, 2023. [Online]. Available: http://www.perseus.tufts%20.edu/hopper/text?doc=Perseus%5C%3Atext%5C%3A1999%20.01%20.0234%5C%3Abook%5C%3D6%5C%3Achapter%5C%3D34.

[9] S. Levy, *Hackers: Heroes of the Computer Revolution*. Sebastopol, CA: O'Reilly Media, 1984, pp. 85–102, ISBN: 978-1449388393.

[10] *CTSS Programmers Guide*, 2nd ed. MIT Press, 1965.

[11] R. Wang, S. Chen, and X. Wang, "Signing me onto your accounts through facebook and google: A traffic-guided security study of commercially deployed single-sign-on web services", en, pp. 365–379, May 2012. DOI: 10.1109/sp.2012.30. [Online]. Available: http://ieeexplore.ieee.org/document/6234424/.

[12] N. I. of Standards and Technology, *Authorization - glossary*, https://csrc.nist.gov/glossary/term/authorization, Accessed: 2024-10-16.

[13] Nist, *Nist glossary: Authentication*, https://csrc.nist.gov/glossary/term/authentication, Accessed: 2024-10-16.

[14] W. Li, P. Wang, and K. Liang, "Hpake: Honey password-authenticated key exchange for fast and safer online authentication", en, *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 1596–1609, 2023, ISSN: 1556-6013, 1556-6021. DOI: 10.1109/tifs.2022.3214729.

[15] L. Lamport, "Time, clocks, and the ordering of events in a distributed system", *Communications of the ACM*, vol. 21, no. 7, pp. 558–565, 1978. DOI: 10.1145/359545.359563.

[16] E. W. Dijkstra, "Self-stabilizing systems in spite of distributed control", *Communications of the ACM*, vol. 17, no. 11, pp. 643–644, Nov. 1974. DOI: 10.1145/361179.361202.

[17] M. Jo, D. Kim, and S. Park, "Analysis of byzantine fault tolerant consensus algorithms", in *2024 International Conference on Information Networking (ICOIN)*, Jan. 2024, pp. 205–207. DOI: 10.1109/icoin59985.2024.10572154. [Online]. Available: https://ieeexplore.ieee.org/document/10572154.

[18] F. Xiao and L. Wang, "Asynchronous consensus in continuous-time multi-agent systems with switching topology and time-varying delays", *IEEE Transactions on Automatic Control*, vol. 53, no. 8, pp. 1804–1816, Sep. 2008, ISSN: 1558-2523. DOI: 10.1109/tac.2008.929381.

[19] [Online]. Available: https://ieeexplore.ieee.org/document/5779706.

[20] Z. Xu, Y. Li, C. Feng, and L. Zhang, "Exact fault-tolerant consensus with voting validity", in *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2023, pp. 842–852. DOI: 10.1109/ipdps54959.2023.00089. [Online]. Available: https://ieeexplore.ieee.org/document/10177399.

[21] L. Lamport, "The part-time parliament", 1998, Accessed: January 26, 2025.

[22] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm", in *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, USENIX Association, 2014, pp. 305–319.

[23] S. Srinivasan and R. Kandukoori, "Solving consensus in true partial synchrony", *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 12, pp. 3478–3490, Dec. 2022, ISSN: 1558-2183. DOI: 10.1109/tpds.2022.3156925.

[24] P. Dutta, R. Guerraoui, and L. Lamport, "How fast can eventual synchrony lead to consensus?", in *2005 International Conference on Dependable Systems and Networks (DSN'05)*, Jun. 2005, pp. 22–27. DOI: 10.1109/dsn.2005.54. [Online]. Available: https://ieeexplore.ieee.org/document/1467776.

[25] B. Bisping, P.-D. Brodmann, T. Jungnickel, *et al.*, "Mechanical verification of a constructive proof for flp", in *Interactive Theorem Proving*, J. C. Blanchette and S. Merz, Eds., Cham: Springer International Publishing, 2016, pp. 107–122, ISBN: 978-3-319-43144-4.

[26] A.-Y. Lu and G.-H. Yang, "Distributed consensus control for multi-agent systems under denial-of-service", *Information Sciences*, vol. 439–440, pp. 95–107, May 2018, ISSN: 0020-0255. DOI: 10.1016/j.ins.2018.02.008.

[27] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony", *Journal of the ACM (JACM)*, vol. 35, no. 2, pp. 288–323, 1988.

[28] J. Mallett, "The role of groups in smart camera networks", en, 2006.

[29] P. Gupta and P. Kumar, "The capacity of wireless networks", *IEEE Transactions on Information Theory*, vol. 46, no. 2, pp. 388–404, Mar. 2000, ISSN: 1557-9654. DOI: 10.1109/18.825799.

[30] T. Li, M. Fu, L. Xie, and J.-F. Zhang, "Distributed consensus with limited communication data rate", *IEEE Transactions on Automatic Control*, vol. 56, no. 2, pp. 279–292, Feb. 2011, ISSN: 1558-2523. DOI: 10.1109/tac.2010.2052384.

[31] T. D. Chandra and S. Toueg, "Unreliable failure detectors for reliable distributed systems", en,

[32] I. Gordin and et al., "Moving forward passwordless authentication: Challenges and implementations for the private cloud", in *2021 20th RoEduNet Conference: Networking in Education and Research (RoEduNet)*, Last Accessed: 2024-01-29, Ieee, 2021. DOI: 10.1109/RoEduNet54112.2021.9638271.

[33] M. Morii, Y. Seki, H. Tanioka, *et al.*, "Research on integrated authentication using passwordless authentication method", in *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, Ieee, Tokushima University, 2017, pp. 895–900. DOI: 10.1109/compsac.2017.198.

[34] S. Ghrobany Lyastani, M. Chilling, M. Neumayer, M. Backes, and S. Bugiel, "Is fido2 the kingslayer of user authentication? a comparative usability study of fido2 passwordless authentication", in *2020 51st IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, 2020, pp. 102–108. DOI: 10.1109/sp40000.2020.00047.

[35] J. Bonneau, "The science of guessing: Analyzing an anonymized corpus of 70 million passwords", in *2012 IEEE Symposium on Security and Privacy*, Last Accessed: 2024-01-24, 2012. DOI: 10.1109/sp.2012.49.

[36] F. Alliance, *Fido uaf overview*, https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-uaf-overview-v1.1-id-20170202.html, Accessed: 2024-01-24, Feb. 2017.

[37] W. W. W. Consortium, *Web authentication: An api for accessing public key credentials level 2*, https://www.w3.org/TR/webauthn-2/, Accessed: 2024-01-24, Apr. 2021.

[38] J. L. F. Paul A. Grassi Michael E. Garcia, "Digital identity guidelines: Authentication and lifecycle management", National Institute of Standards and Technology, Tech. Rep. Sp 800-63-3, 2017. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP%20.800-63-3.pdf.

[39] N. I. of Standards and Technology. "Special publication 800-171 revision 2: Protecting controlled unclassified information in nonfederal systems and organizations". Page 24, Chapter 3. (2020), [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP%20.800-171r2.pdf.

[40] V. Parmar, H. A. Sanghvi, R. H. Patel, and A. S. Pandya, "A comprehensive study on passwordless authentication", in *2022 International Conference on Smart Computing and Data Science (ICSCDS)*, 2022. DOI: 10.1109/icscds53736.2022.9760934.

[41] E. Huseynov, "Passwordless vpn using fido2 security keys: Modern authentication security for legacy vpn systems", in *2022 4th International Conference on Data Intelligence and Security (ICDIS)*, 2022, pp. 453–455. DOI: 10.1109/icdis55630.2022.00075.

[42] J. A. Sava, *Biometrics-enabled consumer device adoption in 2020*, Statista, Last Accessed: 2023-03-28, Jun. 2022. [Online]. Available: https://www.statista%20.com/statistics/1226096/biometrics-enabled-devices-by-region/?locale=en.

[43] S. Consortium, *Shibboleth - federated identity solutions*, Accessed: 2023-03-28, 2022. [Online]. Available: https://www.shibboleth.net/.

[44] W3Techs, *Market share trends for ssl certificate authorities, february 2024*, Feb. 2024. [Online]. Available: https://w3techs.com/technologies/history%5C_overview/ssl%5C_certificate.

[45] N. Shaikh, K. Kasat, and S. Jadhav, "Secured authentication by single sign on (sso): A big picture", en, in *2022 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)*, Greater Noida, India: Ieee, Nov. 2022, pp. 951–955, ISBN: 978-1-66546-200-6. DOI: 10.1109/icccis56430.2022.10037708. [Online]. Available: https://ieeexplore.ieee.org/document/10037708/.

[46] Ietf, *Oauth 2.0 assertion profiles for jwts*, https://datatracker.ietf.org/doc/html/rfc7522, Accessed: January 26, 2024, 2015.

[47] Ietf, *The oauth 2.0 authorization framework: Bearer token usage*, https://datatracker.ietf.org/doc/html/rfc6750, Accessed: January 26, 2024, 2012.

[48] Ietf, *The oauth 2.0 authorization framework*, https://datatracker.ietf.org/doc/html/rfc6749, Accessed: January 26, 2024, 2012.

[49] D. Fett, R. Küsters, and G. Schmitz, "Spresso: A secure, privacy-respecting single sign-on system for the web", in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, ser. Ccs '15, New York, NY, USA: Association for Computing Machinery, Oct. 2015, pp. 1358–1369, ISBN: 978-1-4503-3832-5. DOI: 10.1145/2810103.2813726. [Online]. Available: https://dl.acm.org/doi/10.1145/2810103.2813726.

[50] A. D. Johnson, I. Alom, and Y. Xiao, "Rethinking single sign-on: A reliable and privacy-preserving alternative with verifiable credentials", Mtd '23, pp. 25–28, Nov. 2023. DOI: 10.1145/3605760.3623767. [Online]. Available: https://dl.acm.org/doi/10.1145/3605760.3623767.

[51] J. Anderson and K. Keahey, "Migrating towards single sign-on and federated identity", en, in *Practice and Experience in Advanced Research Computing*, Boston MA USA: Acm, Jul. 2022, pp. 1–8, ISBN: 978-1-4503-9161-0. DOI: 10.1145/3491418.3530770. [Online]. Available: https://dl.acm.org/doi/10.1145/3491418.3530770.

[52] P. I. S. Stian Thorgersen, *Keycloak - identity and access management for modern applications: Harness the power of keycloak, openid connect, and oauth 2.0 protocols to secure applications*, Packt Publishing, 2021. [Online]. Available: https://ieeexplore.ieee.org/document/10163450.

[53] L. S. Ramamoorthi and D. Sarkar, "Single sign-on: A solution approach to address inefficiencies during sign-out process", en, *IEEE Access*, vol. 8, pp. 195 675–195 691, 2020, ISSN: 2169-3536. DOI: 10.1109/access.2020.3033570.

[54] J. Wiele, "Pebkac revisited – psychological and mental obstacles in the way of effective awareness campaigns", en, in *ISSE 2011 Securing Electronic Business Processes: Highlights of the Information Security Solutions Europe 2011 Conference*, N. Pohlmann, H. Reimer, and W. Schneider, Eds. Wiesbaden: Vieweg+Teubner Verlag, 2011, pp. 88–97, ISBN: 978-3-8348-8652-1. DOI: 10.1007/978-3-8348-8652-1_9. [Online]. Available: https://doi.org/10.1007/978-3-8348-8652-1%5C_9.

[55] N. A. Lynch, *Distributed Algorithms*. San Francisco, CA: Morgan Kaufmann, 1996, ISBN: 978-1-55860-348-6. [Online]. Available: https://www.elsevier.com/books/distributed-algorithms/lynch/978-1-55860-348-6.

[56] Y. Zhang, C. Xu, H. Li, K. Yang, N. Cheng, and X. Shen, "Protect: Efficient password-based threshold single-sign-on authentication for mobile users against perpetual leakage", *IEEE Transactions on Mobile Computing*, vol. 20, no. 6, pp. 2297–2312, Jun. 2021, ISSN: 1558-0660. DOI: 10.1109/tmc.2020.2975792.

[57] N. Hossain, M. A. Hossain, M. Z. Hossain, M. H. I. Sohag, and S. Rahman, "Oauth-sso: A framework to secure the oauth-based sso service for packaged web applications", en, in *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, New York, NY, USA: Ieee, Aug. 2018, pp. 1575–1578, ISBN: 978-1-5386-4388-4. DOI: 10.1109/TrustCom/BigDataSE.2018.00227. [Online]. Available: https://ieeexplore.ieee.org/document/8456096/.

[58] Z. Triartono, R. M. Negara, and Sussi, "Implementation of role-based access control on oauth 2.0 as authentication and authorization system", en, in *2019 6th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, Bandung, Indonesia: Ieee, Sep. 2019, pp. 259–263, ISBN: 978-602-07-3728-7. DOI: 10.23919/eecsi48112.2019.8977061. [Online]. Available: https://ieeexplore.ieee%20.org/document/8977061/.

[59] E. Hammer, *OAuth 2.0 and the road to hell*, https://web.archive .org/web/20130325140509/http://hueniverse .com/2012/07/oauth-2-0-and-the-road-to-hell/, [Accessed 20-03-2024].

[60] L. Team, *Notice of recent security incident*, Accessed: 2024-01-30, Dec. 2022. [Online]. Available: https://blog.lastpass%20.com/2022/12/notice-of-recent-security-incident/.

[61] D. Winder, *Why you should stop using lastpass after new hack method update*, Accessed: 2024-01-30, Mar. 2023. [Online]. Available: https://www.forbes.com/sites/daveywinder/2023/03/03/why-you-should%20-stop-using-lastpass-after-new-hack-method-update/.

[62] R. Dhanalakshmi, N. Vijayaraghavan, S. Narasimhan, and S. Basha, "Password manager with multi-factor authentication", in *2023 International Conference on Networking and Communications (ICNWC)*, Apr. 2023, pp. 1–5. DOI: 10.1109/icnwc57852.2023.10127424. [Online]. Available: https://ieeexplore.ieee.org/document/10127424.

[63] H. Padalia, H. Patel, A. Deshmukh, M. Patil, A. Kumar, and N. Kumar Nrip, "A study on password manager: Users' perspective", in *2023 International Conference on Computational Intelligence for Information, Security and Communication Applications (CIISCA)*, Jun. 2023, pp. 72–75. DOI: 10.1109/ciisca59740.2023.00024. [Online]. Available: https://ieeexplore.ieee.org/document/10403587.

[64] P. Pandare, S. Uniyal, R. Vani, S. Mali, and P. Rumao, "Enhanced password manager using hybrid approach", in *2023 International Conference on Inventive Computation Technologies (ICICT)*, Apr. 2023, pp. 1793–1798. DOI: 10.1109/icict57646.2023.10134398. [Online]. Available: https://ieeexplore.ieee.org/document/10134398.

[65] M. Shirvanian, N. Saxena, S. Jarecki, and H. Krawczyk, "Building and studying a password store that perfectly hides passwords from itself', en, *Ieee Transactions On Dependable And Secure Computing*, vol. 16, no. 5, 2019.

[66] S. Bellovin and M. Merritt, "Encrypted key exchange: Password-based protocols secure against dictionary attacks", in *Proceedings 1992 IEEE Computer Society Symposium on Research in Security and Privacy*, May 1992, pp. 72–84. DOI: 10.1109/risp.1992.213269. [Online]. Available: https://ieeexplore.ieee.org/document/213269.

[67] R. Editor, *Guidelines for writing an iana considerations section in rfcs*, Accessed: 30-Jan-2024, 2017. [Online]. Available: https://www.rfc-editor.org/rfc/rfc8125.txt.

[68] I. E. T. F. (IETF), *The opaque asymmetric password-authenticated key exchange protocol*, Accessed: 30-Jan-2024, 2024. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-opaque.

[69] A. Scaglione and S. D. Servetto, "On the interdependence of routing and data compression in multi-hop sensor networks", en, in *Proceedings of the 8th annual international conference on Mobile computing and networking*, Atlanta Georgia USA: ACM, Sep. 2002, pp. 140–147, ISBN: 978-1-58113-486-5. DOI: 10.1145/570645.570663. [Online]. Available: https://dl.acm.org/doi/10.1145/570645.570663.

[70] T. Bray, *The javascript object notation (json) data interchange format*, Internet Requests for Comments, RFC, Dec. 2017. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc8259.

[71] G. Developers, *Protocol buffers: Proto3 language guide*, https://protobuf.dev/programming-guides/proto3/, Accessed: 27 December 2024, 2024.

[72] NIST, *Safer languages*, https://www.nist.gov/itl/ssd/software-quality-group/safer-languages, Accessed: 16-12-2024.

[73] Speykious, *Cve-rs*, https://github.com/Speykious/cve-rs, Accessed: 19-12-2024.

[74] Rust Language Team, *Issue #25860*, https://github.com/rust-lang/rust/issues/25860, Accessed: 19-12-2024.

[75] M. Ben-Or, "Another advantage of free choice: Completely asynchronous agreement protocols", pp. 27–30, 1983. DOI: 10.1145/800221.806707. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/800221.806707.

[76] A. Name, "How your stolen iphone ends up in one chinese city market", *The Times*, 2024, Accessed: 27-12-2024. [Online]. Available: https://www.thetimes.com/uk/society/article/how-your-stolen-iphone-ends-up-in-one-chinese-city-market-v6rf6w6pv.

[77] A. Sumaray and S. K. Makki, "A comparison of data serialization formats for optimal efficiency on a mobile platform", en, in *Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication*, Kuala Lumpur Malaysia: Acm, Feb. 2012, pp. 1–6, ISBN: 978-1-4503-1172-4. DOI: 10.1145/2184751.2184810. [Online]. Available: https://dl.acm.org/doi/10.1145/2184751.2184810.