CryptoChronicles

LessPM's Quest to a Passwordless Utopian Ecosystem

Draft

T-742-CSDA

Håvard Nordlie Mathisen
Reykjavik University
havard22@ru.is

Instructor
Jacky Mallet

Department of Computer Science
Reykjavik University
April 1, 2023

Contents

Draft

Abstract

We describe the implementation of LessPM, a passwordless password manager that utilizes WebAuthn's asymmetric cryptographic nature to attempt to create a more secure authentication experience with the help of smartphones and other user-owned devices. Despite traditional passwords' long and respectable history, the limitations of conventional password authentication have become increasingly apparent through brute-force and phishing attacks, password reuse, keylogging, and shoulder surfing. This has led to the development of more sophisticated methods, such as WebAuthn.

To help secure the server where LessPM's passwords are, we use AES-256 and Argon2 for encryption, combined with WebAuthn's Credential ID, randomly generated salts, and pepper for key derivation. Additionally, we secured the server using HTTPS, configured CORS, and utilized an encrypted variation of JWT (RFC7519), inspired by JWE (RFC7516), to maintain user authentication between requests from a client.

We created a separate client as a proof-of-concept to adhere to The Law of Demeter, while the server handles requests and securely stores user data. This approach offers promising prospects for a passwordless future in digital security.

This report provides valuable insights into the implementation and potential benefits of using WebAuthn and a passwordless approach, along with the security measures used to protect user data on the server.

1 Introduction

In today's digital landscape, utilizing various identifiers (such as usernames, email addresses, or phone numbers) combined with passwords has become a prevalent method for verifying an individual's identity and ensuring their authorization to access restricted materials.

This is not a novel approach. Polybius' *The Histories* [Pol23] contains the first documented use of passwords, describing how the Romans employed "*watchwords*" to verify identities within the military. This provided a transparent, simple way to allow or deny entry to restricted areas of authorized personnel only. The story of secret writing (in this context referenced as cryptography) goes back the past 3000 years [Doo18], where the need to protect and preserve privacy between two or more individuals blossomed.

Fernando J. Corbató is widely credited as the all-father of the first computer password when he was responsible for the Compatible Time-Sharing System (CTSS) in 1961 at MIT [Lev84]. The system had a `"LOGIN"` command, which, when the user followed it by typing `"PASSWORD"`, had its printing mechanism turned off to offer the applicant privacy while typing the password [65]. Given the long history of passwords and their importance, one could argue that it was a natural and judicious step in the evolution of computer systems.

The study "The Memorability and Security of Passwords", conducted in 2004, provides insights into password creation strategies, including tips on improving password entropy and methods for easy recall of passwords [Yan+00]. With an emphasis on diversity in character selection, password length, and avoiding dictionary words, the study suggests that acronym-based passwords offer a delicate balance between memorability and security [Yan+00].

However, as technology has advanced, the limitations of password-based authentication have become increasingly apparent, leading to the development of more sophisticated methods like Universal Authentication Framework (UAF) [FID17] and WebAuthn [Wor21] through the Fast IDentity Online Alliance (FIDO) and The World Wide Web Consortium (W3C).

WebAuthn, short for Web Authentication, is an open standard for web-based authentication that enables users to securely access online web services without relying on a traditional password. Through a collaborative effort between FIDO and W3C, We-

Draft

bAuthn is developed to leverage asymmetric cryptography[1][2] and biometric or hardware-based authenticators to provide a more secure and robust authentication experience.

This report explores the implementation of LessPM, a passwordless password manager that leverages WebAuthn to provide a secure authentication experience, free from the constraints of traditional passwords, while placing a strong emphasis on security. By examining recent advancements in authentication mechanisms and the related innovative potential of WebAuthn, we hope to illuminate the prospects of a passwordless future in digital security. The findings in this report is not an attempt of getting rid of trivial concepts such as session hijacking, nor is it an empirical study of user's perception of the technology seen.

2 Background

In today's world of rapidly evolving technology, it is essential to have a strong foundation in the underlying concepts and protocols that drive modern systems. This background chapter aims to provide a comprehensive understanding of the key technologies and principles that are relevant to our report. By delving into these fundamental topics, we can better appreciate their significance and application in the context of the implementation, design, and architecture presented in the subsequent chapters.

We will provide background information on topics such as WebAuthn (Section 2.1), JSON Web Tokens (JWT. Section 2.4), JSON Web Encryption (JWE. Section 2.4), hashing through Argon2 (Section 2.3), and Advanced Encryption Standard (AES. Section 2.2).

[1] Asymmetric cryptography uses a key pair consisting of public and private keys. The public key encrypts data, while the private key decrypts it. The keys are mathematically related, but deriving one from the other is infeasible, ensuring secure communication and data exchange.

[2] **Note:** According to the library used to implement jsonwebtokens in Rust it is the private key that encrypts and the public key is responsible for decrypting. *Last Accessed: 2023-03-25.*

2.1 WebAuthn

WebAuthn, short for Web Authentication, is a collaborative project between the FIDO Alliance and W3C that aims to implement a secure, robust key-based authentication system for the web, to strongly authenticate users [Wor21]. The concept relies on the use of a third-party device, called an Authenticator, which leverages asymmetric cryptography. These devices employ biometric or hardware-based mechanisms to provide a secure and reliable means of authenticating a user.

Upon registration, the Authenticator Device (AD) generates a key pair called a Passkey. This Passkey contains a Credential ID (CID) uniquely generated for each registered key-pair [21a; 21b] on the Authenticator, per service registered. The unique generation of each key pair offers the advantage of making it much more difficult for trackers to follow a user[3]. Further, if an attacker gains access to an individual's Passkey, they might compromise one specific service, whereas a traditional password could potentially compromise multiple services where password reuse occurs [Wan+18]. This suggests that WebAuthn could serve a stronger level of security, whereas a traditional password exposed in a leak will expose users who are reusing password across services and devices.

2.1.1 Registration

In order to register in a WebAuthn authentication system, the user (i.e., client, browser, phone, etc.) issues a registration request to a WebAuthn implemented server, called a Relying Party (RP), asking to be registered [W3C21b]. The request contains a body with the relevant user identifier (UID. i.e., username, phone number, email, etc.). The server responds by initiating a Registration Ceremony (RC) and generates a challenge, which is sent to the client [W3C21c].

The client then calls the browser-integrated We-

[3] This is subject to the key-pair alone. A willing party could still track the user through their email or similar.

Draft

bAuthn API, requesting the Authenticator (i.e., phone, hardware authenticator device, or similar) to create a new public key credential through the Client to Authenticator 2 Protocol (CTAP2).[4] During this process, the Authenticator generates a new key-pair (public and private keys). The Authenticator then signs the challenge received from the server with the private key [W3C21d].

The newly created public key, signed challenge, and additional metadata are combined into a public key credential object, which the client sends back to the server.

The server verifies the authenticity of the signed challenge and the public key credential object. If successful, the server stores the user's public key and other relevant information (e.g., user identifier, credential ID) for future authentication.

Thus completing the RC, the process can be seen in Figure 1.

2.1.2 Authentication

When a user wishes to perform authentication (commonly referred to as `logging in`), much of the same procedure occurs. The user issues an authentication request to the RP, asking for authentication with the UID that the user used to sign up included in the request body. If the server can find an associated user with the UID in the persisted storage, the server responds by initiating and issuing an Authentication Ceremony (AC) containing a challenge.

The client calls the browser-integrated WebAuthn API, prompting the use of the Authenticator to validate and sign the challenge. Unlike the registration process, the Authenticator now yields a signature, which is forwarded to the RP along with Authenticator-specific data [W3C21a]. Upon receiving the data, the Relying Party (RP) validates the signed challenge using the stored credentials. If the validation succeeds, the server authenticates the user.

The RP can then determine the next steps, such as deciding the authentication duration and establishing methods for persisting and validating the authentication.[5] If the authentication process is unsuccessful at any point in the ceremony, the ceremony is aborted and considered invalid.

This process can be seen in Figure 2.

2.2 Advanced Encryption Standard with 256-bit

The Advanced Encryption Standard (AES) is a symmetric[6] key encryption algorithm. Since its inception in 1998, it has become the gold standard for encrypting various information across applications [Sch15; DR02], being adopted as the successor of the Data Encryption Standard (DES) by The National Institute of Standards and Technology (NIST) in 2001 [NIS00]. AES operates on fixed-sizes units of data referred to as `blocks` [ST01a], supporting keys of sizes 128-, 192-, and 256-bit [ST01b]. A Substitution-Permutation Network (SPN) structure forms the basis of the design, which achieves a high level of security through multiple rounds of processing by combining substitution and permutation [ST01c]. AES with 256-bit key length (hereafter referred to as AES-256 in the rest of the report), employs a 256-bit key and consists of 14 rounds of encryption, offering an advanced level of security compared to its counterparts with shorter key lengths and fewer rounds [ST01d]. In each round of encryption, AES-256 undergoes four primary transformations, as seen in Figure 3, operating on a 4×4 block:
The larger key-size in AES-256 provides an expo-

Draft

[4] The user is prompted to use their Authenticator to prove their presence, which can involve scanning a QR code, providing a fingerprint, or any other modality supported by the device.

[5] In the case of LessPM, the system sets an authentication duration of 15 minutes. It stores this information within an AES-256 encrypted 2.2 JSON Web Token. For more details about this process, refer to Section 2.4.

[6] Symmetric in this context refers to the same key to encrypt and decrypt.

[7] The term `state` refers to an intermediate result that changes as the algorithm progress through its phases.

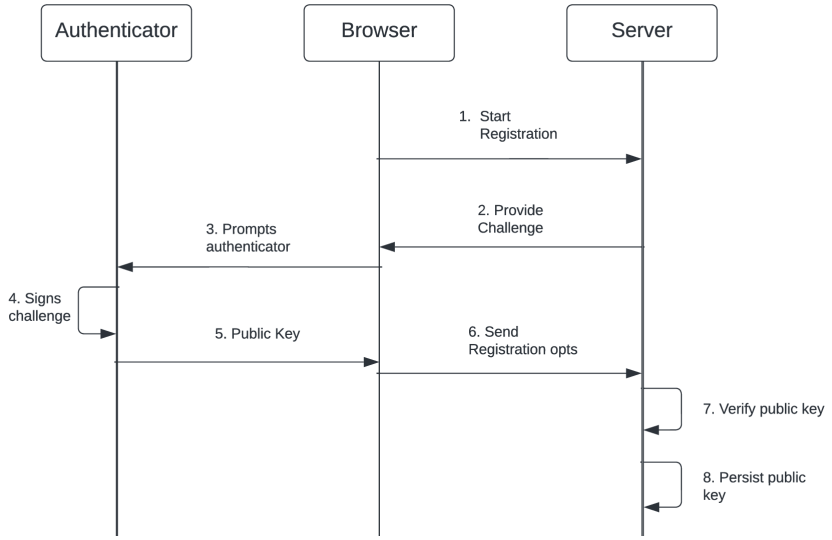

Figure 1: A diagram dipicting the registration process through WebAuthn and Authenticator Device.

Draft

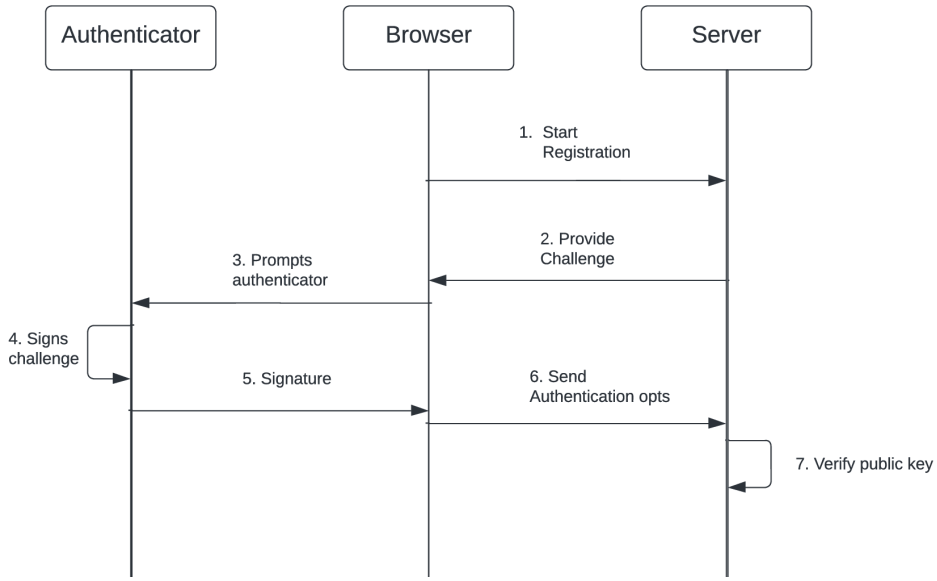

Figure 2: A diagram dipicting the authentication process through WebAuthn and Authenticator Device.

- **SubBytes** is a non-linear substitution step where each byte is replaced with another according to a predefined lookup table.
- **ShiftRows** cyclically shifts each row of the state over a certain number of steps. of the State over varying numbers of bytes while preserving their original values.
- **MixColumns** is a process that works on the columns of the state by combining the four bytes in each column through a mixing operation.
- **AddRoundedKey** involves combining a subkey with the state[7] by applying a bitwise XOR operation.

Figure 3: The steps AES takes when encrypting and decrypting data [ST01c].

nential increase in the number of possible keys, making it significantly more resilient to brute-force attacks and further solidifying its position as a robust encryption standard for safeguarding sensitive information.[8]

2.3 Hashing through Argon2

Argon2 is a hasing/key-derivation function developed by Alex Biryukov, Daniel Dinu, and Dmitry Khovratovich [al17]. Argon2 aims to provide a highly customizable function tailored to the needs of distinct contexts [al17]. Additionally, the design offers resistance to both time-memory trade-off and side-channel attacks as a memory-hard function [al17].

The function fills large memory blocks with pseudorandom data derived from the input parameters, such as the password and salt.[9] The algorithm then processes the blocks non-linearly for a specified number of iterations [al17].

The function offers three configurations, depending on the environment where the function will run and what the risk and threat models are:

- **Argon2d** is a faster configuration and uses data-depending memory access. This makes it suitable for cryptocurrencies and applications with little to no threat of side-channel timing attacks.[10]
- **Argon2i** uses data-independent memory access. This configuration is more suitable for password hashing and key-derivation functions.[11] This configuration is slower due to making more passes over the memory as the hashing progresses.
- **Argon2id** is a combination of the two, beginning with data-dependent memory access before transitioning to data-independent memory access after progressing halfway through the process.

Figure 4: The three configurations of Argon2 [al17].

Draft

Argon2, as a memory-intensive hashing function, demands substantial computational resources from attackers attempting dictionary attacks.[12] This characteristic significantly hampers the feasibility of cracking passwords using such attacks. The algorithm's customizability allows users to adjust its behaviour based on memory, parallelism, and iterations, catering to specific security requirements and performance needs. As these configurations are crucial for computing the original hash, Argon2 provides robust resilience against brute-force and side-channel attacks [al17]. The resulting enhanced security makes Argon2 suitable for password storage

[8] The practical number of potential keys for an AES-256 implementation is 2^{256} possibilities. This gives us an approximation of 1.1579209×10^{77} options. The number is theoretical, as this is a worst-case scenario of options an attacker must go through to find the right key.

[9] A salt is a randomly generated sequence of characters, unique to each instance that gets hashed. Argon2's intension is to have a 128-bit salt for all applications but this can be sliced in half, if storage is a concern [al17].

[11] Side-channel timing attacks analyze execution time variations in cryptographic systems to reveal confidential data, exploiting differences in time caused by varying inputs, branching conditions, or memory access patterns.

[11] Due to the nature of prioritizing security, LessPM uses the third configuration. This is expanded upon in Section **??**.

[12] A dictionary attack is an approach where an attacker tries to find a hash by searching through a dictionary of precomputed hashes or generating hashes based on a dictionary commonly used by individuals or businesses.

and key-derivation in various applications and systems.

In 2015, Argon2 won the Password Hashing Competition [The].[13]

2.4 JSON Web Token & JSON Web Encryption

Although JSON Web Tokens (JWTs) [JBS15] are not inherently encrypted, they still serve an essential purpose for some forms of secure data transfer. By combining JWTs with JSON Web Encryption (JWE) [IET15], secure intercommunication between two or more parties can be achieved.

2.4.1 JSON Web Token (RFC 7519)

JWTs were introduced as part of RFC 7519 in 2015 [JBS15]. It is a compact, URL-safe string intended to transfer data between two entities. They can be used as part of an authentication and authorization scheme in a web service, application, or API. The data in the string is intended as a payload and is referred to as a `claim` [JBS15].

A JWT typically consist of three parts: header, payload, and signature. The header and payload are serialized into JavaScript Object Notation (JSON) and then encoded using a Base64Url encoding to ensure a URL-safe format [JBS15].

The token contains an expiry timestamp, which, when decoded, is validated if the timestamp is not passed at the time of decoding. JWTs can be cryptographically signed using various algorithms like Hash-Based Message Authentication (HMAC), Rivest-Shamir-Adleman (RSA), or Elliptic-Curve Digital Signature Algorithm (ECDSA) ensuring the integrity and authenticity of the token [JBS15]. This prevents unknown authorities from constructing or hijacking existing tokens.[14]

The URL-safe format of a JWT is often performed through a Base64 encoding, which permits larger bits of data to be sent in a compressed, safe[15] format [JBS15]. JWTs are widely used for scenarios such as single sign-on (SSO), user authentication, and securing API endpoints by providing an efficient and stateless mechanism for transmitting information about the user's identity, permissions, and other relevant data [Kar18].

The process functions as follows:

1. The client completed an authentication request.
2. A token was constructed and created through a claim on the server. The claim could have been any data the server wished to use to authenticate the legitimacy of a future request.
3. The claim was signed with the desired algorithm. This could also have been a secret, stored on the server.
4. As part of the response to a request, the server appended the token.[16]
5. The client received the token and carried it upon the next performed request.
6. When the server received the token again, it validated the `exp` property of the JSON object and took action accordingly.

2.4.2 JSON Web Encryption (RFC 7516)

Complimenting the JWT standard are JWEs (RFC 7516). Introduced around the same time as JWTs, in May 2015, there are many parallels between the

[13] NIST's competition to find an encryption algorithm inspired the Password Hashing Competition, but it took place without NIST's endorsement.

[14] Hijacking a token could happen by a man-in-the-middle attack. This is done by a third-party individual listening and intercepting traffic in order to either read data or input their own in a client's request. This would allow an attacker to gain access to privileged information.

[15] Safe in this context should not be interchanged with secure. We reference safe as a way to transfer the data over the selected protocol, in most cases HTTP(S).

[16] A common place to embed these tokens is in the Authorization part of the HTTP header [JBS15].

Draft

two, but the significant distinction between them is that JWEs are encrypted [IET15]. This encryption can happen through AES (see Section 2.2) and provides integrity and authenticity for the token. This prevents eavesdropping or tampering with the token during transit. Combined with JWT, JWE enhance the overall security of JWT-based implementations, making them more suitable for transferring sensitive data between intercommunicating entities on the Web.

3 Literature Review

Passwordless Authentication (PA) is a growing field of study within Computer Science, as traditional authentication methods like passwords are increasingly recognized as vulnerable to attacks such as phishing[17] and credential stuffing.[18] Passwords and sensitive information can also be a victim of successful brute-force attacks [Bon12] through data leakages by hacking or purchasing information on the dark Web.

Following NIST [Nat20; Pau17], authentication should consist of covering one of the three principles in Figure 5:

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

Figure 5: The Three Principles of Password Security [Sch00; Nat20].

[17] Phishing is a form of attack where a hacker tries to leverage Social Engineering to act as a trusted entity to dupe a victim to give away credentials by opening an email, instant message, or text message, then signing into a spoofed website, seeming legitimate [OWAnd].

[18] Credential stuffing refers to the practice of using automated tools to inject compromised or stolen username and password combinations into web login forms to gain unauthorized access to user accounts [Mulnd].

There are many approaches to passwordless authentication or a second step to authenticate with a common password.[19] In 2022, Parmar et al.[Par+22] described several attractive solutions, along with their advantages and drawbacks, for performing PA using the most common methods. The study discovered that PA commonly gets accepted as the most frictionless authentication system for User Interfaces (UI) [Par+22]. Biometrics was mentioned as one of the authentication methods, concluding that it captures universal human traits, encouraging differentiation from one another [Par+22]. The same study raises the caution surrounding the loss of authentication devices and how fingerprints can be compromised [Par+22].[20]

One promising approach is using the FIDO Alliance's collaborative work with W3C to create WebAuthn. WebAuthn permits users to authenticate through biometric information stored on a device in the user's possession (i.e. phone, computer) or a physical security key (i.e. YubiKey, Nitrokey, etc.) [Wor21].

In [Hus22], Huseynov utilized a Web interface with WebAuthn to create credentials that users could use for a VPN. The client required a user to authenticate through the procedure of WebAuthn (see Section 2.1). On a successful request, the Remote Authentication Dial-In User Service (RADIUS) creates a temporary username and password, which would then be transferred as a response to the end-user, permitting them to copy and paste it into the necessary client, alternatively to construct a batch file which would establish the correct connection [Hus22]. The study suggested creating a solution for a VPN client which embedded some browser components [Hus22].

Gordin et al. [Gor21] implemented PA into an

[19] Often referenced as Two-Factor Authentication or Multi-Factor Authentication.

[20] The security implication of using the core concept of FIDO2's WebAuthn is subject to storage in the system on Apple-specific devices [App23a]. On an Android device, the implementation is up to the manufacturer of the device, where Samsung has implemented a Physically Unclonable Function [Lee+21].

Draft

OpenStack environment using WebAuthn, which increases security and bypasses the risk of malefactors employing leaked passwords on other services.[21] The PA process considerably reduces the number of attacks towards a server because the server no longer has user authentication secrets [Gor21]. They, however, utilized a fingerprint as the primary biometrics, citing cost as a primary factor to discourage the use of FIDO2 [Gor21].[22]

Statistia reported in 2020 that between 77–86% of smartphones now have a form of biometric scanner built into their device [Sav22]. Gorin, Et al. continue by mentioning that some individuals have trouble using their biometric scanner or getting it to work correctly on their device [Gor21], which can be a potential drawback for user adaptability.

According to a study conducted by Lyastani et al. in 2020 [Ghr+20], a significant portion of users found the usage of WebAuthn and Fido2 standard to be easy and secure, but with some concerns about losing access to their accounts or fear of others accessing their accounts [Ghr+20]. The study utilized a fingerprint Yubikey for the authentication process. Despite some reservations, the study found that overall, WebAuthn and Fido2 have good usability for passwordless authentication.

The authors reported that users automatically associated the loss of the AD with losing access to the account [Ghr+20] – and vice versa – indicating that users are slightly unwilling to replace the initial principle of *"Something you know"* with the second *"Something you have"* and third *"Something you are"* principle. Additional research is necessary to educate users, increase trust and confidence in the technology, and address concerns about the potential loss of account access.[23]

Morii et al. [Mor+17] investigated the potential of FIDO as a viable PA solution in 2017 when the FIDO2 and WebAuthn standards were yet to be widely adopted. The authors noted that, at that time, only the Edge browser had implemented proper support for FIDO2 and WebAuthn [Mor+17]. Despite the limited browser support, the study demonstrated the feasibility of integrating PA into the well-established authentication system, Shibboleth [Shi22; Mor+17].[24]

As the technology evolved from FIDO to FIDO2.0, some security concerns persisted, such as session hijacking[25], which can compromise user accounts [Mor+17], highlighting the need to protect these.

Since the publication of Morii et al. 's research, browser support for FIDO2 and WebAuthn has significantly improved, with major browsers like Google Chrome, Mozilla Firefox, Apple Safari, Microsoft Edge, and Vivaldi now offering support for these standards. This broader adoption has enabled the more widespread deployment of PA solutions, providing increased security and improved user experiences across various online services.

However, the ongoing evolution of security threats and the increasing sophistication of attackers highlights the need for continuous research and development in the field of passwordless authentication, ensuring that new methods and standards are both secure and user-friendly.

Having explored various studies and developments in the field of PA, it is evident that this area has been continuously evolving to provide secure, user-friendly solutions. However, the implementation of these solutions into stand-alone, real-world applications, such as password managers, independent of other login systems, is a critical aspect that requires thorough investigation.

Draft

[21] See Section 2.1 for further explanation as to how this works.

[22] Authentication is provided through the Keystone environment on the OpenStack platform.

[23] We believe that these concerns are mostly raised due to using a YubiKey and that using a phone-based authenticator would reveal other results.

[24] Shibboleth is a widely-used, open-source federated identity solution that enables secure single sign-on across multiple applications and organizations.

[25] Session hijacking refers to an attacker gaining unauthorized access to a user's authenticated session, often exploiting weaknesses in the handling of cookies, sessions, or JSON Web Tokens (JWT). See Section 2.4.

In the following section, we will examine the methodology employed in this study to integrate a passwordless system into a passwordless password manager, taking into account the challenges and concerns identified in the literature. By doing so, we aim to contribute to the growing body of knowledge on passwordless authentication and its practical applications.

4 Methodology

Exploring the development and implementation of LessPM, a passwordless password manager, our focus will be on the key components, technologies, and steps that form the system's development process. A crucial part of the development and system was its security and robustness.

Our approach encompasses an explanation of the system architecture, the technologies and tools utilized and the development process to create the prototype. By providing a comprehensive account of the system development process, we aim to enable readers to understand the technical aspects of our work as well as to assess the validity and relevance of our findings.

Draft

4.1 Environment

To implement the system, we approached the development from a type-safety environment.

4.1.1 Server

We chose Rust as the programming language for our backend, which provided significant benefits regarding the software development environment.

Rust's emphasis on safety and performance allowed us to create a highly secure and efficient environment for our implementation without the risks typically associated with memory-related vulnerabilities [Riv19]. The built-in memory management and focus on concurrency ensures that the software runs smoothly, which is crucial [FLP85] when dealing with sensitive user data.

Additionally, Rust's surrounding ecosystem is at rapid growth [Kor], containing a vast library of high-quality crates[26], which enables expeditious development and easy integration of various functionality. Utilizing Rust's distinctive features, the passwordless password manager provides enhanced security and reliability in the context of user authentication [Riv19], showcasing the benefits of utilizing a modern programming language.

The backend ran an instance of an HTTPS server, serving as a wrapper for the sensitive data.[27] The Chromium developers mandate this constraint to guarantee that the pertinent API is invoked exclusively within a secure context [web16]. During development, we established a secure connection by using a self-signed certificate for the "`localhost`" domain.

To serve as persistent storage to maintain the passwords, user accounts, and other related data, LessPM adopted MongoDB. MongoDB allowed us to use their NoSQL architecture, permitting storing Object-like structures [Mon21].

4.1.2 Client

An essential part of the implementation was to create a viable client that could function as a visual entry point to the server. For the simplicity of the project, we chose React as a framework. React is a JavaScript library for building user interfaces, offering an efficient and flexible approach to web development. We focused on following The Law of Demeter (LoD) [LH90], and using a least-knowledge principle.

This approach ensured that the client only retained the necessary information to function, maintaining no knowledge of the server or its implementation between each performed request.

As is customary when developing a system con-

[26] `Crate` is the Rust-specific name for a package or library.

[27] We took advantage of the Authorization header during development, as specified in RFC 7519. However, the framework we used for the server required us to expose the usage of `Authorization header` to access it in the client. See Section 6 for further explanation.

taining authentication and authorization, we took advantage of JWT/JWE (Section 2.4) to retain some information to authorize the client between requested resources. We strongly encrypted the Base64-encoded data passed between the server and client using AES-256 (Section 2.2).

We took advantage of React's ability to construct a single-page application with no routing capabilities, avoiding the possibility of utilizing any URL tampering or manipulation[28] to attempt privilege escalation or accessing data restricted from them. Further, the server checked and verified the token before proceeding with any request made to it. This information is updated and inspected between each re-render [Rea] of the client.

When the development of the project first started, we had a strong wish for the client to be an extension for Chromium-based browsers.[29] However, we quickly discovered that the Rust cargo used to perform WebAuthn requests did not implement this support. We reported this issue upstream to the authors of the cargo, and we will continue to work with the authors to ensure a proper implementation of this functionality in the future.

Draft

4.2 WebAuthn

We used WebAuthn to perform passwordless authentication. As an open standard, WebAuthn aims to provide a key-based authentication scheme to authenticate users strongly. When users sign up for a new service, their device generates a new key-pair. (See Section 2.1).

WebAuthn attempted to mitigate phishing, man-in-the-middle, and brute-force attacks through its intuitive design [Wor21]. By leveraging the increasing market of smartphones with biometrics [Sav22], WebAuthn becomes a natural extension in terms of authentication.

[28] Malefactors can perform URL Manipulation, which involves modifying a URL to request resources that would otherwise be inaccessible to a user.

[29] In the client's project folder are traces of a manifest.json file and build scripts to have the client run in an extension.

4.2.1 Registration

When a user tried registering in LessPM, we first check whether a user with a similar name existed. We would deny the registration if the user existed. If no user had that name within the system, we generated a unique ID using UUID V4 and started the necessary Registration Ceremony.

We generated a JWE during this process, which was signed with RSASSA-PSS using SHA-512 before getting encrypted with AES-256 (See Section 6). This claim is then attached to the request's `Authorization` header, along with the Creation options from WebAuthn (See Section 2.1). The `expiration` time for this claim is set to one minute to allow the user some time to authenticate.[30] The claim expired after this minute, and the user would then have to restart the process. This expiration timer was a decision we made to emphasize security within LessPM further.

When the response returned to the user, LessPM used the browser's built-in WebAuthn API to prompt the user for their authentication (See Section 2.1). At this point, it is entirely up to the user to decide what device to use to authenticate. We used an Apple iPhone 13 Pro Max, a Samsung S21, and a Samsung Galaxy A52 to test authentication.[31]

We scanned the QR codes prompted through our phones, which initiated the key-pair generation on the device after a successful facial recognition. [32] The public key then gets transmitted to the browser and sent to the Relying Party through a new request.

Before the request reaches the Relying Party,

[30] WebAuthn describes a timeout performed within the system. In our case, this timeout is one minute. However, we chose to add this extra step to secure the project further, and JWEs require an expiry time. See Section 2.4.

[31] Other alternatives included a YubiKey, NitroKey, Etc.

[32] There is a question of concern that photography can bypass facial recognition. Apple uses built-in sensors to scan depth, colours and a dot projector to create a 3D scan of a person's face. However, this approach prevents the use of photography to authenticate through their FaceID and TrueDepth technology [App23b].

an authentication middleware checks for the JWE sent earlier and denies the request with an `Unauthorized` status code if the request is invalid. [33] The user is then stored in the database if WebAuthn can validate the key and metadata sent with the request, and then the user gets stored in the database, along with the UUID generated.

4.2.2 Authentication

Authentication worked similarly to the registration process. Upon receiving an incoming request, the server checked the database. The server immediately rejected the request if the user did not exist in the database.

Further, the server attempted an Authentication Ceremony by collecting the public key from the database. Upon validation[34] of the public key, the server creates a new JWE, which also receives an `expiry` of one minute for the user to authenticate (See Section 2.1), before sending that and the challenge in the response. We then prompt the user to authenticate with their original authenticator that they used in the registration step, having the authenticator sign the challenge before performing a new request to the server with the signed challenge. The server considered the user authenticated if the Relying Party accepted the signed challenge.

Finally, the user had a 15-minute timeframe to perform the necessary activity within LessPM. The specification for JWT does not specify bounds for this value [JBS15].

4.2.3 Password Creation & Retrieval

Passwords are sensitive in nature, so it seems only natural in a security context to enforce an extra level of authentication upon retrieving and creating one unique password.[35]

A user is presented with the following options when they initiate a password creation:

- **User Identifier**: An identification that the user wants to associate with the password entry they are storing.
- **Website**: A URL or similar where the password belongs.
- **password**: The user is prompted with the input to create a strong password automatically, choosing options such as numbers, special symbols, smaller or larger characters, and the length. As an option, the user was also permitted to construct their password but warned by a warning saying that this option is less secure.

As a final step before a password can is created and stored, the user is prompted to reauthenticate with their AD. The password is created and stored in the database if this authentication process is successful.

Users needed to make a new authentication request when they wanted to retrieve the plaintext version of the password. This step is done in part to validate the user's presence[36] but also to gather the necessary bits of the key reconstruction (See Section 4.5).

4.3 Cors

With the server and client running separately on different ports, Cross-Origin Resource Sharing (CORS) needed to be configured correctly.

When a web page tries to access a resource hosted on another domain, browsers perform an additional request to the server, called a `"preflight"`.

[33] Validity in this context means not timed out, tampered with, or similar.

[34] Validation in this context only means that the key stored in the database is valid.

[35] We retrieved a complete list of the user's passwords upon successful authentication. The hashed version of the password is stripped of the returned values to protect and enforce security.

[36] This approach is also used in password managers on phones to avoid situations where a user might have left their computer unlocked.

Draft

The preflight request is responsible for determining whether the actual request that the web page is trying to make to the server is allowed. This request is done through the `OPTIONS` method in HTTP, and contain some information about the origin, accepted Content-Type, and similar of the actual request. The server response to this with what methods and headers are allowed, denying the actual request from ever happening if the preflight is not successful.

We constructed a CORS `layer`[37] which contained the two domains for the server and client, including credentials[38] and then permitting the two HTTP methods `POST` and `GET`. We made sure the `Content-Type, Authorization`, and `Cookie` headers are permitted.

Any other methods or headers should abort the request in the preflight.

4.4 Cookie

JavaScript can access and manipulate `Cookies` [HXC18]. We are utilizing the browser's local cookie storage to attempt secure authentication between requests.[39] Attempting a couple of strategies listed below, we aim to fortify the cookie that LessPM sets in the browser, against a malefactor:

- **Strict SameSite**: This ensures that the cookie is protected against Cross-Site Request Forgery (CSRF) and inaccessible to domains of other origins than the one where the cookie got sent from.
- **Expires**: Since the JWE is only good for 15 minutes after the user authenticated, the cookie gets a similar Time-to-Live (TTL) mechanism.
- **Secure**: Making sure that a cookie is only transmitted over a secure connection through HTTPS. This encrypts the data being sent back and forth between the client and the server, attempting to avoid eavesdroppers.
- **HttpOnly**: Setting HttpOnly tells the browser to make this cookie inaccessible through JavaScript. This is important to avoid session hijacking.

4.5 Password Encryption

A password can be stored and hashed using a `salt` in a typical authentication scheme. The user will provide their UID along with their password, this will get collected from the database and checked with a random salt[40] that was generated when the user registered.

This complicates the process compared to what is described above, since the passwords should be a randomly generated string that is unknown to the applicant, we cannot hash it and compare the input from the user. We decided on the AES-256 (See Section 2.2), as a high form of encryption is paramount to secure our passwords. AES requires a 256-bit key to encrypt and decrypt the data. Since the CID generated by WebAuthn is unique and random for each application, this serves as a basis to construct the key.

The key for each password is based on the following premise:

1. We take advantage of the fact that each CID is unique in every application (See Section 2.1). So we grab 192-bits of the string that is the CID. This string is converted to integers. Since every CID is unique depending on device, we generate a padding in the situations where the key does not contain 192-bit to reach the remaining difference.

[37] A layer is referred to in this context as a wrapper around all other requests.

[38] To pass the JWT token back and forth between the server

[39] The cookie storage in a browser is subject to any vulnerabilities that can be performed on a SQLite database while having access to the computer where it is running.

[40] Salting is the process of adding a randomly generated string consisting of arbitrary characters to the password before creating a hash [See15].

Draft

2. Each key consists of a random salt of 128-bit of integers. These numbers are stored with the entry in the database.

3. Then we add 128-bit of pepper which is collected from the environment variable to finish it off.

These 448-bit are then used to serve as the input for the key-derivation function, Argon2 (See Section 2.3).

4.6 Hashing

When searching for a good key-derivation function, we first came across PBKDF2. We saw this as a good solution for the project but after performing more research into the topic, we ended up with Argon2. Since LessPM is required to run in as safe of an environment as possible, Argon2's configuration option seemed like the perfect solution. Upon hashing, each key is also subject to a 128-bit salt generated specifically for the key.

Draft

4.6.1 Configuring Argon2

Argon2's hashing output is dependable on configurations [al17].[41] Given that we emphasize security, we opted for the `Argon2id` configuration, giving us equal protection against side-channel and brute force attacks.

A part of the brilliant aspect that Argon2 has to offer is the option to set a required amount of memory to do the hashing. This forces a malefactor to use a specific amount of memory for each attempt at constructing the hash. The only way a malefactor can get passed this requirement is to purchase more memory.[42]

For LessPM, we are using 128 Megabyte of memory when constructing the hash.[43] We went for the default suggestion of two iterations to compliment the memory. To finalize the configuration, we added 8 degree of parallelism, since the system where we developed it consists of 8 cores [44]

4.7 JWT & JWE

A JWT is a base64-encoded string containing some data that can be used to validate an authenticated user between HTTP-requests (See Section 2.4). According to [JBS15], these should be attached to the authorization part of a request, along with a `Bearer` part. These tokens are cryptographically signed with RSASSA-PSS using SHA-512 with a key-pair stored on the server to avoid forgery, before being encoded and to a Base64-encoded string. For each step in registration 2.1.1, authentication 2.1.2, and password creation/retrieval 4.2.3, different tokens with different information are constructed, verified, and discarded.[45]

The tokens are not encrypted by default. It is out of the question to expose any sensitive details about the server's inner workings unnecessarily. Therefore, we decided to incorporate a part inspired from JWE.[46] Before any token is attached to a response from the server, the token is encrypted using AES. The Base64 gets encrypted into a ciphertext and then turned into a new Base64 encoding. An example of this can be found in A.1. We decode the Base64, decrypt the ciphertext, and then decode the JWT-Base64 to reconstruct the claim when a new request is performed to the server. This claim

[41] Dependable in this context is referring to each configuration that can possibly be constructed. An instance of Argon2 with 256 Megabyte of `memory` will not return the same hash as 255 Megabyte. The same is true for amount of `iterations` and `parallelism`.

[42] As a sidenote, we imagine that the increase of memory usage will scale as technology evolves and more memory is installed in computers.

[43] According to [al17], the reference implementation using Argon2d with 4Gb of memory and 8 degree pararellism should let the hashing process take 0.5 seconds on a CPU with 2Gz. However, we were unsuccessful in seeing anywhere close to similar results.

[44] The degree of parallelism is affected by many cores a CPU contains [al17].

[45] The token for password creation/retrieval is the same as being verified that a user is signed in authorized to create entries on behalf of the user.

[46] In a proper implementation of JWE, some metadata about the algorithm should be encoded into the token, such as what algorithm is used to perform the encryption [IET15]. For further expansion, see Section 6.

is then used to gather any details about the user that performed the request to further respond accurately to the request.

5 Summary

This report describes the details of implementing a passwordless password manager, using WebAuthn to authenticate users. The aim of the project was to create a secure and efficient solutions for managing passwords that doesn't rely on the need for a traditional password.

We came up with an intuitive way to encrypt passwords, instead of the traditional hash. To achieve this, AES-256 was applied and Argon2 was used to construct a hash out of a key supplied through WebAuthn's Credential ID and a randomly generated salt for both key-derivation function and the AES-256 key, combined with a pepper for the latter.

The backend where WebAuthn and the password manager was running consisted of a HTTPS server, serving content through a self-signed certificate. Because the server and client were independent of each other, we also configured CORS.

To keep a user authenticated between requests, we used an encrypted version of a JWT, inspired by the JWE RFC. This was achieved by storing the token in a fortified cookie in the client, being encrypted on the server before it was stored.

Despite its potential benefits, our report also highlights some limitations and challenges associated with passwordless authentication. These include the potential lack of adaptability by users and the possibility of losing the authenticator device.

Our findings suggest that future studies should explore ways to address the issue of device loss in the context of passwordless authentication. In particular, research should focus on developing methods for securely recovering access to accounts and data when a user's primary authentication device is lost or stolen. This could include the use of backup authentication methods, such as biometric verification or recovery codes, as well as the development of secure protocols for remotely revoking access to lost devices. By addressing this challenge, we can enhance the security and reliability of passwordless authentication systems.

6 Future work

Through implementing LessPM, we aimed to create a barebone implementation that could serve as a reliable Minimal Viable Product (MVP). However, we recognize that more work is needed to further enhance and compliment the product. The related topics to further improve LessPM are listed below and briefly discussed as a way to highlight potential drawbacks of the current version.

- **Authorization Headers**
  Even if [JBS15] specifies the `Authorization` header as the appropriate place to append the header, in a system using WebAuthn it would be better to construct a cookie with similar settings as mentioned earlier in the report. This is of particular concern as the HTTP framework we used requires to specify exposure of the `Authorization` header in order for JavaScript to read it. Exposing it to the client entails exposing it to malefactors as well.

- **Hashing the stored password**
  We emphasized and recognized that only constructing a hash for the AES key leaves the stored password exposed **iff** AES gets broken or have an unknown zero-day failure. Adding some form of hashing for the password before encrypting it would serve an exceptional benefit. As of thies writing, we are unsure how this hash would be properly implemented, given that we have nothing to construct the hash for the password.

- **Hardcoded AES for JWT**
  Each JWT is encrypted with the same AES. This serves as a great treat to the whole system and was implemented this way due to the lack

Draft

of time during the implementation phase. In a perfect scenario, we would have applied the same technique for the JWT token as for the passwords.

- **Encrypted Passkey**
  The Passkey that gets stored on the user object could be beneficial to encrypt. While the Argon2 hash of the key serves as a great way to require an attacker to get access to the codebase as well as the database, the passkey could be encrypted with AES and the Credential ID serve as part of the key to decrypt it.

- **Attaching connecting IP to JWT**
  We would have liked to attach the connecting IP address to the JWT. Seeing that a JWT token is exposed to the client and then a form of session hijacking, attaching an IP address to the token serves as a first-line defence in order to avoid exposure unencrypted tokens.[47]

- **Properly implement JWE**
  During development, we thought that JWTs were encrypted and not just signed. We discovered the JWEs [IET15] late in the process and these were new to us. Due to lack of time, we therefore took the shortcut of just implementing the encryption process of JWE, not the remaining metadata. In the future, we would like to properly implement these and follow the standard.

- **Multiple Authenticators**
  In its current implementation, LessPM supports a single registered authenticator per username to maintain a focused security approach. During registration, the server checks the database for an existing username similar to the incoming one and aborts the registration ceremony if a match is found. While WebAuthn permits users to have multiple authenticators, limiting this feature in the initial iteration of LessPM helps ensure a more controlled security environment. As the system evolves, considering addition of support for multiple authenticators can be weighed against potential security risks and benefits.

- **Return Passwords with Authentication**
  As part of the authentication process, we could make sure that the password list is returned with the last request to authenticate. This would further emphasize security by not exposing passwords in a separate end-point. On the same note, it would be good to handle the decryption part of the JWE better, seeing that a misconfigured cookie sent to this endpoint now returns an error. Yet another technical debt as a cause of lack of time. This is, however, not something that brings the system to a halt, rather an error that needs handling.

[47] This would not work on a mobile device connected to mobile data, seeing that the IP address switches between connections.

Draft

7 Acknowledgement

The implementation of this project was only possible with the dedicated and hardworking community surrounding Rust. As a thank you, we would like to acknowledge the following cargos and their hardworking developers: jsonwebtoken, ring, webauthn-rs.

While there are other libraries to mention, these are the most prominent ones that serve as the basis of the project. All the libraries are well-respected within the community, with the former two having several million downloads and the latter well over 100.000.

ChatGPT Throughout the report, the authors utilized ChatGPT to thoroughly reiterate concepts used during development.

References

[21a] *Web Authentication: An API for accessing Public Key Credentials Level 2*

- *Credential ID*. https://www.w3.org/TR/webauthn-2/#credential-id. Accessed: 2023-03-25. World Wide Web Consortium, 2021.

[21b] *Web Authentication: An API for accessing Public Key Credentials Level 2 - PublicKeyCredential identifier slot*. https://www.w3.org/TR/webauthn-2/#dom-publickeycredential-identifier-slot. Accessed: 2023-03-25. World Wide Web Consortium, 2021.

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

[al17] Alex Biryukov Et al. *Argon2: The memory-hard function for password hashing and other applications*. Tech. rep. Accessed on: March 25, 2023. University of Luxembourg, 2017. URL: https://github.com/P-H-C/phc-winner-argon2/blob/master/argon2-specs.pdf.

[App23a] Inc Apple. *Secure Enclave*. 2023. URL: https://support.apple.com/en-gb/guide/security/sec59b0b31ff/web.

[App23b] Apple Inc. *How to use Control Center on your iPhone, iPad, and iPod touch*. https://support.apple.com/en-us/HT208108. Accessed: March 30, 2023. 2023.

[Bon12] Joseph Bonneau. "The Science of Guessing: Analyzing an Anonymized Corpus of 70 Million Passwords". In: *2012 IEEE Symposium on Security and Privacy*. Last Accessed: 2023-03-28. 2012. DOI: 10.1109/SP.2012.49.

[Doo18] John F. Dooly. *History of Cryptography and Cryptanalysis: Codes, Ciphers, and Their Algorithms*. Springer, 2018, p. 5. ISBN: 978-3-319-90442-9.

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer-Verlag Berlin Heidelberg, 2002. ISBN: 978-3-540-42580-9.

[FID17] FIDO Alliance. *FIDO UAF Overview*. https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-uaf-overview-v1.1-id-20170202.html. Accessed: 2023-03-25. Feb. 2017.

[FLP85] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. "Impossibility of Distributed Consensus with One Faulty Process". In: *Journal of the ACM (JACM)* 32.2 (1985), pp. 374–382.

[Ghr+20] S. Ghrobany Lyastani et al. "Is Fido2 the kingslayer of User authentication? A comparative usability study of FIDO2 Passwordless Authentication". In: *2020 51st IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. 2020, pp. 102–108. DOI: 10.1109/SP40000.2020.00047.

[Gor21] Ionel et al. Gordin. "Moving forward passwordless authentication: challenges and implementations for the private cloud". In: *2021 20th RoEduNet Conference: Networking in Education and Research (RoEduNet)*. Last Accessed: 2023-03-28. IEEE. 2021. DOI: 10.1109/RoEduNet54112.2021.9638271.

[Hus22] Emin Huseynov. "Passwordless VPN Using FIDO2 Security Keys: Modern Authentication Security for Legacy VPN Systems". In: *2022 4th International Conference on Data Intelligence and Security (ICDIS)*. 2022, pp. 453–455. DOI: 10.1109/ICDIS55630.2022.00075.

Draft

[HXC18] Xincheng He, Lei Xu, and Chunliu Cha. "Malicious JavaScript Code Detection Based on Hybrid Analysis". In: *2018 25th Asia-Pacific Software Engineering Conference (APSEC)* (2018), pp. 1–5.

[IET15] IETF. *RFC 7516: JSON Web Encryption (JWE)*. https://www.rfc-editor.org/rfc/rfc7516.txt. Accessed: 2023-03-26. May 2015.

[JBS15] Michael B. Jones, John Bradley, and Nat Sakimura. *RFC 7519 - JSON Web Token (JWT)*. https://www.rfc-editor.org/rfc/rfc7519. Accessed: 2023-03-26. Internet Engineering Task Force (IETF), May 2015.

[Kar18] Chetan Karande. *Securing Node Applications: Protecting Against Malicious Attacks*. O'Reilly Media, Inc., 2018. ISBN: 9781491999644.

[Kor] Kornelski. *Lib.rs Stats*. https://lib.rs/stats. [Accessed: Mar 29, 2023].

[Lee+21] Yongki Lee et al. "Samsung Physically Unclonable Function (SAMPUF™) and its integration with Samsung Security System". In: *2021 IEEE Custom Integrated Circuits Conference (CICC)*. Last Accessed: 2023-03-28. Access through IEEE Explore. IEEE. 2021, pp. 1–4.

[Lev84] Steven Levy. *Hackers: Heroes of the Computer Revolution*. Sebastopol, CA: O'Reilly Media, 1984, pp. 85–102. ISBN: 978-1449388393.

[LH90] Karl J. Lieberherr and Ian M. Holland. "Assuring Good Style for Object-Oriented Programs". In: *IEEE Software* 7.5 (1990), pp. 38–48. DOI: 10.1109/52.35588.

[Mon21] MongoDB. *NoSQL Explained*. Last Accessed: 2023-03-29. 2021. URL: https://www.mongodb.com/nosql-explained.

[Mor+17] Michitomo Morii et al. "Research on Integrated Authentication Using Passwordless Authentication Method". In: *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*. IEEE. Tokushima University, 2017, pp. 895–900. DOI: 10.1109/COMPSAC.2017.198.

[Mulnd] Neal Muller. *Credential Stuffing*. https://owasp.org/www-community/attacks/Credential_stuffing. Last Accessed: 2023-03-28. n.d.

[Nat20] National Institute of Standards and Technology. *Special Publication 800-171 Revision 2: Protecting Controlled Unclassified Information in Nonfederal Systems and Organizations*. Page 24, Chapter 3. 2020. URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-171r2.pdf.

[NIS00] NIST. *Commerce Department Announces Winner of Global Information Security Competition*. Web page. Last Accessed: 2023-03-27. Oct. 2000. URL: https://www.nist.gov/news-events/news/2000/10/commerce-department-announces-winner-global-information-security.

[OWAnd] OWASP. *PHISHING IN DEPTH*. https://owasp.org/www-chapter-ghana/assets/slides/OWASP_Presentation_FINAL.pdf. Last Accessed: 2023-03-28. n.d.

[Par+22] Viral Parmar et al. "A Comprehensive Study on Passwordless Authentication". In: *2022 International Conference on Smart Computing and Data*

Draft

Science (ICSCDS). 2022. DOI: 10.1109/ICSCDS53736.2022.9760934.

[Pau17] James L. Fenton Paul A. Grassi Michael E. Garcia. *Digital Identity Guidelines: Authentication and Lifecycle Management*. Tech. rep. SP 800-63-3. National Institute of Standards and Technology, 2017. URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-63-3.pdf.

[Pol23] Polybius. *The Histories*. Accessed on: March 23, 2023. 2023. URL: http://www.perseus.tufts.edu/hopper/text?doc=Perseus%3Atext%3A1999.01.0234%3Abook%3D6%3Achapter%3D34.

[Rea] React developers. *React Component*. https://react.dev/reference/react/Component#setstate. Last Accessed 2023-03-23.

[Riv19] Elijah E. Rivera. *Preserving Memory Safety in Safe Rust during Interactions with Unsafe Languages*. Tech. rep. Defense Technical Information Center, 2019. URL: https://apps.dtic.mil/sti/trecms/pdf/AD1188941.pdf.

[Sav22] Justina Alexandra Sava. *Biometrics-enabled consumer device adoption in 2020*. Statista. Last Accessed: 2023-03-28. June 2022. URL: https://www.statista.com/statistics/1226096/biometrics-enabled-devices-by-region/?locale=en.

[Sch00] Bruce Schneier. *Secrets and Lies: Digital Security in a Networked World*. Wiley, 2000.

[Sch15] Bruce Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. 20th Anniversary Edition. Wiley, 2015. ISBN: 978-1119096726.

[See15] Alok Sharma Seema Kharod Nidhi Sharma. "An Improved Hashing Based Password Security Scheme Using Salting and Differential Masking". In: *2015 4th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO)*. IEEE, 2015, pp. 1–6. DOI: 10.1109/ICRITO.2015.7359225.

[Shi22] Shibboleth Consortium. *Shibboleth - Federated Identity Solutions*. Accessed: 2023-03-28. 2022. URL: https://www.shibboleth.net/.

[ST01a] National Institute of Standards and Technology. *Announcing the Advanced Encryption Standard (AES)*. Federal Information Processing Standards Publication 197. 3.1 Input and Output, p7-8. Last Accessed: 2023-03-26. NIST, Nov. 2001. URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf.

[ST01b] National Institute of Standards and Technology. *Announcing the Advanced Encryption Standard (AES)*. Federal Information Processing Standards Publication 197. Introduction, p5. Last Accessed: 2023-03-26. NIST, Nov. 2001. URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf.

[ST01c] National Institute of Standards and Technology. *Announcing the Advanced Encryption Standard (AES)*. Federal Information Processing Standards Publication 197. Algorithm Specification, p13-14. Last Accessed: 2023-03-26. NIST, Nov. 2001. URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf.

[ST01d] National Institute of Standards and Technology. *Announcing the Advanced*

Draft

Encryption Standard (AES). Federal Information Processing Standards Publication 197. Appendix C, C3, p42-46. Last Accessed: 2023-03-26. NIST, Nov. 2001. URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf.

[The] The Password Hashing Competition. *Password Hashing Competition*. https://password-hashing.net. Accessed on: March 25, 2023.

[W3C21a] W3C. *Web Authentication: An API for accessing Public Key Credentials - Level 2. Section 6.1 Authenticator Data*. Accessed: 2023-03-26. 2021. URL: https://www.w3.org/TR/webauthn-2/#authenticator-data.

[W3C21b] W3C. *Web Authentication: An API for accessing Public Key Credentials Level 2*. https://www.w3.org/TR/webauthn-2/#webauthn-relying-party. Accessed on: March 25, 2023. 2021.

[W3C21c] W3C. *Web Authentication: An API for accessing Public Key Credentials Level 2*. https://www.w3.org/TR/webauthn-2/#registration-ceremony. Accessed on: March 25, 2023. 2021.

[W3C21d] W3C. *Web Authentication: An API for accessing Public Key Credentials Level 2*. https://www.w3.org/TR/webauthn-2/#sctn-registering-a-new-credential. Accessed on: March 25, 2023. 2021.

[Wan+18] Chun Wang et al. "The Next Domino to Fall: Empirical Analysis of User Passwords across Online Services". In: Mar. 2018. DOI: https://doi.org/10.1145/3176258.3176332. URL: https://people.cs.vt.edu/gangwang/pass.pdf.

[web16] web.dev. *Credential Management API*. Last Accessed: 2023-03-29. 2016. URL: https://web.dev/security-credential-management/.

[Wor21] World Wide Web Consortium. *Web Authentication: An API for accessing Public Key Credentials Level 2*. https://www.w3.org/TR/webauthn-2/. Accessed: 2023-03-25. Apr. 2021.

[Yan+00] Jianxin Yan et al. *The Memorability and Security of Passwords: Some Empirical Results*. Tech. rep. UCAM-CL-TR-500. University of Cambridge, Computer Laboratory, Sept. 2000. URL: https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-500.pdf.

Draft

A Appendix

A.1 Base64 and ciphertext

```
uAaQghfTS0jpMA1WaYozepQ4/TpN13Y6tlKSzXG0l+epVZK+
   vjvD8BwMIlWVxvTaV3mcFxL665qBNsFg//81hpU8I660lq/LAXsPcdfq2vr8YRa14+GH+
   Gtw6YlqLrDU3E8Rhb/
   IlAvJ8u5VN8pwV1SmBZLTwL7AyEWlp4GodSrX4NSl5grIFoVwRq7kXofVu1aUToD6KJcmo0X

BnwG0KWpEPUZFPd76KcA/QfXHnJHQsaR2jKWFJdRCpbtJAacAbssJk/
   bJjioo3AS1caEVZbNJctp9xgqVvgQJPyhmYtMLqdjq/
   SocUscTrLSPiR2X0g5sWByNIm6ses2SJ3dtMYYeOr7+qtVOcoX+U7w2+
   uLorawsCcXCmXRunEKd5jXiydwXZjzPoKHaT8hGwDB8CNSHxg/
   JXrezEJ5JKwq3Gio3xEyjD09Cfq5qLn9kENbcDZ/uRZK6+
   Swxcqg1DYhGngPJbbPkbOpqcXxdLutybYgdGpFN0zCQw3/LNbxYzOBeVhVsXM+
   GOVEAcKgYpnyCILwKKtFUyaRX2Q7IjdBbKP8NpG7RWZKFtRBVc4YVdWSIRpfek1/
   lFkq1JrvgK/6KjyyR+
   m6sb2RzUzxNO1V5uTkH2m8cBUwQBUqjiNEgVQDzTeaYIZqH0j2Is4cblRcCsjoFgX3Nvh4/
   OvgpgQ==
```

```
KHO
Vi3z8:Mv:RqU;
```

```
" U W y 6 ‘5<#{qakpj.O U7pWTEu*
pF^ VN (&E)jD=FE=)?A    v % B I n              *(qeIr}
        ?(fb*RlX"nz  ’wm1: N r S 6  ,     tnB5f  <  OFO  |   ^I+
*7LOB~j.-d
E,1r5
Fmljq|].rmMO-5X^Vl\   9Qp   |
*T  W  [(
nYUsudF _J&+,nodsS<M;Uy9oLJ  U M !=:}  ?:)
```

Draft

Figure 6: The Base64 encoding and relevant ciphertext after AES-256.