1-2023

Token Based Authentication and Authorization with Zero-Knowledge Proofs for Enhancing Web API Security and Privacy

Michael Lodder

Token Based Authentication and Authorization with Zero-Knowledge Proofs for Enhancing Web API Security and Privacy

by

Michael Lodder

A Dissertation Presented in Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy in Cyber Operations

DAKOTA STATE UNIVERSITY

January 2023

DAKOTA STATE UNIVERSITY

DISSERTATION APPROVAL FORM

This dissertation is approved as a credible and independent investigation by a candidate for the Doctor of Philosophy degree and is acceptable for meeting the dissertation requirements for this degree. Acceptance of this dissertation does not imply that the conclusions reached by the candidate are necessarily the conclusions of the major department or university.

Student Name: Michael Lodder

Dissertation Title: Token Based Authentication and Authorization with Zero-Knowledge Proofs for Enhancing Web API Security and Privacy

Dissertation Chair/Co-Chair: DocuSigned by: Michael Ham 499B29D70C7542A... Date: April 25, 2023

Name: Michael Ham

Dissertation Chair/Co-Chair: DocuSigned by: E94723CA282F45C... Date: April 25, 2023

Name: Austin O'Brien

Committee member: DocuSigned by: Stephen Krebsbach 3D7C49A27237465... Date: April 25, 2023

Name: Stephen Krebsbach

Committee member: _____ Date: _____

Name:

Committee member: _____ Date: _____

Name:

Committee member: _____ Date: _____

Name:

Original to Office of Graduate Studies and Research
Acid-free copies with written reports to library

Token Based Authentication and Authorization with Zero-Knowledge Proofs for Enhancing Web API Security and Privacy

by

Michael Lodder

March 2023

Approved:

Dr. Michael J. Ham, Chair

Dr. Austin F. O'Brien, Co-Chair

Dr. Stephen Krebsbach, Committee

Accepted and Signed: _____ _____
Michael J. Ham Date

Accepted and Signed: _____ _____
Austin O'Brien Date

Accepted and Signed: _____ _____
Stephen Krebsbach Date

_____ _____
Mark L. Hawkes Date

Dean of Graduate Studies and Research

Dakota State University

Abstract

This design science study showcases an innovative artifact that utilizes Zero-Knowledge Proofs for API Authentication and Authorization. A comprehensive examination of existing literature and technology is conducted to evaluate the effectiveness of this alternative approach. The study reveals that existing APIs are using slower techniques that don't scale, can't take advantage of newer hardware, and have been unable to adequately address current security issues. In contrast, the novel technique presented in this study performs better, is more resilient in privacy sensitive and security settings, and is easy to implement and deploy. Additionally, this study identifies potential avenues for further research that could help advance the field of Web API development in terms of security, privacy, and simplicity.

TABLE OF CONTENTS

1 Chapter 1: Introduction

This study demonstrates the benefits for an artifact that utilizes Zero-Knowledge Proof (ZKP) based Tokens for protecting Web Application Programming Interface (API) security and privacy authentication, authorization and data exposure and compares its effectiveness and impact against existing protocols.

An API is a set of rules and protocols that specifies how programs interact with each other. It serves the contract between different systems, allowing them to communicate and exchange information.

For example, when building applications needing access to a database of information, an API can be used to retrieve that information from the database. API's handle the communication between the application and the database, allowing the application to request specific data and the database to return it in a standardized format. APIs are a central component of modern software development, as they allow different systems and applications to integrate and work together effectively.

According to O'Neill, Zumerle, and D'Hoinn (2017), they predicted in 2022, API abuses would become the most-frequent attack vector, with more data breaches for enterprise web applications which Google (2022) and Akamai (2022) have confirmed. The Open Web Application Security Project (OWASP) maintains a list of the ten most important considerations for API design (2021) with the top three listed as "Broken Object Level Authorization", "Broken User Authentication", and "Excessive Data Exposure". The remainder of the top 10 ranking reflects the ongoing challenge and significance of ensuring API endpoint security through proper authentication, authorization, and protection against data breaches.

Despite guidelines and best practices, numerous APIs are still inadequately set up, resulting in major security lapses for companies such as Checkr, Twilio, Scale, Segment, Facebook, Cisco, and GitLab (Akamai, 2021). These lapses have resulted in unintended client data exposure, unauthorized data manipulation, disrupting the availability of end

clients and consumers and hindering their proper functioning (Wallarm Inc., 2022). Despite these challenges, the use of APIs continues to grow in importance. The variety of applications for APIs is increasing where the intended purpose is currently web (57%), mobile (56%), automation (49%) and Internet of Things (IoT) (46%) (Google, Inc., 2021). API traffic also increased 46% year-over-year to 2.21 trillion calls between 2019 and 2020 (Google, Inc., 2021). The variety of data exposed via APIs ranges from HealthCare, Financial, Telecommunications, Academic, Media & Entertainment, Retail, Travel, and many more. The risk for API misconfiguration is significant, and incidents of abuse have seen a three- to four-fold increase from 2020 to 2021 (2021, Salt Security), (2021, Akamai).

A ZKP is a procedure where one party can prove to another they know a piece of data, without revealing any information about the actual content of that data. Alice, for example, can prove to Bob that she knows a secret key to a certain account, without actually revealing the key to Bob. Alice uses a ZKP to demonstrate to Bob that she knows the key, without revealing the key itself (Blum et al., 1988).

Zero-Knowledge Proof (ZKP)s are used in a variety of contexts, to allow one party to prove their knowledge or identity without revealing any sensitive information. They are considered a useful tool for maintaining the privacy and security of sensitive information (Wu & Wang, 2014). Such contexts include but are not limited to: cryptography to prove the authenticity of a message or the possession of a secret key, without revealing the actual contents of the message or key, computer security to authenticate users and grant access to secure systems without revealing passwords or other sensitive information, identity verification to validate an identity without revealing their personal details, such as a name, address, or date of birth, to prove the authenticity of financial transactions without revealing the details of the transactions, and privacy-preserving data analysis to perform statistical analysis on sensitive data without actually accessing the underlying data.

Chapter 1 focuses on measuring how the proposed approach to use ZKP tokens can enhance or diminish the security and privacy of API design and implementation. This

chapter also compares against known existing API solutions. This chapter covers the impact of the proposed experiment, the design, and possible results. Additional topics include further research questions that should be explored. The research goals were to determine whether ZKP based tokens were more secure and measured their effectiveness in enhancing privacy.

1.1 Background of the Study

As the internet gained popularity in the mid-2000s, companies began making their APIs available for others to use. One widely adopted protocol for API communication is Representational state transfer (REST), introduced by Roy Fielding in his doctoral thesis (2000). REST leverages the same communication protocol that underlies web pages to enable communication between applications through APIs. REST relies on HyperText Transfer Protocol (HTTP) verbs to convey the intended action. These verbs are part of the HTTP protocol and signify the client or browser's intent. The four verbs used by REST are GET, POST, PUT, and DELETE. GET is used for querying or retrieving data from the API, while POST is for creating or adding data to the API's resource pool based on the enclosed request. PUT is utilized for updating or replacing existing data with information from the enclosed request, while DELETE is for removing specified data. While different APIs may not use REST, they generally follow similar principles for data processing and access. Other API protocols include Simple Object Access Protocol (SOAP), Remote procedure call (RPC), and Windows Communication Foundation (WCF), with RPC being the foundation for many popular derivatives such as Google's gRPC, Apache Thrift, and Facebook's GraphQL (Reselman, 2020) (Source, 2022).

These approaches often lack clear guidance on correct handling of authentication and authorization, usually left for API providers to figure out on their own. Custom methods, which may or may not involve cryptography such as passwords, or pre-built solutions like OpenID Connect (OIDC), Open Authorization Framework (OAuth 2), and Security Assertion Markup Language (SAML), are among the possible solutions. Custom

solutions can be implemented quickly, but tend to be less secure, whereas pre-built solutions have some security issues addressed, but can be challenging to integrate or deploy. Enterprise-level issues like these are typically handled by specialized companies, such as Okta, Auth0, Microsoft's Active Directory, and Amazon's Web Services. This experiment aims to explore the various authentication and authorization deployment methods for web APIs, which have seen a 46% year-over-year increase in external API use from 2019 to 2020, with further growth expected in 2021 (Google, Inc., 2021). However, the proliferation of API data has resulted in a constantly evolving landscape where security is often an afterthought in API operation (Akamai, 2021). This explains the wide range of differing options available.

The varied contexts in which APIs are used may contribute to the range of security challenges encountered. Industries such as Healthcare, Finance, and Telecommunications face stringent regulations that require robust security measures to ensure privacy and data protection. Large companies in these industries tend to have dedicated Information Security (InfoSec) teams and the resources to invest in strong security measures. In contrast, industries such as Media, Retail, Travel, and Marketing typically face fewer regulatory requirements, and smaller companies in these sectors may not prioritize security to the same extent. However, both regulated and non-regulated industries have experienced security breaches. According to a report by Akamai (2021), attackers are outpacing the security industry in recruiting skilled personnel and developing more sophisticated techniques to exploit security vulnerabilities in APIs. One issue that can contribute to security problems with APIs is that web developers often work at several levels of abstraction away from the underlying technologies and may not have a strong understanding of the HTTP protocol (Akamai, 2021). According to Chris Eng from Veracode, we are repeating the same security mistakes with APIs that we made with web security 20 years ago (Akamai, 2021). This lack of progress has resulted in a decline in modern security measures for APIs, leading to an increased risk of security vulnerabilities

that can be exploited by malicious actors to gain unauthorized access to sensitive data or systems.

Developer tools have advanced to a point where they enable developers to design APIs quickly and easily, often without considering security implications. While this can be beneficial for productivity, it also presents a challenge for ensuring that APIs are adequately guarded. In fact, according to a 2021 report, only 6% of companies reported no security issues related to their APIs (Korolov, 2021). Cheng lists more than 30 major data breaches in which vulnerabilities in API data were the root cause (2020). In one instance, a lateral attack using VMWare Workspace One was carried out to breach Solarwinds by exploiting the API (Cheng, 2020). This led to a supply chain attack that affected a wide range of organizations that rely on or utilize Solarwinds. Consequently, there has been a decrease in trust in both the software product and the company behind it. In another case, a significant security flaw in APIs was uncovered, affecting numerous automobile manufacturers such as Nissan, Toyota, and Ford. This vulnerability provided access to sensitive internal systems and user data, and weaknesses that enabled attackers to remotely execute code. This flaw is particularly dangerous as it allows adversaries to issue arbitrary commands to approximately 15.5 million vehicles and even update vehicle firmware (Lakshmanan, 2023). Another significant event happened where the improper configuration of T-Mobile's API led to a data leak of 37 million customer accounts, which marks the eighth such incident in the last five years (Gatlan, 2023).

Similar findings were reported in cases like Twitter's Fleet API, which allowed access to tweets that were supposed to disappear after 24 hours, and Tesla's Backup Gateway API, used for arbitrating charging batteries and sending power back to the grid, which exposed Personal Identifiable Information (PII) and sometimes required no authentication. These incidents highlight the importance of applying best practices, which led to the creation of the OWASP Top 10 list. However, it's worth noting that the OWASP Top 10 list doesn't recommend any particular protocol or standard to use for

authentication or authorization.

Another challenge comes from the lack of prioritization for development security, as businesses prioritize releasing products to generate revenue over investing time and money in security. Security is often viewed as a low-risk tradeoff, and patching is seen as a task that can be deferred until later or not done at all. As a result, fixes are often only applied when they are considered critical (Akamai, 2021, pg. 12). According to Krishnaswamy, API developers did not consider security a priority until recently (Krishnaswamy, 2022). This neglect of development security has contributed to the increase in security vulnerabilities in APIs.

In order to ensure API security, it is essential to address the three fundamental tests that determine the validity of a request: authentication, authorization, and access control. These three aspects are commonly referred to as the 'three A's of security'.

Authentication is the process of verifying the identity of an entity. This process is usually done using traditional methods such as usernames and passwords or more advanced methods that use a combination of something the entity knows (such as a password or PIN), something they have (such as a cryptographic key, smart card, phone, or fob), and something they are (such as biometric data like retina scans, voice recognition, facial recognition, or fingerprints). However, automated systems like software processes do not have biometric data, so other items are used like environment values or hardware specifications such as Media Access Control (MAC) addresses, Trusted Execution Environments (TEE), Hardware Security Module (HSM), Trusted Platform Module (TPM), or Physical Unclonable Function (PUF)s are used instead.

Authorization is the process of determining whether an entity has the correct privilege to access a particular resource or perform a specific operation. Access control involves allowing or denying access to resources and operations based on predefined criteria. If the entity satisfies the criteria, access is granted; otherwise, it is denied. Access control policies are typically configured according to predefined rules. Various methods of

API authentication, authorization, and access control exist, all of which require a secure channel to protect the credentials sent between communicating parties. Transport Layer Security (TLS) is commonly used to provide this secure channel. However, it is important to note that these protocols are vulnerable in the absence of a secure channel.

TLS is a protocol that provides secure communications over a network between two parties typically a client and a server. It is widely used to secure the transmission of data such as email and web traffic that might include sensitive information like credit card numbers and personal data via cryptographic encryption. To ensure that clients can verify server identity and not connecting to a malicious party, TLS requires servers to present signed certificates. Client certificates, while optional, are not commonly used, so clients are typically authenticated through other means. If a client does present a certificate, their authentication is similar to that of the server. Certificates are signed by trusted certificate authorities that validate the server's identity information and a public encryption key.

Basic authentication is the simplest method of authentication for Web APIs. The caller supplies a username and password in an HTTP header with each call. This means the credentials are sent over and over again. Browsers often cache these credentials so the user isn't bothered to reenter them each time but the credentials are still communicated. The system creates a password file that the web server uses to validate the credentials for the incoming request. If the credentials are not found or incorrect, the request is denied. Rules for group membership may be included but cannot account for more rigorous security definitions to create access control policies. Password files can be configured to use databases instead but the security of the system more or less stays the same (F5, 2022).

SAML replaces basic authentication by having the user's credentials stored with an identity provider (Organization for the Advancement of Structured Information Standards, 2005). The identity provider gives the user an attestation token that is used instead of sending the credentials. Tokens and other system data are exchanged using Extensible Markup Language (XML) documents between the identity provider and a service provider.

Service providers receive tokens and they query the identity provider to check if the token is valid. If attackers compromise tokens they can impersonate a user as if they had the user's credentials until the token is revoked or expires. The downside to SAML is the specification is complex and difficult to understand. The upside is any service that uses SAML can allow access to their API without the worry of authenticating users with credentials. The tokens are sent as HTTP Authentication header with a Bearer tag (Wierenga & Lear, 2005). OAuth 2 is a standard for determining whether a request is authorized. OAuth 2 does not handle authentication. OAuth 2 is centered around sharing resources based on user's information and permissions. OAuth 2 accomplishes its mission using two tokens: identification tokens and access tokens. Identity tokens are used by the identity provider to know who's resources are request. Access tokens are used by the service provider to determine if the request is authorized. OAuth 2 is widely deployed and used for sharing data between providers (Parecki, 2022). Like SAML, the tokens are sent in an HTTP Authentication header with a Bearer tag.

OIDC builds on top of OAuth 2 by standardizing in areas where OAuth 2 leaves the choice up to the implementers. Like SAML, users sign in with an identity provider allowing them to access other websites or APIs without entering their credentials anywhere else. OIDC is an open standard that can be used by anyone. Okta, Facebook, Google, and others offer their solutions as identity providers so users can log in with Google, Facebook, Okta, Microsoft, etc. The flow for OIDC is similar to OAuth 2.

The downside to using identity providers is two fold: privacy and availability. Service providers accessed by users are disclosed to identity providers allowing some of their activities to be tracked. A dishonest identity provider can build a profile for a user with such information or worse disclose it to another party. Identity providers control user's sensitive data and should they be compromised, the attacker can apply techniques like offline dictionary attacks to discover passwords or biometrics and other authentication credentials. These protocols also require the identity provider always be available or online.

If the provider is offline, the user or system can no longer sign in or access any services. OIDC, OAuth 2, and SAML rely on trusted third parties to function properly. This problem will increase as more APIs move to identity provider solutions (Gravitee.io, 2022) first over token based approaches. Figure C2 shows these approaches in a coherent diagram.

1.2 Statement of the Problem

There are several reasons why this experiment is being conducted. Firstly, this study shows API attacks are on the rise, despite existing security recommendations, indicating that traditional solutions are insufficient for providing adequate protection. Secondly, similar mistakes made in the past are still being repeated today. Thirdly, the damage caused by API breaches is much more significant than before. Fourthly, traditional methods do not scale effectively with the number of users. Fifth, Multi-Factor Authentication (MFA) is not available to API consumers since their programs run autonomously or non-interactive. Finally, newer hardware is problematic as protection parameters must be adjusted for each environment and system. The artifact presented in this study provides a practical solution to all of these issues.

Additionally, it is important to note that despite advancements in advanced cryptography that could potentially address many of these issues, there has been little to no research on applying them to replace traditional methods or efforts to make them accessible to everyday API implementers. This highlights the need for further exploration and dissemination of these newer techniques in the field of API security.

Current authentication and authorization schemes for securing Web APIs require storing sensitive user credentials either on the local system or with a trusted third party that is always accessible. All too often, these credentials are stored in a way that makes them susceptible to compromise (Taujenis, 2022) or exposure (Buchanan, 2022).

A significant challenge with APIs is the absence of simple and secure setup options, and the difficulty of maintaining the security of those setups. Local setups are often

misconfigured using insecure methods and practices (Gatlan, 2022), (Lakshmanan, 2022). Integrating with third parties also tends to involve complex setups and processes. Attempts have been made to use other techniques with Camenisch, 2014, Haböck and Krenn, 2019, and Mir et al., 2020 to mitigate and simplify this process but have not seen wide adoption. Furthermore, there has been little research on the current state of various techniques and their effectiveness. (Akamai, 2021). Thus main problems for API security can be defined as:

1. Traditional solutions don't deliver adequate API protection.
2. Storing sensitive credentials and policies and transmission of sensitive credentials (Team Traceable, 2023).
3. Lack of simple and secure authentication setups.
4. Inability to implement and deploy advanced cryptographic techniques.
5. No research in applying advanced cryptographic techniques to APIs.

In spite of current security guards API attack traffic has risen 681% in the finance sector (Balmas, 2022). Companies typically deploy runtime security stacks that consist of several layers of security tools, including bot mitigation, Web Application Firewalls (WAF), and API gateways. While these conventional tools offer essential security capabilities and protection for standard applications, they fall short for identifying and preventing attacks that target the API specific logic (Mishra, 2023) especially authentication.

To achieve a more secure setup for APIs and determine the possibility for reducing the need to store or transmit sensitive credentials, it is crucial to conduct research in this area that includes applying advanced cryptographic techniques. This study aimed to investigate potential security issues to avoid and propose a simple yet secure setup for APIs through the utilization of advanced cryptographic techniques.

In order to enhance the security of APIs and reduce or eliminate the necessity of storing or transmitting sensitive credentials, it is essential to conduct extensive research in this area, particularly focusing on the application of advanced cryptographic techniques. Such techniques can significantly improve security by ensuring that only authorized users can access sensitive data or functions. The main objective of this study is to explore potential security vulnerabilities that should be avoided when setting up APIs and suggest a secure setup that utilizes advanced cryptographic techniques. This research aimed to provide insights into the current state of API security and identify potential risks and threats, as well as offer solutions to mitigate these risks through the use of advanced cryptographic techniques. By conducting this research, the goal is to contribute to the development of more secure and reliable APIs that can be used in a variety of contexts, from financial services to healthcare and beyond.

1.3 Purpose of the Study

The study's objective was to gather information about existing API architectures and compare them against the newer techniques proposed in this study's artifact which aimed to overcome the limitations of trusted third parties, constant online connectivity, and sensitive credential storage for authentication, authorization, and access control in Web APIs. The artifact's techniques are not restricted to Web-based APIs and could be implemented anywhere, while also enhancing privacy by reducing user tracking. The efficacy of this approach in achieving these objectives was assessed in this study, with inspiration drawn from Bonneau et al., 2012.

- How was security handled with existing current API solutions? Various API's authentication techniques were examined and contrasted against this study's proposal. The findings show equal or better results.
- What tradeoffs did these solutions offer?
  - *Complexity* refers to how straightforward it is to implement the authentication

methods used by an API. It was discovered in this study that it is common for APIs to use multiple authentication methods, including both simple and complex techniques. This suggests that the API's design may have begun with a simpler approach and then integrated more complex methods later on meaning the techniques in this study can be added without replacing existing solutions.

- *Usability* refers to how easy it is for end users to consume the APIs examined in this study, as they have a large number of consumers.
- *Efficient* is evaluated by measuring the time required to complete authentication. Existing techniques require multiple network trips or intensive CPU cycles against sensitive credentials to confirm valid authentication, whereas the proposed techniques in this study do not, making them a more efficient alternative.
- *Succinct* how much data is exchanged? Current authentication methods exchange either minimal data that is not secure enough or exchange a large amount of data. In contrast, the proposed techniques in this study is as concise as the simple methods but provides the required level of security similar to the more complex methods.
- *Recovery-from-loss* if consumers lose their credentials or their credentials are stolen, can they conveniently regain the ability to authenticate? Also how complicated is it for the API service to revoke lost or stolen credentials? This study demonstrates that the advanced cryptographic techniques proposed are not only more secure against credential theft but also simplifies the process of credential revocation.
- *Resilient-to-observation* An attacker (external or internal like a curious administrator) cannot impersonate a consumer. The proposed techniques in this study provides complete protection against impersonation by external or

internal attackers since credentials are not stored on the system.

- *Resilient-to-theft* how difficult it is for attackers to steal consumer credentials. The proposed techniques in this study includes multiple factors that make it significantly harder for attackers to steal credentials. Furthermore, if attackers do manage to steal credentials and move them to a different environment, the functionality of the stolen credentials is broken.
- *Resilient-to-leaks* If API credentials are compromised, will consumers be vulnerable to impersonation attacks, or is the system resilient to leaks? The proposed techniques ensure that the credentials are never exposed during transmission or storage in the authentication system by using ZKPs, thereby limiting the possibility of leakage.
- *Resilient-to-correlation* Can API endpoints determine if the entity is the same or not. Worse can multiple endpoints collude to complete this determination? The proposed techniques in this study utilize ZKPs that make each presentations unique and It has been demonstrated that some of the most advanced correlation techniques currently available are ineffective.

1.4 Significance of the Study

Google, Inc., 2021, Security, 2021, Akamai, 2021 show the growth of Web APIs over the year 2021 to be 3-4 times greater than 2020 with no sign of slowing down. Moreover the security of Web APIs continues to be a major source of concern and exploitation (O'Neill et al., 2017), (Security, 2021) in spite of existing recommendations and protocols. This highlights the necessity for further research on analyzing the various and extensive implementations for typical security problems and providing recommendations for resolving these issues. Moreover, there has been limited exploration of utilizing advanced cryptographic techniques to address these issues. This study represents one of the pioneering attempts to implement these newer techniques and mitigate the security

challenges currently afflicting APIs. The primary objective of this study was to address several challenges related to complexity, usability, efficiency, succinctness, loss recovery, and resilience against observation, theft, leaks, and correlation as described. As a result of this research, API consumers and implementers can leverage the findings and compare to their own implementations. Additionally, they will have gained knowledge on how intricate authentication and authorization workflows can be simplified through ZKP-based alternatives and the tradeoffs associated with such an approach, including reduced transmission and storage of sensitive information. The hypothesis has been verified in that ZKP-based methods are less complex, equally secure, and diminish or eliminate access to sensitive information in comparison to existing deployments. Furthermore, the study demonstrated that the utilization of advanced cryptographic techniques can satisfy the following requirements: 1) simplicity of deployment in current solutions, 2) no degradation of system performance, and 3) enhancement of the overall security of the system.

1.5 Nature of the Study

Due to the scarcity of previous research in this domain and the absence of few quantitative measurements, two possible methodologies emerge as potential options: qualitative or design science. While a qualitative study can be useful in understanding the reasons behind API designs by interviewing their developers, it may not be the most feasible option for also testing advanced cryptographic techniques. This is because many existing APIs are managed by large companies and/or groups, and locating the appropriate personnel to answer such questions can be time-consuming and challenging. Moreover, these personnel may not have the necessary knowledge to explain the design decisions adequately. Additionally, these individuals may not have sufficient knowledge about advanced cryptography to secure their endpoints, rendering a qualitative research project less suitable. The design science methodology allows the researcher to focus on the creation of innovative and effective solutions to complex problems through the development and evaluation of artifacts, such as systems, models, methods, and processes (Larsen et al.,

2020). It can be applied to tackle practical problems related to the design, implementation, and use of information technologies and assess the validity of the results. The typical design science research process involves a series of iterative cycles, where the researcher first identifies a problem and explores its root causes and requirements, in this case simplifying deployments, limiting performance degradation, and enhancing system security. Then, the researcher creates a solution in the form of an artifact and evaluates it based on its fitness for use and usefulness to solve the problem. The researcher may also derive theoretical contributions from the artifact, which can help to advance the knowledge in the field. The ultimate goal of design science is to create practical solutions that can be used to address real-world problems while also contributing to the development of theoretical knowledge in the field (Creswell, 2018). As the reasons for API design choices are largely unknown and trying something new, Creswell, 2018 suggests that a design science experiment would be appropriate. Therefore, given the research objectives and the challenges associated with conducting a qualitative study, design science research is the most appropriate methodology for this study. The design science experimental component of this study involved contrasting the findings with the deployment of advanced cryptographic techniques. The goal of the experiment was to evaluate the impact of an alternative solution on current methodologies and compare its effectiveness with existing approaches. To guide the experiment, some quantitative data was collected based on factors such as the use of similar methods, payload sizes, network traffic, computational requirements, and security tradeoffs among different APIs. It's important to note that the experiment did not involve human subjects; rather, the design was to control variables for comparison and contrast for validation. As part of the validation process, data was collected from publicly available Web APIs of various internet companies to investigate the authentication techniques employed, without delving into the reasons behind their selection. The selected companies were among the most heavily trafficked in the world, as reported in (DuVander, 2023).

1.6 Research Questions

The hypothesis of this study was to determine whether utilizing advanced cryptographic techniques to implement an improved authentication and authorization approach with ZKPs provided any advantages over conventional web API designs. The research aimed to investigate if the approach simplified any of the three A's of design choices and whether there were any improvements or drawbacks regarding payload size, computational costs, and deployment simplicity. To guide the research, the following questions were employed: What were the required setup costs in a ZKP-based model, and did they impact performance, payload size, and privacy?

The study evaluated the impact on various parameters, such as CPU usage, RAM consumption, payload sizes, number of network trips, offline availability, design complexity, and security and privacy benefits. The experiment assumed that API designers had a basic understanding of security but lacked expertise in cryptography. The purpose of the research was to examine how the set of computations impacted the aforementioned parameters.

To conduct a more comprehensive evaluation of the hypothesis, the study implemented all the required cryptography in a user-friendly artifact. The artifact offers comparable methods to conventional procedures, minimizing the learning curve and demonstrating that complex techniques can be made simplified. By using this artifact, the researcher could determine its validity. By providing a similar interface to existing methods, the researcher was able to make more accurate comparisons, ensuring that the methods and processes used were comparable. This allowed for a more precise evaluation of the impact of the implemented methods on the various parameters. The study measured the privacy impact of ZKPs in terms of the information that can be gleaned through observation, influence, and correlatability. The observation aspect sought to determine what an observer could learn from observing the Zero-Knowledge Proof. The attacker influence aspect investigated whether an attacker could weaken the proof generation process enough to gain any secret information. The correlatability aspect aimed to

determine whether an attacker could distinguish between a repeated presentation from the same entity and that of a different entity.

The design science methodology is appropriate in this case for measuring the intended outcomes because it involves developing a solution (the advanced cryptography ZKP-based approach), implementing it, and evaluating it based on objective and subjective criteria. The methodology allows for an iterative process where the solution can be refined or modified to improve its performance and meet the intended outcomes. This approach ensures that the research study is not just a theoretical exercise but produces a practical solution that can be implemented and tested in real-world scenarios.

The design science methodology proved to be effective in the study of the ZKP-based approach for web API designs. By developing, implementing, and evaluating a solution to the research problem, the study revealed advanced cryptographic techniques that confirmed the hypothesis questions demonstrating that the use of ZKPs in web API designs can offer several benefits. The study also highlighted some potential limitations and trade-offs, such as the setup cost and computational complexity of the ZKP-based approach. The findings of this study can be used to inform future research and development of web API designs and cryptographic techniques.

1.7 Assumptions

No research study is flawless in its design, approach, or certainty, so a researcher must make assumptions about the study. It was considered that the measurements taken from the computers and environment, which reflected the researcher's API usage, were accurate. For example, ZKPs applied directly to the API with minimal effect on work flows and used a dedicated environment to control variables. The study implemented an open source library that was deployed to a server and local client which enabled the researcher to collect accurate data measurements. The dedicated environment included an Amazon AWS web server running only a single service at a time to limit measurement variance. Open source web server software Nginx (Sysoev, 2011) was used to handle HTTP

communication and therefore assumed to be following standardized approaches. Nginx further reversed proxied traffic to software running the researcher's software alleged to perform the correct work.

1.8 Theoretical Framework

Theoretical frameworks as described by Kumar, 2014 provided the structure for understanding the relationships between different variables in this study. The frameworks help researchers understand and make sense of complex phenomena. The research is typically developed from existing relevant literature and theories in the field and guided the design of the study and the interpretation of the results. It helped to focus the research and provide a logical structure for the study, as well as to identify areas where further research is needed.

The model of this study was a ZKP based approach and estimated the cost benefits/flaws versus existing deployments. Specifically, 1) simplicity of deployment in current solutions, 2) no degradation of system performance, and 3) enhancement of the overall security of the system. One way of measuring these is by their original design goals and results from existing deployments in production systems.

In comparison to SSO-based solutions, the study found that such solutions involve significant network traffic and multiple round trips between parties, making them more susceptible to network latency and multiple hops. In contrast, the ZKP-based approach required minimal communication between parties, typically involved only one or two interactions. This minimized the impact of network latency and hops, making the ZKP-based approach a more efficient and effective solution for web API designs.

1.9 Scope and Limitations

Scope

The scope of this experiment was to evaluate the benefits of using a ZKP-based approach for web API authentication, authorization, and access control. The experiment was conducted in a controlled cloud environment using virtual applications and

open-source software, which enabled the researcher to have complete control over all the variables measured. To ensure the results were representative of real-world scenarios, the study was conducted over the live internet using rented server space with cloud vendors and server hardware available directly to the researcher. The data collected from the experiment included measurements from virtual and physical web-based servers, mobile and desktop devices. The experiment revealed that the impact of hardware differences among mobile, desktop, and server platforms on the performance of the ZKP-based approach was minimal, indicating the approach can be effective across a range of devices and hardware configurations.

The study results are valid for Linux and Unix-based servers. While Windows-based servers were not included in the study, it is reasonable to expect that the same ZKP-based approach would be effective for Windows-based servers as well. The results of this study can be reproduced by using the same tools and environments as presented herein. It should be noted that the techniques presented in this study were implemented in a chosen programming language. While the results obtained in this study were specific to the chosen language, it is reasonable to expect similar results in other programming languages with the exception of performance. If the techniques presented in this study are implemented in a different programming language, the performance results may differ. The artifact implementation was done in a compiled language without garbage collection, just-in-time compiling, or runtime engines. This may have a significant impact on the performance of the techniques presented, and thus the results may differ if the techniques are implemented in a different language.

Limitations

While this study presented a comprehensive evaluation of the ZKP-based approach to API authentication, authorization, and access control, it is important to note that the research was limited to evaluations and deployments of existing solutions against the proposed approach. Further investigation into the design choices behind API development

and the implications on security and privacy is necessary. Additionally, while the study considered the most common and some less common attack scenarios, it is not possible to account for every possible threat or external influence. Moreover, the wide range of data types available for a given API poses a challenge in measuring the impact of the proposed approach on different data types, and further research is needed to address this limitation. This research is also limited in scope as it does not include various aspects such as the frequency of API usage, the nature of the data it provides, the deployment setup of the API system (including load balancers, operating systems, hardware, and cloud or colocated environment), and how the techniques would scale under heavy load. These factors can significantly impact the performance and security of the API setup. Future studies could explore the impact of these variables on the effectiveness and scalability of the proposed ZKP-based approach.

Delimitations

The study was conducted in a controlled web server environment where traffic between clients and API endpoints was measured. However, on the Internet, traffic is typically routed through different hops each time, which can add variance to delay measurements and is a difficult external influence to manage. Nonetheless, multiple measurements were taken to account for this variability. Moreover, cloud servers are often shared among multiple tenants, which can also affect results depending on the location, timing, and usage (peak or off hours). Therefore, the impact of server ownership, hardware, and location should be considered when shifting the experiments to other evaluations. These considerations can influence the performance and timing characteristics of the techniques. These limitations should be taken into account when interpreting the study's results and applying the techniques in different scenarios.

1.10 Summary

In Chapter 1, the background and need for more research concerning Web APIs were presented, with a focus on the challenges and weaknesses surrounding the security and

privacy of these systems. The significance and nature of the study were discussed, including the proposed cryptographic technique and its potential benefits, highlighting its advantages and disadvantages in terms of its ease of deployment in existing solutions, its lack of impact on system performance, and its enhancement of overall system security in existing systems.

The proposed technique was discussed in the context of its impact on system performance, including the potential for degradation in speed or efficiency. Further, the potential benefits of the technique were explored in terms of the overall security of the system, with a focus on the benefits of using a ZKP-based approach. The study's artifact and the evaluation of models and processes were shaped by the use of design science, which was also discussed in detail and why its the most appropriate for this research.

Finally, the scope and limitations of the study were explained, including the specific aspects of Web APIs that were considered in the study, as well as the potential external factors that may have influenced the results. By carefully considering the scope and limitations of the study, this research provides a valuable contribution to the ongoing discussion surrounding the security and privacy of Web APIs.

Chapter 2 presents an in-depth review of the current state of Web APIs, as examined through a comprehensive literature review. The chapter covers the historical evolution of Web APIs, analyzing their successes and shortcomings, and exploring how they have developed over time. It provides critical analyses of existing literature and explores key issues, challenges, and trends in the field. The chapter also examines the factors that have shaped the design, implementation, and deployment of today's Web APIs, and discusses the impact of these factors on performance, security, and scalability. Overall, the chapter provides a holistic view of the Web API landscape, including its challenges, limitations, and future directions.

Chapter 2 will provide an overview of cryptography used in web application development, including currently used cryptography and more advanced cryptographic techniques that serve as the foundation for the proposed ZKP-based approach. The

chapter will cover various advanced cryptographic primitives and related concepts, with the goal of providing a comprehensive understanding of the range of possibilities that can be explored and developed in the context of web API security and privacy. The discussion will also highlight the strengths and weaknesses of each technique, as well as their suitability for various use cases.

2 Chapter 2: Literature Review

Chapter 2 provides a comprehensive literature review of the current and historical state of the art in the field of Web APIs. This chapter offers a perspective on the diverse range of technologies and methodologies that have been used in the field and the industries that they serve. Furthermore, this chapter delves into the background and evolution of the cryptographic primitives that form the basis of this study's proposed approach, along with related primitives. The aim is to establish a solid foundation for understanding and measuring the impact of the study's approach. The review explores the challenges and shortcomings of current solutions and highlights the problems that the study aims to solve. By examining the limitations of current solutions, the review provides a foundation for understanding how advanced cryptography can be used to overcome these limitations including security, privacy, and efficiency concerns. Lastly, this chapter explores measurement techniques for evaluating the effectiveness of advanced cryptography primitives against various statistical attack methods, including correlation and deanonymization. By covering these topics, Chapter 2 lays the groundwork for evaluating the study's proposed ZKP-based approach, which aims to address the limitations of existing solutions through the use of advanced cryptography.

2.1 Authentication Methods

This section discusses the evolution and deployment of authentication methods in use by Web APIs. The literature review examines the security, privacy, and ease of deployment tradeoffs for each one and how they pertain to the proposed model.

Basic and Digest Authentication

Basic Authentication over HTTP was initially proposed in Section 11 of the Internet Request for Comment (RFC) 1945 (Nielsen et al., 1996). The scheme employs HTTP Headers to indicate a request for credentials and the response. The request is sent from the server with the HTTP Header *WWW-Authenticate: Basic realm="domain"*. The response is sends the HTTP Header *Authorization: Basic ....* The header includes the username and

password separated by a ":" then base64 encoded. Base64 encoding changes binary data into plain text so it can be easily transmitted over text based protocols like images or files. The text can be decoded back again into the original binary format losslessly.

The RFC states the scheme is insecure without first securing the communication via another method like TLS since the username and password are sent in clear text–without any cryptography to maintain confidentiality. This was more of a problem in the early days when TLS was costly and difficult to configure. Now TLS has become more ubiquitous with the rise of Let's Encrypt. Google's reports in 2021 that 97% of indexed websites use TLS (Google, 2021).

Two changes have been made with Digest Authentication as proposed in RFC2069 (Hallam-Baker et al., 1997), and RFC2617 (Franks et al., 1999) which sends a cryptographic hash of the password instead. However, this still suffers from the same problem, its dependent upon using a secure channel. Hashing the password offers little to no benefit since malicious actors can quickly deduce the hash preimage with credential stuffing (Alliance, 2021) and rainbow table attacks. RFC7235 (Fielding & Reschke, 2014) adds a random challenge from the server which permits authorization requests/responses but doesn't change how the credentials are sent or calculated. The scheme also supports Proxy server authentication (Mozilla, 2021)–an intermediary used to observe network traffic.

Basic Authentication is simple to set up for both the client and server entities since the entire scheme uses HTTP headers to handle the credentials. The server prepares a file or database containing valid usernames and passwords or password hashes. The client only needs to know one of these credential pairs to send over the correct HTTP Header (F5, 2022). Other variants include combining with many different cryptographic and password hashing algorithms.

Another limitation with this approach is that it creates identity silos–the credentials can only be validated at the same application/domain where they reside. To be reusable,

the credentials must be copied to other locations or stored in a shared central repository (F5, 2022).

Mutual TLS

TLS can also be used to authenticate the client as well as the server. The same difficulty to setup a server also applies to a client. For this to work correctly, the server and client must have **mutual** authentication enabled, otherwise the server doesn't perform client side authentication using TLS. This protocol uses X.509 certificates where the certificate chains must be trusted by both sides. This setup reduces or eliminates many classes of attacks man-in-the-middle, credential stuffing, password brute-force, and phishing (Cloudflare, 2022). However, managing certificates for each end client is a difficult task (Cloudflare, 2022) and is usually implemented with enterprise systems that have dedicated security teams.

Token Based Authentication

Token Based Authentication extend Basic and Digest Authentication where the verifier generates a unique token for a client to use instead of sending their credentials with each request. This avoids the necessity of requiring users to enter their credentials each time. The security token contains all the necessary information for the verifier to continue to interact with the client for a limited time or until the client logs out. The client sends the token as a web cookie, form data, or both (Rackspace, 2022a). Tokens can be encrypted or signed with a digital signature or Message Authentication Code (MAC). The relying party checks the token using cryptographic keys that correspond to the method used. TLS is required to maintain the token's confidentiality since tokens are replayed until they expire or are invalidated. Without confidentiality, an attacker can track the interaction. The scheme is also vulnerable if an adversary obtains the server's token signing or encryption key i.e. she can forge tokens. A variant of this method is to use API tokens that are randomly generated values that are used for credentials (Rackspace, 2022b), (AWS, 2019), (Oracle, 2022) instead of a password or pin.

2.1.1 Markup Languages

Markup languages are a way to encode data objects using tags and delimiters to indicate how the data is displayed or structured. These play an important role in how security tokens are used for authentication and authorization.

XML

Extensible Markup Language (XML) is a markup language used for data storage and transfer (W3C, 2008). It is similar to HTML in that it uses tags to structure the data, but unlike HTML, the tags used in XML are not predefined and can be created by the user to suit the specific needs of the data being stored. XML is often used for data exchange between different systems and organizations, as it allows for data to be represented in a standardized format that can be easily understood by different systems. XML can be accompanied by an XML Schema Definition (XSD).

XSD

XSDs are used to specify constraints on the tags and attributes that can appear in an XML document, as well as the relationships between those tags (W3C, 2012). XSD facilitates describing the structure of an XML document using a set of rules, similar to a database schema. It also supports data validation to help ensure that the data in an document is accurate and consistent. It validates an document during development and also at runtime, in order to catch errors early, before they cause problems downstream.

JSON

The Javascript Object Notation (JSON) format is also used for data storage and transfer but is much easier for humans to read and write and easy for machines to parse and generate (Crockford, 2017). JSON is similar in structure to XML, but it is more succinct. It is a text-based format that uses a simple syntax to represent objects, collections of objects, and their properties and values and is often used for data exchange between systems that are written in different programming languages. The majority of modern day

APIs opt for JSON over XML given its more compact and easier to read format.

JWT

JSON Web Token (JWT) are a standard for securely transmitting information as a JSON object for authentication and authorization purposes, as well as for securely transmitting information between parties. The token is broken into three parts: a header, a payload, and a signature. The header consists of two parts: the type of the token, which is JWT, and the cryptographic signing algorithm being used. The payload contains claims. Claims are data about an entity (typically, the user) and additional metadata. There are three types of claims: registered, public, and private claims. Registered claims are a set of predefined claims which are not mandatory but recommended, to provide a set of useful, interoperable claims. Some example of registered claims are: iss (issuer), sub (subject), aud (audience), exp (expiration time). Public claims are claims that are defined by those using JWTs. Private claims are claims that are specific to an implementation, that might not be shared between implementations. The signature is used to verify the authenticity of the sender of the JWT and tamper resistance. It can be used to authenticate users in a stateless way, so the server doesn't need to keep track of the user's session. Due to this flexibility in the claims, there have been vulnerabilities that allow attackers to bypass these checks (Database, 2022).

CBOR

Concise Binary Object Representation (CBOR) is a data storage and transfer format similar to JSON but is an even more compact and efficient binary form, while maintaining the same level of expressiveness (Bormann & Hoffman, 2020). CBOR is designed to be easily extensible, allowing for new data types to be added, is self-describing, which means that data encoded in CBOR can be decoded without prior knowledge of the schema or structure of the data. Similar technology that exists for JSON can also be used in CBOR given their similarities in structure.

2.1.2 Protocols

Security Assertion Markup Language

The SAML standard defined by the Organization for the Advancement of Structured Information Standards (OASIS) (2005) promotes authentication using XML tokens. The standard was the first to support Single Sign On (SSO) by identity providers. User's are directed to an identity provider who stores their credentials to determine authentication status. If valid, a SAML token is issued to the user who passes it to any service that supports the protocol. Service providers are relieved from storing and checking user credentials but still reliant on identity providers. The standard has been updated to SAML 2.0 (Wierenga & Lear, 2005) to allow authorization policies and other standards like Simple Authentication and Security Layer (SASL) (Myers, 1997) and the Generic Security Service Application Program Interface (GSS-API) (Wray, 1993).

SAML2 enabled the proliferation of SSO providers and simultaneously enhancing and enabling authentication and authorization (Wilson & Hingnikar, 2019). The benefits provided by SAML are a solution for web single sign-on across domains and federated identity. Web APIs redirect clients to their identity providers for authentication and authorization. Clients only use and/or have to remember a single set of credentials across multiple services and domains while companies benefit from a central location to enable and disable accounts. SAML2 supports stronger authentication methods like Two-Factor Authentication (2FA) or Multi-Factor Authentication (MFA)–a combination of something the client knows (pin, password), something they have (cryptographic key, smart card, phone, fob), and something they are (biometric like retina, voice recognition, face or fingerprints). Configuring strong authentication is complex and not often done by systems that don't use SSO (Wilson & Hingnikar, 2019) and can't be used by autonomous systems without creating bespoke solutions. SAML is not without weaknesses. Recently, an attack was successfully executed that permits an adversary to modify authentication responses.

This enables them to change signed assertions in order to gain unauthorized access to arbitrary user accounts, inject additional roles, change the recipient of the assertion, or to inject an entirely new username to compromise another user's account. (Roberts, 2021). Any problem in the SSO flow can be catastrophic given the multiple services and data that can be exposed to unauthorized parties. SAML only supports identity validation which by itself is not a practical business model (Wilson & Hingnikar, 2019). This new business model came in combination with OAuth 2.

OAuth 2

OAuth 2 allows applications to access client information and services with the user's consent, facilitating the authorization process for accessing third-party APIs. Before OAuth 2, the only method of sharing resources across services was to copy them, effectively giving ownership of the resource to multiple end points. OAuth 2 does not authenticate clients but instead relies on other methods like SAML and OIDC. OAuth 2 generates authorization tokens to indicate the authorization intent. The authorization server receives requests for an API and returns the security token that can be used by the application to access the API. The authorization request gives the application the scope of what it wants from the API. The authorization server evaluates the request and, if authorized, returns a token to the application (Wilson & Hingnikar, 2019).

OAuth 2 is based on JSON instead of XML with JWT (Jones et al., 2015, RFC 7519) because JWTs are much smaller and simpler to process than the XML packets used by SAML. Used in combination with OIDC a service is able to provide a more secure identity and authorization workflow.

OIDC

OpenID Connect (OIDC) is a more API friendly version of OpenID 2.0 built on top of OAuth 2. OIDC is more opinionated about the cryptography and data formatting so users only use what's available rather than having to make those decisions. OIDC uses JWT to send tokens and receive responses. OIDC "provides an identity service layer on top

of OAuth 2.0, designed to allow authorization servers to authenticate users for applications and return the results in a standard way" (Wilson & Hingnikar, 2019, OpenID Connect, para. 1). The standard employs similar roles to OAuth 2: the authorization server is called the identity provider that authenticates a user and return the user's claims and a relying party that delegates authentication to the provider and requests claims about the user from the provider. The service operates by passing ID tokens as JWT's that include a lot of metadata and claims concerning the user.

The protocol defines many different possible workflows to use and each comes with its benefits and complications (Wilson & Hingnikar, 2019). Many benefits are the conjunction between OAuth 2 and a secure login mechanism. However, it requires the client to define the security policies and controls in the provider and stores these somewhere that is tamper protected. Maintenance has shown this is also difficult. The FBI showed a case where even MFA can be exploited if not done properly (Gatlan, 2022). Okta, Auth0, Microsoft Active Directory, Google, Amazon AWS, Facebook and Github are examples OIDC providers in use today. As identity providers adoption becomes more prevalent, so does their likely of of exploitation (Vaughan-Nichols, 2022a). Okta one of the biggest was recently shown to have implemented poor security measures creating problems for their customers (Vaughan-Nichols, 2022b).

2.2 Summary of Authentication and Authorization Methods

This study has covered a brief description the current popular methods in use by APIs and their data formats. The next section covers newer cryptographic methods that have come to light recently in the past 10 years that may be used in place of the aforementioned methods and some older techniques still in use today. The section begins by describing some basic cryptographic primitives that will be used by this study and apply to the discussed topics.

2.3 Cryptography

2.3.1 Cryptographically Secure Random Number Generators

Cryptographically Secure Pseudorandom Number Generator (CSPRNG)s are algorithms that generate a sequence of seemingly random numbers that can be safely used for cryptographic purposes. The numbers generated by a CSPRNG are meant to be indistinguishable from truly random numbers and unpredictable, even if an attacker has some knowledge of the internal state of the generator.

XorShift

XorShift has good distribution properties but it not deemed suitable for cryptographic purposes. However, it is commonly used due to its distribution properties in statistics (Marsaglia, 2003). This study used XorShift to determine if a CSPRNG is needed or not.

Chacha

ChaCha is a stream cipher (Bernstein, 2008) that is an improved variant of the Salsa20 cipher family. It has been selected as one of the stream ciphers suitable for widespread adoption by (ECRYPT II, 2023, eSTREAM). It has also found integration in the Linux kernel random number generator (Federal Office of Information Security, 2022).

Operating System Random Number Generators

Operating System PRNGs are random number generators that are built into operating systems and are suitable for cryptographic purposes. They are usually implemented as part of the operating system's library or API and can be accessed by different programs running on the system. They serve as strong sources of entropy for cryptographic key generation and generating nonces, which are unique values used to prevent replay attacks in protocols such as TLS.

2.3.2 Summary of Random Number Generators

Random number generators play a major role in cryptography as signing and encryption keys depend on strong sources of randomness to prevent guessing or predicting their values. The next section covers another important primitive used in this study and in cryptogrpahy.

2.3.3 Cryptographic Hashes

Cryptographic hashes take a message and produce a fixed-size output, known as a 'hash' or 'digest'. They are designed to be a one-way function–it is computationally infeasible to determine the original input (message) from the output (hash). One of the main uses of cryptographic hashes is to verify the integrity of data. For example, when a file is downloaded, the hash of the downloaded file can be calculated and compared to the known hash of the original file. If the two hashes match, it can be assumed that the file has not been tampered with. The security of a cryptographic hash depends on the algorithm used, the hash length, and the collision resistance. Hash functions become vulnerable to attacks over the years and it's important to use the most recent and secure hash functions. This study relies on the Secure Hash Algorithm 2 (SHA-2) family (National Institute of Standards and Technology, 2002) and Secure Hash Algorithm 3 (SHA-3) (of Standards & Technology, 2015). The algorithm is specified with the number of output bits. SHA-256, for example, is the SHA-2 family that produces a 256-bit output with 128-bits of security–current computational power requires hashing 2^{128} possibilities to find two messages that yield the same output (Handschuh, 2011). SHA-512 produces a 512-bit output. SHA3-256 uses the SHA-3 family to produce a 256-bit output. The SHA-3 family also includes Extendable-Output Function (XOF)s hashes which can output any number of bits which a minimum length to ensure security. The SHA-3 XOFs are SHAKE128 and SHAKE256. SHAKE128 provides 128-bits of security and requires at least 256-bits be output. SHAKE256 is twice that.

Other uses of cryptographic hashes include preprocessing a variable length message

before computing a signature or hashing passwords. In the context of passwords, attackers compromise databases of user credentials and use precomputed large lists of common passwords in an attempt to guess what the original passwords were. Thus simply hashing a password is not strong enough in this context. Instead Password Hashing Algorithms exist for this purpose.

Password Hashing Algorithms

Password hashing algorithms are designed to securely store user passwords in a way that makes it difficult for an attacker to recover the original password even if they gain access to the hashed value. These algorithms take the password and produce a fixed-length output that is difficult to reverse or build large tables of precomputed hashes. The most common algorithms in use today are: PBKDF2, BCrypt, SCrypt, and Argon2. PBKDF2 and BCrypt rely on a number of iterations to loop over the input until the output is generated. Scrypt, and Argon2 take a different approach. They are designed to be memory-hard, making it more expensive for an attacker to use specialized hardware such as ASICs to speed up the cracking process. All algorithms use salting, which is the process of adding random data to the password before hashing it, a common practice when storing password hashes. This makes it even more difficult for an attacker to use precomputed tables to quickly crack a large number of hashed passwords.

Elliptic Curve Cryptography

Elliptic curves have two fields that they operate in, the base field for scalar values, and the modulus field for points (Brown, 2009). Scalars and Points each operate in a finite field (a field that contains a limited number of elements). An elliptic curve contains a set of parameters that express its form. A Weierstrass curve for example has the form $y^2 = x^3 + ax + b$ where a and b are the curve coefficients and x and y are the point. In cryptography the curve does not include every possible point and is limited by a curve modulus p i.e. values for x and y are reduced by p. The curve includes a base point G which is a finite field generator with an order q also known as the base order. Scalar

elements are values that operate in order q. Scalars represent elements that multiply points to result in another point. Since the points and scalars operate in finite fields, this makes it difficult given a random point to find a scalar that mapped to it from the base point. This is called the Elliptic Curve Discrete Log Problem (ECDLP). Private keys are values in the base field called scalars while public keys are random points on the curve computed from the base point multiplied by the scalar. As described by ECDLP, the private key should be hard to find given a public key (Barker et al., 2018).

Diffie-Hellman

The Diffie-Hellman public key cryptosystem is the oldest public key system still in use since it was first published (Diffie & Hellman, 1976). The idea is based on the fact that its easy to compute but hard to reverse like the ECDLP. It has been used in cryptographic proofs to describe the difficulty of finding a secret from a given value (den Boer, 1988). The algorithm works with elliptic curves where two parties each generate a scalar, compute a point using the scalar and curve base point and exchange their results. The two parties use their secret scalars against the exchanged points to arrive at a shared secret, even though they can only exchange in public settings. The protocol is widely deployed in almost every cryptosystem like Kerberos and TLS to facilitate secure key exchanges. For example, Alice generates a random scalar a and computes $A = a \cdot G$ and sends A to Bob. Bob generates a random scalar b and computes $B = b \cdot G$ and sends B to Alice. Alice computes $S = a \cdot B$ and Bob computes $S = b \cdot A$. The results are equivalent $S = a \cdot b \cdot G$ due to the associative property of multiplication. Since a and b are never disclosed, no observer can solve the ECDLP (Barker et al., 2018). Protocol variants have been described for authentication purposes given the simplicity of the computation (Teseleanu, 2021).

Cryptographic Signatures

Cryptographic signatures are a way for a party to prove their identity and the authenticity of a message or document. It is a generated using a private key, and can be verified using the corresponding public key. They prove the authenticity, non-repudiation

and integrity of a document or message. They are created by first applying a hash function to the message, then transforming the hash with the party's private key into a signature. The recipient can then use the party's public key to run a computation with the signature and verify that the message hash corresponds to the signature and key. As related to this study, the signature algorithms are Elliptic Curve Digital Signature Algorithm (ECDSA), Edwards-curve Digital Signature Algorithm (EdDSA), and Schnorr as these are the algorithms used by OIDC and OAuth 2. Signing algorithms rely on a combination of secure hash functions and the hardness of ECDLP.

Schnorr Algorithm

The Schnorr signature algorithm generates a signature given an elliptic curve with a base point G of order q, create a secret key a and the corresponding public key $A = aG$ (Schnorr, 1991). To create the signature σ over the message m, the following steps are used:

- Choose a random $0 < k < q$
- Compute $R = kG$
- Compute $e = H(m||A||R) \mod q$ where $H()$ is a cryptographic hash function. Different variants hash different values in different orders but this is inconsequential.
- Compute $s = k - e \cdot a \mod q$. Different variants add rather than subtract here, but this is also inconsequential.
- The signature is $\sigma = (e, s)$

Verifying a signature with the public requires computing $R = sG + eA$ and accepting if $e = H(m||A||R)$. Schnorr signatures are also a simple ZKP in that they prove the signer knows the private key a without revealing it. To expand on this concept, ZKPs can be created in a similar manner. Instead of calling them signatures, they are called Schnorr Proofs. This also makes ZKPs non-interactive–no interaction between the prover

and verifier is needed other than sending the ZKP itself from one to the other. This technique and the Fiat-Shamir heuristic are used to convert ZKPs to be non-interactive. Other reasons for using Schnorr signatures over ECDSA include they are more efficient, secure against existential forgery under chosen-message attacks and support for multisignatures–multiple signatures can be aggregated into a single signature.

Fiat-Shamir Heuristic

The Fiat-Shamir heuristic is a method for converting a interactive proof of knowledge into a non-interactive one (Fiat & Shamir, 1986). The basic idea is to replace the interactive exchange between the prover and verifier with a publicly verifiable cryptographic hash function. The prover commits to a random value, which is then used as the seed for the hash function. The prover then uses the output of the hash function as the verifier's challenge. The verifier can then verify the proof by checking that the prover's response is consistent with the commitment and the challenge. The technique also can turn ZKPs that are otherwise difficult or expensive to perform non-interactively, into much more efficient non-interactive proofs. It has been widely used in various cryptographic protocols and has been found to be a powerful technique for constructing efficient and secure non-interactive proofs of knowledge.

El-Gamal Encryption

ElGamal encryption (ElGamal, 1985) is an alternative to the RSA algorithm based on the difficulty of finding discrete logarithms like ECDLP. Two parties like Alice and Bob agree on an elliptic curve to use. Bob generates a private key, b, and a corresponding public key, $B = bG$. To encrypt a message, m, Alice generates a random value k and calculates a pair of values, $(C_1, C_2) = (kG, mG + kB)$. The pair (C_1, C_2) is the ciphertext. To decrypt the ciphertext, Bob uses his private key to calculate $M = C_2 - bC_1$, which is the original message in the exponent. Bob can then use other techniques to extract m or M can be the original message. One important property of ElGamal encryption is the fact that it can be used to encrypt and sign digital data, it is therefore a digital signature system too. This

technique can be used to create a ZKP that a message was indeed encrypted to a public key i.e. Verifiable Encryption.

Oblivious Transfer

Oblivious transfer is defined as a protocol in which the sender transfers information to the receiver, but the sender is unaware of what information the receiver receives (Schoenmakers, 2011). In other words, the sender is unaware of the choices made by the receiver during the protocol, which were used as the basis for the result. The cryptosystem is used by other protocols like oblivious access control (Camenisch et al., 2009) and multi party signature generation (Doerner et al., 2018). A more complicated and network efficient protocol has been discovered called Adaptive Oblivious Transfer and can be used in place of the non adaptive version (Libert et al., 2021). The difference between them is that instead of making a choice for all possibilities, the receiver can choose a subset while the security guarantees remain as strong.

Pairing Friendly Cryptography

Pairing friendly cryptography or pairing based curves are described as "a pairing is a function that maps a pair of points on an elliptic curve into a finite field" (Moody et al., 2015, page 11) i.e. a pairing maps a point or set of points to a finite field similar to scalars. An elliptic curve that supports the pairing function is said to be pairing friendly. Instead of a single field for points there are two. This function of two points is called a Bilinear map. These curves consist of the points in $\mathbb{G}_1$ of prime order p and $\mathbb{G}_2$ of order p^2. A pairing function e receives a point from both fields and returns a result in a large finite field $\mathbb{G}_T$ of order p^{12} known as the target group. The target group must be sufficiently large or it is not secure. The pairing function essentially takes all the scalars on both points and outputs the product of them all in the target group. This is written as $e(aG_1, bG_2) == e(G_1, G_2)^{ab}$ where a and b are the point multipliers or scalars. The scalars may be present in either or both fields. Regardless of which point to which they apply, the result is the same. In other words, $e(aG_1, bG_2)$, $e(G_1, abG_2)$, $e(abG_1, G_2)$ are all equivalent. If the result did not end in

a target group, this function could be used to solve ECDLP.

Galbraith et al., 2008 defines three pairing types: Type 1: $\mathbb{G}_1 = \mathbb{G}_2$ also known as symmetric pairings, Type 2: $\mathbb{G}_1 \neq \mathbb{G}_2$ but there exists an efficient homomorphism $\phi : \mathbb{G}_2 \to \mathbb{G}_1$, while no efficient one exists in the other direction, Type 3: $\mathbb{G}_1 \neq \mathbb{G}_2$ and no efficiently computable homomorphism exists between $\mathbb{G}_1$ and $\mathbb{G}_2$, in either direction. Pointcheval and Sanders, 2016 state "[type 3 pairings] offer a better efficiency and are compatible with several computational assumptions" making them more secure than type 1 or type 2 pairings. Type 1 and Type 2 pairings are no longer used in practice. This study employs type 3 pairings when referring to pairings. This unique property enables many new cryptographic protocols that were not previously possible.

BLS signatures

BLS signatures (short for "Boneh-Lynn-Shacham") are a pairing-based digital signature (Boneh et al., 2001). BLS signatures have succinct signature sizes, about half the size of ECDSA and Schnorr, fast signing and verification. They are used in various protocols such as threshold signatures, multisignatures, signature aggregation, and multiparty computation. They are useful because they offer several key benefits over other types of digital signatures. One of the main advantages is that they are very efficient in terms of the amount of computation required, making them well-suited for use in resource-constrained environments such as smart contracts on blockchain networks. BLS signatures are the only cryptographic signature that can be computed in a threshold setting with one round and no precomputation unlike Schnorr, its variant EdDSA and ECDSA. These same benefits apply to all signatures that are similar, covered later in this section. They form the basis for the other signature schemes used in this study.

Private Set Intersection

Private Set Intersection (PSI) is a ZKP scheme where parties have sets of data and want to learn the overlap without disclosing their respective sets to each other (Badrinarayanan et al., 2021). PSI operates with Diffie-Hellman computations, oblivious

transfer, cryptographic accumulators (Ghosh et al., 2016), pairings (Stefanov et al., 2012), or discrete-log equalities (Camenisch & Zaverucha, 2009). This has seen some deployment with respect to authorization and access control policies (Stefanov et al., 2012) called Authorized PSI. Other applications include database data joins, social networking, and password breach checks.

Short Group Signatures

Chaum and van Heyst, 1991 introduced group signatures which provide signer anonymity i.e. any group member can sign a message but the identity of the member remains hidden. In the event of a problem, a group manager can revoke the identity of the signer for a specific message.

Boneh et al., 2004 proposed BBS short group signatures–a group signature over a vector of messages. The signature is *short* because it's length is fixed and does not depend number of messages. This protocol as since been improved by Au et al., 2008 and Camenisch, Drijvers, and Lehmann, 2016 calling it BBS+ for use in Intel's Enhanced Privacy ID (EPID) and TPM's Direct Anonymous Attestation (DAA). The signature holder presents a signature proof of knowledge ZKP and selective disclosure ZKP which hides the signature and any subset of messages. For example, if messages A, B, C are signed with signature σ, all messages can be hidden, revealed or any mixture of them. The verifier checks the ZKP using the group public key, but cannot link any holder to the presentation using the proof. Because each presentation looks unique, it is said to be *unlinkable* i.e. the verifier cannot determine if two presentations are from the same holder or a different holder (Camenisch & Lysyanskaya, 2002b). A problem with this approach occurs when linking to a revocation status which has proven difficult to get right in practice (Sanders & Traoré, 2020).

Pointcheval and Sanders, 2016 proposed a simpler and smaller short group signature with a security update in (Pointcheval & Sanders, 2018). The other difference between PS signatures and BBS+ signatures is public key size. PS and BBS+ signatures require a

unique generator point per message to be signed. BBS+ signatures do not require the generators to be related to the signing key whereas PS signatures do. Thus, BBS+ public keys can be optimized into a single point whereas PS public keys require a point per message. BBS+ signature size is a point with two scalar values where as PS signatures are two points. Both group signatures require pairings to verify signatures and proofs. Another benefit to PS signatures is the easier ability to use in a threshold context with Coconut Attribute Based Credentials (Sonnino et al., 2020) and Vehicle to Vehicle communication (Camenisch et al., 2020).

Accumulators

An accumulator allows a set of elements to be represented by a fixed sized value. The scheme also permits proving set membership or nonmembership without revealing the element being inspected (Camenisch & Lysyanskaya, 2002a). Accumulators that support membership ZKPs are *positive* and those that support nonmembership are *negative*. Accumulators that support both are *universal*. If accumulators support adding and removing elements they are *dynamic*. Otherwise they are *static*. Accumulators membership and nonmembership proofs are constant time regardless of the number of elements in the set.

An accumulator can be constructed using an RSA modulus by Boneh et al., 2019 or using pairing friendly elliptic curves by Nguyen, 2005 and Camenisch et al., 2008 and Vitto and Biryukov, 2020. Both accumulators support efficiently adding and removing elements using a trapdoor function called a trusted setup, or a trustless setup where the trapdoor function is not known by any party and adding and removing elements requires intense computational work. In a trusted setup, an accumulator manager secures the trapdoor function like a private key and publishes updates whenever a change occurs.

ZKPs for membership and nonmembership proofs require the use of a witness in addition to knowledge of the element. The witness can be constructed quickly using the trapdoor and latest accumulator value, or high computational cost and knowledge of all set

elements without. When an accumulator value changes, all witnesses become invalidated and must be updated to become current. Generating ZKPs uses a witness and the set element to produce an unlinkable proof that can be verified by any party. Pairing friendly accumulators require pairings during verification.

Accumulators have largely been used to for a revocation scheme for anonymous credentials (Camenisch & Lysyanskaya, 2002a), (Camenisch et al., 2008), (Camenisch, Drijvers, & Hajny, 2016a), (Baldimtsi et al., 2017), but recently have been used to enhance verifiable computation (Ozdemir et al., 2019), blockchain ZKP checks (Boneh et al., 2019), and limited use tokens (Hölzl et al., 2019), (Camenisch, Drijvers, & Hajny, 2016b), but none have been deployed with APIs.

2.4 Summary of Cryptography

The previous section covered various cryptography schemes that can be used for digital signatures and revocation mechanisms. There are a few cases where these more advanced methods have seen deployment like IBM's Identity Mixer (Team & Switzerland, n.d.) and Microsoft's ("U-Prove Cryptographic Specification V1.1 Revision 3," 2013). Identity Mixer hasn't seen much adoption (if any) in Web APIs and U-Prove has been found to be insecure (Baldimtsi & Lysyanskaya, 2013) and discontinued. U-Prove requires the holder to visit the credential issuer for each presentation which further limits its use with Web APIs. Identity Mixer is similar token issuance in that the token can be reused multiple times until expiration or revocation halt it. Even so, Identity Mixer employs XML instead of JSON which deters adoption.

2.5 Measuring techniques

Normally, statistical measurements are metrics used to quantify and describe certain aspects of a dataset or population such as central tendency (e.g. mean, median, mode), measures of dispersion (e.g. range, variance, standard deviation), and measures of correlation (e.g. Pearson's correlation coefficient, Spearman's rank correlation coefficient). These measurements are used to summarize and analyze data, and to make inferences

about a population based on a sample. Although some of these metrics are utilized in this study, a significant number fail to accurately assess the objectives of the research. This section describes the methods employed to enhance the precision of this experiment evaluation by checking that cryptographic data does not exhibit any statistically significant patterns.

Shannon Entropy

Shannon entropy measures the amount of uncertainty or randomness in data such as a sequence of characters in a text or the distribution of values (Shannon, 1948). The entropy in data is calculated by summing the negative of the product of each probability and the logarithm of that probability for each unique item in the set. Higher entropy indicates more randomness or uncertainty, while lower entropy indicates less randomness or more predictability. It is commonly used to measure CSPRNG output to determine if the entropy is high enough for private keys or nonces. This experiment used Shannon's entropy test to quantify the randomness in the ZKPs generated before sending their contents to an API. In the case of random bytes for private keys, a value of 7.9 or higher was acceptable (Hamming, 1980).

Arithmetic Mean

The arithmetic mean, also known as the "average," is the measure of central tendency that is calculated by adding up all the values in a dataset and dividing by the total number of values. Private keys should have an even number of random bytes and thus a value between 127 and 128 was used since the lowest byte value is 0 and the highest 255.

Serial Correlation Coefficient

The serial correlation coefficient measures the correlation between observations in a time series dataset. It measures the similarity between a given observation and the previous observation and can determine whether there is a pattern or trend in the data over time.

A positive coefficient indicates a positive correlation between observations, meaning that as the value of one observation increases, the value of the other observation also tends

to increase. A negative coefficient indicates a negative correlation, meaning that as the value of one observation increases, the value of the other observation tends to decrease. If the coefficient is close to zero, it indicates that there is no correlation between the observations.

Perfectly random data has a coefficient of 0. However, since no data is perfectly random, this study allowed coefficients to be between -0.004 and 0.004 (Knuth, 1969, pp. 64-65).

Estimating PI using Monte Carlo

Estimating the value of π using Monte Carlo methods is a statistical technique that takes advantage of the properties of random sampling to approximate the value of π. The idea is to use random points within a square to approximate the area of a circle inscribed within that square.

As used in this experiment, the test was run over sequences of six bytes as the X and Y coordinates within a square, repeatedly. If the randomly generated point is located at a distance less than the radius of the circle inscribed within the square, it is considered a "hit". The ratio of hits to total number of points generated is used to approximate the value of pi. With large number of points, this approximation converges slowly to the true value of π, and it is suitable for cryptographic private keys. The Monte Carlo method for estimating π results in a value of 3.143580574 with an error of 0.06 percent (Park & Miller, 1988, pp. 1192).

ANOVA Framework

The ANOVA (Analysis of Variance) (Montgomery, 2017) is a statistical framework used to compare the means of two or more groups. It is a widely used statistical method for analyzing experimental data such as found in this study.

ANOVA partitions the total variation in the data into two sources: variation between groups and variation within groups. The variation between groups represents the

extent to which the group means differ from each other, while the variation within groups represents the random variation within each group.

The ANOVA framework involves testing the null hypothesis that there is no significant difference among the group means, against the alternative hypothesis that at least one group mean is significantly different from the others. To do this, ANOVA uses the F-test, which compares the ratio of the variation between groups to the variation within groups (Kutner et al., 2005).

ANOVA is useful for handling situations where there are multiple groups to compare, which is challenging to do using pairwise comparisons. It provides more information than a simple t-test by detecting if there are significant differences among the group means while controlling for the overall variability in the data (Faraway, 2005).

ANOVA can be used in a wide range of experimental designs, including one-way ANOVA (one independent variable with multiple levels), factorial ANOVA (two or more independent variables), repeated measures ANOVA (within-subjects design), and mixed ANOVA (combination of within-subjects and between-subjects factors).

Overall, ANOVA is a useful tool for analyzing the study's experimental data, allowing the researcher to test hypotheses about group means and draw conclusions about the differences among groups. This experiment employs ANOVA to compare variables and evaluate their collective impact on computations for CPU timing.

K-means clustering

K-means clustering is a technique for grouping or partitioning a set of objects based on their feature values. The k in K-means refers to the number of clusters to be generated. The goal of K-means is to divide data into k clusters, where each cluster is characterized by its centroid, the mean of all the data points in that cluster. K-means is a simple and efficient algorithm, however, it is sensitive to the initial placement of the centroids, and it may not work well if the clusters have different shapes or sizes. It assumes that the clusters are spherical, which may not be the case in many real-world datasets. There are variations

of the K-means algorithm that try to overcome some of these limitations but are not relevant for this experiment. Here, the study applies K-means in an attempt to cluster ZKPs. A variation of K-means was used called Lloyd's algorithm, where the iterative method assigns each data point to the nearest centroid, then updates the centroid of each cluster by computing the mean of all the data points assigned to that cluster. This process is repeated until the assignments no longer change or a stopping criteria is reached. For cryptographic purposes, if the ZKPs are random enough, the error value for K-means will be very high meaning unable to effectively cluster the values, the desired result.

LSTM Networks

Long Short-Term Memory networks are a type of recurrent neural network capable of processing input sequences of variable length and maintaining a memory of the past inputs (Hochreiter & Schmidhuber, 1997). LSTM networks are particularly useful for processing sequential data, such as time series, natural language, speech, and other sequence-to-sequence tasks. They are considered to be one of the most powerful architectures for sequential data processing. An LSTM network consists of a series of cells, which are units treated as "memory blocks" that maintain a state over time. Each cell has several gates that control the flow of information into and out of the cell, including an input gate, an output gate, and a forget gate. These gates allow the LSTM cell to selectively retain or discard information, enabling it to maintain a memory of past inputs. The downside to LSTMs is they are more complex than the traditional methods, requiring more computational power and more data to be trained effectively. They also have a tendency to forget or lose the information about the past inputs over time, specially for long sequences, which is called the vanishing gradient problem. While this issue can be addressed by using variants (Sutskever et al., 2014) it was not considered for this study. LSTMs are not employed in this study at all but are mentioned here for potential future work for measuring and attacking cryptographic ZKPs along with other time series analysis, regressions and forecasting techniques.

2.6 Summary of Measuring techniques

The previous section covered statistical measurements for testing and correlating datasets generated in this study. The measurements assess randomness and correlation to describe and analyze the data to infer information. Using these techniques we can increase the story told by the data generated by the cryptography and ZKPs.

2.7 Summary

The literature studied in Chapter 2 presented a comprehensive outlook on the state of API technology highlighting the unique features and properties of various authentication and authorization models. These technologies include HTTP authentication, SAML OAuth 2, and OIDC. Chapter 2 defined advanced methods for authentication and authorization with advanced cryptographic techniques like private set intersection, short group signatures, cryptographic accumulators and discussed techniques for refining the measurements used to evaluate the data. Despite these advancements in security and privacy, these advanced cryptographic techniques have not seen widespread use or adaptation, but they are crucial for understanding the research presented in this study.

Chapter 3 provides a comprehensive overview of the precise application of advanced cryptographic techniques, including the specific cryptographic tools and protocols used in the implementation. The chapter delves into the design considerations and technical details of the cryptographic approach, highlighting the unique features and properties of the approach and how they address specific security and privacy challenges. Additionally, the chapter discusses the validation process of the approach, including the testing, evaluation, and verification methods used to ensure the security and reliability of the solution. Overall, Chapter 3 provides a detailed exploration of the advanced cryptography approach used in the study, including its design, implementation, and validation.

3 Chapter 3: Research Methods

Chapter 2 reviewed the literature and current marketplace that applies to this study. A background of possible cryptographic techniques for authentication and authorization were surveyed and introduced. Chapter 3 discusses the research methods that were applied for the study. Details justifying the model and methods covered include: model design, data collection, data analysis, compare and contrast, and success criteria.

3.1 Research Approach

Design science is a research methodology that focuses on creating and evaluating artifacts (e.g., new approaches, tools, systems, etc.) that solve practical problems. This approach is appropriate for this study due to introducing a new approach to web API authentication, a practical problem that needs a better solution.

Chapter 1 outlined the design requirements and objectives to guide the development of the proposed approach to API authentication. These requirements include the need for the approach to be simple to deploy within existing solutions, maintain system performance, and enhance overall security. Additionally, the research examined the setup costs of implementing a zero-knowledge proof (ZKP)-based model, and how these costs affected performance, payload size, and privacy.

To ensure a thorough evaluation of the design's validity, specific variables were selected to align with the established requirements and objectives. This approach allowed for a comprehensive assessment of the proposed approach's ability to meet the identified needs and improve upon existing solutions then incorporate these improvements on the proposed artifact.

To enable end consumers to replicate the experiments conducted in this study for problem-solving, the artifact was developed using the design science methodology approach (Hevner et al., 2004) since the primary goal of design science is to create novel ideas, methods, technologies, and products that enhance the ability to analyze, design, implement, manage, and utilize information systems efficiently and effectively. This type of

research involved investigating a naturally occurring or artificially created phenomenon to improve our comprehension of that phenomenon or its particular aspects. In this study, the primary research activity was dedicated to constructing the artifact, with a willingness to accept a higher degree of potential failure to explore uncharted territory.

3.2 Data collection and Experiment

To assess the effectiveness of the proposed approach to web API authentication, a set of specific variables were identified and analyzed, including time to complete authentication or authorization, the number of steps required, network delay, and network traffic volume, observability, corruptability, and correlatability each of which played a critical role in evaluating the approach's ability to meet the established design requirements and objectives. The selection and analysis of these variables were driven by the research questions/hypothesis, which aimed to determine the extent to which the proposed approach satisfies the identified design requirements and objectives, and how it compares to existing authentication methods in terms of performance, security, and ease of deployment.

Many of these variables can be quantitatively measured using network tools such as ping, wireshark, and qperf, as well as operating system clock measurement commands like time. However, observability, corruptibility, and correlatability require alternative measurement methods. Observability is measured by analyzing the data passing between the endpoints (authenticator and authenticatee), the amount and what is human readable, including the ZKP payloads and associated metadata. Corruptibility is measured by assessing the ability to correctly authenticate with incorrect data, while correlatability evaluates the ability to correctly identify when the same authenticatee is attempting to authenticate.

In this study, various approaches to API design that are currently in use by production systems were examined. To gain a comprehensive perspective, web APIs from a diverse range of industries, including cloud infrastructure, single-sign-on, finance, know-your-customer, security, marketing research, search engines, version control,

academia, government, telecommunications, and medical, were analyzed.

Web APIs typically have public documentation that do not require creating an account to access and a sandbox environment to test the API's credentialing, authorization, and access control settings (AWS, 2019), (Rackspace, 2022b). While many APIs do support sandbox environments (AWS, 2021), (Rackspace, 2022b) where the API doesn't apply to real resources, the study aimed to assess the security and performance of APIs in real-world scenarios.

To identify areas where improvements would be most effective, the data was subjected to quantitative analysis as part of the design science validity cycle (Larsen et al., 2020). This analysis was used to evaluate the impact of changes implemented by the researcher, allowing for a comprehensive assessment of the effectiveness of the proposed approach.

3.3 Artifact details

To validate the effectiveness of the proposed approach, this study used ZKP-based tokens as a drop in replacement for any existing design. The ZKP approach employed several different cryptosystems, including Pointcheval-Sanders (PS) short group signatures (Pointcheval & Sanders, 2016), cryptographic accumulators (Vitto & Biryukov, 2020), enhanced with techniques from (Jaques et al., 2022), and Private Set Intersection (PSI) (Badrinarayanan et al., 2021). Short group signatures and accumulators were chosen due to their ability to satisfy the privacy, payload size, and performance requirements of the system. These cryptographic primitives enable the transmission of minimal clear text information, provide succinct ZKPs, and are computationally efficient.

Each approach is designed to serve specific purposes and is referred to as *Oberon*, with a suffix indicating the specific function is serves. The different offerings are discussed in detail in this chapter, highlighting their respective strengths and weaknesses. To begin, the notation followed is documented here

3.4 Notation

1. $\mathbb{Z}_q*$ represents the message space used by the elliptic curve.
2. $\mathbb{G}_1$ represents the point space used by the elliptic curve in group 1.
3. $\mathbb{G}_2$ represents the point space used by the elliptic curve in group 2. This group is bigger than $\mathbb{G}_1$ and slower to perform computations. Thus it is preferred to defer to $\mathbb{G}_1$ when possible.
4. $\mathbb{G}_T$ represents the value in the target group after computing a bilinear map or pairing function.
5. $H_{\mathbb{G}_1}$ is a hash function that maps a byte sequence to a point in $\mathbb{G}_1$ (Faz-Hernández et al., 2021) and is assumed to not output the point at infinity.
6. $H_{\mathbb{Z}_q*}$ is a hash function that maps a byte sequence to a value in $\mathbb{Z}_q*$ and never outputs a zero value.
7. $r \xleftarrow{\$} \mathbb{Z}_q*$ symbolizes a random integer drawn between the range $0 < r < q$.
8. The base point in $\mathbb{G}_1$ is P
9. The base point in $\mathbb{G}_2$ is $\widetilde{P}$.
10. $a \cdot P$ represents an elliptic curve scalar multiplication between the left hand side scalars a and the right hand side points P.
11. $e()$ is the bilinear map or type-3 pairing function $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$. Reminder that $e(aG_1, bG_2) == e(G_1, G_2)^{ab}$.

Oberon-ID

Oberon-ID is a novel approach that employs PS signatures, which enable the creation of a cryptographic token using a short group signature over a unique identifier.

These signatures are similar to BLS signatures and can be computed with threshold schemes to enhance security. With Oberon-ID, the public key is the only piece of information that needs to be stored by the API endpoint, and no database of passwords or credentials is required for validation. The token signing key can be kept in separeate secure locations to enhance security. When presenting the token, a signature proof of knowledge is generated and transmitted to the verifier instead of the token itself, ensuring that no valid or useful values for accessing the API can be obtained by any observer. The token is valid for a single use only, as it includes a nonce or timestamp to prevent replay attacks. Nevertheless, the token can generate unlimited ZKP presentations. This approach is an excellent alternative to traditional username and password or static API token systems that are long-lived and vary according to the user. The API manager can quickly retire tokens by rotating the public keys.

The scheme consists of 6 functions: DeriveInternals, Keygen, Signing, Blinding, Verifying, Proving, and Open.

$DeriveInternals(I) \rightarrow \{m, m', U\}$ is used to compute the three values used for each function. The following three equations hash the input to a message m, which is then hashed again to produce a digest of the message as a separate value m', then hashes this value to a random point U. U serves as a generator for computing the rest of the signature. This allows values that are not dependent on the secret key to be derived deterministically, which avoids the need for storage. However, these values can be stored to same computation. The signature is computed by combining the messages with the secret key then multiplied to the generator.

1. $m = H_{\mathbb{Z}_q*}(I)$
2. $m' = H_{\mathbb{Z}_q*}(m)$
3. $U = H_{\mathbb{G}_1}(m')$

4. $\{m, m', U\}$

$KeyGen$ PS signatures has three random elements $\{w, x, y\} \xleftarrow{\$} \mathbb{Z}_q*$ for the secret key and $\{\widetilde{W}, \widetilde{X}, \widetilde{Y}\}$ in $\mathbb{G}_2$ for the public key where $\{\widetilde{W} = w \cdot \widetilde{P}, \widetilde{X} = x \cdot \widetilde{P}, \widetilde{Y} = y \cdot \widetilde{P}\}$.

$Sign(sk, I) \rightarrow \sigma$ - takes the secret key sk and a random identifier I and outputs a token σ.

1. $\{m, m', U\} = DeriveInternals(I)$
2. $\sigma = (x + m.y + m'.w) \cdot U$

σ is the token value represented by a point on the curve.

$Blinding(\sigma, \beta) \rightarrow \sigma'$ - takes the token σ and a value β and outputs a blinded token σ'. *beta* can be any value that is used for MFA like a 6-digit pin, secure enclave value, or biometric. The blinding value is hashed to a point that is subtracted from the token. Since the token is just a point, the computation is simple.

1. $B = H_{\mathbb{G}_1}(\beta)$
2. $\sigma' = \sigma - B$

Note: *Blinding* can be applied any number of times using the latest σ' instead of σ with different values for β. Each applied blinding is a different authentication factor that will be needed when the token is used later.

$Verify(pk, I, \sigma) -> \{0, 1\}$ - is run by the token recipient to check the token validity.

1. $\{m, m', U\} = DeriveInternals(I)$
2. return 1 if and only if $e(U, \widetilde{X} + m \cdot \widetilde{Y} + m' \cdot \widetilde{W}).e(\sigma, -\widetilde{P}) == 1$, otherwise return 0

The idea here is balance the equations using the pairing function. This works because

$$e(U, \widetilde{X} + m \cdot \widetilde{Y} + m' \cdot \widetilde{W}) == e(\sigma, \widetilde{P})$$

$$e(U, (x + m.y + m'.w) \cdot \widetilde{P}) == e((x + m.y + m'.w) \cdot U, \widetilde{P})$$

$$e(U, \widetilde{P})^{x+m.y+m'.w} == e(U, \widetilde{P})^{x+m.y+m'.w}$$

$Prove(I, \sigma', [\beta_i, \ldots, \beta_n], n) \rightarrow \tau$ - takes the unique ID, the blinded token, any blinding factors, and a nonce and outputs a ZKP. This is sent to the server for verification. The server never sees the actual token. Any observer only sees the identifier I and a proof that is valid for the specific context. A timestamp with millisecond precision is sufficient to serve as the nonce however this study plans to use more than a timestamp. The proving steps randomize both the generator U and token σ using two randomizers r, t. Since two randomizers r, t are used, one of them must be sent to the verifier (t). However, if the randomizer t is known, the proof can be reversed. So a schnorr proof is sent instead which proves knowledge of the randomizer instead T. The proof is made non-interactive using the fiat-shamir heuristic to produce the challenge value c.

1. $\{m, m', U\} = DeriveInternals(I)$
2. $r, t \leftarrow \mathbb{Z}_q*$ creates the two randomizers.
3. $\{B_i, \ldots, B_n\} = \{H_{\mathbb{G}_1}(\beta_i), \ldots, H_{\mathbb{G}_1}(\beta_n)\}$ hashes each blinding factor to a point.
4. $R = r \cdot U$ randomizes the generator point U as a commitment to be sent to the verifier.
5. $T = t \cdot U$ randomizes the generator point T that will be recreated by the verifier as part of the schnorr proof.
6. $\tau = t \cdot \sigma' + t \cdot (\sum_i^n (B_i)$ randomizes the token.
7. $c = H_{\mathbb{Z}_q}(pk, I, m, m', U, n, R, T, \tau)$ is the fiat-shamir heuristic that computes the challenge.
8. $s = t - c.r$ is the schnorr proof.
9. $\pi = \{I, R, \tau, c, s, n\}$ is the final proof.

$Open(pk, I, \pi) \rightarrow \{0, 1\}$ - is executed by the relying party or API endpoint to check the validity of the ZKP.

1. $\{m, m', U\} = DeriveInternals(I)$
2. $T_v = s \cdot U + c \cdot R$ recreates the same T value as the prover.
3. $c_v = H_{\mathbb{Z}_q}(pk, I, m, m', U, n, R, T_v, \tau)$ recreates the same fiat-shamir heuristic.
4. if $c_v \neq c$ return 0. The challenges should be equal if the correct values are used.
5. return 1 if $e(T_v, \widetilde{X} + m \cdot \widetilde{Y} + m' \cdot \widetilde{W}).e(\tau, -\widetilde{P}) == 1$, otherwise return 0

This works because

$$
\begin{aligned}
e(T_v, \widetilde{X} + m \cdot \widetilde{Y} + m' \cdot \widetilde{W}) &== e(\tau, \widetilde{P}) \\
e((s + c.r) \cdot U, (x + m.y + m'.w) \cdot \widetilde{P}) &== e(t.(x + m.y + m'.w) \cdot U, \widetilde{P}) \\
e((t - c.r + c.r) \cdot U, (x + m.y + m'.w) \cdot \widetilde{P}) &== e(t.(x + m.y + m'.w) \cdot U, \widetilde{P}) \\
e(U, \widetilde{P})^{t.(x+m.y+m'.w)} &== e(U, \widetilde{P})^{t.(x+m.y+m'.w)}
\end{aligned}
$$

A modified version of the *Prove* function, called *AnonProve*, provides anonymity to the user by concealing their identifier during authentication at the endpoint. Additionally, the identifier can be signed blindly into the token and presented without being revealed, allowing the user to retain control over their identifier while keeping it and the token unknown to the endpoint. While this feature adds a layer of privacy for both parties, it does come with the potential for abuse. It is important for the web API to exercise caution when providing this option, as any value may be signed. This particular use case is not examined in this study, but mentioned for its potential for future research.

Oberon-Time

Oberon-Time extends Oberon-ID by limiting the longevity of the token to a specific epoch. The epoch is additionally signed into the random identifier. Each algorithm requires the following changes.

$DeriveInternal(I, \Upsilon) \rightarrow \{m, v, m', U\}$ derives an additional value v to be hashed into m'

1. $m = H_{\mathbb{Z}_q*}(I)$
2. $v = H_{\mathbb{Z}_q*}(\Upsilon)$
3. $m' = H_{\mathbb{Z}_q*}(m, v)$
4. $U = H_{\mathbb{G}_1}(m')$

$KeyGen$ outputs $sk = \{w, x, y, z\}$ and $pk = \{\widetilde{W}, \widetilde{X}, \widetilde{Y}, \widetilde{Z}\}$ calculated as before.

$Sign(sk, I, \Upsilon)$ takes as input sk, identifier I, timestamp Υ and outputs a token σ. The updated steps are

1. $\{m, v, m', U\} = DeriveInternals(I, \Upsilon)$
2. $\sigma = (x + m.y + m'.w + v.z) \cdot U$

$Verify(pk, I, \Upsilon, \sigma) -> \{0, 1\}$ - is updated to

1. $\{m, v, m', U\} = DeriveInternals(I, \Upsilon)$
2. return 1 if $e(U, \widetilde{X} + m \cdot \widetilde{Y} + m' \cdot \widetilde{W} + v \cdot \widetilde{Z}).e(\sigma, -\widetilde{P}) == 1$, otherwise return 0

The works just as before but with an additional value v

$Prove(I, \sigma', [\beta_i, \ldots, \beta_n], n) \rightarrow \pi$ - is updated to

1. $\{m, v, m', U\} = DeriveInternals(I, \Upsilon)$
2. $r, t \leftarrow \mathbb{Z}_q*$
3. $\{B_i, \ldots, B_n\} = \{H_{\mathbb{G}_1}(\beta_i), \ldots, H_{\mathbb{G}_1}(\beta_n)\}$
4. $R = r \cdot U$

5. $T = t \cdot U$
6. $\tau = t \cdot \sigma' + t \cdot (\sum_i^n (B_i)$
7. $c = H_{\mathbb{Z}_q}(pk, I, \Upsilon, m, \upsilon, m', U, n, R, T, \tau)$
8. $s = t - c.r$
9. $\pi = \{I, \Upsilon, R, \tau, c, s, n\}$

This is the same as Prove without the timestamp. The additional timestamp value doesn't add any additional work for prove.

$Open(pk, I, \pi) \rightarrow \{0, 1\}$ - is updated to

1. $\{m, \upsilon, m', U\} = DeriveInternals(I, \Upsilon)$
2. $T_\upsilon = s \cdot U + c \cdot R$
3. $c_\upsilon = H_{\mathbb{Z}_q}(pk, I, \Upsilon, m, \upsilon, m', U, n, R, T_\upsilon, \tau)$
4. if $c_\upsilon \neq c$ return 0
5. return 1 if $e(T_\upsilon, \widetilde{X} + m \cdot \widetilde{Y} + m' \cdot \widetilde{W} + \upsilon \cdot \widetilde{Z}).e(\tau, -\widetilde{P}) == 1$, otherwise return 0

This is the same as Open without the timestamp. The additional timestamp value doesn't add any additional work for open.

Oberon-Counter

Oberon-Counter leverages accumulators to restrict the frequency with which the token can be presented. The API endpoint defines the limited count for the token, i.e., the maximum number of times the token can be used for one-time verification proofs. Similar to a punch card that becomes invalid after a certain number of punches, the token can be utilized until all available uses have been exhausted. The elliptic curve based accumulator used in Oberon-Counter indicates the revocation status of a particular use. During

enrollment, the API endpoint computes the accumulator elements by hashing a random identifier I, the counter start value c_s, and the current index i until the limit c_e is reached. Each one-time element rt_i is then multiplied by the curve base point P. In this way, each use of the token is derived by hashing a counter and a fixed domain-specific string.

1. $s, f \leftarrow \mathbb{Z}_q*$ generates a secret signing key s and an accumulator secret key f.
2. $\widetilde{S}, F, \widetilde{F} = x \cdot \widetilde{P}, f \cdot P, f \cdot \widetilde{P}$ creates a two public verification keys and a base accumulator.
3. $rt_i = H_{\mathbb{Z}_q*}(I, P, c_s, i, c_e)$ computes the punch value based on the identifier, a start count, the current index, and the curve parameters.
4. $\sigma_i = \text{BLS-Sign}(s, rt_i)$ creates a BLS signature for each punch.
5. $A = \left(\prod_{i=c_s}^{c_e}(rt_i + f)\right) \cdot P$ computes the accumulator value by adding each punch to the secret accumulator key a, then multiplying all the resulting values together into a field element then multiplying that result by the base point.

$A, \widetilde{S}, \widetilde{F}$ are stored on the server, $\{I, c_s, c_e, \{\sigma\}_{i \in c_s..c_e}, F\}$ are sent to the client and securely deleted. The main security objectives of the study were to ensure that the server only stores the accumulator value and it cannot determine any current entries or create new ones. This is important because anyone can reconstruct the accumulator with this information. The protocol allows the accumulator manager and verifying server to be two distinct entities or the same.

However, the verifying server can observe a small amount of information regarding accumulator state, and could perform limited modifications to this state:

- The server can keep track of when a specific token is presented and how many times a user has presented it by correlating the data. The system might have a fixed limit that is the same for all users, and in that case, the server could determine how many

presentations remain. However, making updates without leaking any information would be costly, as clients would need to create a separate and more complicated ZKP and proof of correctness over the entire state. Despite this, the server does not gain any additional information beyond the count, making it more difficult to conceal. Thus the extra ZKP and proof of correctness yield little benefit that does not outweigh the cost.

- The server could delete accumulators or corrupt the accumulator by writing invalid or inconsistent values. This doesn't grant the server any capability beyond interrupting service to users, which the server can do more easily by simply not responding to client queries.
- The server could reinstate old accumulator values. This only benefits the client (or a malicious actor) with the ability to replay tokens.

The client can recompute the accumulator value or the server can send A to save on computation. This study has the client recompute it and relies as little on the server as possible. In a real world scenario, this can be specified by the API endpoint. rt_i represent one-time punches sent back to the server along with an accumulator witness α and σ_i. Witness construction generates the disposable token and next value of the accumulator. This scheme is better when combined with Oberon-Time or Oberon-Z to redeem a punch to gain access to an Oberon signature based token.

$Prove(A, I, c_s, c_e, i) \rightarrow \pi$ - takes an accumulator value, the unique ID, the starting value, the end value, and the current index which must be between the start and end values

1. if not $c_s < i < c_e$ return $\perp$
2. $rt_i = H_{\mathbb{Z}_q*}(I, P, c_s, i, c_e)$ recomputes the punches.
3. $\alpha = \left(\sum_{j=c_s \neq i}^{c_e} (rt_j \cdot P + F)\right)$ computes the witness for checking the punch against the accumulator.

4. $\pi = \{\alpha, rt_i, \sigma_i\}$

The user sends π to the API endpoint. A is replaced with α if the API accepts the proof.

$Open(rt_i, \sigma_i, \pi, A) \rightarrow \{0, 1\}$ - is executed by the API endpoint to check the validity of the one time token

1. if BLS-Verify(rt_i, σ_i, $\widetilde{F}$) == False, abort
2. if $e(\alpha, rt_i \cdot \widetilde{P} + \widetilde{F}).e(A, -\widetilde{P}) == 1$ replace A with α
3. if $\alpha == F$ delete

The endpoint solely keeps track of the accumulator's current value, preventing anyone, including the server, from discerning the constituent elements of the accumulator or the number of elements remaining. This guarantees that only clients with authorized access to the original information can generate tokens and that no observer, including system administrators or malicious actors, can extract valuable information from the accumulator values. The BLS signatures ensure the client cannot add extra punches.

Oberon-Z

Oberon-Z utilizes a hybrid approach of PS signatures and PSI to generate Zero-Knowledge Authorization Data Sets (ZKADS). This allows a client and server to treat their credential values as sets and execute intersection operations PSI for authorization evaluations. After authentication, the token contains an identification with a set of all the authorization privileges that the client possesses in the system. It is important to note that the entity issuing the token may differ from the one performing verification, and thus can be multiple entities responsible for verifying the token. Each verifier may have specific permission requirements to fulfill a request or access information, and authorization is granted if the client can demonstrate possession of the necessary set of permissions. The

client engages in an interactive protocol with the server through APIs, while only disclosing the permissions that intersect with a particular request. For example, the token may comprise 10 permissions, but only 2 to 3 are shared with the server at a time. The server learns about the client's lack of required permissions only in case of protocol failure, whereas the client is unaware of the server's expected permissions in any scenario unless communicated back. This architecture preserves the confidentiality of the server's permissions, while thwarting any attempts by observers to uncover the permissions held by the client. Similar to Oberon-Time, this protocol is an extension of Oberon-ID that includes a set of permissions, thus enabling precise access control. Each permission is incorporated into the token in a comparable manner, which increasing the key size while keeping a constant token size. By signing the permissions in this way, the client is unable to modify, add, or remove permissions at will, thus guaranteeing the legitimacy and tamper-resistance of the resultant set of permissions (Camenisch & Zaverucha, 2009).

$DeriveInternals(I, \Lambda) \rightarrow \{m, m', \{\alpha\}, U_1\}$ creates the field elements necessary to be signed similarly to the other Oberon methods

1. $m = H_{\mathbb{Z}_{q^*}}(I)$ converts the identity into a field element
2. $\alpha_i = H_{\mathbb{Z}_{q^*}}(a_i)$ converts each permission into a field element
3. $m' = H_{\mathbb{Z}_{q^*}}(m, \alpha_1, \ldots, \alpha_n)$ creates a digest field element of all elements to be signed.
4. $U_1 = H_{\mathbb{G}_1}(m')$ Computes the generator point for the token.

$KeyGen$ outputs $sk = \{w, x, y, \{z\}\}$ and $pk = \{\widetilde{W}, \widetilde{X}, \widetilde{Y}, \{\widetilde{Z}\}\}$ calculated as before in Oberon-ID. The set of z values indicates each permission granted by the issuing authority. The permission set can be represented by $\Lambda = \{a_1, \ldots, a_n\}$

$Sign(sk, I, \Lambda)$ takes as input sk, identifier I, the permissions Λ and outputs the token σ. The identifier may be omitted if the server is willing to allow the anonymous

authentication. In such cases, the server only learns that the client was valid, but not the identity of the requester. For the sake of simplicity, the steps outlined below include the identity of the requester.

1. $\{m, m', \{\alpha\}, U_1\} = DeriveInternals(I, \Lambda)$
2. $\sigma_1 = (x + m.y + m'.w + \alpha_i.z_i, \ldots) \cdot U_1$ computes the token that includes all the permissions and identifier.
3. Compute the polynomial $f(t) = \sum_i^n (\psi_i.t_i) = \prod_i^n (t - \alpha_i)$ with coefficients $\{\psi\}$ that are used to as part of the PSI protocol.
4. $\overline{m'} = H_{\mathbb{Z}_q*}(m, \psi_1, \ldots, \psi_n)$ computes the field element digest of all the polynomial coefficients.
5. $U_2 = H_{\mathbb{G}_1}(\overline{m'})$ creates the generator for the signature over the polynomial coefficients.
6. $\sigma_2 = (x + m.y + \overline{m'}.w + \psi_i.z_i + \ldots) \cdot U_2$ computes the signature over the polynomial coefficients.
7. $\sigma = \{\sigma_1, \sigma_2\}$

Verify is similar to previous descriptions.

$Prove_1(I, \Lambda, \sigma', [\beta_i, \ldots, \beta_n], K) \rightarrow \pi$ - is updated to include the Multi-Party Computation (MPC) steps of the PSI. The prover generates a random point J and sends it to the server. The server generates a key pair $k \leftarrow \mathbb{Z}_q*, K = k \cdot J$ and sends back K

1. $r_1, t_1, r_{t_1}, r_2, t_2, r_{t_2}, r_{m'}, r_{\overline{m'}}, r_{a_i}, r_{\psi_i}, r_{r_{\psi_i}} \leftarrow \mathbb{Z}_q*$
2. $\{B_i, \ldots, B_n\} = \{H_{\mathbb{G}_1}(\beta_i), \ldots, H_{\mathbb{G}_1}(\beta_n)\}$
3. $\{m, m', \{\alpha\}, U_1\} = DeriveInternals(I, \Lambda)$
4. $U_1' = r_1 \cdot U_1$ randomizes the generator for the permissions

5. $\tau_1 = t_1 \cdot U_1' + r_1 \cdot \sigma_1' + r_1 \cdot (\sum_i^n (B_i))$ randomizes the token signature over the permissions and undoes the blinding factors.

6. $D_1 = t_1 \cdot \widetilde{P} + m' \cdot \widetilde{W} + \alpha_i \cdot \widetilde{Z}$ creates a commitment to the permissions.

7. $T_1 = r_{t_1} \cdot \widetilde{P} + r_{m'} \cdot \widetilde{W} + r_{a_i} \cdot \widetilde{Z}_i + \ldots$ creates a randomized generator point to be recreated by the verifier.

8. $\overline{m'} = H_{\mathbb{Z}_q *}(m, \psi_1, \ldots, \psi_n)$

9. $U_2 = H_{\mathbb{G}_1}(\overline{m'})$

10. $U_2' = r_2 \cdot U_2$

11. $\tau_2 = t_2 \cdot U_2' + r_2 \cdot \sigma_2' + r_2 \cdot (\sum_i^n (B_i))$ randomizes the token signature containing the polynomial coefficients

12. $D_2 = t_2 \cdot \widetilde{P} + m' \cdot \widetilde{W} + \psi_i \cdot \widetilde{Z}$ creates a commitment to the coefficients.

13. $T_2 = r_{t_2} \cdot \widetilde{P} + r_{\overline{m'}} \cdot \widetilde{W} + r_{\psi_i} \cdot \widetilde{Z}_i + \ldots$ creates a randomized generator point to be recreated by the verifier.

14. $E_i = (r_\psi \cdot J, \psi_i \cdot J + r_{\psi_i} \cdot K)$ creates the El-Gamal ciphertext first part for the coefficient.

15. $E_{r_i} = (r_{r_{\psi_i}} \cdot J, r_{\psi_i} \cdot J + r_{r_{\psi_i}} \cdot K)$ creates the El-Gamal ciphertext second part for the coefficient.

16. $c = H_{\mathbb{Z}_q}(pk, I, m, U_1', U_2', \tau_1, \tau_2, D_1, D_2, T_1, T_2, E_i, \ldots, E_{r_i}, \ldots, J, K)$ is the fiat-shamir heuristic that computes the challenge.

17. $s_{m'} = r_{m'} - c.m'$, $s_{\overline{m'}} = r_{\overline{m'}} - c.\overline{m'}$, $s_{t_1} = r_{t_1} - c.t_1$, $s_{t_2} = r_{t_1} - c.t_2$ computes the schnorr proofs for the hidden message m' and the hidden permissions.

18. $s_{\alpha_i} = r_{a_i} - c.\alpha_i$, $s_{\psi_i} = r_{\psi_i} - c.\psi_i, s_{r_{\psi_i}} = r_{r_{\psi_i}} - c.r_{\psi_i}$ computes the schnorr proofs for the hidden coefficients.

19. $\pi = \{I, D_1, D_2, U_1', U_2', \tau_1, \tau_2, E_i, c, s_{m'}, s_{\overline{m'}}, s_{t_1}, s_{t_2}, s_{\alpha_i}, s_{\psi_i}, s_{r_{\psi_i}}\}$ is the completed proof.

$Open_1(pk, \pi) \rightarrow \{0, 1\}$ - is updated to

1. $m = H_{\mathbb{Z}_q*}(I)$

2. $T_1 = c \cdot D_1 + s_{t_1} \cdot \widetilde{P} + s_{m'} \cdot \widetilde{W} + s_{\alpha_i} \cdot \widetilde{Z}$ recreate the random commitment for the permissions.

3. $T_2 = c \cdot D_1 + s_{t_2} \cdot \widetilde{P} + s_{\overline{m'}} \cdot \widetilde{W} + s_{\psi_i} \cdot \widetilde{Z}$ recreate the random commitment for the coefficients.

4. Let $E_i = \mu_i, \nu_i$, compute $E_{r_i} = (c \cdot \mu_i + s_{r_{\psi_i}} \cdot \widetilde{P}, c \cdot \nu_i + s_{\psi_i} \cdot \widetilde{P} + s_{r_{\psi_i}} \cdot K)$. This is the El-Gamal ciphertext for the coefficients.

5. $c_v = H_{\mathbb{Z}_q}(pk, I, m, U_1', U_2', \tau_1, \tau_2, K, D_1, D_2, T_1, T_2, E_i, \ldots, E_{r_i}, \ldots, n)$ is the verifier's computation of the challenge.

6. if $c_v \neq c$ abort

7. if $U_1' == 1$ or $U_2' == 1$ abort

8. if $\tau_1 == 1$ or $\tau_2 == 1$ abort

9. if $e(U_1', \widetilde{X} + m.\widetilde{Y} + D_1).e(\tau_1, -\widetilde{P}) \neq 1$ abort

10. if $e(U_2', \widetilde{X} + m.\widetilde{Y} + D_2).e(\tau_2, -\widetilde{P}) \neq 1$ abort

The last two equations are identical to verifying the tokens from the previous Oberon schemes.

The server then homomorphically evaluates f for all elements in the permission set $V = \{v_1, \ldots, v_n\}$ by computing

$$w_i = \left(\prod_j^n (v_i^j \cdot E_{j,1}), \prod_j^n (v_i^j \cdot E_{j,2}) \right)$$

Afterwards server computes for each $w_i = k^{-1} \cdot E_{i,1} + E_{i,2}$ which will equal $-kr_i \cdot J + (a_i + kr_i) \cdot J = a_i \cdot J$ for the case where $a_i = 0$. The server creates a partition of $\{w_1, \ldots, w_n\}$:

$$C_1 = \{w_i := 1\}; C_y = \{w_i := y \neq 1\}$$

If any elements of V are present in C_1, it signifies that the prover possesses a legitimate set of permissions, and the server has acquired knowledge of the intersection between V and Λ. While the server can offer evidence that would enable the prover to obtain the same information, this study omits that possibility since the server intends to preserve the confidentiality of its collection.

3.5 Code Organization

The Oberon code in this study was written using Rust and compiled natively to match the target operating system and hardware platform. Rust was chosen due to its rapid run-time performance, ease of development, and extensive collection of cryptographic libraries. Compared to C/C++, Rust is now gaining popularity for several reasons. Rust provides numerous benefits that are found in C/C++ languages, while addressing the long-standing memory safety concerns that have plagued these languages. One of Rust's primary benefits is its built-in memory safety features, including a borrow checker, which assists in preventing programming errors that cause memory-related issues, such as buffer overflows and use-after-free errors. These memory safety characteristics make Rust programs more resilient and less susceptible to security vulnerabilities. As a result, Rust has been demonstrated to be a secure programming language (Jung, 2020). Additionally, Rust provides a more expressive type system and modern syntax, making it simpler and more efficient to write and maintain large, complex codebases. Rust also prioritizes concurrency, enabling more effective utilization of multi-core processors and proving advantageous when creating high-performance systems. Recently, Microsoft declared its

intention to embrace Rust for future projects and phase out the use of C/C++ internally (Clayburn, 2022).

Rust organizes and distributes library units of code as crates. A crate can encompass one or more modules, each of which can include functions, types, other items, or additional crates. Crates are self-contained, meaning they are defined by the code incorporated within them, rather than the code that invokes them. Crates can be produced by the user or downloaded from public or private registries, such as crates.io, which is the official Rust package registry. Crates serve the purpose of code sharing across different projects, and can be shared as either a binary file or source code. Crates are included as dependencies in the Cargo.toml file of the project, and the Rust package manager, *cargo*, manages the process of downloading, building, and linking the code. The crate system is a robust feature that empowers developers to generate, share, and utilize libraries and other code modules in a simple and efficient manner. The design artifact created in this study is written as a crate. As Rust does not use a garbage collector, it allows for the generation and compilation of code as C-compatible binaries. As a result, the final artifact is a C callable library authored in Rust (Matsakis & Klock II, 2014). This research leveraged this capability to enable the execution of all the methods discussed in alternative programming languages like Node.js and Python by consuming the C callable library. This way, the code was written once and utilized across multiple languages with minimal overhead. Each Oberon version is divided into separate crates, allowing each component to be employed either independently or in combination as a whole.

3.6 Instrumentation

This section outlines the data collection methods employed by the information instruments, which included measurements of network traffic, latency, CPU and RAM usage, and code complexity. In addition, human-readable metadata, along with information such as the time of day and type of API request, and the ZKP data was collected to assess privacy concerns. The main objective was to assess the effectiveness of

the ZKP in enhancing privacy and security, by measuring the correlation between ZKPs payloads and the potential for an attacker to deduce if the same user is making different API requests. However, evaluating the security of the system when the server is compromised was a challenging task, and it remains an area for future research. This analysis is more effectively measured qualitatively from a risk based approach. Nevertheless, there are still unanswered questions regarding the use of ZKPs to secure data in transit and at rest, particularly regarding potential risks and their impact, and possible mitigation strategies if necessary. These are areas for future research. The study was conducted across multiple operating systems, including Windows, Mac OSX, and various Linux-based systems, for both server and client environments, and the results indicated that the operating system did not have a significant impact on the findings.

3.7 Validity and Reliability

In research, validity refers to whether the researcher is measuring what they intend to measure or not (Kumar, 2014). It is therefore essential for the researcher to understand the significance of describing validity with respect to the current study. In this research, user feedback was not collected, and it is planned for future work. To establish the validity of this project, three factors were considered. Firstly, the artifact and how easily it can be integrated into APIs. Secondly, comparison benchmarks were included with the code that can be run independently, allowing for verification of the results. Finally, a report was generated, describing the tradeoffs and costs associated with using the new system compared to existing systems. This information will enable readers to understand what to expect when using this design science project and the associated costs and benefits. The design science research validity framework (Larsen et al., 2020) was used to iterate on the artifact, consisting of three components: Design Antecedent Validity, Development and Context Validity, and Design Outcome Validity. The first measures validity by examining the internal composition of the artifact and its relationship to the broader environment. The second emphasizes the importance of the artifact, while the third demonstrates its

utility. The framework provides researchers with guidance on how to select and utilize validities throughout the research process to enhance their designs, evaluations, and arguments. Design science validity refers to the degree to which the design solution satisfies the requirements and objectives of its target audience and successfully resolves the problem for which it was created. The validation of a design solution is established through testing and evaluating its effectiveness in practical situations, in addition to user feedback and other forms of user-centered evaluations. One effective approach for such evaluation is the test-retest methodology, which involves testing the solution in multiple iterations to evaluate its consistency over time. However, it is important to note that while test-retest methodology is a valuable technique for assessing the usability and effectiveness of design solutions, there are other methods available to evaluate design science validity. The test-retest methodology was a valuable approach as it enabled the researcher to assess the reliability of a variable or measure over time by administering the same test or measure on two occasions and comparing the results (Riedl & Robertson, 2010). This method was also useful for evaluating the stability and consistency of the design solution over time and comparing against different design alternatives. It is well-suited for technical studies that do not involve human subjects and where potential confounding factors, such as user education during testing, can be avoided. In the current research, the test-retest methodology was employed to gather quantitative data on CPU performance and network latency, ensuring that numerical data was acquired effectively and consistently. Additionally, the observability of the system was assessed by evaluating the amount of metadata sent along with the ZKP as this data is human readable and is more easily correlatable.

Validity and Reliability Performance

Network traffic and latency were measured to evaluate the performance of the ZKP-based approach compared to traditional API methods. By measuring the network traffic and latency, the researcher was able to compare the speed and efficiency of the two

methods, and determine whether the ZKP-based approach had any negative impact on network performance.

CPU and RAM usage were measured to evaluate the computational cost of the ZKP-based approach. By measuring the CPU and RAM usage, the researcher was able to compare the resource requirements of the two methods, and determine whether the ZKP-based approach was more or less efficient than traditional API methods.

Validity in this study refers to the extent to which the research accurately measures what it intends to measure, and whether the results are reliable and trustworthy. The validity of the instruments used in this study was ensured through rigorous testing and evaluation, including the use of test-retest methodology to measure the consistency of the results over time. The researcher compared the results of the ZKP-based approach to traditional API methods, and included benchmarks that can be run independently and verified to ensure the accuracy and reliability of the results. The Rust crate criterion was used which gives data over multiple repeated trials. However, these same trials were run manually using the linux command time and the compiled binary for the same number of times to ensure validity for CPU and RAM.

Reliability in this study refers to the consistency and stability of the instruments used to measure network traffic, latency, CPU and RAM usage. The reliability of the instruments was ensured through repeated measurements and comparisons, as well as through the use of standardized testing procedures and protocols. The instruments were measured on the same operating systems to ensure that the results were consistent and reliable and the same times of day to account for traffic spikes and lulls. Thus the same server was expected to have the same CPU load and network traffic each time.

Thus, the network traffic, latency, CPU and RAM usage measurements were used to evaluate the performance and computational cost of the ZKP-based approach compared to traditional API methods. The validity and reliability of the instruments used in the study were ensured through rigorous testing and evaluation, including the use of test-retest

methodology and standardized testing procedures.

The tests were run thousands of times to produce the means according to the law of expected averages. By running many simulations by the law of large numbers the average converges to the true expected values.

Validity and Reliability Privacy

Human readable metadata, the ZKP payload, time of day, and type of API request were used to measure the privacy implications of the ZKP-based approach compared to traditional API methods. Additionally, the ZKP data was statistically analyzed using randomization tests and clustering algorithms as described in chapter 2.5. Different PRNGs were used to detect any potential weakening of the ZKP for correlation attacks. These data points were collected to evaluate the ability of an attacker to deduce whether the same user was making different API requests and to compare the privacy of the two approaches.

To ensure the validity of the study, the human-readable metadata collected was carefully chosen to represent the most important information that an attacker could use to identify a user or their behavior like identifiers and credentials. Additionally, the time of day and type of API request were chosen to represent different scenarios that could occur in a real-world setting such as behavior based usage. The data was collected consistently across all test runs, and the same data points were collected for both the ZKP-based approach and traditional API methods to enable direct comparison using the network tools ping, wireshark, and qperf for network timing and package inspection. The results were analyzed and interpreted carefully to ensure that the conclusions drawn were supported by the data.

To ensure the reliability of the study, the payloads collected were checked for accuracy and consistency across test runs. The time of day and type of API request were carefully recorded to ensure that the data was representative of a real-world scenario. The data was collected using the same instruments and methods across all test runs to ensure consistency, and the results were analyzed using statistical methods to ensure that the

conclusions drawn were supported by the data. The reliability of the instruments was also checked by running repeated tests and ensuring that the results were consistent.

In summary, the human-readable metadata, ZKP payload, time of day, and type of API request were important data points for measuring the privacy implications of the ZKP-based approach compared to traditional API methods. The validity and reliability of the study were ensured by carefully choosing and collecting the data points, checking for accuracy and consistency, and analyzing the results using statistical methods.

Validity and Reliability Ease of Deployment

Ease of deployment is an important factor in evaluating the effectiveness and usefulness of a design solution, especially in the context of software development. While test-retest methodology can be used to measure certain aspects of a design solution, it is not suitable for evaluating ease of deployment. Instead, ease of deployment is evaluated by measuring the number of steps required to integrate the artifact into an existing API framework, as compared to other existing models. The number of steps in this context refers to the number of lines of code required for integration, as well as any necessary setup or maintenance tasks, such as configuring an SSO provider, inserting passwords into a database, updating hashing algorithms, and deleting records.

In order to ensure the validity and reliability of this measurement, it is important to use a consistent and well-defined methodology for measuring the number of steps required for integration. This can be done by providing simple code for integration, as well as by conducting tests and evaluations with multiple developers and teams. Overall, the ease of deployment of the artifact is an important factor to consider when evaluating its effectiveness and practicality for real-world use.

3.8 Data Analysis

Conducting statistical analysis and monitoring computational performance data presents certain difficulties. One of the main difficulties is the vast amount of data that can be collected (Adams & Heard, 2014). Another issue that specifically pertains to network

data is the correlation and timing of events. For this study, ZKPs do not affect network latency directly, only in the reduced number of trips to various endpoints. Therefore, the main focus is to evaluate the influence of ZKP on clients and API endpoints. Additionally, it is valuable to examine the correlation and sequence of ZKPs in relation to one another in order to better comprehend if an attacker could exploit this information for malicious purposes. Such data encompasses deanonymization and weak randomization. Computational tools were used for data gathering and experiments.

To meet these goals, data was gathered to asses these areas that includes using various CSPRNGs in proof generation to evaluate randomness and anonymization using the bytes entropy. Other models included clustering for correlation and timing attacks. Future work should investigate time series regressions and sequential data processing. Together these calculations and analyses paint a more detailed picture about the effectiveness of ZKPs as they are used for APIs. These calculations were largely utilized during the data analysis phase to create descriptive statistics. Each PRNG sample was compared against the others using ANOVA to test for enough evidence to suggest that there is a significant difference between them.

The method used for data generation made data cleansing redundant since the large number of simulations carried out effectively neutralized any influence from outliers. Additionally, all data points were used in the analysis without any consideration for outliers as extreme value were considered valid and integral to the dataset. The raw data was processed from researcher code to yield the desired output format.

3.9 Summary

Chapter 3 provides a comprehensive analysis of the ZKP-based token authentication and authorization artifact. The chapter delves into the mathematics and methods used in detail, outlining the architecture for each one and its various applications. In addition, the chapter explores the design science methodology, data collection, and methods employed in this research. The instrumentation, validity and reliability, and data analysis aspects are

also covered in depth. Moving forward, Chapter 4 offers a detailed account of the study's findings, including insights, discoveries, and conclusions regarding the design characteristics and a thorough examination of the outcomes.

4 Chapter 4: Experiment Results

This chapter provides the results for the Oberon authentication and authorization protocols developed in the design artifact in the previous chapter. Sections 4.1 to 4.2 cover the implementation details and a notable finding. Sections 4.5 to 4.6 discuss the data collection process and measurements used to validate Oberon against existing designs and protocols.

Primary interest was in exploring how Oberon compared in simplicity to more complicated setups and setups with weak security like no protection for password credentials, single iteration based hashing for password credentials, and recommended password credential hashing methodologies. The experiment further explored setups that use SAML, OAuth 2, OIDC. This experiment compared data payloads, network latency and round trips, and computation time. This work experimented with additional authentication factors and its effects to the overall time to complete authentication. Oberon data was also analyzed through statistical tests to measure the probability of correlation attacks.

The results covered here show a positive comparison and effective protocol for deploying a multi-factor enabled ZKP based authentication and authorization setup for APIs and additional authentication factors can be added with minimal impact to time and complexity.

The experiment also shows each structure serialized to a human readable format as JSON and CBOR.

4.1 Oberon implementation

Oberon was written to enable the main functions described in the previous chapter namely: key generation, token signing and verification, proof generation and opening, and applying blinding factors. The method interfaces was designed to be similar to other standard cryptographic signature libraries. Key generation is done using two methods, randomly or by hashing a seed.

```
impl SecretKey {
    pub fn new(mut rng: impl RngCore + CryptoRng) -> Self

    pub fn hash<B: AsRef<[u8]>(data: B) -> Self
}
```

Hash uses SHAKE-256 to compute a 64-byte digest that is treated like a big integer and reduced by the group order. This produces a deterministic yet uniform distribution of possible values. Random does something similar but uses the supplied CSPRNG to extract 64 bytes as described in the previous chapter. The serialization encodings are shown below

```
json = {
"w":"10857e0bc099160e35d6632842c2601e1356896b963168cad15080b60f9a2239",
"x":"6433af6c0c38064c2eadf7ffb8a5e47f9141abc32ca40310842b6c523f887403",
"y":"5d01e298c5983dd4b920c3d325cef21f484f5347c566ca815f136922162e7c5e"
}
cbor = \
10857e0bc099160e35d6632842c2601e1356896b963168cad15080b60f9a2239\
6433af6c0c38064c2eadf7ffb8a5e47f9141abc32ca40310842b6c523f887403\
5d01e298c5983dd4b920c3d325cef21f484f5347c566ca815f136922162e7c5e
```

JSON shows the individual components separated whereas CBOR shows the data as a single byte sequence. Binary sequences are encoded as hexidecimal strings for readability. HTTP based APIs use the JSON version whereas binary protocols can use CBOR.

Token signing takes two inputs, the signing key and the identifier or message to be signed. In Rust, it returns an option in case the secret key is invalid. An invalid secret key is a *zero* value.

```
impl Token {
```

```
    pub fn new<B: AsRef<[u8]>>(sk: &SecretKey, id: B) -> Option<Self>
}
```

There is no difference in serialization for the token since its simply a 48 byte value.

Token verifying takes the verification key, the identifier, and the token. *Choice* in Rust is a constant-time value to indicate true or false as **1** or **0.**

```
impl Token {
    pub fn verify<B: AsRef<[u8]>>(&self, pk: PublicKey, id: B) -> Choice
}
```

Blinding factors can be created from any binary octet string and applied to or removed from the token. This interface is consistent across Oberon implementations.

```
impl Blinding {
    pub fn new<B: AsRef<[u8]>>(data: B) -> Self
}
```

Code Example

```
    let pin = Blinding::new(b"1234");
    let blind_token = token - pin;
```

These same blinding factors serve as authentication factors that must be included when generating proofs. If blinding factor inclusion is neglected during proof generation, the proof validation will always fail even if the token, identifier, and verification key are correct.

```
impl Proof {
    pub fn new<B: AsRef<[u8]>, N: AsRef<[u8]>>(
        token: &Token,
        blindings: &[Blinding],
```

```
        id: B,
        nonce: N,
        mut rng: impl RngCore + CryptoRng,
    ) -> Option<Self>
}
```

Proof::new returns an option in case the *token* or *blindings* are invalid values.

The serialization for a proof is 256 bytes for CBOR and 580 bytes for JSON.

```
json = {
"proof":
"ac91b29f66d5d8014307b5bc15d6b75d5ee7a2ab444c1ac6\
cd1180f3b1cc26c5280d9b57cbfd88421774f40acdff3ac3",
"u_tick":
"a121c8b246270b682fe8677587df236a1f5f7fcf190a098d\
bc0f29e6d3b0ea47ba3f4670a81d558e0cb865f850d4de20",
"commitment":"ae1e714d7e37d310c1c6d3bde7eecfdb9e1\
b2c1656abc54f9e61ea3075f121c711542a39fe277ee669e0b\
88a6595cc4f0fddf242d4be7d9cafd24b305bc8ad90ea2c3fc\
0927f31ec93460b4806559e5e828d3c455fec0ead5c3682498d\
0f4027",
"challenge":
"db7d1cf6b02e42e8194a56725cd87e1c\
8c7e5cfda43cb1d77a2f2c1ae300a917",
"schnorr":
"ea826a98f136eb97baa91ec1a94383da\
d60f807f23ebbd29c775a920f9b0006b"
}
```

```
cbor =
ac91b29f66d5d8014307b5bc15d6b75d5ee7a2ab444c1ac6cd1180f3b1cc26
c5280d9b57cbfd88421774f40acdff3ac3a121c8b246270b682fe8677587df
236a1f5f7fcf190a098dbc0f29e6d3b0ea47ba3f4670a81d558e0cb865f850
d4de20ae1e714d7e37d310c1c6d3bde7eecfdb9e1b2c1656abc54f9e61ea30
75f121c711542a39fe277ee669e0b88a6595cc4f0fddf242d4be7d9cafd24b
305bc8ad90ea2c3fc0927f31ec93460b4806559e5e828d3c455fec0ead5c36
82498d0f4027db7d1cf6b02e42e8194a56725cd87e1c8c7e5cfda43cb1d77a
2f2c1ae300a917ea826a98f136eb97baa91ec1a94383dad60f807f23ebbd29
c775a920f9b0006b
```

Byte counts in the next sections only refer to the CBOR size since this is more compact and JSON doesn't add much readability since the values are byte sequences anyway.

Proof verification is identical to signature verification with the addition of the nonce where *id* is the message signed.

```
impl Proof {
    pub fn open<
        B: AsRef<[u8]>,
        N: AsRef<[u8]>
    >(
        &self,
        pk: PublicKey,
        id: B,
        nonce: N
    ) -> Choice
}
```

4.2 Changes from the original design for Oberon-Counter

The original design of Oberon-Counter used only an accumulator. However, during development, it was discovered to be insecure. The flaw allows a malicious prover to prove punches in the accumulator that were never included–to have unlimited punches when they should not. The fix is to cryptographically sign each punch so no new elements can be added or altered. The fix adds to the size of the token and token generation time but adds no computational time to proof generation and verification.

4.3 Oberon-Counter API changes

Key generation, creating and applying blinding factors, proof verification remains unchanged from Oberon-ID and Time. The only additional parameter for creating a new token is the maximum number of times it can be presented. It returns an additional value–the accumulator that determines if the presentations are valid. This accumulator is used in conjunction with the public key to verify proofs.

```
impl Token {
    pub fn new<B: AsRef<[u8]>>(
        sk: &SecretKey,
        id: B,
        count: usize,
    ) -> Option<(Self, Accumulator)>
}
```

Verifying tokens requires the *Accumulator* created at the same time as the token.

```
impl Token {
    pub fn verify<B: AsRef<[u8]>>(
        &self,
        pk: &PublicKey,
        acc: &Accumulator,
```

```
        id: B,
    ) -> Choice
}
```

An additional method was added for consumers to know how many times they have left to present which is just an unsigned integer.

```
impl Token {
    pub fn punches_remaining(&self) -> usize
}
```

Creating a proof only needed a minor update–make the token mutable since a punch is essentially burned when the proof is created.

```
impl Proof {
    pub fn new<B: AsRef<[u8]>, N: AsRef<[u8]>>(
        token: &mut Token,
        blindings: &[Blinding],
        id: B,
        nonce: N,
    ) -> Option<Self>
}
```

4.4 Oberon-Z API Changes

Key generation was modified to include the number of permissions as a parameter.

```
impl SecretKey {
    pub fn new(
        permission_count: usize,
        rng: impl RngCore + CryptoRng
```

```
    ) -> Option<Self>


    pub fn hash<B: AsRef<[u8]>>(
        permission_count: usize,
        data: B
    ) -> Option<Self>
}
```

Token signing takes a list of permissions in addition to the identifier and signing key. The API remains the same otherwise.

```
impl Token {
    pub fn new<
        M: AsRef<[u8]>,
        L: AsRef<M>
    >(
        sk: &SecretKey,
        id: M,
        permissions: L
    ) -> Option<Self>
}
```

Token verification is the same as previous Oberon methods. Creating proofs is more complicated that previous protocols. The server creates its secret k and associated public key K.

```
impl PresentationSecretKey {
    pub fn new(mut rng: impl RngCore + CryptoRng) -> Self
}
```

The user creates a proof using the updated API after receiving the presentation public key and the permissions requested.

```
pub enum PermissionProof<'a> {
    Hidden {
        value: &'a [u8],
        signature: Signature,
    },
    Revealed {
        value: &'a [u8],
        signature: Signature,
    }
}


impl Proof {
    pub fn new<
        'a,
        B: AsRef<[u8]>,
        L: AsRef<PermissionProof<'a>>,
    >(
        token: &Token,
        blindings: &[Blinding],
        id: B,
        permissions: L,
        presentation_key: &PresentationPublicKey,
        mut rng: impl RngCore + CryptoRng,
    ) -> Option<Self>
}
```

Proof verification returns the permissions that overlap with the request.

```
impl Proof {
    pub fn open<
        M: AsRef<[u8]>,
        L: AsRef<M>
    >(
        &self,
        public_key: PublicKey,
        permission_key: &PermissionSecretKey,
        permissions: L,
    ) -> Result<Vec<M>>
}
```

The server can then read the result. If $Result :: Ok$, then the list of overlapped permissions can be compared to what was expected and confirm or fail.

4.5 Data Collection

The Rust code was measured using Criterion, a benchmark framework over all the standard functions used for Oberon and all its variants. Criterion allowed extracting the raw samples into a text file which could analyzed with other tools. The issuer, prover, and verifier code was run on a client machine running Ubuntu 22.04 with a 12th Gen Intel Core i7-12700H, up to 4.7 GHz - 24 MB cache - 6 P-cores and 8 E-cores and a server machine to gather measurements and impact on cloud environments.

A C5.Large server was created in the Amazon AWS Cloud running Ubuntu 22.04, configured with the Nginx web server and basic and digest authentication turned on. Nginx was configured to reverse proxy to a running Rust service listening on a Unix socket that verified the credentials or ZKP and returned the result. Time was measured at each endpoint for generating and verifying the messages. The average network latency was

observed to be 225 ms for this portion of this experiment from networks like ping, qperf, and wireshark. The setup is shown in Figure C1. Latency was measured between data send and receive with log files recording timestamps on both sides. The server and client clocks were kept in sync to ensure time readings were consistent. The tests were conducted during the same time of day at 6 pm MST for consistency to account for possible shared environment work loads and shared network traffic. Criterion is a rust tool designed to be statistically useful for measuring code performance and accurately measuring optimizations and useful benchmarks (Heisler, 2022). Criterion measured computational time for each function out outputs results to the command line. Each piece of Oberon was also measured in terms of byte size. Other code was written to run the exact simulations without Criterion to check its validity and reliability. All measurements produced by Criterion were cross compared with samples generated without it with additional measurements manually. These measurements analyzed total mean time, standard deviations and the 95% confidence interval. The numbers were also analyzed using ANOVA to check whether PRNGs affected the times. B8 shows the results from this. Given the p-value is greater than 0.05 with a value of 0.117, this indicates that there is a 11.7% chance of obtaining an F-value as extreme or more extreme than 2.149 under the assumption that there is no difference among the group means. This is above the conventional significance level of 0.05, meaning that the observed F-value is not significant at the 0.05 level. Therefore, it is a failure to reject the null hypothesis and conclude that there is not enough evidence to support the claim that at least one of the PRNG group means is significantly different from the others.

4.6 Non functional measurements

To verify the *unlinkability* property of zero-knowledge, the privacy aspect of the design, or resilience to leaks, observation, theft, correlation, fingerprinting, multiple proofs were analyzed to check for any discernible patterns. The experiments aimed to determine if any such patterns were a result of the random number generator or elliptic curve used. Three different random number generators were used: XorShift as described by Marsaglia,

2003, Chacha as described by Bernstein, 2008, and the operating system cryptographically secure random number generator. The output of CBOR encoded proofs were examined using the K-Means clustering algorithm to identify any similarities and correlatability across multiple proofs. This analysis aimed to detect any potential issues with the random number generator or elliptic curve, which could result in the ability to *link* the proofs. The high error values observed indicate that the accuracy of the clusters identified is poor. The exact error numbers are shown in Table B9. This shows that given Oberon proof payloads they are truly unlinkable as long as the random number generator is decent. XorShift was used to show that even a non-CSPRNG could be used without hindering the strength of the unlinkable property. This is not advised in practice since XorShift and other non-CSPRNG are not resistant to side channel attacks and as shown Table B8, the CSPRNG only add a few microseconds of overhead which is not noticeable to a user or program as this will be dwarfed by the latency of sending data across a network. An ANOVA calculation again showed no statistical difference between the three PRNGs meaning no noticeable difference between them in privacy timings or performance.

4.7 Data for Basic and Digest Authentication

Basic and Digest Authentication are similar to usernames and passwords. For this experiment, three different modes were used: (1) a username and password without utilizing any hash function, similar to a static API token, (2) a username and one pass of a hash digest of the password, (3) a username and password with a password hashing algorithm.

Passwords were generated at random with a length of 8, 12, and 20 which are the common lengths of passwords used in the United States (Statista, 2022). The average time is shown in the Table B10 as measured with the three password lengths. The hashing algorithms were not accelerated with assembly instructions as implemented using RustCrypto code. The rounds and parameters were taken from (Owasp, 2022).

The metrics are Oberon's performance is 2 to 10 times faster than password hashing

based techniques like PBKDF2 and Argon2id but 10-15 times slower than insecure techniques as shown in in Tables B1- B3 and Figures C3- C4. The performance for the two functions the server performed are show in table B4. Against secure digest based techniques the proof sizes are 2.5 to 4 times larger. However, the proof sizes are still < 1KB which can be sent across modern networks in a few milliseconds.

4.8 Data for SAML, OAuth 2 and OIDC

Data collected for SAML OAuth 2, and OIDC was network traffic payloads and latency joining a dummy portal created following instructions based on (Google, Inc., 2022). Packets and latency were collected using WireShark (Wireshark Organization, 2022) and timestamps recorded in logs just like the previous experiment. Since these protocols encode various data in messages like authentication and authorization. This experiment used 10 permissions like in a Linux based file system with a sticky bit, read, write, and execute for the user, group, and world.

SAML

SAML messages ranged from 400 bytes to 4 kilobytes in size. The SAML 2.0 protocol requires 12 network passes. This translated to 2.7 seconds based on the average network latency to complete all rounds. SAML messages include policies which tend to be large even for a simple policy. Oberon ID and time match SAMLs message sizes with no policy attached and Oberon-Z maps to more complicated policies.

The results showed Oberon is 2-10X smaller, required one network trip and Oberon-Z required 3 network trips. This translated to 10-100X performance and latency difference. With 10 permissions, Oberon-Z size was 2.5 KB vs 5.4 KB for SAML thus a 2X reduction in traffic size.

OAuth2 and OIDC

Since OIDC is built on top of OAuth 2, the flows for these two protocols is identical. OIDC was solely measured against Oberon ID and Time since its purpose is to authenticate users and convey information about them. OAuth 2 controls authorization to

access a protected resource like APIs and thus was compared to Oberon-Z. OIDC messages are smaller than SAML due to using JSON instead of XML. Oberon proofs were similar in size to OIDC being 1.1 to 1.7 times smaller as seen in Tables B1- B3 and Tables B5- B7. The number of network trips was the same meaning the same latency delays effect. Identical findings are between OAuth 2 and Oberon-Z where Oberon-Z–1 to 1.2 times smaller and the same number of network trips.

Oberon proof verification time was slower than the previous two protocols using only ECC based signature verification. ECDSA on the NIST P-256 curve is 55 μs/permission vs Oberon-Z 20 ms and Oberon ID's 13.5 ms. The consequence is 30X slow down for Oberon-Z and 18X for Oberon ID. Inspite of the performance gap, the time required to use Oberon is still less than 50 ms which is fast enough to not be noticeable by API callers and includes a minor win for network traffic.

These results show Oberon is more resilient to certain attacks, more efficient and simpler.

4.9 Compared to existing API Data

An analysis of existing APIs from different industries, such as cloud providers, AI tools, databases, container sandboxes, secrets managers, email, pages, and sms sender applications, code repositories, social media, and website software was conducted. The APIs were classified according to their product offerings, technology, and features of their API authentication. These APIs were chosen due to their popularity among businesses and developers (DuVander, 2023) based on the number of API calls per month, specifically industries most commonly used by web developers and the latest trending technologies. The authentication methods were categorized by password-based, token-based, mutual TLS, OAuth, public key cryptography, and preshared keys based on symmetric cryptography. Some of the APIs support LDAP authentication, which can be classified into one or more of the categories previously mentioned. OAuth in this grouping includes single-sign on methods like OpenID Connect. Some APIs offered multiple options meaning

they could allow a ZKP offering without sacrificing existing customers. Tables B11- B19 show the findings for each API authentication.

A total of 45 APIs were analyzed in this study. Of these, 62% (28) support Token-based authentication, with 44% (20) using it exclusively. 24% (11) support Password-based authentication, with 7% (3) using it exclusively. 44% (20) support some form of OAuth, with 18% (8) using it exclusively. Other forms of authentication such as preshared key with symmetric cryptography, public-key cryptography, and mutual TLS are used by 3 or less. Password-based, Token-based, and OAuth account for 98% of the API's authentication techniques. It is also worth mentioning that OAuth, which supports MFA, is not commonly enabled for service-based APIs used by autonomous programs since these programs run without human interaction and MFA usually requires a human response. Thus, OAuth methods can be inferred to be similar to password-based authentication where the identity information is located on separate security domains than the API itself. This indicates that the majority of existing APIs (98%) have the potential to adopt or integrate the ZKP approach presented in this research for their authentication methods. As Oberon can serve as a replacement for OAuth where only a username and password are supplied, 18% of APIs could experience a significant reduction in authentication times and an additional 51% (20-Token and 3-password-based) would not only see the same performance improvements but also gain additional security benefits as no credentials are stored on the same system as the API, suggesting that the majority (98%) of current web offerings could potentially see advantages.

The findings of this study demonstrate that out of the 45 APIs analyzed, a majority of them (62%) support Token-based authentication, while 24% support Password-based authentication, and 44% support some form of OAuth. Notably, Token-based, Password-based, and OAuth authentication methods account for 98% of all authentication techniques used by the APIs analyzed in this study. Only a small percentage of APIs use other forms of authentication, such as preshared key with symmetric cryptography,

public-key cryptography, and mutual TLS.

Although OAuth supports MFA, it is not commonly enabled for service-based APIs used by autonomous programs. Thus, OAuth methods can be inferred to be similar to password-based authentication, where the identity information is located on separate security domains than the API itself.

The results show the majority of existing APIs (98%) have the potential to adopt or integrate the ZKP approach presented in this research for their authentication methods. Oberon, which can serve as a replacement for variants where only a username and password are supplied, can result in a significant reduction in authentication times for 18% of APIs. Furthermore, an additional 51% (20-Token and 3-password-based) of APIs would not only see the same performance improvements but also gain additional security benefits, as no credentials are stored on the same system as the API. Therefore, the majority (98%) of current web offerings could potentially experience significant advantages by adopting or integrating the ZKP approach presented in this study first and foremost of no credentials files or databases stored on the servers.

4.10 Summary of Experiment Results

Oberon was demonstrated to be congruent with existing secure authentication and authorization protocols. In comparison to Basic and Digest-based authentication methods, which are prevalent in modern deployments, Oberon is more efficient, lightweight, and requires fewer network interactions. When compared to OIDC, network round trips and data size are comparable, with slightly inferior but still acceptable performance. The main divergence between OIDC and Oberon is in terms of privacy–Oberon only reveals the requested permissions and nothing else, due to its use of ZKPs. By utilizing Oberon, consumers maintain token privacy. Oberon does not depend on the security of TLS to safeguard the authentication tokens and thus attackers would need to target the user's token directly where it is used to generate the proof. In the case of a man-in-the-middle attack, the attacker gains no information from either the user or by compromising the

authentication endpoint as storing only the verification key is enough and no credentials will exist on the system.

Oberon's API calls are structured to be similar to traditional cryptographic signature libraries allowing consumers to implement secure Web APIs just like they would employ signing tokens. The last section gives data collected from existing web APIs and their current authentication methods and how the ZKP method could be applied to their solutions.

4.11 Summary

In Chapter 4, the implementation details of Oberon and its variants were covered, including the design decisions that were made. Additionally, chapter 4 discussed the data collection and findings gathered for the experiments and simulations conducted to evaluate the performance, security, and simplicity of Oberon. The primary objectives of the study were to assess these key aspects of the new proposal, and the results demonstrated both the advantages and disadvantages of Oberon compared to existing deployments and the most commonly used APIs.

Chapter 5 presents the overall conclusions and interpretations of the study's findings, as well as recommendations and guidance for future work in this area. The results of the study offer insights into the potential benefits of adopting or integrating the Oberon approach for authentication in web offerings. By summarizing the key takeaways from the study, Chapter 5 aims to provide readers with a comprehensive understanding of the implications and significance of the research.

5 Chapter 5: Conclusion

This chapter summarizes the findings of the design science research conducted on Oberon and its variations. The study showed that these approaches provide a secure and uncomplicated method for API authentication, similar to traditional username and password or token-based systems, but with the added benefit of improved privacy, performance, and simplicity. Additionally, this research introduced novel cryptographic techniques for ZKP-based authentication and analyzed the trade-offs associated with using this approach. Overall, the results of the study demonstrate the potential of the Oberon approach to improve the security and privacy of web offerings, while maintaining ease of use for end-users.

5.1 Contributions

5.1.1 Oberon-ID and Time

These two protocols provide a streamlined approach to API development by eliminating the need to store credentials such as passwords or API tokens in secure storage or use computationally expensive password hashing techniques or data encryption. By doing so, they significantly reduce the risk of man-in-the-middle attacks and the potential for leaked credentials or tokens. Instead, API endpoints only store a public verification key to validate API calls, which is useless to attackers without the ability to create ZKPs. Additionally, API endpoints securely store a single signing key for creating new tokens, reducing the amount of sensitive information that needs to be managed. The experimental results show that the performance of these protocols is sufficient for most use cases and even improves in some cases. For example, password hashing methods don't scale when the system is under heavy load, requiring many system resources and tuning when underlying platforms change.

Moreover, integrating these protocols is straightforward, requiring only a few lines of code. The privacy benefits are substantial, even when relatively weak PRNGs are used. Overall, this study suggests that hesitation to adopt these protocols must stem from

reasons other than their effectiveness and ease of integration. The results of the ANOVA calculation indicate that the use of strong hashing and cryptography reduces the impact of the PRNGs on performance. The standard deviations also suggest that the performance remains relatively stable even with randomizations in the signing keys and inputs. The Shannon, Mean, Monte Carlo Value for Pi, and Serial Correlation Coefficient entropy tests demonstrate that the PRNGs have negligible impact on the randomness of the ZKPs. Furthermore, clustering techniques such as K-Means cannot be used to correlate the ZKPs.

5.1.2 Oberon-Counter

A new variation of Oberon, called Oberon-Counter, was introduced to improve API services by adding fixed number use tokens. This feature eliminates the need to track token usage in a database, which can be susceptible to tampering by malicious parties. The API endpoint only needs to track the current state of the accumulator and determine when the usage limit has been reached. The storage requirement for this approach is small, requiring only a 48-byte count value in addition to verification keys. If tampering occurs, it only results in a denial of service rather than unauthorized access. The performance, privacy, and simplicity benefits of this approach are similar to those of the previous two protocols.

5.1.3 Oberon-Z

Oberon-Z is an extension of Oberon that offers policy-based code and authorization capabilities, which are signed alongside the authentication token in a manner similar to other widely-used protocols such as OIDC, OAuth 2, and SAML. While this approach operates similarly to these protocols in terms of bandwidth and network usage, it does require higher computational requirements. However, the added computational costs are not extreme enough to rule it out remaining subsecond. The main advantage of Oberon-Z is increased privacy and simplicity, as not all of the user's capabilities are revealed, even when included with the credentials and permissions can be retained with the signer versus distributed among the entire system. Unlike protocols such as OIDC, which require the identity provider to withhold information before token creation, Oberon-Z offers greater

granularity in terms of redacting sensitive information. This increased flexibility in managing and controlling access privileges helps organizations to secure their systems against unauthorized access while maintaining the privacy of user data. The results of the experiments demonstrate that the performance of Oberon-Z is comparable to existing protocols, and the use of ZKPs provides additional security benefits, such as resistance to certain types of attacks. Moreover, using Oberon-Z does not require the API provider to depend on or configure permissions with external identity providers. This means that the API provider has full control over the authorization policies and can manage them directly on their own system. This approach offers more flexibility and reduces the reliance on third-party services.

5.2 Lessons Learned

Nonces To ensure the security of ZKP based protocols, preventing replay attacks is paramount by including a number used once, or nonce, demonstrating the proof presentation is recent and unique. Without proper nonce management, Oberon's security measures may be compromised, as presentations can be replayed as if they were the original unknown token, leading to unauthorized access. But this isn't a different weakness than is present in OIDC and TLS.

To prevent replay attacks, established protocols like TLS and OIDC handle nonce management by using techniques such as time-based or incremental nonce generation. The same techniques can be applied to Oberon to ensure that all presentations are current and unique. If an API endpoint fails to check for the nonce, limited additional security is offered by Oberon. For example, if presentations are not persisted, replayed presentations will go undetected, allowing attackers to bypass the client's security measures.

Proper nonce management is crucial to the security of the protocol. Commonly used nonce methods include server-generated random numbers, time-based, or incremental. By implementing proper nonce management, the risk of replay attacks is mitigated, ensuring the security of the ZKP based protocols.

Popular APIs The vast majority of the busiest APIs, accounting for approximately 98%, continue to rely on methods that have been in use for the past 20 years with little changing in the foreseeable future. This includes both token-based and OAuth 2-based approaches, which have become the de-facto standard regardless of the domain and business use case. While these methods have proven to be effective and widely adopted, they also come with their own set of challenges, such as maintaining secure storage for tokens and access keys, preventing token leakage, and mitigating man-in-the-middle attacks. As evidenced by the many news sources, these methods have done little to mitigate modern day attacks leading to the research conducted in this study to design a better method. Additionally, as more businesses move towards digital transformation and the use of APIs to power their applications, the need for more robust and privacy-preserving authentication and authorization methods is becoming increasingly urgent.

5.3 Limitations

5.3.1 Libraries

Oberon was developed in Rust and can be run on any platform that supports Rust. To increase adoption, Oberon can be integrated into popular Web API languages such as Golang, .NET, Java, NodeJS, and PHP. Although multiple languages have been utilized for Oberon ID implementation, its utilization is contingent on its ease of integration into existing development and production processes. Making the integration process simpler through language wrappers and pre-packaged options would require considerable resources, which wasn't thoroughly evaluated in this study.

5.3.2 Client token persistence

The study does not offer a definitive solution for storing and managing client tokens. This could be explored qualitatively in the future. Conventional techniques like read-only environment variables, configuration files, and database tables are feasible but do not offer the full range of capabilities provided by Oberon, such as local-based multi-factor authentication. Moreover, Oberon's multi-factor authentication feature can be linked with

hardware devices, allowing for mobility constraints, which can provide an added layer of protection.

5.4 Conclusion

Oberon is a new zero-knowledge proof protocol for enhancing traditional password and token-based authentication. The protocol eliminates the need for expensive password-hashing authentication, removing the necessity of managing password files that can be stolen or compromised, benefits from newer hardware, and is enables better security. The concept of punches, which are used to limit the number of times a token can be used, and private set intersection can also be used for greater flexibility such as policy-based code and authorization capabilities, similar to widely used protocols such as OIDC and OAuth 2, but with increased privacy and granularity. Additionally, Oberon enables local-based MFA and hardware-based MFA for added protection which are missing from non-interactive API based authentication. Overall, Oberon represents a promising new approach to token-based authentication with increased security, privacy, and simplicity compared to existing methods.

Acronyms

2FA Two-Factor Authentication. 28

API Application Programming Interface. 1–23, 27–31, 42, 46–49, 51, 54, 56–60, 65–71, 73, 86–93

CBOR Concise Binary Object Representation. 27, 73, 74, 76, 77, 84

CSPRNG Cryptographically Secure Pseudorandom Number Generator. 31, 42, 71, 74, 84, 111

DAA Direct Anonymous Attestation. 39

ECC Elliptic Curve Cryptography. 86

ECDLP Elliptic Curve Discrete Log Problem. 34–36, 38

ECDSA Elliptic Curve Digital Signature Algorithm. 35, 36, 38, 86

EdDSA Edwards-curve Digital Signature Algorithm. 35, 38

EPID Enhanced Privacy ID. 39

GSS-API Generic Security Service Application Program Interface. 28

HSM Hardware Security Module. 6

HTTP HyperText Transfer Protocol. 3, 7, 8, 17, 23, 24, 46, 74

InfoSec Information Security. 4

JSON Javascript Object Notation. 26, 27, 29, 41, 73, 74, 76, 77, 86

JWT JSON Web Token. 27, 29, 30

LDAP Lightweight Directory Access Protocol. 86

LSTM Long Short-Term Memory. 45

MAC Media Access Control. 6

MAC Message Authentication Code. 25

MFA Multi-Factor Authentication. 9, 28, 52, 87, 88, 94

MPC Multi-Party Computation. 61

OASIS Organization for the Advancement of Structured Information Standards. 28

OAuth 2 Open Authorization Framework. 3, 8, 9, 29, 30, 35, 46, 73, 85, 86, 91, 93, 94

OIDC OpenID Connect. 3, 8, 9, 29, 30, 35, 46, 73, 85, 86, 88, 91, 92, 94

OWASP Open Web Application Security Project. 1, 5

PII Personal Identifiable Information. 5

PRNG Pseudorandom Number Generator. 31, 69, 71, 83, 84, 90, 91

PS Pointcheval-Sanders. 49, 50

PSI Private Set Intersection. 38, 39, 49, 59, 61

PUF Physical Unclonable Function. 6

REST Representational state transfer. 3

RFC Request for Comment. 23, 24

RPC Remote procedure call. 3

SAML Security Assertion Markup Language. 3, 7–9, 28, 29, 46, 73, 85, 86, 91

SASL Simple Authentication and Security Layer. 28

SHA-2 Secure Hash Algorithm 2. 32

SHA-3 Secure Hash Algorithm 3. 32

SOAP Simple Object Access Protocol. 3

SSO Single Sign On. 18, 28, 29, 70

TEE Trusted Execution Environments. 6

TLS Transport Layer Security. 7, 24, 25, 31, 34, 86–88, 92, 119

TPM Trusted Platform Module. 6, 39

WAF Web Application Firewalls. 10

WCF Windows Communication Foundation. 3

XML Extensible Markup Language. 7, 26–29, 41, 86

XOF Extendable-Output Function. 32

XSD XML Schema Definition. 26

ZKADS Zero-Knowledge Authorization Data Sets. 59

ZKP Zero-Knowledge Proof. 1–3, 13, 14, 16–21, 23, 35–42, 45–49, 51, 53, 58, 65–71, 73, 82, 87–92, 110, 111

References

Adams, N., & Heard, N. (2014). *Data analysis for network cyber-security* (N. Heard & N. Adams, Eds.). Imperial College Press.

Akamai. (2021). Api: The attack surface that connects us all. *State of the Internet*, *7*. https://www.akamai.com/content/dam/site/en/documents/state-of-the-internet/soti-security-api-the-attack-surface-that-connects-us-all.pdf.

Akamai Technologies. (2022). *State of the internet / report | api: The attack surface that connects us all* (Technical Report). Akamai Technologies. https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/report/state-of-the-internet-security-api-report-2022.pdf

Alliance, C. S. (2021). Understanding the owasp api security top 10.

Au, M. H., Susilo, W., & Mu, Y. (2008). Constant-size dynamic k-taa. *IACR Cryptology ePrint Archive*, *2008*, 136. https://doi.org/10.1007/11832072_8

AWS. (2019). Configuration and credential file settings.

AWS. (2021). New aws solutions implementation: Aws innovation sandbox.

Badrinarayanan, S., Miao, P., & Xie, T. (2021). Updatable private set intersection. *IACR Cryptol. ePrint Arch.*, *2021*, 1349.

Baldimtsi, F., Camenisch, J., Dubovitskaya, M., Lysyanskaya, A., Reyzin, L., Samelin, K., & Yakoubov, S. (2017). Accumulators with applications to anonymity-preserving revocation. *2017 IEEE European Symposium on Security and Privacy (EuroS P)*, 301–315. https://doi.org/10.1109/EuroSP.2017.13

Baldimtsi, F., & Lysyanskaya, A. (2013). Anonymous credentials light. *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, 1087–1098. https://doi.org/10.1145/2508859.2516687

Balmas, Y. (2022). Main factors accelerating api security risks in financial services [https://www.financederivative.com/main-factors-accelerating-api-security-risks-in-financial-services/].

Barker, E., Chen, L., Roginsky, A., Vassilev, A., & Davis, R. (2018). Recommendation for pair-wise key-establishment schemes using discrete logarithm cryptography.

Bernstein, D. (2008). Chacha, a variant of salsa20.

Biryukov, A., & Tikhomirov, S. (2019). Deanonymization and linkability of cryptocurrency transactions based on network analysis. *2019 IEEE European Symposium on Security and Privacy (EuroSP)*, 172–184. https://doi.org/10.1109/EuroSP.2019.00022

Blum, M., Feldman, P., & Micali, S. (1988). Non-interactive zero-knowledge and its applications. *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, 103–112. https://doi.org/10.1145/62212.62222

Boneh, D., Boyen, X., & Shacham, H. (2004). Short group signatures. In M. Franklin (Ed.), *Advances in cryptology – crypto 2004* (pp. 41–55). Springer Berlin Heidelberg.

Boneh, D., Bünz, B., & Fisch, B. (2019). Batching techniques for accumulators with applications to iops and stateless blockchains. https://doi.org/10.1007/978-3-030-26948-7_20

Boneh, D., Gorbunov, S., Wahby, R. S., Wee, H., Wood, C. A., & Zhang, Z. (2022). *BLS Signatures* (Internet-Draft draft-irtf-cfrg-bls-signature-05) [Work in Progress]. Internet Engineering Task Force. Internet Engineering Task Force. https://datatracker.ietf.org/doc/draft-irtf-cfrg-bls-signature/05/

Boneh, D., Lynn, B., & Shacham, H. (2001). Short signatures from the weil pairing. *Journal of Cryptology*, *17*(4), 297–319.

Bonneau, J., Herley, C., Oorschot, P. C. v., & Stajano, F. (2012). The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. *2012 IEEE Symposium on Security and Privacy*, 553–567. https://doi.org/10.1109/SP.2012.44

Bormann, C., & Hoffman, P. E. (2020). Concise Binary Object Representation (CBOR). https://doi.org/10.17487/RFC8949

Brown, D. R. L. (2009). Standards for efficient cryptography. elliptic curve cryptography, version 1.5.

Buchanan, B. (2022). Stop hardcoding your secrets...how to securely keep a secret [https://medium.com/asecuritysite-when-bob-met-alice/stop-hardcoding-your-secrets-how-to-securely-keep-a-secret-adbf638f86e1].

Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., & Maxwell, G. (2017). Bulletproofs: Short proofs for confidential transactions and more [https://eprint.iacr.org/2017/1066]. https://eprint.iacr.org/2017/1066

Camenisch, J. (2014). Concepts Around Privacy-Preserving Attribute-Based Credentials [Part 1: Invited Papers]. In M. Hansen, J.-H. Hoepman, R. Leenes, & D. Whitehouse (Eds.), *Privacy and Identity Management for Emerging Services and Technologies: 8th IFIP WG 9.2, 9.5, 9.6 / 11.7, 11.4, 11.6 International Summer School, Nijmegen, The Netherlands, June 17-21, 2013, Revised Selected Papers* (pp. 53–63). Springer. https://doi.org/10.1007/978-3-642-55137-6_4

Camenisch, J., Drijvers, M., & Hajny, J. (2016a). Scalable revocation scheme for anonymous credentials based on n-times unlinkable proofs. *Proceedings of the 2016 ACM on Workshop on Privacy in the Electronic Society*, 123–133. https://doi.org/10.1145/2994620.2994625

Camenisch, J., Drijvers, M., & Hajny, J. (2016b). Scalable revocation scheme for anonymous credentials based on n-times unlinkable proofs. *Proceedings of the 2016 ACM on Workshop on Privacy in the Electronic Society*, 123–133. https://doi.org/10.1145/2994620.2994625

Camenisch, J., Drijvers, M., & Lehmann, A. (2016). Anonymous attestation using the strong diffie hellman assumption revisited. *9824*, 1–20. https://doi.org/10.1007/978-3-319-45572-3_1

Camenisch, J., Drijvers, M., Lehmann, A., Neven, G., & Towa, P. (2020). Short threshold dynamic group signatures. https://doi.org/10.1007/978-3-030-57990-6_20

Camenisch, J., Dubovitskaya, M., & Neven, G. (2009). Oblivious transfer with access control. *IACR Cryptology ePrint Archive*, *2009*, 131–140. https://doi.org/10.1145/1653662.1653679

Camenisch, J., Kohlweiss, M., & Soriente, C. (2008). An accumulator based on bilinear maps and efficient revocation for anonymous credentials. *IACR Cryptology ePrint Archive*, *2008*, 539.

Camenisch, J., & Lysyanskaya, A. (2002a). Dynamic accumulators and application to efficient revocation of anonymous credentials, 61–76. https://doi.org/10.1007/3-540-45708-9_5

Camenisch, J., & Lysyanskaya, A. (2002b). A signature scheme with efficient protocols, 268–289.

Camenisch, J., & Zaverucha, G. (2009). Private intersection of certified sets. *5628*, 108–127. https://doi.org/10.1007/978-3-642-03549-4_7

Chaum, D., & van Heyst, E. (1991). Group signatures. In D. W. Davies (Ed.), *Advances in cryptology — eurocrypt '91* (pp. 257–265). Springer Berlin Heidelberg.

Cheng, L. (2020). Api data breaches in 2020.

Clayburn, T. (2022). In rust we trust: Microsoft azure cto shuns c and c++.

Cloudflare. (2022). What is mutual tls (mtls)?

Creswell, J. W. (2018). *Research design: Qualitative, quantitative, and mixed methods approaches 5th edition*. SAGE Publications.

Crockford, D. (2017). The JavaScript Object Notation (JSON) Data Interchange Format [RFC 8259].

Database, N. V. (2022). Cve-2022-23529 [https://nvd.nist.gov/vuln/detail/CVE-2022-23529].

den Boer, B. (1988). Diffie-hellman is as strong as discrete log for certain primes. *Advances in Cryptology - CRYPTO '88 Proceedings*, 530–539.

Diffie, W., & Hellman, M. (1976). New directions in cryptography. *IEEE Transactions on Information Theory*, *22*(6), 644–654. https://doi.org/10.1109/TIT.1976.1055638

Doerner, J., Kondi, Y., Lee, E., & Shelat, A. (2018). Secure two-party threshold ecdsa from ecdsa assumptions. *2018 IEEE Symposium on Security and Privacy (SP)*, 980–997. https://doi.org/10.1109/SP.2018.00036

DuVander, A. (2023). 15 apis developers need to know [https://www.creativebloq.com/web-design/apis-developers-need-know-121518469].

ECRYPT II. (2023). Estream: The ecrypt stream cipher project [https://www.ecrypt.eu.org/stream/].

ElGamal, T. (1985). A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, *31*(4), 469–472.

F5, I. (2022). Restricting access with http basic authentication.

Faraway, J. (2005). *Linear models with r*. Chapman & Hall/CRC.

Faz-Hernández, A., Scott, S., Sullivan, N., Wahby, R. S., & Wood, C. A. (2021). *Hashing to Elliptic Curves* (Internet-Draft draft-irtf-cfrg-hash-to-curve-16) [Work in Progress]. Internet Engineering Task Force. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-hash-to-curve-16

Federal Office of Information Security. (2022). Documentation and analysis of the linux random number generator [https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/Studies/LinuxRNG/LinuxRNG_EN_V4_5.pdf?___blob=publicationFile&v=8].

Fiat, A., & Shamir, A. (1986). How to prove yourself: Practical solutions to identification and signature problems. *Advances in Cryptology—CRYPTO'86*, 186–194.

Fielding, R. T., & Reschke, J. (2014). Hypertext Transfer Protocol (HTTP/1.1): Authentication. https://doi.org/10.17487/RFC7235

Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures* (Doctoral dissertation). University of California, Irvine.

Franks, P. J., Hallam-Baker, P., Stewart, L. C., Hostetler, J. L., Lawrence, S., Leach, P. J., & Luotonen, A. (1999). Http authentication: Basic and digest access authentication. https://doi.org/10.17487/RFC2617

Galbraith, S. D., Paterson, K. G., & Smart, N. P. (2008). Pairings for cryptographers. *Discrete Applied Mathematics*, *156*(16), 3113–3121.

Gatlan, S. (2022). Fbi warns of mfa flaw used by state hackers for lateral movement.

Gatlan, S. (2023). T-mobile hacked to steal data of 37 million accounts in api data breach [https://www.bleepingcomputer.com/news/security/t-mobile-hacked-to-steal-data-of-37-million-accounts-in-api-data-breach/].

Ghosh, E., Ohrimenko, O., Papadopoulos, D., Tamassia, R., & Triandopoulos, N. (2016). Zero-knowledge accumulators and set algebra. *Proceedings, Part II, of the 22nd International Conference on Advances in Cryptology — ASIACRYPT 2016 - Volume 10032*, 67–100. https://doi.org/10.1007/978-3-662-53890-6_3

Google. (2021). Google transparency report.

Google Cloud. (2022). *Api security: Latest insights & key trends* (Technical Report). Google Cloud. https://cloud.google.com/solutions/api-security-latest-insights-and-key-trends

Google, Inc. (2021). The state of api economy 2021 report.

Google, Inc. (2022). Open id connect [https://developers.google.com/identity/openid-connect/openid-connect].

Gravitee.io. (2022). The future of apis: 7 trends you need to know. *3*.

Haböck, U., & Krenn, S. (2019). Breaking and fixing anonymous credentials for the cloud [https://ia.cr/2019/1061].

Hallam-Baker, P., Franks, P. J., Stewart, L. C., Sink, E. W., Hostetler, J. L., Leach, P. J., & Luotonen, A. (1997). An Extension to HTTP : Digest Access Authentication. https://doi.org/10.17487/RFC2069

Hamming, R. W. (1980). *Coding and information theory*. Prentice-Hall.

Handschuh, H. (2011). Sha-0, sha-1, sha-2 (secure hash algorithm). In H. C. A. van Tilborg & S. Jajodia (Eds.), *Encyclopedia of cryptography and security (2nd ed.)* (pp. 1190–1193). Springer. http://dblp.uni-trier.de/db/reference/crypt/crypt2011.html#Handschuh11b

Heisler, B. (2022). Criterion.rs goals [https://github.com/bheisler/criterion.rs].

Hevner, A., March, S., Park, S., & Ram, R. (2004). Design science in information systems research. *Journal of management information systems*, *21*(1), 9–30.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780.

Hölzl, M., Roland, M., Mir, O., & Mayrhofer, R. (2019). Disposable dynamic accumulators: Toward practical privacy-preserving mobile eids with scalable revocation. *International Journal of Information Security*. https://doi.org/10.1007/s10207-019-00458-7

Jaques, S., Lodder, M., & Montgomery, H. (2022). Allosaur: Accumulator with low-latency oblivious sublinear anonymous credential updates with revocations [https://eprint.iacr.org/2022/1362]. https://eprint.iacr.org/2022/1362

Jones, M., Bradley, J., & Sakimura, N. (2015). JSON Web Token (JWT). https://doi.org/10.17487/RFC7519

Jung, R. (2020). *Understanding and evolving the rust programming language*. https://doi.org/http://dx.doi.org/10.22028/D291-31946

Knuth, D. E. (1969). *The art of computer programming, volume 2 / seminumerical algorithms*. Addison-Wesley.

Korolov, M. (2021). Api attacks, breaches piling up.

Krishnaswamy, K. (2022). How platform ops teams should think about api strategy.

Kumar, R. (2014). *Research methodology: A step-by-step guide for beginners* (4th). SAGE Publications Ltd.

Kutner, M., Nachtsheim, C., Neter, J., & Li, W. (2005). *Applied linear statistical models*. McGraw-Hill/Irwin.

Lakshmanan, R. (2022). French electricity provider fined for storing users' passwords with weak md5 algorithm [https://thehackernews.com/2022/11/french-electricity-provider-fined-for.html?_m=3n.009a.2901.oj0ao43uy3.1vbq&m=1].

Lakshmanan, R. (2023). Millions of vehicles at risk: Api vulnerabilities uncovered in 16 major car brands [https://amp.thehackernews.com/thn/2023/01/millions-of-vehicles-at-risk-api.html].

Larsen, K., Lukyanenko, R., Mueller, R., Storey, V., Vander Meer, D., Parsons, J., & Hovorka, D. (2020). Validity in design science research.

Leto, D., & Developers, T. H. (2020). Attacking zcash for fun and profit [https://eprint.iacr.org/2020/627]. https://eprint.iacr.org/2020/627

Libert, B., Ling, S., Mouhartem, F., Nguyen, K., & Wang, H. (2021). Adaptive oblivious transfer with access control from lattice assumptions. *Theoretical Computer Science*, *891*, 210–229. https://doi.org/https://doi.org/10.1016/j.tcs.2021.09.001

Marsaglia, G. (2003). Xorshift rngs. *Journal of Statistical Software*, *08*. https://doi.org/10.18637/jss.v008.i14

Matsakis, N. D., & Klock II, F. S. (2014). The rust language. *ACM SIGAda Ada Letters*, *34*(3), 103–104.

Mir, O., Roland, M., & Mayrhofer, R. (2020). Damfa: Decentralized anonymous multi-factor authentication. *Proceedings of the 2nd ACM International Symposium on Blockchain and Secure Critical Infrastructure*, 10–19. https://doi.org/10.1145/3384943.3409417

Mishra, A. (2023). Top 10 fintech api security risks and challenges [https://www.valuebound.com/resources/blog/top-10-fintech-api-security-risks-and-challenges].

Montgomery, D. (2017). *Design and analysis of experiments*. John Wiley & Sons.

Moody, D., Peralta, R., Perlner, R., Regenscheid, A., Roginsky, A., & Chen, L. (2015). Report on pairing-based cryptography. *Journal of Research of the National Institute of Standards and Technology*, *120*, 11–27.

Mozilla. (2021). Http authentication.

Myers, J. G. (1997). Simple Authentication and Security Layer (SASL). https://doi.org/10.17487/RFC2222

National Institute of Standards and Technology. (2002). Secure hash standard.

Nguyen, L. (2005). Accumulators from bilinear pairings and applications to id-based ring signatures and group membership revocation. *IACR Cryptology ePrint Archive*, *2005*, 123.

Nielsen, H., Fielding, R. T., & Berners-Lee, T. (1996). Hypertext Transfer Protocol – HTTP/1.0. https://doi.org/10.17487/RFC1945

of Standards, N. I., & Technology. (2015). Secure hash standard (shs).

O'Neill, M., Zumerle, D., & D'Hoinne, J. (2017). How to build an effective api security strategy. (G00342236).

Oracle. (2022). Api management.

Organization for the Advancement of Structured Information Standards. (2005). *Security assertion markup language (saml) v2.0*.

Owasp. (2022). Password storage cheat sheet [https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html].

Ozdemir, A., Wahby, R. S., Whitehat, B., & Boneh, D. (2019). Scaling verifiable computation using efficient set accumulators [https://ia.cr/2019/1494].

Parecki, A. (2022). Oauth 2.0 specification.

Park, S. K., & Miller, K. W. (1988). Random number generators: Good ones are hard to find. *Communications of the ACM*, *31*(10).

Pointcheval, D., & Sanders, O. (2016). Short randomizable signatures, 111–126. https://doi.org/10.1007/978-3-319-29485-8_7

Pointcheval, D., & Sanders, O. (2018). Reassessing security of randomizable signatures. https://doi.org/10.1007/978-3-319-76953-0_17

Rackspace. (2022a). Authentication tokens.

Rackspace. (2022b). Get your credentials.

Reselman, B. (2020). An architect's guide to apis: Soap, rest, graphql, and grpc.

Riedl, R., & Robertson, T. (2010). A methodology for evaluating design alternatives. *Journal of the Association for Information Systems*, *11*(5), 1–31.

Roberts, A. (2021). https://research.nccgroup.com/2021/03/29/saml-xml-injection/

Sanders, O., & Traoré, J. (2020). Epid with malicious revocation [https://eprint.iacr.org/2020/1498]. https://eprint.iacr.org/2020/1498

Schnorr, C.-P. (1991). Efficient signature generation by smart cards. *Journal of Cryptology*, *4*(3), 161–174.

Schoenmakers, B. (2011). Oblivious transfer. In H. C. A. van Tilborg & S. Jajodia (Eds.), *Encyclopedia of cryptography and security* (pp. 884–885). Springer US. https://doi.org/10.1007/978-1-4419-5906-5_9

Security, S. (2021). State of api security report q3 2021.

Shannon, C. E. (1948). A mathematical theory of communication. *The Bell System Technical Journal*, *27*, 379–423, 623–656.

Sonnino, A., Al-Bassam, M., Bano, S., Meiklejohn, S., & Danezis, G. (2020). Coconut: Threshold issuance selective disclosure credentials with applications to distributed ledgers.

Source, A. O. (2022). The top 120 application programming interfaces topics on github.

Statista. (2022). Average number of characters for a password in the united states in 2021 [https://www.statista.com/statistics/1305713/average-character-length-of-a-password-us/#:~:text=Averagenumberofcharactersforapasswordin theU.S.2021&text=Approximatelysixoutoften,passwordswithover12characters.].

Stefanov, E., Shi, E., & Song, D. (2012). Policy-enhanced private set intersection: Sharing information while enforcing privacy policies. In M. Fischlin, J. Buchmann, & M. Manulis (Eds.), *Public key cryptography – pkc 2012* (pp. 413–430). Springer Berlin Heidelberg.

Sutskever, I., Vinyals, O., & Le, Q. (2014). Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems*, 3104–3112.

Sysoev, I. (2011). Nginx web server [https://www.nginx.com/].

Taujenis, R. (2022). How to safely store api keys in python [https://python.plainenglish.io.how-to-safely-store-your-api-keys-in-python-1dc5aadf93f9].

Team, S. G. S., & Switzerland, R. (n.d.). Specification of the identity mixer cryptographic library version 2 . 3 . 0 *.

Team Traceable. (2023). The business case for api security: Why api security? why now? [https://www.traceable.ai/blog-post/the-business-case-for-api-security].

Teseleanu, G. (2021). Lightweight swarm authentication [https://ia.cr/2021/1540].

U-prove cryptographic specification v1.1 revision 3. (2013).

Vaughan-Nichols, S. J. (2022a). Multifactor authentication is being targeted by hackers.

Vaughan-Nichols, S. J. (2022b). The okta mess is even worse than it appears.

Vitto, G., & Biryukov, A. (2020). Dynamic universal accumulator with batch update over bilinear groups [https://ia.cr/2020/777].

W3C. (2008). Extensible markup language (xml) 1.0 (fifth edition) [W3C Recommendation].

W3C. (2012). Xml schema definition language (xsd) 1.1 part 1: Structures [W3C Recommendation].

Wallarm Inc. (2022). Three new api exploits causes gitlab data privacy and availability issues [https://lab.wallarm.com/gitlab-security-issues-cve-2022-1352/].

Wierenga, K., & Lear, E. (2005). *A SASL Mechanism for SAML* (Internet-Draft draft-wierenga-ietf-sasl-saml-01) [Work in Progress]. Internet Engineering Task

Force. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-wierenga-ietf-sasl-saml-01

Wilson, Y., & Hingnikar, A. (2019). Evolution of identity. In *Solving identity management in modern applications: Demystifying oauth 2.0, openid connect, and saml 2.0* (pp. 19–28). Apress. https://doi.org/10.1007/978-1-4842-5095-2_3

Wireshark Organization. (2022). About wireshark [https://www.wireshark.org/].

Wray, J. (1993). Generic Security Service API : C-bindings. https://doi.org/10.17487/RFC1509

Wu, H., & Wang, F. (2014). A survey of noninteractive zero knowledge proof system and its applications. *TheScientificWorldJournal*, *2014*, 560484. https://doi.org/10.1155/2014/560484

Appendix A

Future Work

Oberon was created to be adaptable, as each of the following modules can be added if it supports cryptographic commitments or Schnorr proofs:

Revocation

At present, revocation in Oberon requires either deleting the server-side verification key or blacklisting the identifier or epoch timestamp, as the protocol does not have built-in support for revocation. However, techniques such as accumulators, as proposed in references such as RSA-based (Vitto & Biryukov, 2020), bilinear pairing based (Boneh et al., 2019), Merkle hash trees, or range proofs, can be used to implement revocation checks while maintaining privacy by not leaking a correlatable identifier. One approach is to embed a unique token ID into the token and present it as a ZKP of set membership to indicate whether the token is revoked or not. This requires adding ≈ 300 bytes to the token with accumulator-based proofs such as those in (Vitto & Biryukov, 2020). Range proofs, such as those generated by Bulletproofs (Bünz et al., 2017), are useful for indicating if the token timestamp falls within a valid or invalid range without revealing it, starting at 600 bytes. Neither of these additions adds much in proof size, but artifact design science iterations could explore the impact they add to computation. Range proofs can confirm the validity of a token without revealing its precise expiration date and time. Additionally, range proofs can perform other time-based verifications, such as confirming the token's issuance date and time, validating age based on birthdate, and ensuring that a value falls within a specific set.

Threshold

The cryptography that powers Oberon supports threshold based cryptography with minimal changes, but no research around its affects and benefits to Oberon were investigated. Threshold signing can be employed during token creation to create additional

security domains for the token signing key. BLS signatures (Boneh et al., 2022) are unique in that they are the only signature to date that facilitates one round signing methods. Oberon signature tokens are based on BLS signatures meaning their threshold requirements are identical.

While the cryptography used in Oberon can be adapted to support threshold-based cryptography, the effects and benefits of this approach on Oberon have not yet been studied. Threshold signing can be implemented during token creation by creating additional security domains for the token signing key. BLS signatures (Boneh et al., 2022) are unique–the only signature that enables one-round signing methods. Oberon signature tokens are based on BLS signatures, so their threshold requirements are the same.

Deanonymization

This work examined various methods for deanonymizing or linking token presentations. One method used was K-means clustering with two groups to detect if a presentation had been reused. However, the results showed that K-means was unable to accurately identify repeated presentations, regardless of the number of presentations analyzed from the same or different tokens or the CSPRNG used.

Potential areas for future investigation include utilizing machine learning and artificial intelligence techniques, such as long-short term neural networks, to assess if token presentations over time could possibly uncover a user's identity or establish correlation. Other techniques that analyze patterns may also be of value, as time is one of the few details that cannot be concealed by ZKPs. Correlation attack methods have not been extensively studied in relation to to ZKPs apart from a few unique deployments like Monero (Biryukov & Tikhomirov, 2019) and ZCash (Leto & Developers, 2020).

Appendix B

Oberon Performance Measurements

Function	Mean Time	Standard Deviation	95% Confidence Interval
Token generation	542 μs	23 μs	$\pm$ 593 ns
Token verification	5 ms	235 μs	$\pm$ 3 μs
Proof generation	3 ms	131 μs	$\pm$ 3 μs
Proof verification	9 ms	331 μs	$\pm$ 6 μs
Blinding factor	253 μs	12 μs	$\pm$ 272 ns

Table B1 *Client Oberon ID and Time Performance with 15000 Samples*

Function	Mean Time	Standard Deviation	95% Confidence Interval
Token generation	420 μs/punch	267 μs	$\pm$ 476 ns
Token verification	3.8 ms/punch	272 μs	$\pm$ 112 ns
Proof generation	3.7 ms/punch	270 μs	$\pm$ 298 ns
Proof verification	7.5 ms	301 μs	$\pm$ 198 ns

Table B2 *Client Oberon Counter Performance with 15000 Samples*

Function	**Mean Time**	**Standard Deviation**	**95% Confidence Interval**
Token generation	600 μs/permission	234 μs	$\pm$ 659 ns
Token verification	5 ms + 1 ms/permission	235 μs	$\pm$ 3 μs
Proof generation	3 ms + 1 ms/permission	131 μs	$\pm$ 6 μs
Proof verification	8 ms + 1 ms/permission	13 μs	$\pm$ 260 ns

Table B3 *Client Oberon-Z Performance with 15000 Samples*

Function	**Mean Time**	**Standard Deviation**	**95% Confidence Interval**
ID & Time token generation	750.5 μs	8 μs	$\pm$ 129.4 ns
ID & Time proof verification	13.5 ms	318 μs	$\pm$ 5 μs
Counter token generation	983 μs/punch	8 μs	131 ns
Counter proof verification	14 ms	313 μs	$\pm$ 5 μs
Z token generation	820 μs/permission	343 μs	$\pm$ 7 μs
Z proof verification	16 ms + 2.5 ms/permission	8 μs	$\pm$ 12 ns

Table B4 *Server Oberon Performance with 15000 Samples*

Item	**Size (Bytes)**
Secret Key	96
Public Key	288
Token	48
Proof	256

Table B5 *Oberon ID and Time size*

Item	Size (Bytes)
Secret Key	292
Public Key	354
Token	675 + 48/punch
Proof	639
Accumulator	256

Table B6 *Oberon Counter Size*

Item	Size (Bytes)
Secret Key	96 + 32/permission
Public Key	228 + 96/permission
Token	96
Proof	640 + 192/permission

Table B7 *Oberon-Z Size*

RNG	Average Proof Generation Time
XorShift	3.699 ms
Chacha	3.702 ms
Native OS	3.703 ms
F-value from ANOVA	2.149
p-value from ANOVA	0.117

Table B8 *Random number generator with Oberon*

RNG	**K-Means Error**	**Shannon**	**Arithmetic Mean**	**Monte Carlo Value for Pi**	**Serial Correlation Coefficient**
		≥ 7.9	≥ 126	$[3.1, 3.2]$	$[-0.004, 0.004]$
XorShift	5448984600	7.999384	126.72	3.161333	-0.0011753
Chacha	5448400400	7.999520	126.82	3.160058	-0.0011765
Native OS	5458848000	7.999564	126.86	3.160382	-0.0014526

Table B9 *Oberon statistical tests*

Method	Iterations	Client	Server	Payload (Bytes)
No Hash	0	10 ns	10 ns	13
SHA-256	1	54 ns	67 ns	32
SHA-384	1	218 ns	274 ns	48
SHA-512	1	219 ns	274 ns	64
SHA3-256	1	298 ns	322 ns	32
SHA3-384	1	302 ns	344 ns	48
SHA3-512	1	300 ns	336 ns	64
Shake-128	1	306 ns	352 ns	32
Shake-256	1	309 ns	363 ns	64
PBKDF2-HMAC-SHA256	320K	23ms	27 ms	95
PBKDF2-HMAC-SHA512	120K	48 ms	51 ms	95
BCrypt	10	47 ms	57 ms	72
BCrypt	11	94 ms	100 ms	72
BCrypt	12	187 ms	202 ms	72
SCrypt	n = 15 r = 8 p = 1	58 ms	70 ms	88
Argon2id	m = 37MB t = 3 p = 1	73 ms	88 ms	97

Table B10 *Password hashing*

API	Authentication Methods
Amazon AWS	Preshared Key symmetric cryptography
Microsoft Azure	Token-based OAuth
Rackspace Cloud	Token-based
Google Cloud	Token-based OAuth Public-key cryptography
Oracle Cloud	OAuth
IBM Cloud	Token-based

Table B11 *Cloud Providers API*

Authentication Methods

API	Authentication Methods
Texti.app	OAuth
ChatGPT	Token-based
Poised	Token-based
AssemblyAI	Token-based
StockAI	Token-based
Transaction.io	Token-based

Table B12 *AI Provider API*

Authentication Methods

API	Authentication Methods
VoltDB	Password-based
NuoDB	Token-based
CockroachDB	Password-based
DataStax	Password-based OAuth
Couchbase	Password-based OAuth

Table B13 *Database API*

Authentication Methods

API	Authentication Methods
LastPass Enterprise	Password-based OAuth
KeyCloak	Password-based OAuth
Hashicorp Vault	Password-based Token-based Public-key cryptography OAuth

Table B14 *Secrets Management API*

Authentication Methods

API	Authentication Methods
Twilio	Token-based
SendGrid	Token-based
SocketLabs	Token-based
MailChimp	Token-based
Pagerduty	Token-based

Table B15 *Sender API*

Authentication Methods

API	Authentication Methods
Openshift	OAuth Mutual TLS
Rancher	Token-based
Canonical Kubernetes	Password-based Token-based
Mirantis	OAuth

Table B16 *Container API*

Authentication Methods

API	Authentication Methods
Github	Token-based
Gitlab	Token-based OAuth
BitBucket	Password-based
SourceForge	Token-based

Table B17 *Code Repository API*

Authentication Methods

API	Authentication Methods
Twitter	Token-based
Fullcontact	Token-based
Facebook	OAuth
Dropbox	OAuth
Stripe	Token-based
Slack	Token-based OAuth
Vimeo	OAuth
TickTock	OAuth
Pinterest	Token-based OAuth
OpenSea	Token-based

Table B18 *Social API*

Authentication Methods

API	Authentication Methods
Wordpress	Password-based Token-based Public-key cryptography OAuth
Drupal	Password-based Token-based Public-key cryptography OAuth
Wix	OAuth

Table B19 *Website Builder API Authentication Methods*

Appendix C

Network Figures

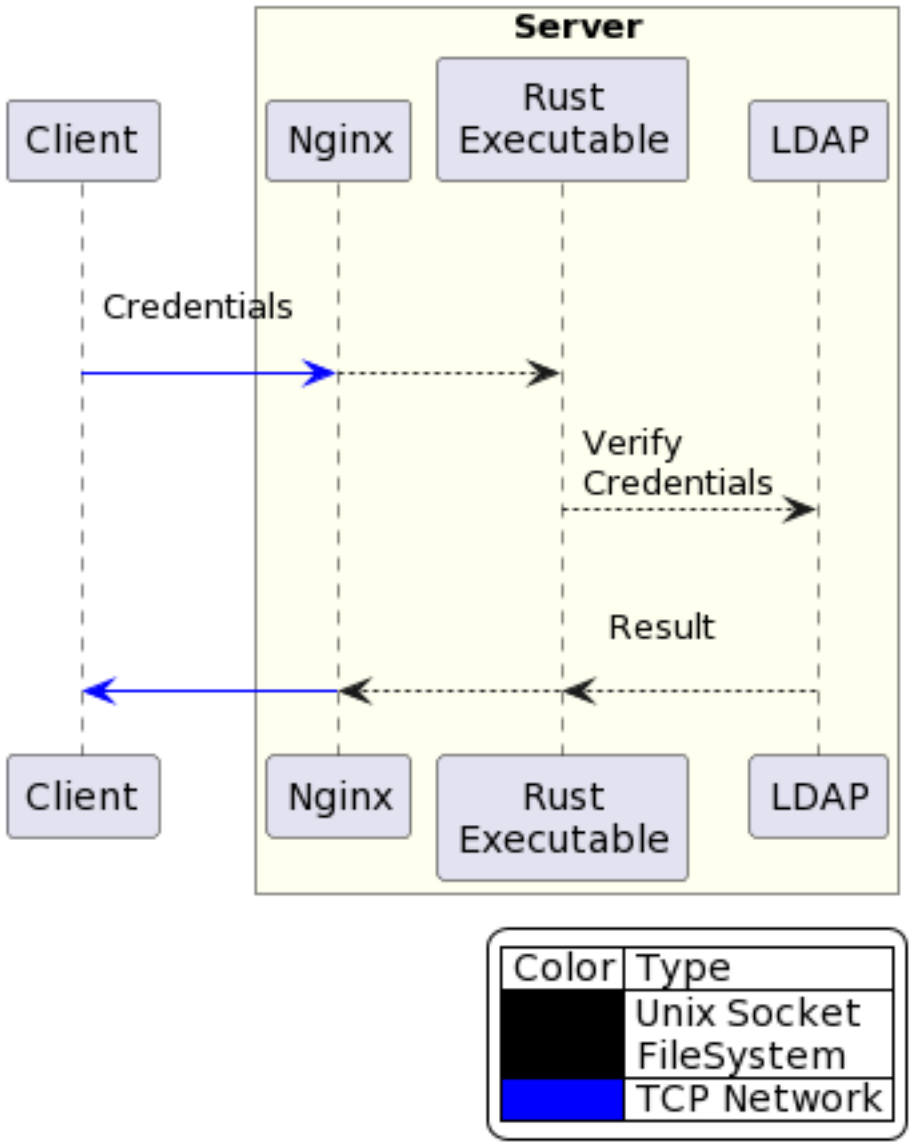

Figure C1 *Basic & Data Authentication Diagram*

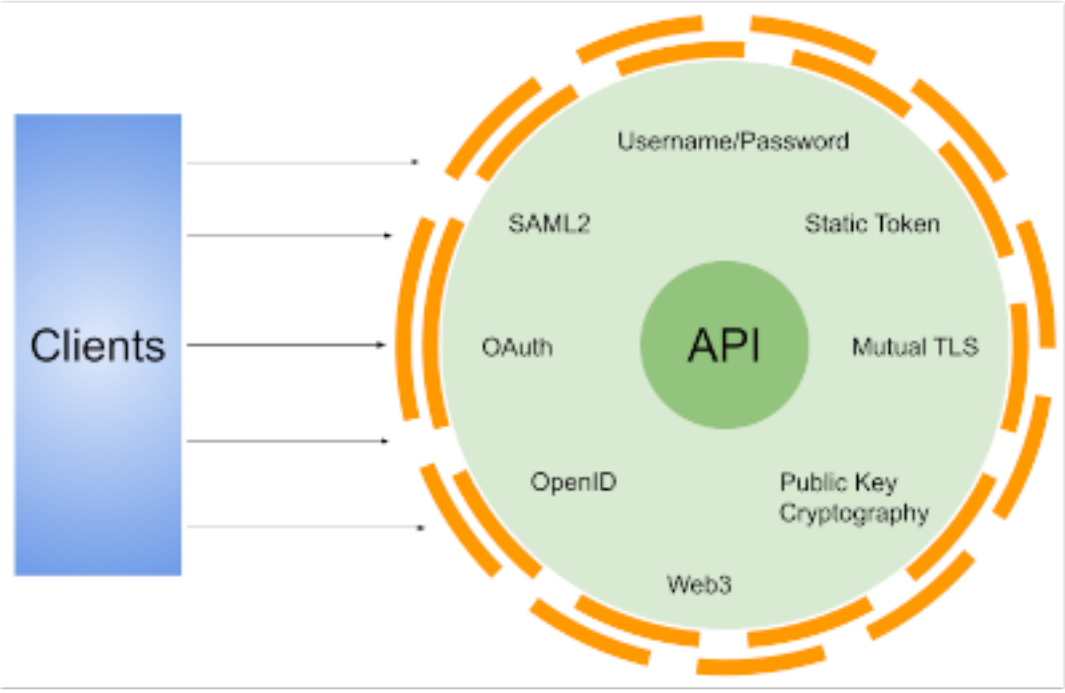

Figure C2 *API Authentication Methods*

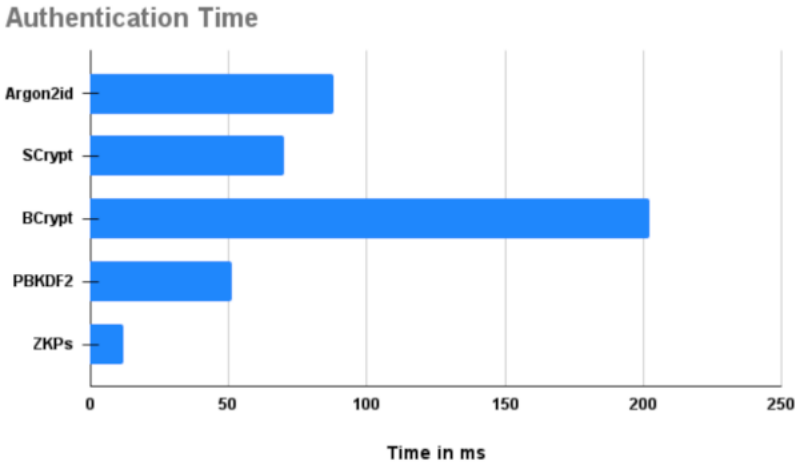

Figure C3 *ZKP vs Hashing Authentication Time*

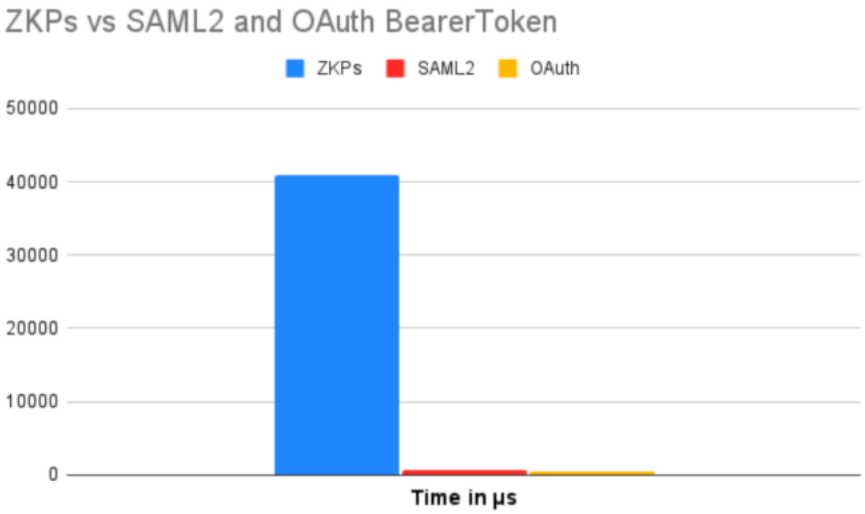

Figure C4 *ZKP vs OAuth/SAML Authorization Time*

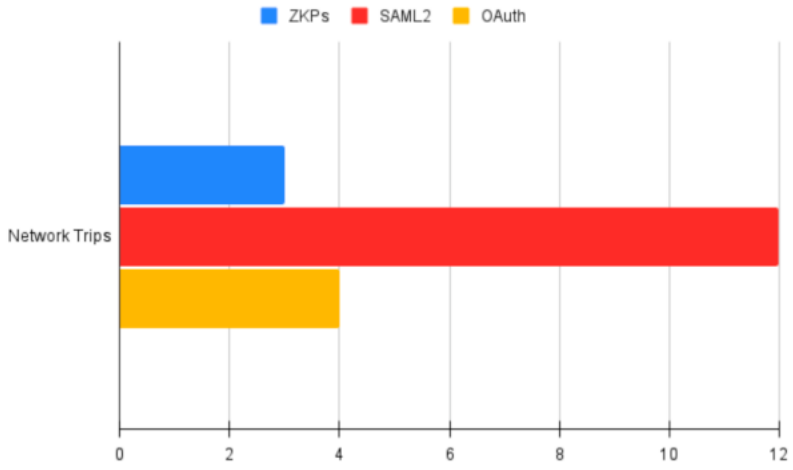

Figure C5 *ZKP vs OAuth/SAML Authorization worst case network trips*

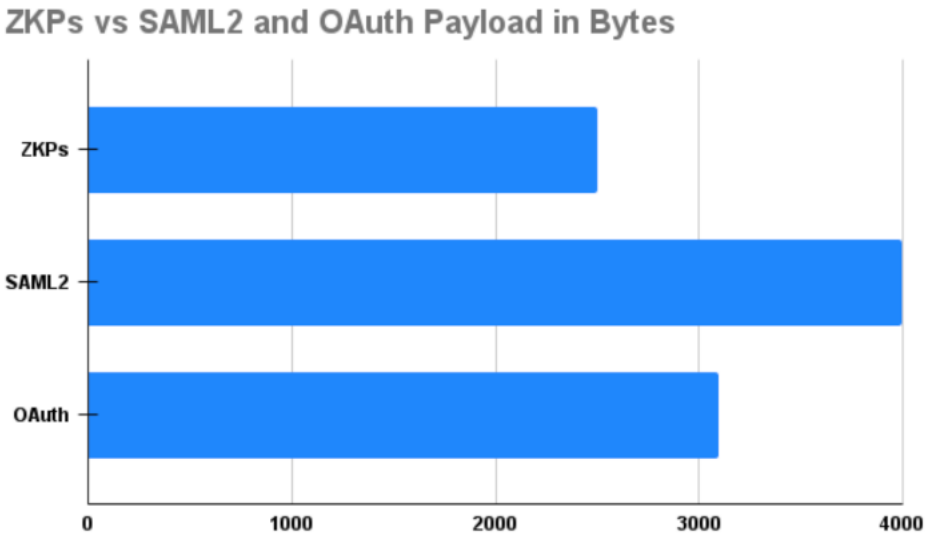

Figure C6 *ZKP vs OAuth/SAML Authorization payload in bytes*