CryptoChronicles

LessPM's Quest to a Passwordless Utopian Ecosystem

Confidential

T-742-CSDA

Håvard Nordlie Mathisen
Reykjavik University
havard22@ru.is

Instructor
Jacky Mallet

Department of Computer Science
Reykjavik University
April 16, 2023

Contents

Confidential

Abstract

We describe the implementation of LessPM, a passwordless password manager using Rust as a programming language. While traditional passwords have a long and respectable history, they have become increasingly vulnerable to attacks such as brute-forcing, phishing, and keylogging. LessPM seeks to address these deficiencies with a sophisticated authentication method, called WebAuthn. Through asymmetric cryptography, WebAuthn provides a secure authentication experience with the help of user-owned devices such as smartphones or hardware authenticator devices. This would eliminate the use of traditional passwords. To secure passwords stored in LessPM's, we employ AES-256 for encryption and Argon2 for hashing, WebAuthn's Credential ID serves as a key, combined with randomly generated salts and pepper.

We also use Hypertext Transfer Protocol Secure (HTTPS) to secure communication between client and server and configured Cross-Origin Request Sharing (CORS). Additionally, we used an encrypted version of JSON Web Tokens (JWT, RFC7519), inspired by JSON Web Encryption (JWE, RFC7516), to maintain user authentication between client requests.

We created a separate client as a visual proof-of-concept to adhere to a principle of least-knowledge, while the server handles requests and securely stores user data. The report provides valuable insights into the implementation and potential benefits of using WebAuthn and a passwordless approach, along with the security measures employed to protect user data on the server, offering promising prospects for a passwordless future in digital security.

Confidential

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 report explores the implementation of LessPM, a passwordless password manager that leverages WebAuthn (Web Authentication). LessPM aims to provide an experience free from the constraints and security implications of traditional passwords by placing a strong emphasis on security.

LessPM is designed with a multi-layer security approach to protect confidentiality and integrity of the user authentication process, and passwords stored for other registered services. By leveraging WebAuthn's asymmetric cryptographic[1] nature, LessPM creates a robust user authentication mechanism for sensitive data. This is achieved by utilizing authenticators, such as smartphones or hardware authenticator devices, with biometric scanners to authenticate a user through strong means. WebAuthn, an open standard developed collaboratively by The Fast IDentity Online Alliance (FIDO) and World Wide Web Consortium (W3C), aims to provide a more secure authentication experience through biometric scanners on these devices [W3C21b].

When a user initiates the process to register, LessPM issues a challenge constructed by WebAuthn to the client. The user is prompted to use a biometric scanner on their device, which then constructs a keypair, signing this challenge, before sending this information back to LessPM. LessPM receives and stores the public key-part of the keypair for later authentication. This process can be seen in Figure **??**.

Rework diagram to relate to LessPM

When the user attempts to sign in, LessPM again issues a challenge, prompting the user to use their

[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.

authenticator's biometric scanner to verify their identity and presence. This time, however, the challenge is signed by the private key on the authenticator device. The device yields the signature back upon the device successfully verifying the authenticity of the user. LessPM then verifies this signature with the help of the public key stored in the previous step. This process can be seen in Figure ??.

The aforementioned described processes constructs an environment such that a malefactor is required to first access the authenticator device, and then to bypass the device's biometric security in order to authenticate as the user in LessPM.

Passwords stored in LessPM are encrypted using AES-256, a widely recognized symmetric encryption algorithm [Sch00; DR02]. The encryption process includes using a Credential ID (CID) that is derived from the metadata of the public key in WebAuthn, a 128-bit randomly generated salt that is unique for each password, and a 128-bit pepper.

To maintain authentication, LessPM uses an encrypted version of JSON Web Tokens [JBS15] (JWT, RFC7519), inspired by JSON Web Encryption [IET15] (JWE, RFC7516) which are encrypted with AES-256 as well and attached to HTTPS requests.

The registration, authentication, and password creation/retrieval processes all use these encrypted tokens but LessPM manages and maintains different tokens for each step, providing an extra level of security. Each token is valid for a limited amount of time before expiring. An expired token prompts LessPM to deny access to unauthorized content and forcing the user to restart the process.

LessPM's client stores the token for authentication in a cookie. This cookie is protected by various built-in functionality in browsers, such as the attribute `secure` to transfer the cookie over a secure connection only, such as HTTPS. Other attributes include `HttpOnly` to make sure the cookie is inaccessible to JavaScript and Cross-Site Scripting (XSS), `SameSite=Strict` to ensure the cookie is only sent to the same origin it came from, and `Expires` to limit the amount the life-time of the cookie.

LessPM uses this cookie to verify the authenticity of requests between the client and the server. All data the server provides to the client is dependent on the validity of this cookie, and no information leaves the server unless it is valid.

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.

Confidential

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 ??), JSON Web Tokens (JWT. Section ??), JSON Web Encryption (JWE. Section ??), hashing through Argon2 (Section 2.1), and Advanced Encryption Standard (AES. Section ??).

2.1 Hashing through Argon2

3 Literature Review

The history of traditional passwords is a long and respectable one. 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 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).

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[2] and credential stuffing.[3] 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 1.

- **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 1: The Three Principles of Password Security [Sch00; Nat20].

Confidential

There are many approaches to passwordless authentication or a second step to authenticate with a one-time password.[4] 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

[2] 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].

[3] 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].

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

how fingerprints can be compromised [Par+22].[5]

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. 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 OpenStack environment using WebAuthn, which increases security and bypasses the risk of malefactors employing leaked passwords on other services.[6] The PA process considerably reduced 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].[7]

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 [Ghr+20].

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.[8]

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].[9]

As the technology evolved from FIDO to FIDO2.0, some security concerns persisted, such as session hijacking[10], which can compromise user accounts [Mor+17], highlighting the need to protect

Confidential

[5] 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] (PUF).

[6] See Section **??** for further explanation as to how this works.

[7] Authentication was provided through the Keystone environment on the OpenStack platform.

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

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

[10] 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 **??**.

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.

In the following section, we will examine the methodology employed in this study to integrate a passwordless system into a 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

In today's world of rapidly evolving technology, it is crucial to have a solid foundation in the underlying concepts and protocols that drive modern systems. This section aims to provide a comprehensive understanding of the key technologies and principles that are relevant to LessPM. As we explore the development and implementation of LessPM, our focus will be on the key components, technologies, principles, and steps that form LessPM's development process an encapsulation.

Our approach encompasses an explanation of the system architecture, as well as an in-depth description of the key components, technologies, tools, and steps that are involved in LessPM's encapsulation and development process to create the prototype. By thoroughly exploring these fundamental topics, we can better appreciate their significance and application in the context of the implementation, design, and architecture presented in LessPM.

One critical aspect of the development and system is its security and robustness. By providing a comprehensive account of the system development process, we aim to enable readers to understand the technical aspects of our work and assess the validity and relevance of our findings.

Confidential

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[11], which enables expeditious development and easy integration of various functionality. Utilizing Rust's distinctive features, the LessPM achieves enhanced security and reliability

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

in the context of user authentication [Riv19], showcasing the benefits of utilizing a modern programming language.

LessPM ran an instance of an HTTPS server, serving as a wrapper for the sensitive data.[12] 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. As a deliberate decision, we opted for MongoDB due to its NoSQL architecture, which facilitated the storage of Object-like structures [Mon21] including passwords, user accounts, and other related data that LessPM stores.

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 LessPM and proof-of-concept. 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. The design of LessPM was influenced by the principles of least-knowledge [LH90]. This influenced the decision to ensure that the client only retained necessary information to function properly, containing no knowledge of the server nor its implementation.

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[13] to attempt privilege escalation or accessing restricted data.

When initiating the project's development, we had a strong vision of creating a client that could seamlessly integrate with Chromium-based browsers as an extension.[14] However, we quickly discovered that the Rust cargo used to perform WebAuthn requests did not implement this functionality. 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.

LessPM's client maintains authentication and authorization through an encrypted JSON Web Token (JWT, RFC7519), inspired by JSON Web Encryption (JWE, RFC7516), passed back and forth between the server and client.

4.2 Authentication & Authorization

A system that maintains authentication and authorization through a stateless protocol such as HTTPS, requires some information to authorize a client between requested resources. LessPM took advantage of JWTs and an inspired variation of JWEs for this purpose. Although JWTs are not inherently encrypted [JBS15], combined with JWE, they serve as an essential backbone for secure data transfer in LessPM.

4.2.1 JSON Web Tokens

JWT is a compact, URL-safe string intended to transfer data between two entities, often represented as a Base64 encoding. They are often used as part of authentication and authorization scheme within a web service, application, or Application Programming Interface (API), and LessPM does this as well. The data in the string is delivered as a payload and is referred to as a `claim` [JBS15].

The 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 Base64 to ensure a URL-safe format. This permits larger bits of data

[12] 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 4.8 for further explanation.

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

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

Confidential

to be sent in a compressed, safe[15] format. 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 and permissions [Kar18]. According to [JBS15], these should be attached to the `Authorization` part of the HTTPS header, along with a hardcoded `Bearer` part, preceding the token as the value of the header.

Mention this in the security analysis.

In LessPM, the JWT claim contained data we could use to validate and authenticate a user between HTTPS requests.

LessPM utilizes two functionalities of JWTs to ensure the validity of the Base64-encoded token.

First, all tokens passed back and forth between the LessPM server and client receives an expiry time. We enforce this technique to ensure and maintain the security of accessed resources, requiring a user to reauthenticate when the expiry timestamp is passed.

Second, The tokens used in LessPM are cryptographically signed with RSA Signature Scheme with Appendix - Probabilistic Signature Scheme (RSASSA-PSS) using Secure Hashing Algorithms with 512-bit size (SHA-512), and is derived from the Rivest-Shamir -Adleman (RSA) encryption. The encryption is performed with a key-pair stored on the server to mitigate the risk of token forgery and so LessPM can maintain the validity and authenticity of the tokens created. This prevents unknown authorities from constructing or hijacking tokens belonging to LessPM.[16] Other signature algorithms that can be used include Hash-Based Message Authentication (HMAC), Elliptic-Curve Digital Signature Algorithm (ECDSA), or a plaintext secret stored on the server. LessPM uses RSASSA-PSS with SHA-512 due to the asymmetric operations the algorithms perform on the encryption and decryption, whereas HMAC and ECDSA is based on symmetric- and elliptic curve cryptography.

4.2.2 JSON Web Encryption

By default, JWTs are not encrypted. It is out of the question to unnecessarily expose any sensitive details about the LessPM server's inner workings. This lead to the decision of finding a way to encrypt the contents of the token before they are sent from LessPM's server. Without the encryption, any attacker could decode the Base64 encoding and gain access to the claim that LessPM uses to authorize and authenticate.

To perform this operation, LessPM seeks inspiration from JWEs and RFC7516 [IET15].[17]

Diagram that shows the operation

Before the RSASSA-PSS signed JWT is attached to the HTTPS header, LessPM encrypted the Base64-encoded data using AES-256, before turning them into a new Base64 encoded string. Encrypting the token with AES-256 elevated the overall security of the JWTs, making them more suitable for transferring sensitive data between the intercommunicating entities in LessPM. As a final step, the ciphertext created by AES-256 is again encoded into a new Base64. When LessPM again receives this claim, we used it to gather details about the user who made the request and responded to the request accordingly. An example of this encoding can be seen in Appendix A.1.

Turn this into a diagram, because Jacky love those.

The process functions as follows in LessPM:

1. The client completed an authentication request.

[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 LessPM's case this HTTPS.

[16] 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.

[17] To properly implement JWE, metadata should be encoded into the token, such as what algorithm performs the encryption [IET15]. For further explanation, see Section 6.

Confidential

2. A claim was constructed and created 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. The claim is encoded into a Base64 string.
5. The Base64 encoded string is passed to AES-256 encryption algorithm, creating a ciphertext.
6. The cipertext is created into a new Base64-encoding, which is the final token.
7. As part of the response to a request, the server appended the token.
8. The client received the token and carried it upon the next performed request.
9. When the server received the token again, it decodes the Base64-encoded ciphertext.
10. The ciphertext is decrypted, decoded and reconstructed to a claim.
11. The `exp` property of the JWT is validated and LessPM takes action accordingly.

Put this paragraph somewhere in the WebAuthn section Further, the server checked and verified the token before proceeding with any requests made to it. This information is updated and inspected between each re-render [Rea] of the client.

4.3 WebAuthn

We used WebAuthn to perform passwordless authentication in LessPM. WebAuthn is a collaborative open standard between the FIDO Alliance and W3C. The aim of the standard is to implement a secure, robust key-based authentication system for the web, to strongly authenticate users through biometrics [Wor21]. The concept relies on the use of a third-party device, called an Authenticator Device (AD), which leverages asymmetric cryptography. These devices employ biometric or hardware-based mechanism to provide secure and reliable means of authenticating a user in LessPM.

Upon registering in LessPM, the AD generated a key-pair called a Passkey. This Passkey contained a CID uniquely generated for each registered key-pair [21a; 21b], per services registered on the AD.

Further, if a malefactor 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 create a stronger level of security, whereas a traditional password exposed in a leak might leave a user susceptible to losing access across services and devices through the reused password. Preventing the potential security threat of password reuse is a top priority for LessPM. That is why WebAuthn has been chosen as the preferred standard for passwordless user authentication in LessPM.

WebAuthn further attempts 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, not just for LessPM, but for general authentication.

In order to increase the security of authenticated requests, each of the steps, Registration 4.3.1, Authentication 4.3.2 and Password- Creation & Retrieval 4.3.3) in WebAuthn receives different tokens with different information and is constructed, verified, and discarded separately.[18] We made this decision to further elevate security in LessPM, so that an exposed token could not serve as more than one entry point for one of the steps.

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

Confidential

4.3.1 Registration

When a user tried registering in LessPM, the client performs a registration request to the server, called a Relying Party (RP), carrying a relevant User Identifier (UID. i.e. username, phone number, email, Etc.). This is done to check whether a user with a similar UID existed[19] in the system, and LessPM denies the registration process if it does.

If there are no users with a similar UID, LessPM responds by initiating a Registration Ceremony (RC), generated a challenge and a Unique User Identifier (UUID) version 4, which serves as the body of the HTTPS response. LessPM also attached a JWE to the HTTPS response header's `Authorization` which is sent to the client.

The `expiration` time for this claim is set to one minute to allow the user some time to authenticate.[20] 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.

Using the issued challenge, LessPM's client called the browser-integrated WebAuthn Application Programming Interface (API)[21], prompting the user to utilize their AD to create a new Passkey credential through the Client To Authenticator Protocol (CTAP2)[22] for LessPM. At this point, it is entirely up to the user to decide what device to use to authenticate. For our development, we used an Apple iPhone 13 Pro Max, a Samsung S21, and a Samsung Galaxy A52 to test authentication.[23] We scanned the QR codes prompted through our phones, which initiated the key-pair generation on the AD after a successful biometric scan, such as facial recognition.[24]

Finally, the AD signs the challenge using the private key stored on the device. The created public key, signed challenge, and additional metadata are combined into a public key credential object, which is forwarded to the client through CTAP2 and then sent to LessPM in a new request. Before the new request reaches the RP/LessPM, an authentication middleware checks for the JWE and denies the request with an `Unauthorized` HTTPS status code if the request is invalid.[25] If the RP/LessPM can validate the authenticity of the signed challenge and public key through WebAuthn, the user was stored in the database, along with the UUID generated. Thus completing the RC.

This process can be sign in figure huhu.

Confidential

4.3.2 Authentication

When a user wishes to perform authentication (commonly referred to as `logging in`), much of the same procedure occurs. The user issued an authentication request in the LessPM's client, carrying the UID the user used to register. The client included this information in the body. Upon receiving an incoming request, the RP/LessPM checked the database for the containing UID. LessPM immediately rejected the request, should the user not exist in the database. If LessPM found an associated user with the UID in the database, the server responds by initiating and issuing an Authentication Ceremony (AC), collecting the public key associated with the user from the database and generating a new challenge. Upon validation[26] of the

[19] WebAuthn does not require to check whether the CID generated by the AD exists. However, accepting users with similar identifier might leave a risk of providing unauthorized access. Hence why LessPM performs such a check.

[20] 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 LessPM further, and JWEs require an expiry time (See Section 4.2).

[21] In LessPM's case, this is the *navigator.credentials* API provided by the browser.

[22] The user is prompted to use their AD to prove their presence, which can involve facial recognition, providing a fingerprint, or any other modality supported by the device that the user chooses.

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

[24] 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].

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

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

public key, LessPM creates a new JWE, which, like registration, also receives an `expiry` time of one minute for the user to authenticate, attached to the HTTPS `Authorization` header. The challenge is issued and LessPM's client then calls the browser-integrated WebAuthn API again, prompting the use of the original AD to validate and sign the challenge through CTAP2. Unlike the registration process, the AD now yields a signed signature based on the challenge issued by the RP/LessPM, which is transferred back to LessPM's client through CTAP2 with some AD specific data [W3C21a]. A new HTTPS request is then issued with the signed challenge and AD specific data and the RP/LessPM then validated the signature using the stored public key. LessPM considers the user authenticated if the RP accepted and validated the signed challenge. If the AC is unsuccessful at any point in the ceremony, the ceremony is aborted and considered invalid.

Upon success a new JWE is generated. This time, the `expiry` time is set to a 15-minute[27] timeframe, allowing the user some time to perform necessary activities within LessPM.
This process can be seen in figure HUHUHUH.

Confidential

4.3.3 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 in LessPM.[28]

The following options are presented to a user when they initiate a password creation process in LessPM:

- **User Identifier**: An identification that the user wants to associate with the password entry they are storing. Such as a username, phone number, or email.
- **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 is created and stored, the user is prompted to reauthenticate with their AD. The password is created and stored in LessPM if tha authentication process proves successful, in a similar manner as described for registration (See Section 4.3.1) and authentication (See Section 4.3.2).

To retrieve the plaintext version of the password, LessPM enforces a new authentication request through the AD. We made this decision in LessPM to attempt to assure the owner[29] of the password's presence[30]. In such a scenario where the owner left their computer open while logged in to LessPM, a malefactor would not be able to simply retrieve a password at will. To decrypt a password, LessPM also required the CID from the AD to perform the key-reconstruction for the encryption algorithm employed in LessPM. Further details about how encryption is achieved can be seen in Section 4.6.

4.4 Cors

Cross-Origin Resource Sharing (CORS) must be configured correctly when the server and client are running separately on different ports.

[27] The specification does not specify any upper- or lower bounds for the expiry time [JBS15]

[28] 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.

[29] In this context, we distinguish between `user` and `owner` as the person that created the password, not the person currently using LessPM.

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

When a web page tries to access a resource hosted on another domain, browsers perform an additional request to the server, called a "`preflight`". The preflight request determines whether the 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 contains some information about the origin, accepted Content-Type, and similar for the actual request.

The server responds 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`[31] which contained the two domains for the server and client, permitted credentials[32] and then permitted the two HTTP methods `POST` and `GET`. We also ensured the `Content-Type, Authorization`, and `Cookie` headers are permitted.

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

4.5 Cookie

JavaScript can access and manipulate `Cookies` [HXC18]. We utilized the browser's local cookie storage to attempt secure authentication between requests.[33] We attempted a couple of strategies listed below to fortify the cookie that LessPM set in the browser against a malefactor:

- **Strict SameSite**: This ensures that the cookie is safeguarded against Cross-Site Request Forgery (CSRF) attacks and remains restricted to its original origin domain.

- **Expires**: Once the system authenticated the user, it gave the cookie a Time-to-Live (TTL) mechanism similar to the JWE, which remained valid for only 15 minutes.

- **Secure**: We applied the `Secure` attribute to ensure that the cookie was only accessible through the HTTPS protocol. This protocol 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 property is important to mitigate session hijacking.

4.6 Password Encryption

A password can be stored and hashed using multiple vectors to increase security in a typical authentication scheme. Such measures include using a `salt`[34] and `pepper`[35]. When trying to authenticate, the user would provide their UID and password. These values are collected from the database, where a password can be stored in any form from plaintext to a hashed variation through a hashing algorithm such as PBKDF2, Argon2, or the SHA-family. After the values are collected from the database, the authentication scheme checks them against the provided values from the user. If a `salt` and `pepper` is part of the authentication scheme, they are both required to be appended (in the correct order) to the user-provided password, before the hashing and validation can take place. Upon successful validation, the user is considered authenticated and logged in.

Securing the user passwords in LessPM made this process more complicated than the one described above. Furthermore, since the passwords in a gen-

[31] In this context, we referred to a layer as a wrapper around all other requests.

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

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

[34] Salting is the process of adding a randomly generated string consisting of arbitrary characters to the password before creating a hash [See15]. This randomness of the salt makes identical passwords hash to different values, which can then be stored in the database.

[35] A pepper is the process of adding a hardcoded string to the password. Unlike the salt, the pepper is often stored in the code and used as an extra measure to increase security. A sufficently large pepper will require a malefactor to spend extra time to compute a hash.

Confidential

eral password manager should be a randomly generated string unknown to the applicant, we cannot hash the user-provided input and compare that hashed value. This influenced the decision to introduce encryption on the passwords in LessPM.

4.6.1 Advanced Encryption Standard

We decided on the Advanced Encryption Standard (AES) with a 256-bit key-size (AES-256). AES has, since its inception in 1998, become the gold standard for encrypting various information across applications [Sch15; DR02]. In 2001, it was adopted as the successor of the leading 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 encryption and security in LessPM through multiple rounds of processing by combining substitution and permutation [ST01c]. AES with 256-bit key length (hereafter referred to as AES-256) employs a 256 -bit key and consist of 14 rounds of encryption, offering an advanced level of security compared to its counterpats with shorter key lengths and fewer rounds [ST01d]. In each round of encryption, AES-256 undergoes four primary transformations, operating on a 4×4 block, as described below:

1. **SubBytes** is a non-linear substitution step where each byte is replaced with another according to a predefined lookup table.

2. **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.

3. **MixColumns** is a process that works on the columns of the state by combining the four bytes in each column through a mixing operation.

4. **AddRoundedKey** involves combining a subkey with the state[36] by applying a bitwise XOR operation.

The larger key-size in AES-256 provides an exponential increase in the number of possible keys for each password encrypted in LessPM, making it significantly more resilient to brute-force attacks and further solidifying its position as a robust encryption standard for safe-guarding sensitive information.[37]

4.6.2 AES-256 in LessPM

AES-256 requires a 256-bit key to encrypt and decrypt data. Since the CID generated by WebAuthn is unique and random for each application and device, this serves as a basis for constructing the key used to encrypt passwords in LessPM.

We based the key for each password on the following premise:

1. We took advantage of the fact that each CID is unique in every application (see Section 4.3). We therefore used 192-bits of the string that is the CID, converting it to integers. Since every CID is unique depending on the device, some CIDs are smaller than 192-bit and some are longer. To compensate for this, LessPM constructed and generated a random padding to reach the remaining difference when the CID is smaller than 192-bit.

2. We appended each key with a random 128-bit salt of integers and stored these bits with the entry in the database.

3. Then we add a 128-bit of pepper collected from an environment variable to in LessPM's code to finish the key.

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

[37] 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.

Confidential

We used these 448 bits as the input for the key-derivation function, Argon2 (See Section 4.7). Upon hashing, each key is also subject to a 128-bit salt explicitly generated for the password's key in LessPM. This process happened individually for all passwords constructed and stored within LessPM.

4.7 Hashing

When we searched for a good key-derivation function, we first came across Password-Based Key Derivation Function 2 (PBKDF2). We saw PBKDF2 as a good solution for the project, but after researching the topic further, we ended up with Argon2.

Argon2 is regarded by some to be more secure than PBKDF2 due to its modern design considerations, including memory-hardness and protection against side-channel attacks, which makes it more resistant to brute-force and rainbow table attacks. PBKDF2 offers to set an amount of iterations to construct the hash and which pseudorandom function to use.

4.7.1 Argon2

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 Key-Derivation Function (KDF) fills large memory blocks with pseudorandom data derived from the input parameters, such as password and salt.[38] The algorithm then proceeds to process these blocks non-linearly for a specific number of iterations [al17].

The KDF offers three configurations, depending on the environment where the function will run and what the risk and threat models are, which we take advantage of in LessPM. These can be seen in Figure 2

- **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.[39]
- **Argon2i** uses data-independent memory access. This configuration is more suitable for password hashing and key-derivation functions.[40] The 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 2: The three configurations offered by Argon2 [al17].

Since LessPM is required to run in as safe of an environment as possible, Argon2's configuration option is an excellent solution.

Argon2, as a memory-intensive hashing function, demands substantial computational resources from attackers attempting dictionary-[41] or rainbow table attack. This characteristic significantly hampers the feasibility of cracking passwords using such attacks, constructing an ideal scenario for LessPM's password vault. The algorithm's customizability allowed us to adjust its behaviour based on memory, parallelism, and iterations, catering to LessPM's security requirements and performance needs. All of these benefits contributed to why LessPM uses Argon2 instead of PBKDF2.

Confidential

[38] 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].

[40] 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.

[40] Due to the nature of prioritizing security, LessPM uses the third configuration.

[41] 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.

4.7.2 Configuring Argon2

Argon2's hashing output is dependable on configurations [al17].[42] Given that we emphasized security, we opted for the `Argon2id` configuration, which gave us equal protection against side-channel and brute-force attacks.

This paragraph needs to go somewhere in this section.
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 and key-derivation in various applications and systems.
In 2015, Argon2 won the Password Hashing Competition [The].[a]

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

A part of Argon2's customizability is the offer to set the option for a required amount of memory to do the hashing.

These options would force a malefactor to use a specific amount of memory for each attempt to construct the hash. The only way a malefactor can get passed this requirement is to purchase more memory.[43]

For LessPM, we used 128 Megabytes of memory to construct the hash.[44] We went for the default suggestion of two iterations to complement the memory. To finalize the configuration, we added 8 degrees of parallelism since the system where we developed it consists of 8 cores. [45]

The following section

4.8 Security Analysis

The security of a passwordless password manager is of paramount importance to protect sensitive user infromation and prevent unauthorized access. In this subsection, we conduct a comprehensive security analysis, identifying potential attack vectors and outlining the defensive measures that have been implemented to mitigate tehse risks.

The defensive measures include industry-standard security protocols such as HTTPS, JWT/JWE, Argon2 hashing, salt and pepper techniques, CORS configuration, and secure cookie management. Additionally, different JWTs are used for registration, authentication, and password creation/retrieval to minimize the risk of exposure. These defensive measures collectively aim to provide a robust and secure architecture for LessPM, safeguarding against various threats and ensuring the confidentiality, integrity, and availability of user data.

- **HTTP**

  Weakness:

  Any server running HTTP (Hypertext Transfer Protocol) is passing data between a server and a connecting client. With HTTP, this data is unencrypted. A malefactor can eavesdrop this data, performing a man-in-the-middle attack, replacing information, reading tokens, perform header inections, Etc. with this unencrypted traffic.

  Defensive Mechanism:

  Upgrading the HTTP connection to HTTPS ensures that communication between the client and server is encrypted through Transport Layer Security (TLS). It provides confidentiality and integrity of data transmitted over the

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

[43] As a side note, the increase in memory usage will scale as technology evolves and more memory becomes common.

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

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

Confidential

network, making it more difficult for attackers to intercept or tamper with sensitive information.

- **JSON Web Tokens (JWT)**

  Weakness:

  JWTs are signed with a secret specific to the server. However, if this secret is discovered or leaked, a malefactor could use this information to sign their own tokens and provide their own information. Since JWTs are signed and not encrypted, any malefactor who receives access to a token can decode the Base64 format and read the tokens information in clear text.

  Defensive Mechanism:

  Properly signining the tokens with a strong algorithm prevents information from being tampered with. An attacker can still read the information in clear-text but can not forge their own tokens. LessPM uses RSASSA-PSS with Sha-512 to sign and encode tokens, decoding them with the keypair when received. This ensures integrity and authenticity of the JWT, as the signature is verified by the server. RSASSA-PSS is a robust and secure signature algorithm that provides protection against various attacks, such as collision attacks and length extension attacks. To further impose a level of security, LessPM takes advantage of different tokens for registration, authentication, and authorization.

- **JSON Web Encryption (JWE**

  Weakness:

  JSON Web Encryption are JWTs signed and encrypted using an algorithm for signing and encryption. However, if this encryption algorithm is not potent or weak (such as a Caesar Cipher), an attacker can break the encryption or even encrypt their own tokens. JWEs are also subject to scenarios such as lack of key rotation, insecure length of the encryption key and improper implementation (See Section 6).

  Defensive Mechanism:

  To combat these weaknesses, LessPM implements encryption partly inspired by the JWE standard. LessPM encrypts the token using AES-256, a strong encryption scheme capable of 2^{256} different keys. This ensures that the data is confidentially kept and protected from unauthorized access. While inspired, it is important to note that JWEs have not been properly implemented in LessPM (See Section 6).

- **Hashing Passwords & Keys**

  Weakness:

  Hashing is a one-way process to convert information (such as passwords) into a fixed-size string of characters, typically a fixed-length hash value. However, the hashing process is susceptible to a malefactor using precomputed hash-values for large number of possible passwords, stored in a lookup table called a Rainbow table attack. Hashes are also vulnerable to dictionary attacks, and not being properly implemented without using a salt and preferable a pepper, or brute-force attacks.

  Defensive Mechanism:

  LessPM takes advantage of the latest within public hashing functionality, through Argon2. Argon2 constructs a memory-hard and computationally expensive hash that provides protection against brute-force, rainbow table, dictionary and side-channel attacks by requiring a significant amount of computational resources and time to compute the hash. Finally, LessPM takes advantage of using both a random salt unique for each password and a pepper stored in the source code of 128-bit for both. This creates a higher threshold for a malefactor by requiring access to the database and the source code to be able to quickly compute a hash. The salt and pepper are added to the key of the AES-256

- **Password Authentication**

Confidential

Weakness:

A password is something a secret that a user knows. Secrets belonging to an individual will always be potentially accessible to a malefactor. There are multiple vectors that can be used, such as a user being careless and writing the password down on a piece of paper or storing the secret in an unencrypted file. Further, a password is required to be stored somewhere for a server to authenticate the user, preferably with an email for recovery. This creates two new vectors for a malefactor to exploit, the persisted storage on the server or the recovery process.

Defensive Mechanism:

LessPM takes advantage of WebAuthn to avoid the usage of passwords to authenticate the user. The user is required to have two vectors for authentication; their authenticator device and the biometrics, which is the user itself. It is, however, important to mention that any authenticator device is vulnerable to being compromised. Should a device be compromised or infected with malicious software (Malware), it could be used to intercept or manipulate authentication requests.

- **Storing passwords in plaintext**

Weakness:

Storing a password can be done in plaintext in a database protected by a password. However, should an attacker get access to the database, all passwords and their plaintext are compromised. Storing the passwords in plaintext makes the information accessible to any individual with access to the database, which causes a privacy risk.

Defensive Mechanism:

LessPM encrypt all passwords with one of the strongest public symmetric encryption processes, AES-256. Each key for each password is unique, and consists of the 192-bit CID from WebAuthn generated when the user registered (See Section 4.6). This CID is unique to each service the user is registered with. Along the 192-bit CID, 128-bit is used from a salt and 128-bit used from a pepper. The salt is stored with the entry in the database. Using both salt and pepper enhances the security of the derived key.

- **Cross-Site Scripting, Cross-Site Request Forgery & Unauthorized Access**

Weakness:

Any HTTP server improperly configured is subject Cross-Site Scripting (XSS) and CSRF. XSS is a process where a malefactor could inject a malicious script from a different origin, and execute said script in the context of a user's browser. This could lead to a situation where a malefactor could get access tokens or cookies, or interact with the webpage a user is viewing through the script. CSRF is a type of security vulnerability where a malefactor tricks a user to unknowingly make unauthorized requests on a trusted website, such as a bank or similar. This can lead to actions being performed on a user's behalf unintentionally, without the user's consent, leading to unauthorized access.

Defensive Mechanism:

CORS is a measure that enforces a strict policy for which domains and services are permitted to access certain resources on the server. LessPM takes advantage of CORS by permitting the server itself and the client associated domain to access resources on the server. This is a preventive measure put in place to allow the communication between the client and the serve, even though they are running on different ports, and potentially different domains. This ensures that only authorized clients can access the server's resources, preventing unauthorized cross-origin requests and protecting against cross-site scripting (XSS) and cross-site

Confidential

request forgery (CSRF) attacks.

- **Cookies**

  Weakness:

  A cookie is vulnerable to many aspects of security. Improperly stored, a cookie can be accessed through JavaScript, sent to different domains, get hijacked, excessive expire time, or even poisoned.[46]

  Defensive Mechanism:

  LessPM only uses one cookie, which contains the encrypted JWT. The cookie is protected through built-in browser-features such as restricted to the same origin that the cookie came from, and cannot be sent anywhere else, further preventing XSS of sensitive information. The cookie is expired after 15 minutes, which is extensive amount of time for a user to have authorized access to their passwords, before the need to reauthenticate their identity. Upon creation, LessPM makes sure that the cookie becomes set to secure. This prevents the cookie from being sent over an insecure HTTP connection, limiting it to HTTPS. Finally, the cookie is HttpOnly, so that the cookie can't be access through JavaScript, reducing the attack vector of SS further.

5 Conclusion

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 high-

[46] Cookie Poisoning is the process where an attacker modifies the content of the cookie to inject malicious data or bypass security controls.

Confidential

light 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 this writing, we are unsure how this hash would be properly implemented, given that we have nothing to construct the hash for the password. A potential suggestion would be to apply similar methods to how the encryption was performed.

- **Hardcoded AES for JWT**
  Each JWT is encrypted with the same AES-256 key. This serves as a great treat to the whole system and was implemented this way due to the lack 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 of 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.

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

Confidential

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 and nothing about what error it is exposed to the requesting party.

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.

Confidential

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.

[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.

Confidential

[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 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:

Confidential

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 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/#registration-ceremony. 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.

Confidential

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 !=:}  ?:)
```

Confidential

Figure 3: The Base64 encoding and relevant ciphertext after AES-256.