Password-less two-factor authentication using scannable barcodes on a mobile device

Grant M. Callant II

PASSWORD-LESS TWO-FACTOR AUTHENTICATION USING SCANNABLE BARCODES ON A MOBILE DEVICE

A Thesis

Presented To

Eastern Washington University

Cheney, WA

In Partial Fulfillment of the Requirements

for the Degree

Master of Science in Computer Science

By

Grant M. Callant II

Spring 2021

Authorization to Submit Thesis

This Thesis of Grant Callant II, submitted for the degree of Master of Science with a major in Computer Science and entitled "**Password-less Two-Factor Authentication using Scannable Barcodes on a Mobile Device**," has been reviewed in final form. Permission, as indicated by the signatures and dates given below, is now granted to submit final copies for approval.

Major Advisor: ____ Date ________
Dr. Stu Steiner

Committee Member: ____________________________________ Date ________
Dr. Dan Tappan

Committee Member: ____________________________________ Date ________
Dr. Viktoria Taroudaki

Abstract

Currently, passwords are the default method used to authenticate users. As hardware continues to advance in speed, breaking these passwords becomes easier. The traditional solution to this problem is ever increasing password complexity and two-factor authentication. However, users become strained under overly complex login systems and often circumvent them. Two-factor authentication also adds to this complexity and many forms of two-factor authentication are inherently insecure. In answer to these problems, this project proposes a password-less multi-factor authentication system, which leverages the tried-and-proven existing technologies, asymmetric cryptography, digital signatures, and biometric authentication. Simulated user testing shows promising results, suggesting that registration can be completed in just over thirty seconds, and authentication in just over two seconds. An analysis of this project's possible attack vectors, preventative steps taken, and their solutions in potential future research are also discussed.

Chapter

List of Figures

Acronyms

2FA	Two-Factor Authentication
AES	Advanced Encryption Standard
ANSI	American National Standards Institute
API	Application Programming Interface
ATM	Automated Teller Machine
CA	Certificate Authority
CNAME	Canonical Name
CPU	Central Processing Unit
CSRF	Cross-Site Request Forgery
DNS	Domain Name System
DSA	Digital Signature Algorithim
ECDSA	Elliptical Curve Digital Signature Algorithm
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JWT	JSON Web Token
JSON	JavaScript Object Notation
MFA	Multi-factor Authentication
MITM	Man-In-The-Middle
MVC	Model View Controller
NIST	National Institute of Standards and Technology
ORM	Object Relational Mapping
OTP	One-time password
PEM	Privacy Enhanced Mail
PHP	PHP: Hypertext Preprocessor
PIN	Personal Identifcation Number
RSA	Rivest–Shamir–Adleman
SDK	Software Development Kit
SHA	Secure Hash Algorithm
SMS	Short Message Service
SPA	Single Page Application
TLS	Transport Layer Security
TOTP	Time-Based One-time Password
TTL	Time To Live
UI	User Interface
URL	Uniform Resource Locator
USB	Universal Serial Bus
XSS	Cross-Site Scripting

1 Introduction

1.1 Problem Statement

Cyber security is a constant battle between those who have information worth securing and those who wish to take that information. As technology has advanced, security has increased in complexity at an exponential rate. Early computers brought about a shift in security design, from little protection by restriction of physical access, to multi-user security and application layering. With the age of the Internet, any device connected is potentially vulnerable. One method to secure login vulnerabilities is with passwords.

In the realm of technology, passwords are implemented by computers, with a computer prompting a challenge that has to be correctly answered in text format. A computer can only parse text input and compare it to the exact correct response, meaning the correct text response to the challenge needs to be remembered precisely by the user. As technology continues to advance, passwords can be guessed, deciphered, or brute-forced, causing requirements for passwords including length, use of symbols, and variances in letter casing to increase. For several decades, it has been Department of Defense (DoD) and National Institute of Standards and Technology (NIST) policy, that a secure password requires [1] [2]:

- At least 9 characters in length
- At least one special character
- At least one number
- At least one uppercase letter
- At least one lowercase letter
- Be at least 4 characters different from last password
- Be changed every 90-150 days

Difficulty in remembering long strings of random text is not a new problem. In a 1956 psychological study, Miller [3] found the average person can only easily remember about seven digits or characters, plus or minus two. In a proceeding to the USENIX

Security Workshop, Klein [4] states users tend to choose a small subset of characters and numbers which are most memorable to them as passwords. These passwords tend to be less secure since they are not random, and patterns can be discovered and exploited from them. In an article titled “If your password is 123456, just make it hackme”, Vance [5] discusses an ever growing pool of common exposed user passwords.

Cheswick [6] in a 2013 study, found requiring frequent password changes tends to make users choose less secure, more memorable passwords while simply changing a few characters to make the password meet the minimum requirements. Adams et al. [7] add that these problems with passwords are compounded by the fact that many users frequently utilize multiple systems that require authentication, many requiring a separate set of authentication credentials. In response to these problems, many methods have been proposed; one of these proposals is Multi-factor Authentication (MFA).

Two-Factor Authentication (2FA), a subset of MFA, has been proposed as a possible solution for combating weak user passwords, but it brings about its own set of problems. Jover [8] found that Short Message Service (SMS) One-time passwords (OTPs) are vulnerable to being stolen in transit by insecure cellular lines and SIM swapping. Kogan et al. [9] in their proposal for T/Key discuss how Time-Based One-time Password (TOTP) requires the secret seed to be stored in the clear on the server, and can be exposed in the event of a server-wide attack. Drew [10] reports on a successful network breach on Lockheed Martin, a National Defense Contractor after Lockheed Martin’s secret seeds were stolen from their servers.

Hardware Universal Serial Bus (USB) tokens, such as YubiKey [11] not only require the need to purchase and carry additional nonstandard hardware, but are also vulnerable to a replay request attack, as demonstrated by Jacomme et al. [12] in their analysis of MFA protocol. Finally, adding another layer of complexity to an already complex security solution does nothing to ensure that users will comply with additional 2FA security measures.

According to Cheswick [6], users’ perception about the need for security greatly influences their choices concerning security. Adams et al. [7] suggests users are typically not fully aware of the risks of security exposure and information elicitation, and in some cases, users elude security policies consciously out of consternation for what they perceive to be a needless system. Sasse et al. [13] explain, if users believe security or password policies are overly complex or do not fully understand the policies, they are more likely to circumvent them. These problems are transparent in the growing threat of phishing, browser drive-by, Cross-Site Request Forgery (CSRF), Cross-Site Scripting (XSS), and Man-In-The-Middle (MITM) attacks, leaving a user confused in how to best protect themselves. A system that provides a standard level of security and is simple to use and understand might allow for greater user compliance.

1.2 Goals

This paper proposes a novel approach for these dilemmas, enabling a password-less login system using asymmetric keys, digital signatures, and biometrics to provide MFA using a mobile device. This project seeks to create a system that is more secure or at least as secure as a traditional user-password authentication system utilizing 2FA. The system must be convenient and understandable enough for the average user to easily employ. Establishing a workable prototype to demonstrate the level of security and the ease of user interaction will be sufficient. Therefore this project's goals are the following:

1. Develop an Apple mobile application that will create and securely store the user's private keys. The application will also be responsible for communicating with the server to provide authentication.
2. Setup an Apache web server to demonstrate authentication and authorization using PHP: Hypertext Preprocessor (PHP) and the Laravel framework to provide back-end functionality.
3. Create a Single Page Application (SPA) front-end, what will be viewable to the user, using Vue.js and Vue Router to prevent unauthorized access.

2 Background

This section explores the multiple pieces of technology that are used in this project. It is split into four distinct parts with each technology subdivided for reader convenience as follows:

2.1 Discusses how authentication is defined and how the traditional password process works. Asymmetric cryptography is explained along with digital signatures and certificates and the roles public and private keys play in identity verification. Cryptographic hashing is included here, since it is essential in providing repudiation when validating signatures. Biometric authentication is discussed and its importance in providing a very high level of identity assurance.

2.2 Explains MFA along with the two most commonly used methods, SMS and TOTP including their benefits and drawbacks.

2.3 Defines authorization and how it differs from authentication. Some common methods of providing authorization such as OAuth and JSON Web Tokens (JWTs) are discussed.

2.4 Discusses several components utilized in this project, along with the reasoning for each component choice.

2.1 Authentication

Authentication is defined as: a way to prove with reasonable certainty that you are the person you declare yourself to be. Outside of the technology domain, this is typically done with an identification card issued by a governmental body. In the state of Washington, an example would be a State Issued Driver's License. The Department of Licensing issues licenses on the contingency that they are able to verify your identity from a list of approved identification documents [14]. In the technology realm, authentication can be performed in a number of ways, including: passwords, digital signatures and certificates, and biometrically. Each of these methods has their own advantages and disadvantages.

2.1.1 Passwords

A password is a form of symmetric cryptography where two parties agree on a secret word or phrase (also called a key). Assuming perfect secrecy of the key between the two parties, when identity is challenged, mere knowledge of this key would provide reasonable certainty of identity. The NIST provides guidelines for the password exchange process outlined in Grassi et al. [15]. The process can be summarized as follows. When a user attempts to login to a service, the service attempts to gain assurance of the user's identity. The service sends a challenge request to the user for their password. When the user inputs their password in response to the challenge request, the service checks to see if the received password matches the stored password for the user. Because it is assumed the password is only known to the user and the service, the password provides reasonable certainty to the user's identity, and the service is able to authenticate the user.

Since the service typically initiates the challenge, the service needs to store the password in order to compare it to the received input. Since passwords can be leaked or stolen from an attack, they need to be stored in a way that keeps them secret. Steven et al. [16] of Open Web Application Security Project cite best practices regarding password storage. Best practices require not storing the actual password itself, but a hashed version using a one way cryptographic hashing algorithm, along with a random string called a salt.

2.1.2 Cryptographic Hashing

A hashing function is a way to transform data into a fixed-length output, called a message digest, that is significantly different than the input, such that repeated hashes of the same input will always produce the same digest, but even small changes to the input will result in a significantly differing digest. Hashing provides integrity that the data has not been changed since it was sent. This repudiation hashing provides is essential when verifying signatures, otherwise information could be signed and changed by someone other than the original signer. Since the same input will always produce the same digest, a party can send data along with a hash of the data. The receiving party can recompute the hash on the received data and compare it to the sent hash.

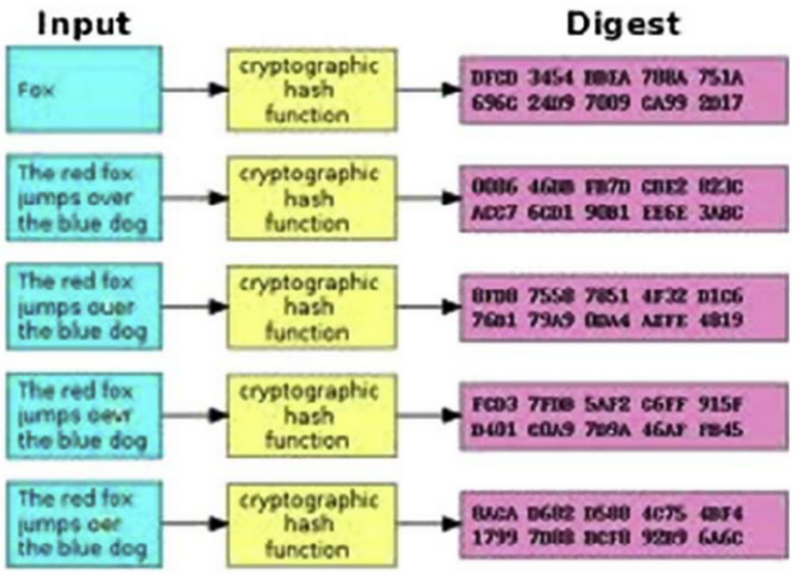

Input
Digest
Fox
cryptographic hash function
DFCD 3454 BBEA 788A 751A 696C 24D9 7009 CA99 2D17
The red fox jumps over the blue dog
cryptographic hash function
0086 46BB FB7D CBE2 823C ACC7 6CD1 90B1 EE6E 3ABC
The red fox jumps ouer the blue dog
cryptographic hash function
8FD8 7558 7851 4F32 D1C6 76B1 79A9 0DA4 AEFE 4819
The red fox jumps oevr the blue dog
cryptographic hash function
FCD3 7FDB 5AF2 C6FF 915F D401 C0A9 7D9A 46AF FB45
The red fox jumps oer the blue dog
cryptographic hash function
8ACA D682 D588 4C75 4BF4 1799 7D88 BCF8 92B9 6A6C

Fig. 1. Data Hash Example. Reprinted from Security Controls Evaluation, Testing, and Assessment Handbook (Second Edition), L. Johnson, Chapter 11 - Security component fundamentals for assessment, Page 527, Copyright 2020, with permission from Elsevier [17]

Figure 1 illustrates a hash function applied to several word examples, where even a single letter change produces a vastly different digest. Hashing algorithms differ in cryptographic strength, block size, and the length of the resulting digest. While the figure does not state the hashing algorithm used, it is important to note that the length of each digest is unchanged. When applied to the same hash algorithm, regardless of the input, the digest length will always remain constant.

While there are several variations of Secure Hash Algorithm (SHA), two frequently used SHA-2 variants, SHA-256 and SHA-512 differ in their block size and digest length. SHA-256 has 512-bit blocks with a 256 bit digest length, while SHA-512 has 1024-bit blocks and a digest length of 512 bits. Hash functions are subject to a phenomenon called collision, where it is theoretically possible to have two different inputs output the same exact message digest. Andress [18] states that larger block sizes and longer digest messages provide increased protection from collision.

2.1.3 Asymmetric Cryptography

Unlike passwords, which are a form of symmetric cryptography and relies on both parties having a secret in common, asymmetric cryptography relies on each party having a private secret key, which is not shared. Instead, a public key is mathematically created from the secret key, such that the private key cannot be discovered from the encrypted data or public key without an unreasonable amount of time and effort. This is known as a key pair.

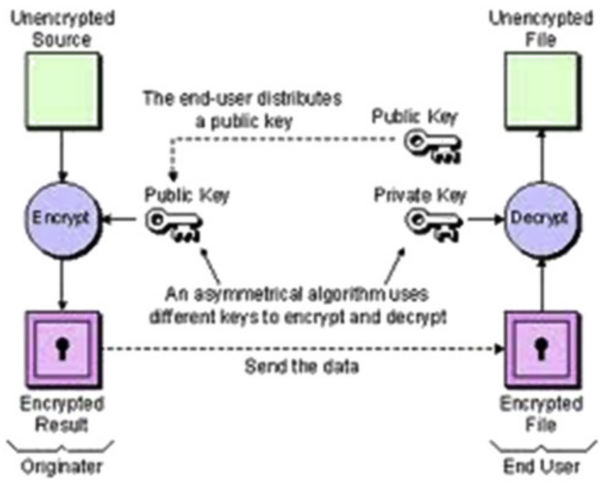


Fig. 2. Asymmetric key encryption. Reprinted from Security Controls Evaluation, Testing, and Assessment Handbook (Second Edition), L. Johnson, Chapter 11 - Security component fundamentals for assessment, Page 525, Copyright 2020, with permission from Elsevier. [17]

As shown in Figure 2, the public key can be used to encrypt data and is shared in the open, while the private key remains secret and is the only means to decrypt information encrypted with the corresponding public key. While asymmetric cryptography can be used to encrypt and decrypt data, it can also be used to sign and verify data with digital signatures.

2.1.4 Certificates and Signatures

Asymmetric Cryptography can also be used to verify the sender of information and that the data has not been tampered with. Rather than using the public key to encrypt, the data is first hashed and then encrypted with the private key. This ensures that anyone with the public key can not only decrypt the data, but can also re-compute the digest of the encrypted data, verifying that the data has not been tampered with and was sent by the holder of the private key. Assuming perfect secrecy of the private key, one can deduce the identity of the sender with reasonable certainty. This is known as a digital signature.

Keys can be contained inside a document called a digital certificate. The digital certificate contains information about the key and its validity. The most common use of these certificates is in secured websites (such as a banking site). The certificate for a site contains its public key and the certificate is signed by its private key. Anyone who visits the site can verify the authenticity of the certificate with the public key contained in the certificate.

2.1.5 Biometrics

Biometric authentication can include any biological attribute that uniquely identifies a user. The most common use of this is fingerprints, which police organizations have been using for decades. Other examples could include retinal scans, facial attributes, voice identification, and heart rhythms. This information must be collected and stored at the time of user registration. Authentication can be performed by comparing presented user biometric data against the previously stored information for that user. Kuhn et al. [19] state all biometric systems require hardware to capture the user information, but can provide a very high level of identity assurance, since the authentication mechanism is based on unique user information that is difficult to duplicate or imitate.

2.2 Multi-Factor Authentication

MFA is a security by redundancy model, where multiple pieces of identification or phases are required for authentication. The process requires all of the pieces of identification or all of the phases to pass else authentication fails. Jover [8] states a common everyday example is an Automated Teller Machine (ATM), where the user has their bank card as the first authenticator, and a Personal Identifcation Number (PIN) only the user knows as the second authenticator. Without both of these authenticators, usage of the ATM is restricted. It also includes the most common implementation of MFA, 2FA which requires exactly two authenticators.

The 2FA process typically combines a password with another form of identity such as: a PIN; an OTP sent through SMS, email, hardware device, or mobile application; an approval notification sent on a trusted device; or a USB token. Jacomme [12] states the additional authentication step(s) may prevent an attacker from gaining access to an otherwise compromised account where the attacker already has knowledge of the user's password or possesses another authenticator.

2.2.1 Short Message Service

The SMS MFA process differs depending on the providing service, but generally the process consists of the user first successfully logging into the service with their username and password. The service then generates and sends an SMS message containing an OTP to the user on their mobile phone. The user then enters the OTP in the message back into the service. The service compares the received OTP with the OTP that was sent. If this second process succeeds, then the user is successfully authenticated and is allowed access to the service [12]. This process can also be done over email, and the approach is very similar, except the OTP is sent to the users' email address rather than the users' mobile phone.

The primary advantage of the OTP over SMS or email approach is in its simplicity to set up and administer with little overhead. Since many services use this approach, it is also familiar to most users. Jover [8] states the primary disadvantage with the SMS MFA approach is that it requires an active cellular network connection and is reasoned generally insecure due to its susceptibility to multiple attack vectors. While email could potentially be a greater risk target, since a compromised account could expose a users' entire online identity, in a 2019 study Mirian et al. [20] found that email accounts protected by Google's 2-step 2FA were difficult if not impossible for hackers to breach without social engineering. Therefore in some cases, OTP sent securely over email, such as accounts protected with Google's 2-step 2FA would not be subject to the same security liabilities as OTP sent over SMS. However, email that is secured with SMS MFA could potentially be subject to the same security issues as other services using SMS MFA, and a compromised email account would allow an attacker to gain access to OTP sent over email.

2.2.2 Time-Based One-Time Password

TOTPs as defined by M'Raihi et al. [21] in RFC-6238, operate by using a pre-defined pseudo-randomly generated secret seed that is stored on both the authenticating service and a user-controlled device. The device can be a specialized hardware device, such as the Rivest–Shamir–Adleman (RSA) SecurID [22] or an application on a mobile device, such as Google Authenticator. The six digit TOTP is continually generated by applying the secret seed to a Hash-based Message Authentication Code (HMAC) function using the seed and time as parameters. In practice, the length of time the TOTP is valid is thirty seconds and is regenerated at this end of this period. The user enters the six digit TOTP displayed on his or her device into the requesting service. The requesting service then separately recreates a TOTP from the seed stored on the service and compares it with the received TOTP.

Kogan et al. [9] state the primary advantage with this approach is that there is no network connectivity needed to generate these codes, since the service and the device can compute them separately using the same pre-defined secret seed. This means that that the TOTP approach is not subject to the same problems as the SMS approach where the code can be stolen in transit.

Kogan et al. [9] explain the primary disadvantage is that the secret seeds must be stored in the clear on the requesting service in order for the service to verify the users' TOTP. Were an attacker to gain unauthorized access to the service, the attacker would be able to steal the secret seed of every registered user and generate valid codes without limit. A very public example of this attack type came to light after Lockheed Martin's network was compromised, as reported by Drew [10]. Despite this, Jover [8] claims, the TOTP method with a mobile application is the most secure way of performing MFA without requiring additional nonstandard hardware.

2.3 Authorization

Rather than providing identity verification, as authentication does, authorization is a way of ensuring clearance or privilege to access a system or service. It is possible for a user to be authenticated but not have authorization to access a resource or vice versa. An everyday example would the difference between an identification badge and a key. The identification badge may prove identity, but it will not open a locked door. Conversely, the possession of a key that will open a locked door does not prove identity, as the key may have been lost or stolen. Most systems have protocols in place to both authenticate and authorize users.

In a single user system, the authenticated owner or primary user of the system would have administrator or root access to all parts of the system. Most modern systems however provide multiple services to many users. Properly configured services should allow for authenticated users to access and configure their own data within a service, while preventing any access from users without permission. The exact method for verifying and enforcing authorization largely depends on the system but according to Harrison et al. [23], authorization can generally be reduced to a relational matrix or access control list (ACL), which maps boolean (true or false) values to users and the areas each user is allowed to access.

2.3.1 Tokens

JWT is a standard for storing encoded JavaScript Object Notation (JSON) [24] claims in a token, and is widely used for granting and proving authorization. JWTs store information about the service or entity which issued the token, the user the token was issued for, the time the token was issued, and the token's expiration time. JWTs can also be digitally signed or encrypted by their issuer, which allows verification that the token was not tampered with since it was issued. Tokens are typically stored within the browser in local or local storage, but can also be stored in a cookie. Jones et al. [25] define the process for exchanging tokes for authorization in RFC-7519. When used for authorization, tokens are typically sent in the Hypertext Transfer Protocol (HTTP) request header in the format: “Authorization: Bearer ⟨token string⟩.”

2.3.2 OAuth

OAuth [26] is an open standard of protocols designed to allow delegate authorization. This means that a service may not be able to directly authenticate a user, but if another service has already authenticated that user, the service may vouch for the users’ identity. A common example of this is using a Facebook account to log into Website A. The user does not have an account established with Website A, but does have an account with Facebook. The user can click the Connect with Facebook button, which sends information about the user to Website A. Depending on Website A’s information request, the information contains at minimum, an authorization token, and a unique user identifier that can be used to identify the user on Website A, but never the user’s Facebook password.

2.4 Technology

Since this project focuses on creating an MFA prototype using asymmetric keys and scannable barcodes, several pieces of technology are required. The following section summarizes the technologies used for this project and the reasons each was chosen. The section is divided into mobile device components and server components for reader convenience.

2.4.1 Mobile Device Components

- The Apple iPhone X is used because of its availability, its ability to secure private keys in the Secure Enclave, and its biometric authentication technology, which allows for password-less login.
- Xcode 12 is used to create a native Apple application that allows access to all the hardware components of the Apple iPhone X.

- Access to the application is secured using biometric authentication or with a legacy PIN passcode.
- BCryptSwift, a Swift implementation of bcrypt is used to securely store the legacy passcode in the keychain. Bcrypt provides additional protection against brute-force and dictionary attacks.
- The Secure Enclave is available to specific Apple products and is used to secure the user's biometric information and the user's private key.
- QR Codes are a type of two-dimensional barcodes which can be scanned from every direction in 360 degrees by virtually every smartphone brand and model with no special hardware or software. QR Codes are used in this project to render a nonce that is scanned by the user with a smartphone.
- 256 bit Elliptical Curve Digital Signature Algorithm (ECDSA) keys are the only key type allowed by the Secure Enclave. ECDSA keys are used in this project to digitally sign the user's payload during authentication to provide identity assurance.

2.4.2 Server Components

- Windows 10 was used as the server operating system due to availability.
- Apache is used to host the server due to its familiarity and ease of setup.
- Transport Layer Security (TLS) 1.3 is used to secure connections to the server. TLS is used due to industry standards for secure server connections and TLS 1.2 is the minimum accepted standard by Apple's Application Transport Security.
- Ngrok is a freemium service that allows a developer to create tunnels to a locally hosted server to expose the server to the Internet. Ngrok is used due to Apple's Application Transport Security requirements for a secured TLS connection with a certificate signed by a trusted root Certificate Authority (CA).
- Let's Encrypt is a free service provided by the Internet Security Research Group that allows anyone to verify a domain name and obtain a free certificate signed by the trusted root CA Let's Encrypt. Let's Encrypt was used on account of its simplicity, cost and Apple's Application Transport Security requirements for a secured TLS connection with a certificate signed by a trusted root CA.
- MySQL is used as the data storage driver due to the need for multiple concurrent reads and writes during authentication. SQLite was originally chosen for its simplicity, however according to SQLite [27], it only supports one write operation with no concurrency. This lack of concurrency frequently caused login to fail due to multiple concurrent writes performed during authentication and authorization.
- Laravel is a PHP developer framework. Laravel is used in this project on account of its Object Relational Mapping (ORM), Eloquent, and Laravel's Model View Controller (MVC) feature set, which allows for rapid development of a front-end agnostic Application Programming Interface (API).

- Laravel Passport, a first-party Laravel library is used on account of its ease of integration with Laravel. Passport is used as the API authorization driver.
- Axios is an HTTP requests client. Axios was chosen due to its integration with Laravel. Axios is used to perform XMLHttpRequests on the user's browser to gain an authorization token after authentication.
- Vue.js is a JavaScript framework that is included by default with Laravel's User Interface (UI) package. Vue.js is used due to its simplicity and ease of integration with Laravel. Vue.js is used in this project to create the front-end SPA that the user interacts with.
- Vue-qrcode is a Vue.js component library that is used to render QR Codes to the front-end SPA. Vue-qrcode is used due to its ease of integration with Vue.js.
- Vue Router is a first-party Vue library that allows for JavaScript routing. Vue Router was used to perform routing for the front-end SPA and to secure protected routes from unauthorized requests.
- Pusher Channels is a freemium service that offers a WebSockets broadcast driver. Pusher Channels is used to broadcast successful authentication events to desktop clients awaiting authorization. Pusher Channels is used due to its simplicity and ease of integration with Laravel.
- Laravel Echo is a first-party Laravel JavaScript library that allows for listening to broadcast events. Laravel Echo is used to listen for successful authentication broadcasts from Pusher Channels on desktop clients awaiting authorization. Laravel Echo is used due to its ease of integration with Laravel.
- OpenSSL is used in this project to verify hashes and signatures sent by the user's mobile device during authentication. OpenSSL is used due to its widespread use in Internet servers and on account of OpenSSL's integration with the PHP library.

3 Architectural Overview

This section provides both a high-level overview of this project's system and an in-depth description of the system's components. This section is divided as follows:

3.1 Contains a high-level overview of the authentication, authorization, and logout process.

3.2 Contains an in-depth description of the components used in this project and the way they were utilized.

3.3 Contains a detailed explanation of the desktop UI and several example figures.

3.4 Contains a detailed explanation of the mobile device UI and several example figures.

3.1 System Overview

The system is presented in logically subdivided components for reader convenience. The authentication process used for this project works as follows.

3.1.1 Registration

This project assumes the user has the application already installed on his or her device. Figure 3 illustrates the user registration process. **1.** The user opens the application on their mobile device. **2.** The user fills out the registration form on the mobile device. **3.** A keypair is generated by the Secure Enclave during registration. **4.** The public key, user's name, phone number, and email address is sent to the server. The server creates a new user and saves the user's information into the database. **5** The server sends a JSON response with the created user's id and an HTTP 201 CREATED status to the phone upon successful user creation, or an error code otherwise.

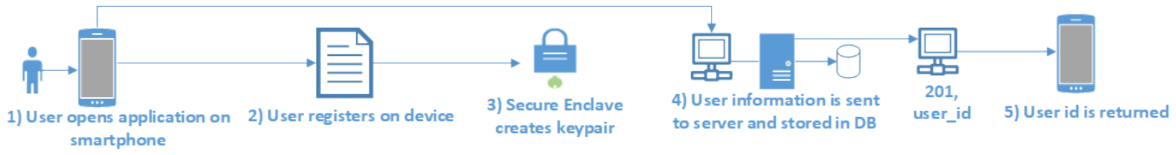

1) User opens application on smartphone
2) User registers on device
3) Secure Enclave creates keypair
4) User information is sent to server and stored in DB
201, user_id
5) User id is returned

Fig. 3. User registration process

3.1.2 Authentication

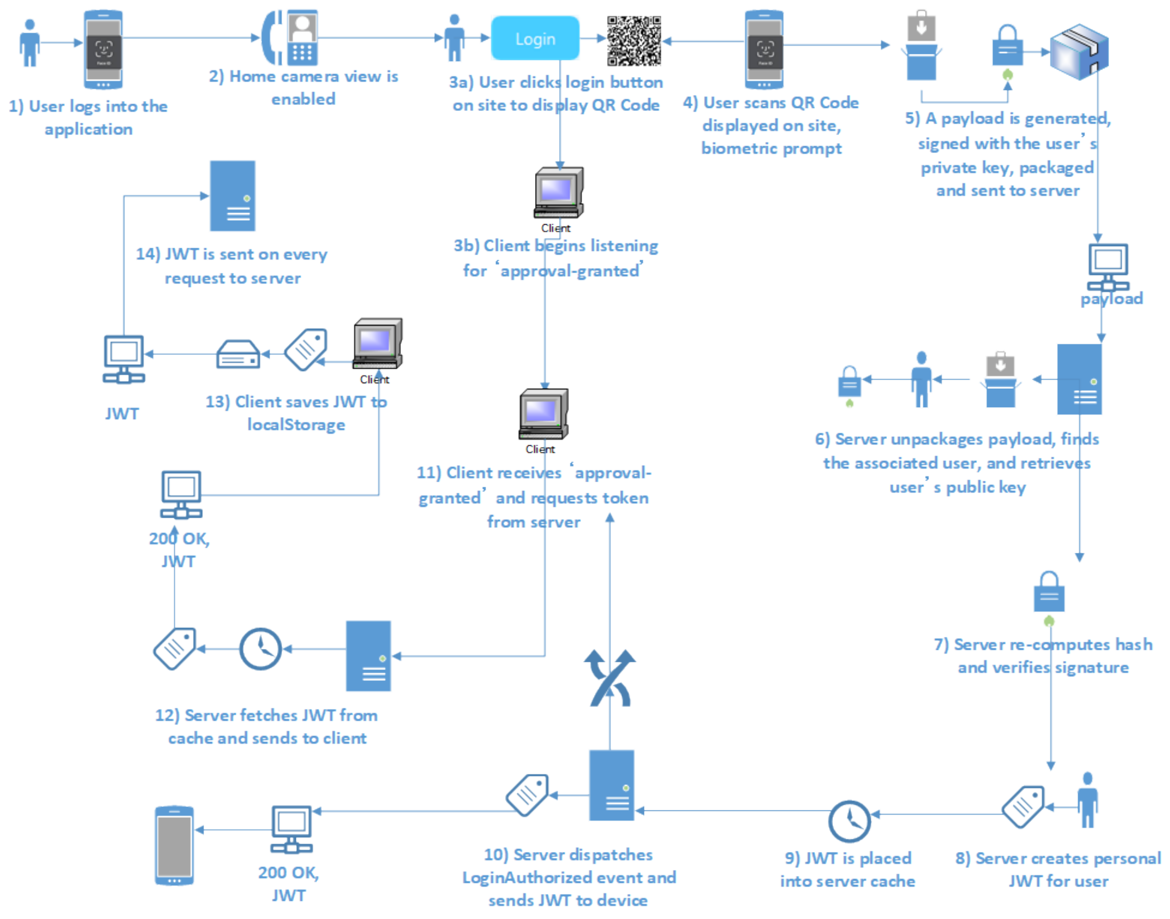

1) User logs into the application
2) Home camera view is enabled
Login
3a) User clicks login button on site to display QR Code
4) User scans QR Code displayed on site, biometric prompt
5) A payload is generated, signed with the user's private key, packaged and sent to server
Client
3b) Client begins listening for 'approval-granted'
payload
14) JWT is sent on every request to server
JWT
Client
13) Client saves JWT to localStorage
Client
11) Client receives 'approval-granted' and requests token from server
6) Server unpackages payload, finds the associated user, and retrieves user's public key
200 OK, JWT
7) Server re-computes hash and verifies signature
12) Server fetches JWT from cache and sends to client
200 OK, JWT
10) Server dispatches LoginAuthorized event and sends JWT to device
9) JWT is placed into server cache
8) Server creates personal JWT for user

Fig. 4. Authentication process

Figure 4 illustrates the authentication process. **1.** If the user is already registered, the user simply logs into the application. **2.** After registration and login, the user's device is redirected to the Home camera view screen. **3a.** The user clicks on the Login button on the website SPA. The SPA makes an Axios API request to the /randomBytes endpoint. A pseudo-random cryptographically secure 64 byte nonce is created and base64 string encoded and is returned to the SPA. A QR Code is generated and rendered in the SPA with the nonce string. **3b.** The SPA begins listening on a channel identified by the nonce for an 'approval-granted' broadcast.

4. The user scans the QR Code on the SPA with the Home camera view on the mobile device. A JSON encoded bundle is created on the device containing the user's id, and the nonce contained in the scanned data from the QR Code. An SHA-512 hash is created from the bundle. The device then sends a request to the Secure Enclave to sign the bundle with the private key on the device. The user is prompted for biometric authentication. If the biometric authentication fails, the process is halted. **5.** Otherwise, the bundle is signed using American National Standards Institute (ANSI) X9.62 [28] with an SHA-512 hash. Then, the bundle, the hash, and the signature are enclosed in a payload. The payload is base64 encoded and is sent to the server in an API request to the /login endpoint in the request body.

6. Once the server receives an authentication request with a payload it then unwraps the payload and base64 decodes, and JSON decodes the bundle. The server looks for a user which matches the user id contained in the bundle. If a user is found, a user object is retrieved using Eloquent ORM, and the user's public key is retrieved from the database. **7.** The server re-computes the SHA-512 hash on the bundle and compares it with the hash contained in the payload. If the values match, the server verifies a signature is present in the bundle, base64 decodes it, and uses PHP OpenSSL verify [29] to validate the signature against the user's public key. If a user is not found, the hash computed does not match the hash sent, a signature is not found, or the signature is not verified, then an HTTP 401 unauthorized code is sent back to the requester.

8. After the signature is verified a personal bearer JWT is created that is unique to the user from the user object. The JWT is given an expiration time of one week from the date of creation. **9.** The JWT is placed into the server cache, using the nonce string as the key and the JWT as the value, with a Time To Live (TTL) of two seconds. If the JWT is not retrieved during this period, it is removed from the cache. **10.** The server then dispatches a LoginAuthorized event, which broadcasts 'approval-granted' on the channel id of the nonce, and returns a JSON HTTP 200 OK status along with the created JWT to the mobile device.

11. Once the listening SPA receives the 'approval-granted' broadcast it sends an Axios API request to the /login/confirm endpoint along with the nonce in the request body. **12.** The server checks the cache for a key that matches the received nonce, if none is found, an HTTP 401 unauthorized status is returned to the requester. Otherwise, the JWT is retrieved from the cache, and the cache entry is deleted to prevent timed-replay attacks. The retrieved JWT is sent back to the requesting client along with an HTTP 200 OK status. **13.** The Axios client receives the JWT and saves it into the browser localStorage. **14.** The Axios client fetches the JWT, if it is present, and sends it in the header on every request to the server. Finally, the user is redirected to the home page. The user's browser client is now authorized to make any resource requests, since the JWT is saved in localStorage and Axios will automatically send the JWT in the header on every request.

3.1.3 Logout

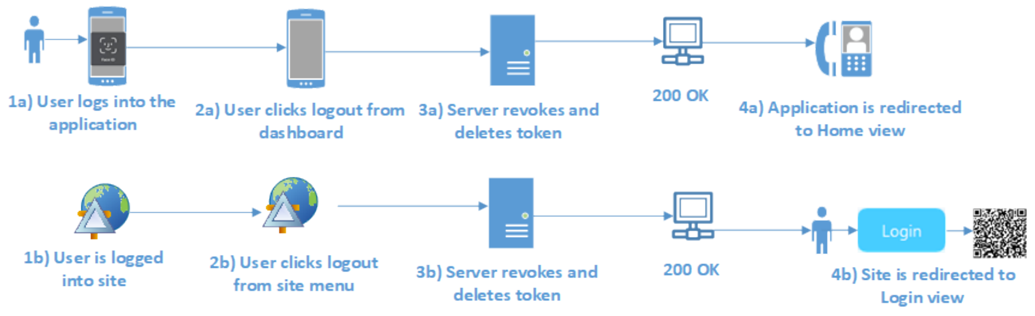

1a) User logs into the application
2a) User clicks logout from dashboard
3a) Server revokes and deletes token
200 OK
4a) Application is redirected to Home view
1b) User is logged into site
2b) User clicks logout from site menu
3b) Server revokes and deletes token
200 OK
Login
4b) Site is redirected to Login view

Fig. 5. Logout process

Since the entire authentication method is stateless, the user will only be logged out when the JWT expires or the user manually logs out. Browser localStorage is persisted across tabs, windows, and when the browser is closed. A /logout API endpoint was defined, which can be accessed from the user's mobile device or directly in the browser. No logout hook was defined, so the browser and mobile device are not automatically notified when the other device logs out, although the logout process is nearly identical for either device. Figure 5 illustrates this logout process. When logging out via either device, the /logout API is called, which revokes and deletes the user's JWT from the server, making any further requests return unauthorized. When logging out from the browser, the JWT is removed from browser localStrage, and the user is redirected back to the login page. When logging out from the mobile device, the user is reminded to close the browser, as the browser will remain on whatever page it was last accessing until closed. When the browser is reopened, the SPA will redirect to the login page and remove any JWTs that were present in the browser localStorage.

3.2 Tools

Since a website with MFA requires multiple pieces of technology, this section will discuss in detail all of the components that this project utilized. Each component has been logically subdivided for reader convenience.

3.2.1 Mobile Device

A device is required to read the QR Codes, since at least one authenticator is required. For this reason, a mobile device with a camera is needed. O'Dea [30] states over three billion people are smartphone owners around the world. Pew Research Center [31] shows that smartphones have become nearly ubiquitous in the United States, with over 85 percent of Americans owning a smartphone in 2021. The previous statistics suggest that most users have a smartphone, and since smartphones are already used for several types of MFA, a smartphone would be a good choice for this project. There are several models and brands of smartphones available, however there are several hardware requirements that could provide the best security with the least amount of inconvenience to the user, such as: biometric authentication, a camera with sufficient resolution, and a mobile device with a secure enclave, an isolated hardware container that will store the user's private key.

Since the user's private key is stored on the mobile device, it is required to secure the key in case of a lost or compromised device. Securing the key is accomplished with the Secure Enclave. Most smartphone devices have a setting in place to allow the user to password protect it, while this is optimal, this project does not assume the user has this setting enabled, so an additional login was added to the application. Since many smartphone models in use have a method of biometric authentication available and this allows for better user ease, integrated biometrics is used where available.

The Apple iPhone X fulfills all these requirements and was readily available. Therefore an Apple iPhone X with iOS 14 was used for this project.

3.2.2 Secure Enclave

The Secure Enclave is an isolated hardware container within a supported Apple main Central Processing Unit (CPU). The Secure Enclave adds another layer of protection to the user's key, since keypairs cannot be imported or taken out of the Secure Enclave. The container has its own processor, which is strictly limited to operations within the Secure Enclave and runs at a lower clock rate to prevent brute-force clock and power attacks. The Secure Enclave container does not have its own memory, but features a Memory Protection Engine which sequesters and encrypts a protected region of memory on bootup from the main memory controller, which is only accessible by the Secure Enclave [32]. The layout of the Secure Enclave components on the chip is shown in Figure 6. The Secure Enclave also stores the user's biometric data, such as their fingerprint or facial data. The biometric data is only ever stored and referenced in the Secure Enclave. The data cannot be exported, cannot be accessed by Apple, and in the event a device is compromised, cannot be easily extracted from the device.

This project used the Secure Enclave to create the user's public and private keypair, store the user's private key, and digitally sign the user's payload during login. Keypairs are generated by the Secure Enclave and while the public key is exportable, the private key remains inside the Secure Enclave itself. Applications which create keypairs by the Secure Enclave never actually see the private key, but only receive a reference to the private key and only receive the result of operations performed by the Secure Enclave. This means in the event a device is compromised, the user's private key cannot be easily extracted from the device.

3.2.3 Keys

The keypairs created in the Secure Enclave are 256 bit ECDSA keys, with the secp256r1 prime random domain parameter [33], which is comparable to the cryptographic strength

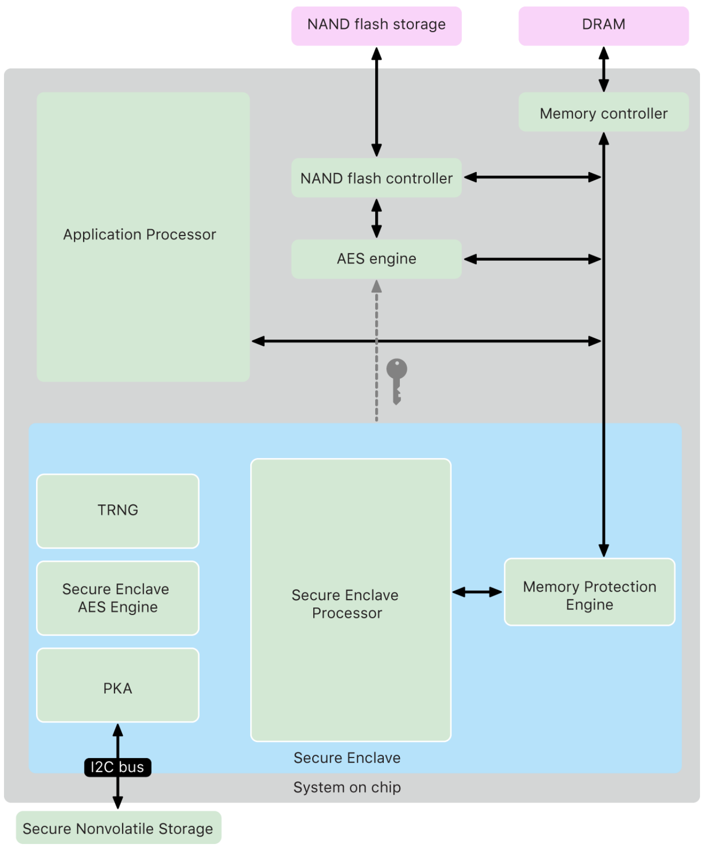
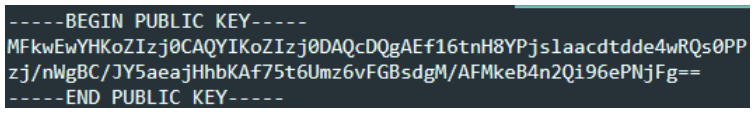


Fig. 6. Secure Enclave components on chip. Screen shot reprinted from Secure Enclave Overview 2021, with permission from Apple Inc. Copyright 2021 by Apple Inc. [32]

of a 3072 bit RSA or Digital Signature Algorithim (DSA) key. This means they are much smaller and are computationally faster to sign and verify than their equivalent RSA or DSA keys. Two additional parameters were added during key creation: kSecAttrAccessibleWhenUnlockedThisDeviceOnly, allowing access to the key only when the device is unlocked and the application is in the foreground [34], and biometryAny, which makes the key available, only if the device is able to authenticate the user using an available biometric authenticator [35].

ECDSA keys are calculated using elliptical curves over a finite field determined by their domain parameter. The signature created from a 256 bit ECDSA key is 512 bits, and is verified by re-calculating the hash using the algorithm specified in the signature, then recovering a point on the curve using the public key and verifying it is the same point that was randomly generated during signing [36] [37].

When preparing the public key for export, the Apple Software Development Kit (SDK) exports public ECDSA keys into ANSI X9.63 [38] format (04 ||X ||Y). While OpenSSL has support for X9.63 keys, the PHP OpenSSL library requires that keys be Privacy Enhanced Mail (PEM) formatted. To allow for key verification with PHP OpenSSL, the key is first converted to Distinguished Encoding Rules (DER) format, and the Abstract Syntax Notation One (ASN.1) object identifier header for the secp256r1 domain parameter is added. The key is then base 64 encoded and the key is wrapped in the PEM ——BEGIN PUBLIC KEY—— heading and ——END PUBLIC KEY—— closing tags. Figure 7 illustrates an example of a PEM formatted sec256r1 ECDSA key exported from the Secure Enclave.



```
-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEf16tnH8YPjslaacdtdde4wRQs0PP
zj/nWgBC/JY5aeajHhbKAf75t6Umz6vFGBsdgM/AFMkeB4n2Qi96ePNjFg==
-----END PUBLIC KEY-----
```

Fig. 7. An example of a PEM encoded exported public key

3.2.4 Mobile App Development

A native application was required in order to utilize all of the hardware components of the Apple iPhone. Apple requires native applications to use their Xcode Integrated Development Environment (IDE), which is only available through Apple's App Store on macOS systems. Several pieces of hardware and parts of the Swift UI required the use of Apple iOS SDK version 13 or greater, which is only available through Xcode 11. However, since iOS 14 was used for this project, which requires Xcode 12, Xcode 12 was used on a Macbook Pro with macOS 10.15.4 Catalina installed.

The mobile application was built using the Swift language, version 4. The mobile application required the use of several third-party libraries. CocoaPods was used as a dependency manager to obtain all third-party libraries. Eureka, a form builder, was used to create the user registration form. BCryptSwift, a Swift implementation of bcrypt was used to securely store user's passcodes in hashed and salted form in the keychain. OAuthSwift was used to process tokens and connections to the server.

3.2.5 BCrypt

BCrypt is a common algorithm that takes advantage of the CPU intensive key setup in eksblowfish to securely store user's passwords in a hashed and salted form. Its main advantage is that its work value is adjustable. The work factor is the amount of time required to hash the password and salt n rounds. The work factor should be adjusted to the highest tolerable value, as advances in hardware will decrease the work factor significantly. According to Pornin [39], the work factor value should be at least 241ms. This time value ensures that user experience is not affected significantly, while providing sufficient protection from brute-force and dictionary attacks. The BCryptSwift library by default sets n at ten rounds but allows the work factor to be adjusted by setting n to an integer between four and sixteen rounds. Each time n is incremented, the work factor increases exponentially.

From testing the BCryptSwift library on the Apple iPhone X, the highest number of rounds tested, fourteen, required a work factor of approximately nine minutes. Four rounds, the minimum, required a work factor of approximately 0.54 seconds, and ten rounds, the default, required approximately 31.3 seconds. Six rounds was chosen with the highest tolerable work factor of approximately two seconds (1.96s). This resulted in an additional two second delay during registration and when logging into the application with a passcode.

3.2.6 Server

The Laravel framework provided the back-end API, along with several libraries, including a first party OAuth 2.0 library, Passport version 10.0.1, to handle authentication and issuing of tokens to clients. A SPA was built with Vue.js, a minimalist JavaScript framework, which enabled storage and retrieval of tokens from browser localstorage, and requests to the back-end API. The PHP OpenSSL library was used along with OpenSSL version 1.1.1 to verify the hash and digital signature during authentication. The server was hosted using Apache 2.4.46 with PHP 7.4.9 and MySQL Community Server version 8.0.21 as the data storage engine on a local Windows 10 machine. The connection to the server was secured using TLS 1.3 with a 2048 bit RSA key and a certificate from the trusted root CA Let's Encrypt.

3.2.7 Laravel

Laravel is a PHP framework that allows for MVC programming strategy, enabling clean, flexible, and scalable server applications. Laravel includes an ORM API, Eloquent, which allows for data and business logic separation. Laravel is also front-end agnostic, enabling developers to choose Laravel's included Blade Engine for traditional PHP written and Hypertext Markup Language (HTML) rendered pages or use API routing to make requests from a separate front-end client. This project used the latter approach with Laravel version 8.7.1. Laravel also includes Axios, a third-party requests library that was used for making XMLHttpRequests from the browser.

3.2.8 Vue.js

Vue.js, or simply Vue, is presented as a progressive framework, minimalist at its core, but with the ability to scale and adapt as needs change. Its main functionality is to power front-end views, but can be integrated with other libraries to allow more functionality. Vue was mainly chosen for its simple integration with Laravel, and the need to store and retrieve tokens from the browser, which is only possible through a client-side application such as JavaScript. Vue version 2.6.12 was used.

Vue operates with the concept of components. Components are small pieces of code that are compiled into the main JavaScript library the client uses. This allows for a more object-oriented code approach. Vue also supports data-binding, which allows for an MVC programming strategy, rather than diving into the Document Object Model (DOM) and changing values directly. Each component has one or more defined sections: template, script, and style. The template section contains standard HTML and any defined data bindings, which are applied directly to the bound element. The script section contains any JavaScript which is local to the component and any code which updates data bindings defined in the template section. The style section contains cascading style sheets (CSS), which can be applied dynamically to any of the elements in the template section.

Since Vue was used to create an SPA in this project, each component was used as a view, which can be thought of as a Uniform Resource Locator (URL) page. When accessing a specific URL, Vue Router loads the component template and any code and styles defined in the script and style sections.

3.2.9 Vue Router

Although Laravel provides routing functionality, a JavaScript SPA consuming a back-end API requires the use of JavaScript routing. This was done through a first-party addon Vue library, Vue Router version 3.5.1. Vue Router also includes the ability to add navigation guards, which was used to guard protected routes from unauthorized and unauthenticated access. This was achieved by first defining a list of slugs, or named URLs. These slugs were then mapped to specific Vue components. For example, when accessing the /login slug, the Login component is returned. When defining these mappings, an asynchronous beforeEnter function was assigned to each protected route. When the slug is accessed via the browser, the router first checks if there is a beforeEnter function on the route. If the router discovers the function, then the function is called.

This project defined an isAuthenticated function, which returns a boolean true value if the browser is authenticated, false otherwise. The isAuthenticated function calls a project defined API route /isAuthenticated. This route is sent through Laravel's Passport API driver, which checks if a valid token was sent in the request. The route returns an HTTP status code of 200 OK if the token is valid or an HTTP status code 401 unauthenticated otherwise. If a status code of 200 was received, then the isAuthenticated function returns true, otherwise it returns false. The beforeEnter function calls the isAuthenticated function each time a protected route is accessed. If the function returns true, then the requested route is returned, otherwise the login route is returned and any JWTs present in local storage are removed.

3.2.10 QR Codes

QR Codes, (a registered trademark of DENSO WAVE INCORPORATED) are a form of universal barcode, which can be scanned from every direction in 360 degrees by virtually every smartphone brand and model with no special hardware or software [40]. Although several types of barcodes exist, QR Codes, being nearly ubiquitous and easy to scan, are used for this project. QR Codes can store up to three Kilobytes of virtually any type of data and are highly resistant to damage. This data storage ability is in contrast to conventional two-dimensional barcodes (such as the barcode used in Universal Product Codes), which are only capable of storing about twenty numeric digits at maximum. Since this project will need to render a 64 byte nonce in an easily assimilable mobile format, QR Codes are best suited for this use.

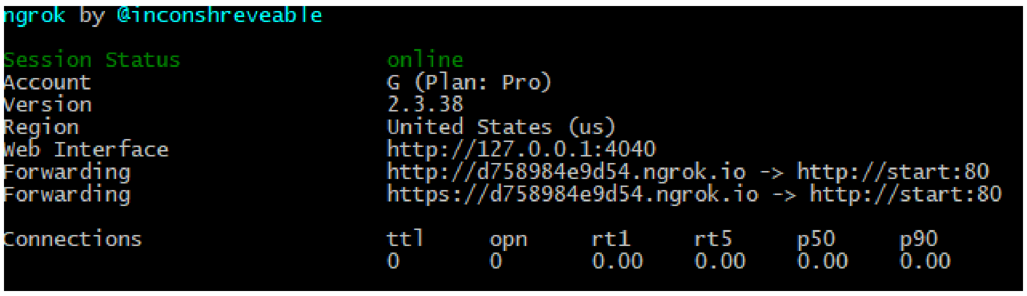
In order to render QR Codes on the server, a Vue component library was used, vue-qrcode, a wrapper for node-qrcode, which is based on "QRCode for JavaScript" by Kazuhiko Arase. The library was installed through the node package manager (npm) and the component was imported into the main Vue application. Finally, rendering a QR Code on the page was accomplished by adding an HTML element into the Login component template section.

3.2.11 Application Transport Security

Starting in iOS 13, Apple made changes to their TLS requirements, requiring all applications to use TLS with certificates verified by a trusted root CA. In order to conform to these specifications while hosting a server on a local machine, a domain name was purchased, and Ngrok was used to create a forwarded secure TLS termination with a Canonical Name (CNAME) record pointing to Ngrok's name servers in the Domain Name System (DNS) registrar. Finally, a free certificate was obtained through the trusted open CA, Let's Encrypt [41] and configured within Apache to create a secured TLS connection over the purchased domain name.

3.2.12 Ngrok

Ngrok is a freemium service which runs an installed program that allows developers to expose their local servers behind Network Address Translation (NAT) and firewalls to the public Internet without resorting to port forwarding. This is done by creating a tunnel from the local machine to the public Internet and provides a public URL to access the server. The URL is created as a subdomain on ngrok.io as a random unique hexadecimal string, such as d758984e9d54, followed by ngrok.io, where the full URL would be http://d758984e9d54.ngrok.io. Figure 8 illustrates Ngrok's console with an active tunnel.



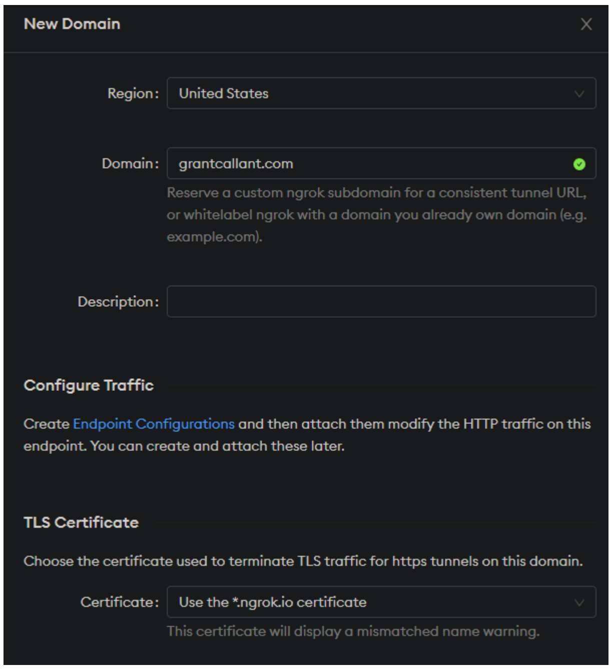
```
ngrok by @inconshreveable

Session Status                online
Account                       G (Plan: Pro)
Version                       2.3.38
Region                        United States (us)
Web Interface                 http://127.0.0.1:4040
Forwarding                    http://d758984e9d54.ngrok.io -> http://start:80
Forwarding                    https://d758984e9d54.ngrok.io -> http://start:80

Connections                   ttl     opn     rt1     rt5     p50     p90
                              0       0       0.00    0.00    0.00    0.00
```

Fig. 8. Ngrok console in command line with example of generated URL

While the core service is free, there are a number of features that are only available for a paid monthly subscription, such as the ability to create a tunnel on a custom domain or a secure TLS tunnel. This project required a valid TLS secured domain name, which typically requires a dedicated Internet protocol (IP) address. Ngrok offers the ability to create a TLS tunnel on a custom domain by first reserving the custom domain in the Ngrok dashboard. Figure 9 illustrates an example of reserving a custom domain in the Ngrok dashboard. The custom domain name is then pointed to Ngrok by creating a CNAME record in the DNS registrar. Figure 10 illustrates an example of a created CNAME record pointing a custom subdomain to Ngrok. This custom domain name was then used for all requests to the server from the mobile application.



New Domain

Region: United States

Domain: grantcallant.com

Reserve a custom ngrok subdomain for a consistent tunnel URL, or whitelabel ngrok with a domain you already own domain (e.g. example.com).

Description:

Configure Traffic

Create Endpoint Configurations and then attach them modify the HTTP traffic on this endpoint. You can create and attach these later.

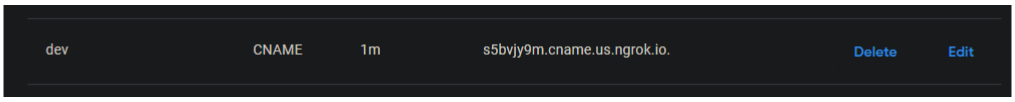
TLS Certificate

Choose the certificate used to terminate TLS traffic for https tunnels on this domain.

Certificate: Use the *.ngrok.io certificate

This certificate will display a mismatched name warning.

Fig. 9. Ngrok dashboard reserving a domain



dev	CNAME	1m	s5bvjy9m.cname.us.ngrok.io.	Delete	Edit

Fig. 10. A CNAME record added to the DNS registrar

3.2.13 Broadcast Notifications

Because authentication is done from the phone, the desktop client needs to be notified on successful authentication. WebSockets provide the ability to send real-time updates to clients, which is more efficient than continually polling to see if data has been updated. Laravel includes broadcast and event functionality that will dispatch events over a WebSocket and enable the handling of an event from a listener. Laravel offers two drivers for client-side broadcasting, Ably and Pusher Channels. This project used Pusher Channels.

3.2.14 Laravel Echo

Laravel includes built-in functionality to listen and handle server-side events, however these events are not visible to separate front-end clients. Laravel has a first-party JavaScript library called Laravel Echo, which allows allows JavaScript clients to listen to broadcasting channels for any dispatched events. This is done by first subscribing to a channel the client wishes to listen on and for what event to listen. The client may then handle the event [42].

3.2.15 Pusher Channels

Pusher Channels is a premium subscription service with a limited free Sandbox plan for developers with less than two-hundred thousand messages per day and less than one hundred concurrent connections [43]. For this project, Pusher acts as a bridge between the Laravel backend, and the Laravel Echo front-end. Laravel dispatches an event and broadcasts it to Pusher, which then redirects the event to Laravel Echo. Pusher includes a web dashboard, which features a debug console. Channel subscriptions, and events are shown in the console in real-time.

3.3 Desktop User Interface

The desktop UI, includes several portions of the hosted SPA that are user intractable. A simple prototype SPA was constructed to hold a collection of assets. These assets would be available to download by any user that is authenticated. There is no administrator interface, and no additional authorization is required by authenticated users. Figure 11 illustrates an example of the SPA to an unauthenticated user. A menu is shown on the left. The menu is always shown, as an SPA does not need to refresh the page in order to update the page content. Bold text separates sections in the menu, with text underneath that are links to content. An unauthenticated user may click on any link but will be continually redirected back to the login page without any feedback. The only thing that an unauthenticated user may do is click on the blue login button.

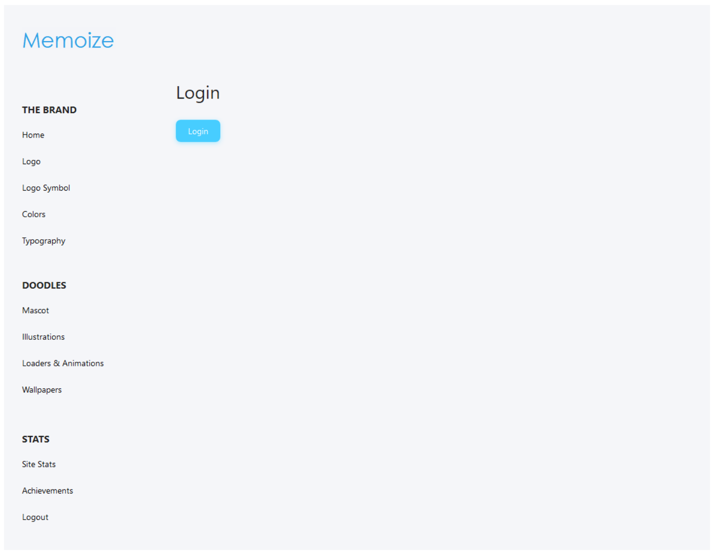
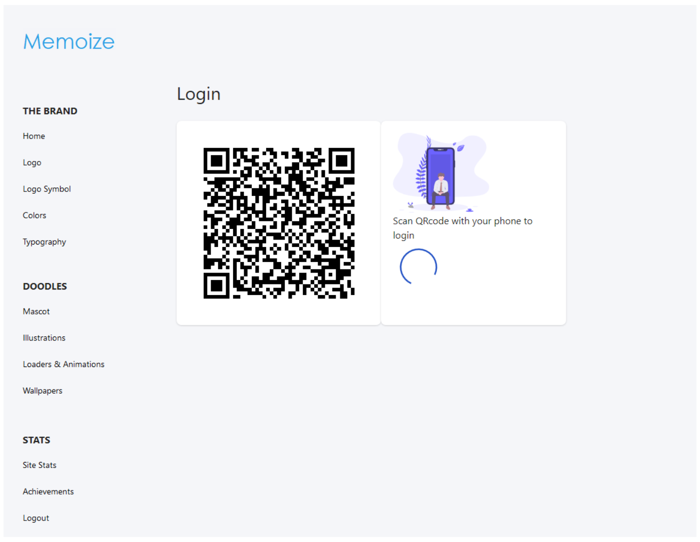


Fig. 11. SPA Login page

After a user clicks on the Login button, new content will be presented on the same page. Figure 12 illustrates the QR Code displayed by the SPA after the login button is clicked. After a user is successfully authenticated, a JWT is issued to the desktop client to provide authorization and the user is automatically redirected to the home page. Figure 13 illustrates the SPA home page. Figure 14 illustrates an assets page an authenticated user would be able to access. Finally, a logout link is provided at the bottom of the menu so a user may logout using only the site. After logout, the user is automatically redirected back to the login page.

3.4 Mobile User Interface

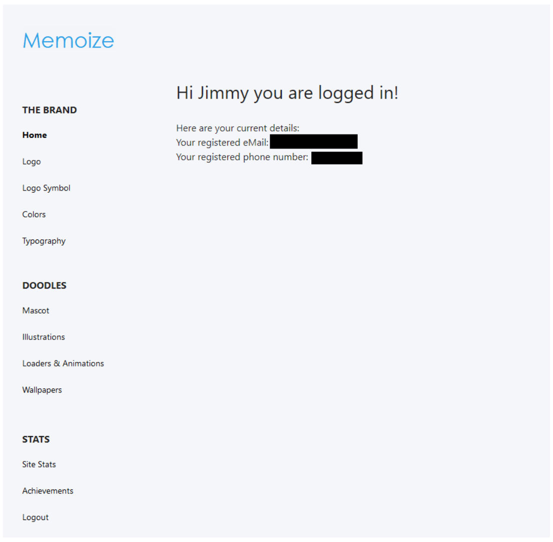
The mobile UI includes several intractable portions of the developed native application. Figure 15 illustrates a very basic welcome screen when opening the application for the first time. Upon tapping the screen, the user will be presented with a registration form. 16 illustrates the registration form. The form gives guidance to users, highlighting entry errors in red. Figure 17 illustrates the registration form with an extra digit inserted into the phone number. The registration form requires: a name, unique email, unique phone number and a passcode with six or more digits. When the form is filled out, the Submit button can be pressed. After the Submit button is pressed, the user's



Login

Scan QRcode with your phone to login

Fig. 12. SPA Login page after user clicks login button



Hi Jimmy you are logged in!

Here are your current details:
Your registered eMail: [redacted]
Your registered phone number: [redacted]

Fig. 13. SPA Home page after user is authenticated—user details are obfuscated

keypairs are created on the device, and the public key and the user's information is transmitted securely to the server. This process involves a brief delay and the user is shown a loading screen while this process completes. Figure 18 illustrates the loading screen shown to the user. In the case there is an error submitting this information to the server, the user is shown an alert, and is given the option to retry. Figure 19 illustrates an alert shown in the case of a failed upload.

Once the user's information has been successfully uploaded to the server, the user is redirected to the Home screen. Figure 20 illustrates the Home camera screen. The

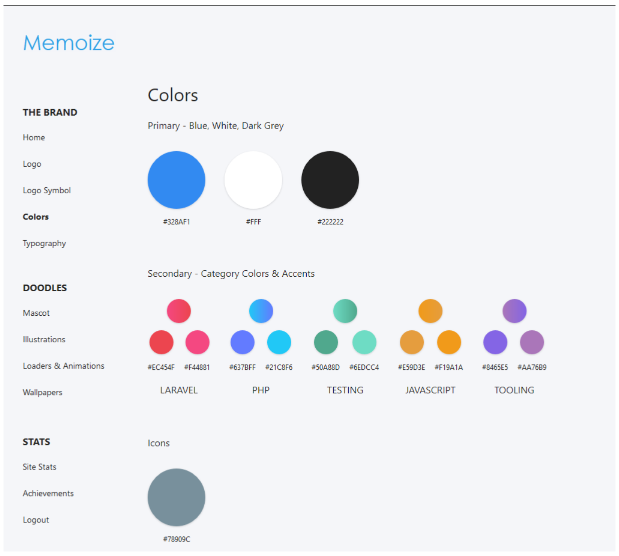


Fig. 14. SPA with example of Colors asset page

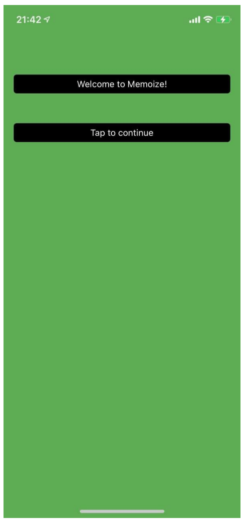
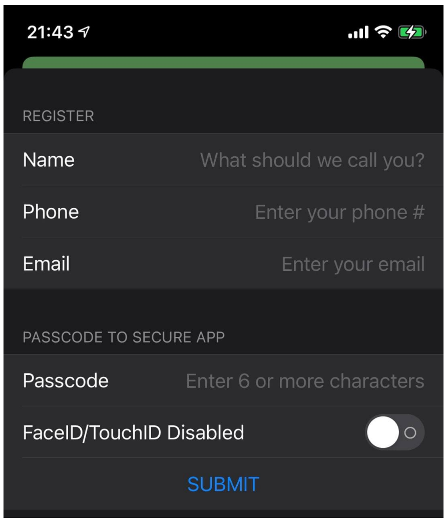


Fig. 15. Mobile application welcome screen

Home screen is a camera view with a navigation bar at the bottom, allowing access to the Settings screen. The Settings screen, mainly contains data used for performance testing, and features a button which allows for deleting all user data. Figure 21 illustrates the settings screen.

From the Home screen, the user scans the QR Code that is generated by the website SPA. Upon successful authentication, the user is redirected to a dashboard, where the user can view his or her information and may logout by pressing the Logout button. Figure 22 illustrates the user dashboard. After a successful logout, the user is redirected

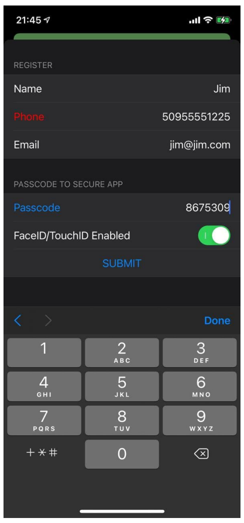

REGISTER

Name	What should we call you?
Phone	Enter your phone #
Email	Enter your email

PASSCODE TO SECURE APP

Passcode	Enter 6 or more characters
FaceID/TouchID Disabled	

Fig. 16. Mobile application registration screen


REGISTER

Name	Jim
Phone	50955551225
Email	jim@jim.com

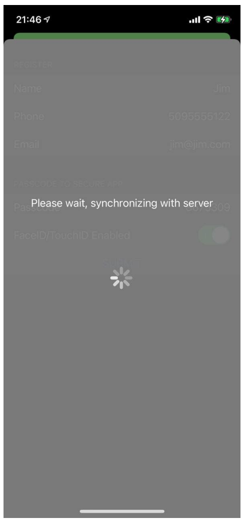
PASSCODE TO SECURE APP

Passcode	8675309
FaceID/TouchID Enabled	

Fig. 17. Mobile application registration form with an extra digit entered in phone number

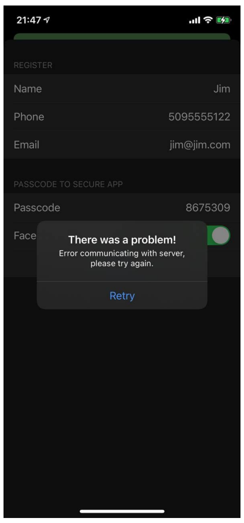
back to the Home screen, and a logout alert is displayed, to remind the user to close the browser to complete the logout process. Figure 23 illustrates the alert shown after successful logout.

If the user exits or navigates away from the application, the application immediately locks, and login is required to re-enter the application. If the user opted to enable biometric authentication during the registration process, then a biometric login is initiated. If the user opted out of biometric authentication, cancels the biometric login, or the biometric login fails, then the user may login with a standard passcode. Figure 24 illustrates the login screen with a biometric authentication in progress.


21:46

Please wait, synchronizing with server

Fig. 18. Mobile application loading screen


21:47

REGISTER

Name Jim

Phone 5095555122

Email jim@jim.com

PASSCODE TO SECURE APP

Passcode 8675309

Face

There was a problem!

Error communicating with server, please try again.

Retry

Fig. 19. Mobile application upload error alert

4 Related Work

The following section discusses current products that are similar to this research.

4.1 CamAuth

CamAuth [44] proposes a method that uses a standard username and password along with an additional security step that utilizes a mobile device's camera, a computer camera, and a browser extension to decode two QR Codes for a 2FA request. The user first enters their username and password then clicks the SecureLogin button. A QR

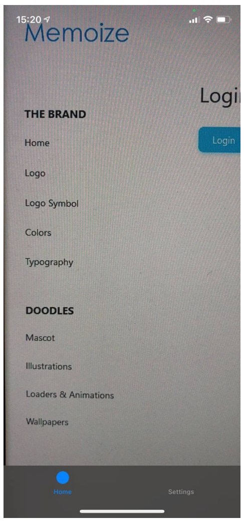
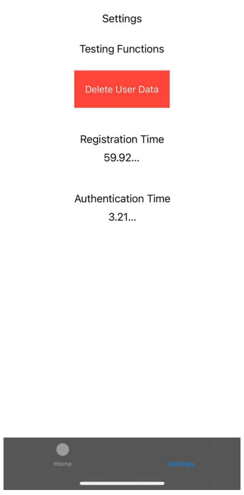


Fig. 20. Mobile application Home camera view screen. The navigation bar is visible at the bottom.


Settings

Testing Functions

Delete User Data

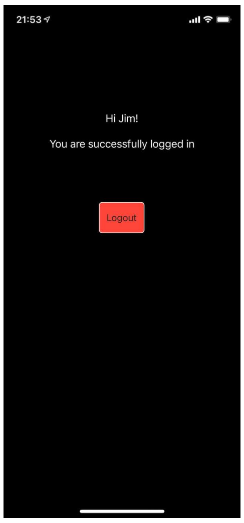
Registration Time
59.92...

Authentication Time
3.21...

Fig. 21. Mobile application Settings screen

Code is generated by the browser extension which is then scanned by the mobile device. The mobile device then computes an additional QR Code which is then scanned by the computer camera. CamAuth does not rely on Internet access or Secure Sockets Layer (SSL)/TLS and provides resilience against Man-In-The-Middle (MITM) and phishing attacks.

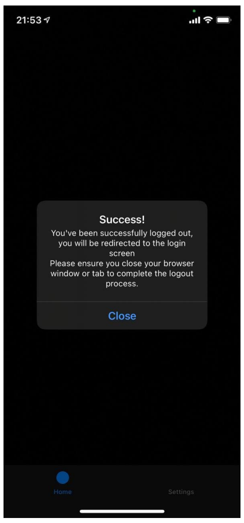
Unlike this project, which uses a digital signature created by a keypair for authentication, CamAuth uses a standard username and password. Similar to this project, CamAuth uses a the concept of QR Codes for an additional security step. However, the



Hi Jim!

You are successfully logged in

Fig. 22. Mobile application Dashboard



Success!

You've been successfully logged out, you will be redirected to the login screen
Please ensure you close your browser window or tab to complete the logout process.

Fig. 23. Mobile application logout alert

QR Code that is generated only provides an additional security step, and the QR Code does not provide authorization information. CamAuth also requires the user to scan two QR Codes, one that is generated on the desktop, and another code that is generated on the phone. While CamAuth does improve upon security with their 2FA method, they do not reduce the complexity of the login process or improve the user experience.

4.2 SC@CCO

SC@CCO [45] is similar in nature to CamAuth including providing an additional 2FA step using a mobile device's camera and a two-dimensional barcode. SC@CCO

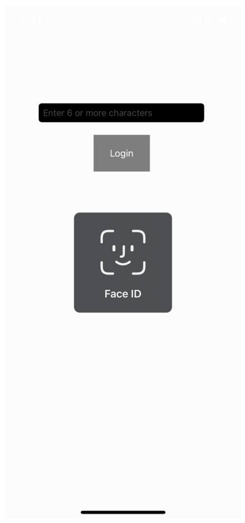


Fig. 24. Mobile application biometric login

also provides resilience against MITM and phishing attacks. SC@CCO's largest difference from CamAuth is that SC@CCO utilizes OTP and does not use QR Codes. SC@CCO instead relies on a pre-shared Advanced Encryption Standard (AES) key that is used to encrypt and decrypt a bi-dimensional Data Matrix barcode. The user first logs into a secured website using a username and password and initiates a transaction. The user's id along with the transaction information is sent to SC@CCO's server. A barcode is generated on the server using the user's id and transaction information. The encrypted barcode is displayed on the secured website, which is scanned by the user's mobile device. The mobile device decrypts the barcode information sent from the server to obtain an OTP, which the user inputs into the service along with the user's chosen PIN. The SC@CCO server compares the received OTP against the sent OTP and verifies the user's PIN. The SC@CCO server then communicates with the web application that the user's identity is confirmed.

Similar to this project, S@CCO utilizes pre-shared keys and two-dimensional barcodes to provide an additional layer of security. Unlike this project, which uses a digital signature created by an asymmetric keypair for authentication, SC@CCO utilizes a symmetric AES key to provide an additional security step after the user is first authenticated into a secured website with a username and password. Since the key is symmetric, the same key can be used to both encrypt and decrypt. Unlike this project, which stores the user's private key in the Secure Enclave, SC@CCO's keys are not stored securely on the mobile device or on the SC@CCO server.

An attacker could potentially gain access to a user's key from a stolen device or from a server breach. A compromised key means an attacker could impersonate the server to send a fake challenge to a user potentially gaining the user's PIN. An attacker could also use the same key to impersonate a user, reading any challenges sent by the server and responding with the correct OTP and a stolen user's PIN. While in theory SC@CCO does improve upon security with their 2FA method, SC@CCO's use of symmetric AES keys provides a significant security drawback while doing nothing to reduce the complexity of the login process or improve the user experience.

4.3 WebTicket

Unlike CamAuth and SC@CCO, which provide an additional 2FA step to an otherwise traditional login process, WebTicket [46] is a form of password management using QR Code tickets. Each password is randomly machine generated and is encoded along with the username and the site URL into a QR Code which can be printed or stored on a user's mobile device. The user then scans the ticket using a workstation's web camera and is directed to the encoded site and automatically logged in. These tickets protect against phishing, since the correct URL is encoded into the QR Code, which prevents a phishing website from being accessed accidentally. Since the passwords are machine generated randomly, they are more likely to be secure and are not known by the user so they cannot be user entered into a phishing website. Unlike SC@CCO, WebTicket does not secure a user's password from MITM attacks.

Similar to this project, WebTicket utilizes QR Codes to encode user and site information into a barcode to simplify the login process. WebTicket differs from this project since it does not provide any additional MFA protection. Although WebTicket does simplify the user login process, WebTicket still utilizes a traditional username and password approach. Since WebTickets are a form of printed password, they have similar drawbacks to written-down passwords. A lost or stolen ticket means an attacker would have a user's complete login information. WebTickets are also vulnerable to over-the-shoulder attacks, as pictures or scans of a user's WebTicket would give an attacker a user's complete login information.

4.4 Discord

Somewhat similar to WebTicket, Discord [47], a popular web-based voice and text communication service, requires a full registration including a username and password but offers a shortcut alternative on subsequent authentications. While Discord's service is proprietary, and the exact underlying methods are unknown, the process works as follows. Similar to CamAuth and SC@CCO, Discord presents a QR Code which is scanned by an already authenticated user's Discord mobile application. The Discord mobile application then prompts the user if they are attempting to log in on the computer along with a picture of his or her avatar and Discord username on the computer. If the user clicks the affirm button, the user is then logged into the Discord application.

Similar to this project, Discord offers additional 2FA protection and offers a shortcut authentication method, which reduces login complexity and improves the user experience. Unlike this project, Discord requires the user to first register with a password and be authenticated with a username and password before this authentication shortcut is available.

5 Results

The password-less authentication performance was evaluated with randomized user data created by PHP Faker [48] estimating theoretical user registration time and repeated testing of authentication time for several random users. Registration and authentication time is reported via the Settings screen in the mobile application. Various attacks were considered and their possible implications on security.

5.1 Registration Time

Twenty random users were created, and their first name, email, phone, and passcodes ranging between 6 and 9 digits were all inputed into the mobile application. This information was then submitted to the server. The total time for registration, including the time inputting the user information, and submitting the information to the server was calculated and averaged together. The average time for user registration was 38.20 seconds. The median was 37.054 seconds. While these results are not completely indicative of an average user's registration experience, they do provide some baseline information that the average user should be able to complete registration in one minute or less.

5.2 Authentication Time

Using five of the simulated users created above, selected at random, authentication was performed ten times for each user. The total time for authentication was calculated at the beginning of the QR Code scan, ending at a JWT being received by the mobile device. The times were then averaged together for each user. This is shown in Figure 25. The average for all authentications for all users was 2.027 seconds. This is again, not necessarily indicative of the average user's authentication experience, but provides baseline information that authentication should take less than five seconds for the average user.

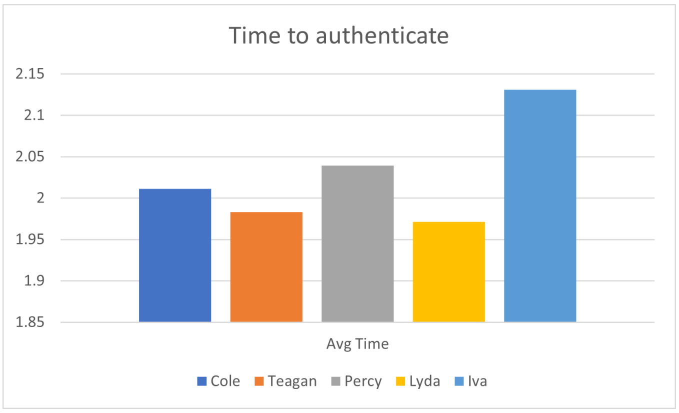


Fig. 25. Average of authentication times for each user

5.3 Server Security

Several means were used to secure the connection to the server including TLS, route guards, middleware, and CSRF protection. This project could not secure against every avenue of potential threat, with many potential attack vectors out of scope. However, the main threats are considered timed replay attacks, and XSS.

5.3.1 Timed Replay Attack

Since the authorization process relies on a nonce that is made available through a QR Code, which is easy to scan and potentially vulnerable to an over-the-shoulder attack, an attacker could scan another user's code and obtain the user's nonce during login. The cache TTL window for obtaining a JWT is very short, two seconds, and the nonce is removed from the cache once used. However, if timed precisely, an attacker could potentially wait for a user to be authenticated and submit a post request to the /login/confirm API endpoint in the second before the user's client to obtain the user's JWT. Testing confirmed this is a potential vulnerability. Using Postman (a Representational State Transfer (REST) API client), and precisely timing a POST request, another user's JWT was able to be obtained. This took multiple attempts, and while clearly possible, in practice it is likely improbable, because the conditions needed to exploit this vulnerability required such precision. The conditions required, a copy of an authenticating user's nonce, which had to be manually copied to Postman before the user authenticated. It also required timing the post request so it sends after the user is authenticated but before the user's browser makes a separate POST request. Nevertheless, users should remain aware of their surroundings, and be wary of persons who seem like they may be attempting to capture a picture of his or her screen. Users should refresh the page to obtain a new nonce if they believe another user has attempted to scan or capture their QR Code. Future work should be done to provide additional security to the barcode to prevent over-the-shoulder scans.

5.3.2 Cross Site Scripting Attack

Because the JWT is stored in browser localStorage, which allows it to be accessed by JavaScript, it is potentially vulnerable to XSS attacks. Although this attack is out of scope and was unable to be performed in testing, one potential solution is to store the JWT in a secure cookie with the HTTPOnly flag, which does not allow JavaScript access. Unfortunately, this also makes it more difficult to use the JWT in an SPA, since the front-end client will not be able to access the JWT. Cookies are also potentially vulnerable to CSRF attacks, and require additional measures to ensure that another domain is not allowed access to secured cookies.

5.4 Application Security

Since perfect security and convenience are usually mutually exclusive, they have to be balanced in favor of usability. Access to the device and the application allows for complete user authentication and is therefore considered a high security risk if the device is compromised. Although the private key itself can be used to identify and authenticate the user, access to the private key is unlikely since it is stored in the Secure Enclave. Therefore the application itself is considered the most vulnerable security vector. Multiple steps were taken to provide additional security to the application and the user's private key. The application itself requires a login, and the private key requires biometric authentication to unlock it for signing. This requires an attacker to not only gain access to the application, but to also have in possession the user's unique biometric authenticator.

Because it is possible for users to possess their mobile device with any kind of authentication, an additional authentication step was added to the application. The main objective of this project was to allow completely password-less login, and this project assumed the majority of users would opt into biometric authentication. However, a standard PIN passcode was provided for legacy purposes. A standard passcode was also provided in the case biometric authentication were to fail or is unavailable. A simple digit passcode was decided because it is the default authentication provided on many devices and because it was the simplest method to implement. A six digit minimum was decided, with no maximum value, in order to provide a greater security surface area than the standard four digit PIN. In order for an attacker to access the application without knowledge of the passcode or possession of a biometric authenticator requires either a brute-force or dictionary attack. Although other attacks are possible, they are outside of scope.

5.4.1 Brute-Force attack

A brute-force attack is simply trying every possible permutation of potential password until access is gained or the attacker gives up. In the case of a six digit minimum, an attacker with access to the device and application would start at 000000, proceeding to 000001...etc trying every possible password combination to 999999. Since many users will comply with only the minimum security requirements, an attack of this nature would likely eventually succeed. The goal therefore is to make the work factor of the attack expensive enough for the attacker to give up prematurely, or give the owner of the device sufficient time to regain access to the device, remotely wipe the device, or change the private key. An analysis of the attack area and the work factor required for this attack follows.

Because there is no enforcement on repeating digits, a user could choose the minimum number of digits and set the passcode to six repeating digits (such as 000000).

Assuming the user has no authentication on the device, a best-case attack would be an attacker gaining immediate access to the application with six repeating 0s. An attacker might heuristically try every other combination of repeating digits and gain access in under a minute. Other heuristic attacks such as attempting linear progressing passwords, 1-6, 2-7, 3-8, etc might also be employed but are outside the scope of this project. The worst-case attack would then be methodically trying every other permutation *P* of ten digits *n* out of every possible six digit combination in order *r* expressed as $P = n^r = 10^6$.

As previously mentioned, passwords are hashed using bcrypt with a work factor of two seconds. Then for an attacker with an attack space of 10^6 possible password permutations yields a total work factor of $2s \cdot 10^6$ or approximately twenty-three days. This is a short time period when considering password requirements for public servers, where password breaches may not be discovered for a long period of time. However, twenty-three days should be sufficient time for a user to realize their device has been compromised and take appropriate action.

5.4.2 Dictionary attack

A dictionary attack or more formally, pre-computed dictionary attack is a method where an attacker is able to gain access to a database of encrypted passwords. The attacker creates a dictionary of all possible hashed values and maps them to their unhashed equivalent. The attacker then simply looks up the hashed password from the stolen database to the unhashed value in the dictionary. However, because bcrypt was used, which hashes the passcode with a randomly generated salt, this dictionary needs to be re-created using bcrypt and the salt value for all possible passcode permutations.

The work factor is difficult to define, because the assumption is the device is not using any authentication, which allows the attacker to download the contents of the device keychain to his or her computer. This means that the two second work factor on the device could be much less on the attacker's machine. Also, when factoring in the brute-force approach, the order of the passcode matters. However, because hashing repeated digits *r*, such as 111111 will always return the same hashed output, the possible ten digit *n* hashed password combinations *P* is significantly decreased. This can be expressed as

$$P\left(n,r\right) = \frac{n!}{(n-r)!} \tag{1}$$

$$P\left(10,6\right) = \frac{10!}{(10-6)!} = \frac{10!}{4!} = 151{,}200 \text{ different combinations} \tag{2}$$

which is significantly less than the 10^6 possible combinations expressed above. Testing a simple implementation of bcrypt in Node.js on a local Windows 10 machine using the salt round value of six, which is the same as on the device, yielded a work factor of approximately 5ms (4.73ms). The total work factor required to generate all possible hashed combinations can be expressed as $5ms \cdot 151{,}200 \approx 12.6$ minutes.

Unfortunately to even gain the recommended work factor of at least 241ms on the Windows 10 machine, required a salt round value of twelve, which required a prohibitively expensive work factor of over six minutes (400 seconds) on the mobile device. This means the application itself is culpable to this type of attack, assuming the device is not protected with any kind of authentication. If the device is protected with authentication, then the device keychain itself is encrypted making access much more difficult. However, even if the attacker gains access to the application, the private key itself is still protected within the Secure Enclave, and generating signatures with the private key still requires a biometric authenticator to unlock it.

6 Conclusion and Future Work

While future work is needed to provide additional validation of results and secure other avenues of attack, this project was able to accomplish its primary objectives. This project leveraged existing, proven cryptography systems, combining them in a novel way. It also succeed in creating a working prototype of a password-less MFA system that is efficient, simple, and easy to use. Simulated user registration suggests that registration can be completed in one minute or less. Repeated simulated authentication tests suggest that authentication can be performed in under five seconds.

This is contrasted against a password system that is archaic and fraught with complexities. 2FA is proposed as the solution but adds additional complexity. SMS 2FA requires additional messages to be sent, which can be intercepted. TOTP provide a high level of additional security, but require a separate application or additional hardware and require the secret seed to be stored in the clear on the server, which was the target of a major national defense breach.

Asymmetric cryptography can provide a high level of security, provided the user's private key is never revealed. Embedded hardware systems like the Secure Enclave on modern mobile devices can allow secure access to private keys, while preventing their exposure if a device is compromised. Biometric authentication can allow for efficient, accurate identify verification without the use of passwords. Combing biometric authentication and digital signatures can allow for a password-less MFA system while providing a much better user experience than a traditional password approach.

Much research has already been completed on new and better authentication systems. This project shows that much more work is still required to stay ahead of attackers. This project used QR Codes because of simple implementation and ubiquity, but bar codes which only allow scanning by authorized users could be created. More secure QR Codes like the SQRC [49] from DENSO WAVE Incorporated could be implemented to prevent over-the-shoulder and timed replay attacks. Encrypted cookies, which contain a CSRF resistant token could protect against XSS and CSRF attacks, while also being available to JavaScript frameworks. An authorization system for identifying authorized devices after authentication would allow for a more secure and seamless device hand off.

This project used barcodes for authentication, but possible advancements with Near-Field Communication, similar to what is currently being done for contactless payment systems could be utilized for simple and efficient authentication. Bluetooth, while being somewhat insecure, could also provide another avenue for a smaller more efficient encrypted authentication system.

References

[1] D. of Defense, “Dod instruction 8520.03, identity authentication for information systems,” Department of Defense, Tech. Rep., May 2011.

[2] J. Foti, “Guideline for the use of advanced authentication technology alternatives,” National Institute of Standards and Technology, Gaithersburg, MD, Tech. Rep. NIST Special Publication (SP) 800-32 Rev 1 (Final), 1994.

[3] G. A. Miller, “The magical number seven, plus or minus two: some limits on our capacity for processing information.” *Psychological review*, vol. 63, no. 2, p. 81, 1956.

[4] D. V. Klein, “Foiling the cracker: A survey of, and improvements to, password security,” in *Proceedings of the 2nd USENIX Security Workshop*, 1990, pp. 5–14.

[5] A. Vance, “If your password is 123456, just make it hackme,” *The New York Times*, Jan. 2010. [Online]. Available: https://www.nytimes.com/2010/01/21/technology/21password.html

[6] W. Cheswick, “Rethinking passwords,” *Commun. ACM*, vol. 56, no. 2, p. 40–44, Feb. 2013. [Online]. Available: https://doi.org/10.1145/2408776.2408790

[7] A. Adams and M. A. Sasse, “Users are not the enemy,” *Commun. ACM*, vol. 42, no. 12, p. 40–46, Dec. 1999. [Online]. Available: https://doi.org/10.1145/322796.322806

[8] R. P. Jover, “Security analysis of sms as a second factor of authentication: The challenges of multifactor authentication based on sms, including cellular security deficiencies, ss7 exploits, and sim swapping,” *Queue*, vol. 18, no. 4, p. 37–60, Aug. 2020. [Online]. Available: https://doi.org/10.1145/3424302.3425909

[9] D. Kogan, N. Manohar, and D. Boneh, “T/key: Second-factor authentication from secure hash chains,” in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS ’17. New York, NY, USA:

Association for Computing Machinery, 2017, p. 983–999. [Online]. Available: https://doi.org/10.1145/3133956.3133989

[10] C. Drew, "Security firm offers to replace tokens after attack," *The New York Times*, Jun. 2011. [Online]. Available: https://www.nytimes.com/2011/06/07/technology/07hack.html?_r=1

[11] Yubico. (2021, Apr) Yubikey 5 series. Yubico. [Online]. Available: https://www.yubico.com/products/yubikey-5-overview/

[12] C. Jacomme and S. Kremer, "An extensive formal analysis of multi-factor authentication protocols," *ACM Trans. Priv. Secur.*, vol. 24, no. 2, Jan. 2021. [Online]. Available: https://doi-org.ezproxy.library.ewu.edu/10.1145/3440712

[13] M. A. Sasse, S. Brostoff, and D. Weirich, "Transforming the 'weakest link' — a human/computer interaction approach to usable and effective security," *BT Technology Journal*, vol. 19, no. 3, pp. 122–131, Jul. 2001. [Online]. Available: http://dx.doi.org/10.1023/A:1011902718709

[14] W. S. D. of Licensing. Steps to getting your first driver license: Proof of identity. Washington State Department of Licensing. [Online]. Available: https://www.dol.wa.gov/driverslicense/idproof.html

[15] P. A. Grassi, M. E. Garcia, and J. L. Fenton, "Digital identity guidelines," National Institute of Standards and Technology, Gaithersburg, MD, Tech. Rep. NIST Special Publication (SP) 800-63 Rev 3, Includes updates as of 03-02-2020, 2017. [Online]. Available: https://pages.nist.gov/800-63-3/sp800-63-3.html#sec3

[16] J. Steven, J. Walton, and K. Wall, "Owasp threat model for secure password storage," Open Web Application Security Project, Tech. Rep., Jul. 2012. [Online]. Available: https://owasp.org/www-pdf-archive/Secure_Password_Storage.pdf

[17] L. Johnson, "Chapter 11 - security component fundamentals for assessment," in *Security Controls Evaluation, Testing, and Assessment Handbook (Second*

Edition), 2nd ed., L. Johnson, Ed. Academic Press, 2020, pp. 471–536. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128184271000112

[18] J. Andress, “Chapter 5 - cryptography,” in *The Basics of Information Security (Second Edition)*, 2nd ed., J. Andress, Ed. Boston: Syngress, 2014, pp. 69–88. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128007440000051

[19] D. R. Kuhn, V. C. Hu, W. T. Polk, and S.-J. Chang, “Introduction to public key technology and the federal pki infrastructure,” National Institute of Standards and Technology, Gaithersburg, MD, Tech. Rep. NIST Special Publication (SP) 800-32 Rev 1 (Final), 2001. [Online]. Available: https://doi.org/10.6028/NIST.SP.800-32

[20] A. Mirian, J. DeBlasio, S. Savage, G. M. Voelker, and K. Thomas, “Hack for hire: Exploring the emerging market for account hijacking,” in *The World Wide Web Conference*, ser. WWW ’19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1279–1289. [Online]. Available: https://doi.org/10.1145/3308558.3313489

[21] D. M’Raihi, S. Machani, M. Pei, and J. Rydell, “Totp: Time-based one-time password algorithm,” Internet Requests for Comments, RFC Editor, RFC 6238, May 2011.

[22] R. S. LLC. Securid® access. RSA Security LLC. [Online]. Available: https://www.rsa.com/en-us/products/rsa-securid-suite/rsa-securid-access

[23] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman, “Protection in operating systems,” *Commun. ACM*, vol. 19, no. 8, p. 461–471, Aug. 1976. [Online]. Available: https://doi-org.ezproxy.library.ewu.edu/10.1145/360303.360333

[24] T. Bray, “The javascript object notation (json) data interchange format,” Internet Requests for Comments, RFC Editor, RFC 7159, Mar. 2014. [Online]. Available: https://tools.ietf.org/html/rfc7159

[25] M. Jones, J. Bradley, and N. Sakimura, "Json web token (jwt)," Internet Requests for Comments, RFC Editor, RFC 7519, May 2015. [Online]. Available: https://tools.ietf.org/html/rfc7519

[26] D. Hardt, "The oauth 2.0 authorization framework," Internet Requests for Comments, RFC Editor, RFC 6749, Oct. 2012. [Online]. Available: https://tools.ietf.org/html/rfc6749

[27] SQLite. Appropriate uses for sqlite. SQLite. [Online]. Available: https://www.sqlite.org/whentouse.html

[28] X. ANSI, "62: public key cryptography for the financial services industry: the elliptic curve digital signature algorithm (ecdsa)," *Am. Nat'l Standards Inst*, 1999.

[29] PHP. openssl_verify. The PHP Group. [Online]. Available: https://www.php.net/manual/en/function.openssl-verify.php

[30] S. O'Dea. Number of smartphone users worldwide from 2016 to 2023. Statista. [Online]. Available: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/

[31] P. R. Center. Mobile fact sheet. Pew Research Center. [Online]. Available: https://www.pewresearch.org/internet/fact-sheet/mobile/

[32] Apple. Secure enclave. Apple. [Online]. Available: https://support.apple.com/guide/security/secure-enclave-sec59b0b31ff/web

[33] ——. Storing keys in the secure enclave. Apple. [Online]. Available: https://developer.apple.com/documentation/security/certificate_key_and_trust_services/keys/storing_keys_in_the_secure_enclave

[34] ——. ksecattraccessiblewhenunlockedthisdeviceonly. Apple Inc. [Online]. Available: https://developer.apple.com/documentation/security/ksecattraccessiblewhenunlockedthisdeviceonly

[35] ——. biometryany. Apple Inc. [Online]. Available: https://developer.apple.com/documentation/security/secaccesscontrolcreateflags/2937191-biometryany

[36] S. Nakov, M. Stefanov, and M. Shideroff, *Practical Cryptography for Developers*. Sofia, Nov. 2018. [Online]. Available: https://cryptobook.nakov.com/digital-signatures/ecdsa-sign-verify-messages

[37] T. Pornin, "Deterministic usage of the digital signature algorithm (dsa) and elliptic curve digital signature algorithm (ecdsa)," *Internet Engineering Task Force RFC*, vol. 6979, pp. 1–79, 2013.

[38] X. ANSI, "63: Public key cryptography for the financial services industry - key agreement and key transport using elliptic curve cryptography," *Am. Nat'l Standards Inst*, 2011.

[39] T. Pornin, "Recommended # of iterations when using pkbdf2-sha256?" Information Security, (version: 2011-05-20). [Online]. Available: https://security.stackexchange.com/a/3993

[40] D. W. INCORPORATED. What is a qr code? DENSO WAVE INCORPORATED. [Online]. Available: https://www.qrcode.com/en/about/

[41] L. Encrypt. How it works. Internet Security Research Group. [Online]. Available: https://letsencrypt.org/how-it-works/

[42] Laravel. Client side installation. Laravel. [Online]. Available: https://laravel.com/docs/8.x/broadcasting#client-side-installation

[43] Pusher. Our plans. Pusher Ltd. [Online]. Available: https://pusher.com/channels/pricing

[44] M. Xie, Y. Li, K. Yoshigoe, R. Seker, and J. Bian, "Camauth: Securing web authentication with camera," in *High Assurance Systems Engineering (HASE), 2015 IEEE 16th International Symposium on*. IEEE, 2015, pp. 232–239.

[45] R. d'Alessandro, M. Ghirardi, and M. Leone, "Sc@ cco: a graphic-based authentication system," in *Proceedings of the 1st European Workshop on System Security*. ACM, 2008, pp. 8–15.

[46] E. Hayashi, B. Pendleton, F. Ozenc, and J. Hong, "Webticket: Account management using printable tokens," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2012, pp. 997–1006.

[47] Discord. Qr code login faq. Discord. [Online]. Available: https://support.discord.com/hc/en-us/articles/360039213771-QR-Code-Login-FAQ

[48] F. Zaninotto. Faker. [Online]. Available: https://github.com/fzaninotto/Faker

[49] D. W. Incorporated. What is an sqrc? DENSO WAVE Incorporated. [Online]. Available: https://www.denso-wave.com/en/system/qr/fundamental/qrcode/sqrc/index.html

VITA

Author: Grant M. Callant II

Undergraduate Schools Attended: Spokane Falls Community College
Eastern Washington University

Degrees Awarded: Associate of Arts, 2007, Spokane Falls Community College
Bachelor of Science, 2011, Eastern Washington University

Honors and Awards: Psi Chi Honor Society

Graduated Magna Cum Laude, Eastern Washington University, 2011

Professional Experience:

Chief Technology Officer, JA Enterprises, 2020 to Current

Software Engineer Intern, Ecova, 2016

Graduate Assistant Systems Administrator,
Eastern Washington University, 2015 to 2017

Director of Engineering, JA Enterprises, 2014 to 2020