JAGZPASS RESEARCH

Password Managers

Password managers are programs that store user credentials such as usernames and passwords securely so users will not have to remember their own credentials and will also be able to use higher entropy passwords that wouldn't be as common or easy to remember. Users access their credentials using a master password, which is first processed by a Key Derivation Function (KDF) to produce a cryptographic key that is then used to encrypt all stored credentials in an encrypted vault file.

There are many different kinds of password managers. Some are cloud-based and offer users access to their vault across multiple devices by syncing encrypted data to remote servers. Others are self-hosted or offline by design, giving users or organisations more direct control over how and where their vaults are stored. Most modern password managers also support features like browser integration for autofilling credentials, mobile app support, password sharing, and breach alerts.

While these features improve usability, they also introduce new points of failure as they increase the threat surface. These risks are further explored in later sections through real-world incidents where these convenience features were exploited.

Security Model of a Password Manager

A password manager is designed to remain secure even if the encrypted vault is exposed. The core of a password manager's security model relies on a single secret: the master password. This introduces a deliberate single point of failure; a trade-off for convenience and centralized credential management.

Understanding what makes a password manager secure starts with identifying the kinds of attacks it needs to defend against. These threats can target everything from the encrypted vault itself to how users interact with the system, and they directly influence the design choices behind secure password storage.

Common Threats to Password Managers

- **Vault Theft**

  The encrypted vault file may be copied or exfiltrated from the local machine. An attacker with persistent access could try multiple attempts over time or move the file to more powerful hardware for cracking.

- **Brute-Force Attacks**

  Systematic attempts are made to guess the master password, potentially leveraging high-performance hardware (e.g., GPUs or ASICs) to compute key derivations quickly.

- **Rollback Attacks**

An attacker may try to revert the .lock file (which enforces lockouts) to a prior state to bypass time-based lockout restrictions and continue making password guesses.

• Credential Stuffing

A user may choose a weak or reused master password, reducing the entropy available to the Key Derivation Function and making the system easier to attack.

• Vault Tampering

An attacker may attempt to modify or entirely replace the encrypted vault file so that users cannot retrieve their credentials.

• Phishing and Social Engineering

While a password manager can protect a user's credentials from attackers, they are entirely reliant on the user keeping the master password confidential. The biggest vulnerability to a password manager is the user unknowingly leaking their own master password to a malicious party who may be manipulating the user.

To address these threats, password managers rely on a range of defenses. These mechanisms are designed to slow down attackers, detect tampering, and make successful compromise significantly more difficult. The following list outlines some of the key defenses commonly implemented in secure password managers.

Defensive Mechanisms

- **Computationally Expensive KDFs**
  A computationally expensive KDF slows dictionary and brute force attacks significantly, whether it be with the time needed with the number of iterations the generate a key or the memory cost.
- **Authenticated Encryption**
  Authenticated encryption ensures that modified cyphertext (credentials in this case) cannot be decoded even if the encryption key is found. This protects against tampering.
- **High Entropy Master Password**
  A master password with high entropy is harder to guess or crack. Enforcing complexity (length, character diversity) reduces the likelihood of successful brute-force or dictionary attacks.
- **Lockout Mechanisms and Rollback Protection**
  Systems can implement login attempt limits or time-based lockouts to reduce the number of guesses an attacker can make. Rollback protection ensures the lockout state cannot be bypassed by restoring a previous version of the state file.
- **Two Factor Authentication**
  2FA adds a second layer of identity verification (e.g., TOTP or hardware token), making it more difficult for attackers to access the system even if the master password is compromised.
- **Password Reuse Detection**
  Password managers can prevent users from reusing passwords within their vault. This limits the impact of any single credential being leaked or guessed, reducing the risk of lateral compromise.

While these defenses work together to protect the overall system, certain components are especially critical to get right. One of the most important among these is the **Key Derivation**

Function (KDF); the mechanism that turns a user's master password into an encryption key. The next section explores what KDFs are, why they matter, and how different algorithms compare in terms of security.

Key Derivation Functions (KDFs)

Purpose and Role

Key Derivation Functions are used to derive a unique cryptographic key from the master password that is used to encrypt all credentials in the vault. They are essentially hashing functions but they have more features that make them more ideal for a password manager.

Features of KDFs

- **Salting**
  A salt is essentially a random string that is generated and added to the master password by the KDF prior to hashing. Adding a salt prevents rainbow table attacks which rely on attackers knowing the hashing function used in the KDF. By adding a salt to the master password, even if the attacker knows the hashing function and the master password, they still won't be able to get the encryption key.
    - **Note:** Some systems also use a *pepper* which is also essentially a salt, but it is not stored alongside the derived key, but in a secure server instead. While peppering adds another layer of defense, it requires secure, separate storage and is less common in local password managers.
- **Iteration/time cost**
  KDFs can hash the salted master password multiple times; this increases the time complexity of the KDF and also helps delay brute force/ dictionary attacks.
- **Memory cost**
  The amount of RAM that a KDF uses is also key in preventing attacks since higher memory usage would make attacks more expensive and less feasible as the KDF would require more physical memory.
- **Parallelism**

  In addition to iteration and memory costs, having the KDF require multiple threads would also make it resource intensive and expensive to run for attackers.
- **Output length**
  KDFs allow for the length of the derived key to be specified. This is important for when a specific key size is needed for the encryption algorithm being used for the passwords.
- **Tunability**
  This is essentially the ability to modify or control all the previous points; some KDFs don't support some of the features above. Being able to change all the above parameters contributes largely to how safe the vault is since knowing what hashing algorithm is being used is not going to be sufficient for attackers if they don't know all these custom parameters.

Comparison of KDFs

SHA-256

SHA-256 is a fast, general-purpose cryptographic hash function. Though it is cryptographically secure when it comes to data integrity, it is not ideal for password hashing particularly because it doesn't support any cost parameters like time or memory. However, SHA-256 is extremely collision resistant, with the chances of a collision being somewhere around 1.2×10^{77}. When it comes to resistance to brute force, dictionary and rainbow table attacks, SHA-256 is vulnerable to modern GPU speeds. The table below presents data collected using a hypothetical RTX 4090.

Number of characters	Numbers Only	Lowercase Only	Upper and Lower	Number, Upper, Lower	Number, Upper, Lower, Symbols
6	Instantly	Instantly	Instantly	Instantly	Instantly
7	Instantly	Instantly	Instantly	Instantly	14 minutes
8	Instantly	Instantly	11 minutes	41 minutes	21 hours
9	Instantly	Instantly	9 hours	2 days	3 months
10	Instantly	27 minutes	19 days	3 months	22 years
11	Instantly	12 hours	2 years	19 years	2052 years
12	Instantly	13 days	141 years	1164 years	195k years
13	2 minutes	9 months	7332 years	73k years	19m years
14	19 minutes	24 years	381k years	4474k years	1760m years
15	4 hours	605 years	19m years	277m years	167.2b years
16	2 days	15732 years	1031m years	18b years	16t years
17	14 days	410k years	54b years	1067b years	1509t years
18	5 months	11m years	2788b years	67t years	144q years
19	4 years	277m years	145t years	4099t years	14Q years
20	37 years	7189m years	8q years	255q years	1294Q years

SPECOPSSOFT

A single 4090 can crack SHA-256 hashed password up to 9 characters in length within 3 months no matter how complex it is. Attackers might have more resources available than a single one of these consumer grade GPUs, which would slash these times further. Considering the likelihood of users using a master password that is "easy to remember", SHA-256 proves to be an non ideal hash for a password manager's KDF. Attackers with more resources will be able to brute force the hash significantly faster than this data shows.

PBKDF2 (HMAC-SHA256)

PBKDF2 is a widely supported and standardised KDF. It uses HMAC (which is the addition of a secret key to ensure integrity) over a hash function (SHA-256) repeated many times to slow down brute-force attacks. HMAC allows for the PBKDF2 to act as a pseudo random function which will make its output seem random to attackers even if it's fixed length, but since HMAC hides a key in the hash, we will be able to verify that it is a valid hash that has not been tampered with in case of a vault tampering attack. Since it runs SHA-256 over several iterations, PBKDF2 does take significantly longer than SHA256 on its own. However,

it does not consume significant memory, which still makes it vulnerable to GPU attacks. For example, benchmark data from Hashcat shows that an RTX 4090 can perform PBKDF2-HMAC-SHA512 at around 2825700 hashes per second with 1023 iterations, which drops to roughly 4800 guesses per second when extrapolated to 600,000 iterations. While this is a major slowdown, it's still fast enough to brute-force low-entropy passwords in days, especially with multiple GPUs.

bcrypt

bcrypt was designed to improve upon PBKDF2 by introducing built-in salting and a work factor (called "cost") that exponentially increases the hashing time. However, bcrypt uses fixed memory and is thus susceptible to optimisation on parallel hardware. Its 72-byte password length limit can also be a constraint. On a single RTX 4090, bcrypt does make a convincing case for itself, with a decently safe password starting with 8 characters, containing digits and symbols along with upper and lower case letters, and a safe upper and lower case letter password taking 9 characters. It is important to keep in mind that the times presented below are of a single GPU; these times might significantly decrease when more resources are thrown at it.. Moreover, because bcrypt does not consume large amounts of memory, it is still not as resistant to brute forcing as some of the other KDFs considered. There is also a paper by (OpenWall), Malvoni and Knezovic describing a hybrid a system of ARM/FPGA SOCs to attack the algorithm.

Number of characters	Numbers Only	Lowercase Only	Upper and LowerCase	Number, Upper, Lower	Number, Upper Lower, Symbols
6	Instantly	7 minutes	7.5 hours	22 hours	11.5 days
7	Instantly	3 hours	16.2 days	8 weeks	3 years
8	3 minutes	4 days	2.4 days	9.5 years	286 years
9	23 minutes	2.8 months	120 years	583 years	27154 years
10	3.8 hours	6 years	6228 years	36160 years	2579596 years
11	38 days	161 years	323856 years	2241941 years	245061585 years
12	15 days	4169 years	16840527 years	139000337 years	23280850.6 thousand years
13	5.2 months	15483 years:	875707453 years	8618021 thousand years	2211681 million years
14	4.3 years	2779344 years	45536787 thousand years	534317295 thousand years	210109676 million years
15	44 years	72262968 years	2367912 million years	33127672 million years	19960419.3 billion years
16	431 years	1878837183 years	123131474 million years	2053916 billion years	1896240 trillion years
17	4309 years	48849767 thousand years	6402837 billion years	127342773 billion years	180142784 trillion years
18	43084 years	1270094 million years	332947505 billion years	7895252 trillion years	17113565 quintillion years
19	430840 years	33022443 million years	17313271 trillion years	489505617 trillion years	1625789 quadrillion years
20	4308396 years	858583501 million years	900291 quintillion years	30349349 quintillion years	154449919 quadrillion years

OWASP recommends that for passwords, bcrypt must have a cost factor of 10 minimum; this equates to about 2^10 rounds of the blowfish cipher being run to hash a password, just to have a safe amount of protection from brute forcing.

scrypt

scrypt addresses bcrypt's shortcomings by introducing memory-hardness, deliberately requiring large amounts of RAM during computation to limit the effectiveness of parallel attacks. Its parameters: N (CPU/memory cost), r (block size), and p (parallelism) can be tuned to increase both time and memory per hash, following the formula:
Memory = 128 × N × r × p bytes.
For example, a typical configuration of $N = 2^{14}$, r = 8, p = 1 requires approximately 16 MB of memory and takes around 500 ms per hash. This makes scrypt highly resistant to GPU and ASIC acceleration. Even an RTX 4090 can only compute about 7126 hashes per second, rendering large-scale brute-force attacks impractical. In the context of a password manager, this delay is acceptable for users but acts as a powerful deterrent to attackers.

Argon2id

Argon2id is the most modern and secure option among mainstream key derivation functions. Like scrypt, it is memory-hard and designed to throttle parallel brute-force attempts using GPUs or ASICs. It combines two earlier variants: **Argon2i**, which resists side-channel attacks through input-independent memory access, and **Argon2d**, which is faster but more aggressive against parallel cracking via input-dependent access. This hybrid design gives Argon2id strong protection on both fronts.

While its parameters: memory size, iteration count, and parallelism function similarly to scrypt's, Argon2id tends to offer better side-channel resistance and flexibility. Real-world benchmarks show that even an RTX 4090 can only perform ~1,900 Argon2id hashes per second (from a source that didn't disclose the parameters used). Meanwhile, research by Red Hat estimates that cracking an 8-character Argon2-derived password could require 75,000 machines running for a decade, at a cost exceeding $4 billion.

Encryption

Purpose and Role

While the master password is considered the primary vulnerability in a password manager, relying solely on authentication for security is insufficient. Attackers may bypass the login mechanism entirely by directly targeting the encrypted vault file.

Encryption addresses this risk by making sure that, even if the vault file is obtained, its contents remain unintelligible without the correct decryption key. It decouples the protection of the credentials from the authentication system giving a password manager its second layer of defence. So even if a user's vault is stolen, encryption ensures that attackers cannot derive credentials from the vault. Additionally, some encryption schemes also protect the integrity of the stored credentials as any modifications would lead to text that cannot be decrypted . These encryption schemes are known as authenticated encryption schemes.

Security Incidents in Existing Password Managers

To better understand the practical risks faced by password managers, several high-profile security incidents were examined. These cases were selected because they highlight different types of failure. From poor infrastructure design and unsafe defaults to vulnerabilities in delivery mechanisms like autofill, they reveal how even widely used and well-resourced password managers can become compromised when assumptions about trust, usage, or security boundaries break down.

LastPass Data Breaches (2022 – 2023)

LastPass, a password manager with a userbase over 25 million strong was breached twice in recent years. The first breach, perpetuated by a compromised developer account, led to some source code and internal documentation being stolen. This led to the attackers having access to LastPass' cloud based storage environment, exposing a backup vault with some unencrypted customer data to the attackers. The second threat, was a targeted attack on a specific senior DevOps engineer, only one of four, who had access to encryption keys to vaults, and in this case, the vaults that were previously obtained. This engineer's home computer was compromised by a keylogger, which recorded his master password, allowing the attackers access to the employee's corporate vault and obtain the keys to decrypt the backup vaults they got in the first breach.

Now, 2 years down the line, the attackers have been confirmed to have used the information they retrieved (customer metadata in addition to their passwords) to have stolen at least $150 million USD.

Passwordstate (2021)

Passwordstate is an enterprise self-hosted password manager by Click Studios. Unlike LastPass, who offered a cloud-based solution, Passwordstate allowed users to store their own vaults locally. In April 2021, attackers gained access to Click Studios' CDN (Content Delivery Network) which provided Passwordstate with regular updates. The attackers were able to upload their own malware onto the CDN, to be distributed to Passwordstate users along with their regular updates. The malware was set to exfiltrate data from user devices once a day. Click Studios did not discover the breach for 28 hours, and that relatively small window of opportunity "limited" the number of customers that were affected. As Click Studios rushed to push out fixes, the attackers took advantage of the breach notes that the customers were receiving. They started phishing customers using notfications similar to official Passwordstate notifications, which led to another link with malware meant to steal more credentials.

AutoSpill (2023)

AutoSpill was a vulnerability disclosed in late 2023 by researchers at the International Institute of Information Technology, Hyderabad. The issue affected the autofill functionality of Android's password management ecosystem, including popular apps like 1Password, LastPass, Keeper, Dashlane, and others. The vulnerability emerged due to a misalignment between Android's Autofill framework, WebView components, and password manager logic.

In affected password managers, credentials autofilled into WebView-based login forms could unintentionally be exposed to the hosting app's native layer, meaning a malicious app could see the credentials even if it wasn't meant to. This violates the common assumption that autofilled credentials are only delivered to the domain or service they belong to. While this didn't lead to mass exploitation in the wild, it broke a core promise of autofill isolation and introduced new attack surfaces on Android.

The attack required a user to install a malicious app that either mimicked a legitimate service or injected JavaScript into a login page to capture autofilled credentials. It only exposed a single credential at a time, but the exploit was silent and difficult to detect. Some password managers, like 1Password and Dashlane, quickly acknowledged the issue and released patches. Others, including Google, shifted the responsibility to developers, calling it a misuse of Android's APIs rather than a flaw in the OS itself.

AutoSpill didn't expose vaults or encryption flaws, but it demonstrated that secure storage isn't enough if credentials are leaked during delivery.

Bitwarden Autofill Exploit (2023)

In early 2023, researchers at Flashpoint disclosed a security issue in Bitwarden, one of the most widely used open-source password managers. The flaw stemmed from Bitwarden's autofill functionality, which would fill in credentials into embedded iframes, even when those iframes originated from a completely different domain. In scenarios like these, the credentials could be harvested silently by a malicious iframe and forwarded to an attacker-controlled server.

Moreover, Bitwarden was found to autofill login credentials on untrusted subdomains. This meant that an attacker who could host content on a subdomain of a legitimate service could trigger autofill and collect the login details of unsuspecting users.

Although autofill on page load was disabled by default, users who enabled it were exposed. Bitwarden initially left the behavior unchanged for compatibility with legitimate sites using iframes (like Apple's login system). However, after public pressure, Bitwarden committed to limiting iframe autofill to trusted domains, and introduced warnings and approval prompts for autofill in risky contexts.

References

Alder, S., 2021. *PasswordState Password Manager Supply Chain Attack Delivers Password-Stealing Malware*. [online] The HIPAA Journal. Available at: www.hipaajournal.com/passwordstate-password-manager-supply-chain-attack/ [Accessed 26 Jul. 2025].

Ars Technica, 2023. *How worried should we be about the "AutoSpill" credential leak in Android password managers?* [online] Available at: arstechnica.com/security/2023/12/autospill-password-manager-leak-vulnerability-android/ [Accessed 26 Jul. 2025].

Chick3nman, [n.d.]. *Hashcat PBKDF2 Benchmarks and Notes*. [online] GitHub Gist. Available at: gist.github.com/Chick3nman/32e662a5bb63bc4f51b847bb422222fd [Accessed 26 Jul. 2025].

Flashpoint, 2023. *Bitwarden flaw can let hackers steal passwords using iframes*. [online] BleepingComputer. Available at: www.bleepingcomputer.com/news/security/bitwarden-flaw-can-let-hackers-steal-passwords-using-iframes/ [Accessed 26 Jul. 2025].

Hashcat.net, 2023. *PBKDF2 Performance Benchmarks*. [online] Available at: hashcat.net/forum/thread-11277.html [Accessed 26 Jul. 2025].

Kost, E., 2025. *The LastPass Data Breach (Event Timeline and Key Lessons)*. [online] UpGuard. Available at: www.upguard.com/blog/lastpass-vulnerability-and-future-of-password-security [Accessed 26 Jul. 2025].

Krebs, B., 2025. *Feds Link $150M Cyberheist to 2022 LastPass Hacks*. [online] KrebsOnSecurity. Available at: krebsonsecurity.com/2025/03/feds-link-150m-cyberheist-to-2022-lastpass-hacks/ [Accessed 26 Jul. 2025].

Malvoni, K. and Martinovic, I., 2014. *Are you sure you want to use bcrypt?* In: 8th USENIX Workshop on Offensive Technologies (WOOT 14). [online] USENIX Association. Available at: www.usenix.org/system/files/conference/woot14/woot14-malvoni.pdf [Accessed 26 Jul. 2025].

Preziuso, M., [n.d.]. *Password Hashing: PBKDF2, scrypt, bcrypt, and Argon2*. [online] Medium. Available at: medium.com/@mpreziuso/password-hashing-pbkdf2-scrypt-bcrypt-and-argon2-e25aaf41598e [Accessed 26 Jul. 2025].

Specops Software, 2023. *SHA256 Hashing Password Cracking*. [online] Available at: specopssoft.com/blog/sha256-hashing-password-cracking/ [Accessed 26 Jul. 2025].

Specops Software, [n.d.]. *Hashing Algorithm Cracking: Bcrypt Passwords*. [online] Available at: specopssoft.com/blog/hashing-algorithm-cracking-bcrypt-passwords/ [Accessed 26 Jul. 2025].