CryptoChronicles

LessPM's Quest to a Passwordless Utopian Ecosystem

T-742-CSDA

Håvard Nordlie Mathisen
Reykjavik University
havard22@ru.is

Instructor
Jacky Mallet

Department of Computer Science
Reykjavik University
March, 2023

1 Executive Summary

Test[**lol**]

2 Abstract

3 Introduction

Polybius' *The Histories* [10] 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 [4], 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 [8]. 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 [2]. 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.

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.

In 2004, a study titled *"The Memorability and Security of Passwords"* [22] was conducted into advising users on the entropy of passwords and ways someone can use to remember a or multiple passwords. A typical standard for larger organizations with a form of password creation system is to emphasize the diversity of smaller characters, capitalized characters, length, and not be commonly referred to in a dictionary [22]. The study analyzed the effectiveness of different password-creation strategies, suggesting that acronym-based passwords offer a delicate balance between memorability and security [22].

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) [5] and WebAuthn [21] 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, WebAuthn 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 delves into 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.

4 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 4.1, JSON Web Tokens (JWT) 4.4, JSON Web Encryption (JWE) 4.4, hashing through Argon2 4.3, and Advanced Encryption Standard (AES) 4.2.

4.1 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 [21]. 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 generates a key pair called a Passkey. This Passkey contains a credential ID uniquely generated for each registered key-pair [19, 20] on the Authenticator. 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[18].

[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.*

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

4.1.1 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. 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.

The client then calls the browser-integrated WebAuthn API, requesting the Authenticator (i.e., phone, hardware authenticator device, or similar) to create a new public key credential.[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.

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 1.

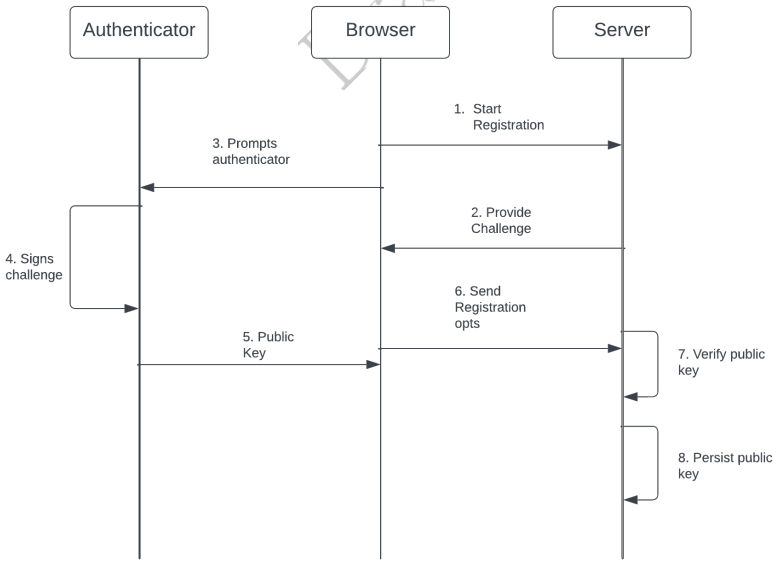

Figure 1: A diagram dipicting the registration process through WebAuthn and Authenticator Device.

[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.

4.1.2 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 [17]. 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.

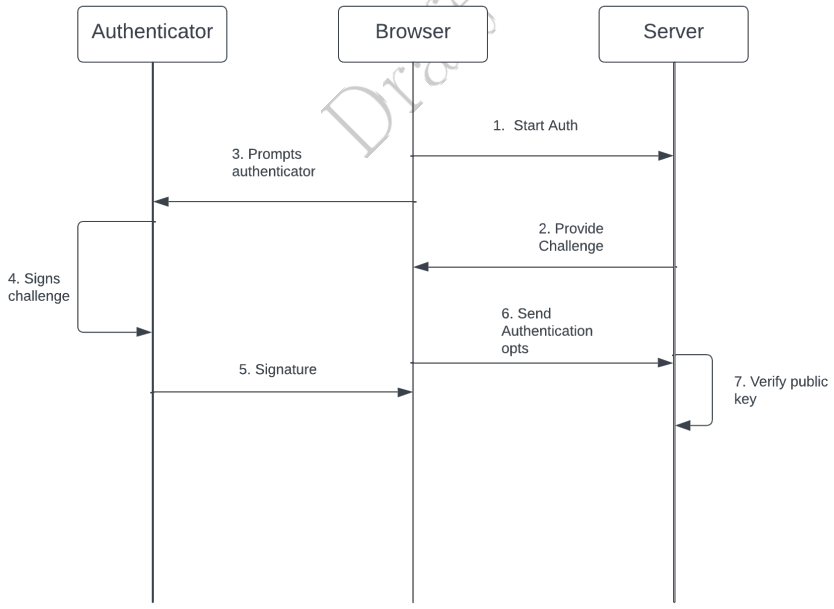

Figure 2: A diagram dipicting the authentication process through WebAuthn and Authenticator Device.

[5] In the case of LessPM, the system sets an authentication duration of 15 minutes. It stores this information within an AES256 encrypted 4.2 JSON Web Token. For more details about this process, refer to Section 4.4.

4.2 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 [11, 3], being adopted as the successor of the Data Encryption Standard (DES) by The National Institute of Standards and Technology (NIST) in 2001 [9]. AES operates on fixed-sizes units of data referred to as `blocks` [12], supporting keys of sizes 128-, 192-, and 256-bit [13]. 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 [14]. 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 [15]. In each round of encryption, AES-256 undergoes four primary transformations, operating on a 4x4 block:

- **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 [14].

The larger key-size in AES-256 provides an exponential 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]

4.3 Hashing through Argon2

Argon2 is a key-derivation function developed by Alex Biryukov, Daniel Dinu, and Dmitry Khovratovich [1]. Argon2 aims to provide a highly customizable function tailored to the needs of separate implementations [1]. Additionally, the design offers resistance to both time-memory trade-off and side-channel attacks as a memory-hard function [1]. The function fills

[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.

[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.

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 [1].

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 [1].

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. 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.[16].[13]

4.4 JSON Web Token & JSON Web Encryption

JSON Web Token (JWT) was introduced as part of RFC 7519 in 2015 [6]. It is a compact URL-safe string intended to transfer data between two entities. They can be used as part

[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 [1].

[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 6.1.

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

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

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 referenced as a `claim` [6].

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 [6].

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 [6]. This prevents unknown authorities from constructing or hijacking existing tokens.[14] The URL-safe format of a JWT is often performed through a Base64 string, which permits larger bits of data to be sent in a compressed, safe[15] format [6]. JWTs are widely used for scenarios like single sign-on (SSO), user authentication, and securing API endpoints by providing an efficient, stateless mechanism for transmitting information about the user's identity, permissions, and other relevant data [7].

5 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. This section will discuss the WebAuthn standard, its effective integration and role in LessPM, and the various security measures contributing to a robust and reliable solution.

6 Key

6.1 Hashing AES-key

7 Conclusion

8 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

[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).

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**
- **AES Key Encryption for Password**
- **Hardcoded AES for JWT**
- **Encrypted Passkey**

References

[1] 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.

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

[3] 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.

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

[5] 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.

[6] 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.

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

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

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

[10] 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.

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

[12] 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.

[13] 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.

[14] 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.

[15] 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.

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

[17] 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.

[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.

[19] *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.

[20] *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.

[21] 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.

[22] 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.