Zero Knowledge (About) Encryption: A Comparative Security Analysis of Three Cloud-based Password Managers

Matteo Scarlata [1], Giovanni Torrisi [2 †], Matilda Backendal [2 †], and Kenneth G. Paterson [1]

[1] *Department of Computer Science, ETH Zurich, Switzerland*
[2] *Faculty of Informatics, Università della Svizzera italiana (USI), Switzerland*

Abstract

Zero Knowledge Encryption is a term widely used by vendors of cloud-based password managers. Although it has no strict technical meaning, the term conveys the idea that the server, who stores encrypted password vaults on behalf of users, is unable to learn anything about the contents of those vaults. The security claims made by vendors imply that this should hold even if the server is fully malicious. This threat model is justified in practice by the high sensitivity of vault data, which makes password manager servers an attractive target for breaches (as evidenced by a history of attacks).

We examine the extent to which security against a fully malicious server holds true for three leading vendors who make the Zero Knowledge Encryption claim: Bitwarden, LastPass and Dashlane. Collectively, they have more than 60 million users and 23% market share. We present **12** distinct attacks against Bitwarden, **7** against LastPass and **6** against Dashlane. The attacks range in severity, from integrity violations of targeted user vaults to the complete compromise of all the vaults associated with an organisation. The majority of the attacks allow recovery of passwords. We have disclosed our findings to the vendors and remediation is underway.

Our attacks showcase the importance of considering the malicious server threat model for cloud-based password managers. Despite vendors' attempts to achieve security in this setting, we uncover several common design anti-patterns and cryptographic misconceptions that resulted in vulnerabilities. We discuss possible mitigations and also reflect more broadly on what can be learned from our analysis by developers of end-to-end encrypted systems.

1 Introduction

Despite the rise of alternative authentication methods, users today still have to deal with passwords, often numbering in the hundreds [60]. Password managers help to tame the problem by providing a tool to securely store passwords, reducing the challenge of remembering many passwords to remembering just the one "master password" for the password manager. Cloud-based password managers outsource the storage to a remote server under the control of a service provider. At an abstract level, a user's passwords are collected in a single object which is then encrypted by the user's client under a cryptographic key derived from the user's master password, creating an *encrypted vault*. The client then uploads the encrypted vault to the server. When a user wishes to access a password for a particular service, their client authenticates to the service, retrieves the encrypted vault, and decrypts it locally with a user-provided copy of the master password.

Importantly in solutions of this type, the service provider does not see the vault plaintext and therefore does not immediately learn the user's passwords or other sensitive data. This is akin to the situation with end-to-end encrypted (E2EE) cloud storage, and while the terms *E2EE* or *client-side encryption* are sometimes used by vendors in this space, the most commonly used term is *Zero Knowledge Encryption*. The term *Zero Knowledge* of course has a specific technical meaning in the context of interactive protocols, but here the term is being used with a different meaning, as we shall see.

The cloud-based approach has multiple advantages: users can access their encrypted vaults from multiple devices; vaults can store other sensitive information beyond passwords (e.g. credit card data, personal documents); and the service can be extended to allow sharing of sensitive data within a family, group or organisation. The "access from anywhere" feature creates work for vendors, who have to support access from web browsers as well as stand-alone applications running on different OSes. Many vendors have offerings which allow the cloud storage element to be self-hosted by an organisation instead of by the vendor.

Three prominent providers in this space are Bitwarden, Dashlane and LastPass. At the time of writing, Bitwarden claims to have 10 million users, Dashlane 19 million users and 24,000 business customers, and LastPass 33 million users and 100,000 business customers [19, 28, 55]. A 2024 report [67]

† Part of this work was done while Giovanni Torrisi and Matilda Backendal were at ETH Zurich.

based on a survey of 1000 US consumers gives further insight into the popularity and market share of password managers. The built-in password managers of Google and Apple now represent 55% of the market, up from a combined share of only 15% in 2021. Bitwarden and LastPass are the next two largest, according to the study, with 11% and 10% market share, respectively. Dashlane now has only 2% market share, down from 7% in 2021 when it was amongst the market leaders.[1] There is a long tail of smaller players in the market.

Threat Model. We conduct our analysis of cloud-based password managers in a threat model in which the servers storing users' vaults are assumed to be fully malicious, meaning that they can arbitrarily deviate from expected behaviour when interacting with clients. This is a stronger model than considered in prior work. We present three arguments to justify this in Section 2, summarised here. First, vendors' explanations of Zero Knowledge Encryption and E2EE terminology imply that security is maintained in this setting. Second, given the large amount of sensitive data that they store, the service providers are likely targets for sophisticated attackers capable of penetrating the servers and then mounting active attacks (indeed, some vendors have been repeatedly breached [43, 51, 70]). Third, in closely related areas such as cloud storage and messaging, security against a malicious server is by now *de rigueur*.

Security Expectations. Given this threat model, what security should users reasonably expect of password managers? Based on prior work and our examination of vendors' security claims, we argue that the appropriate guarantees are:

1. Confidentiality of all data in a user's vault. This includes the *items* – that is, credentials (usernames and passwords), but also credit card information and notes. Most metadata is not kept confidential in the systems we studied, but achieving this is also desirable.

2. Integrity of vault data (and metadata). This should apply at the level of individual items, but also at the level of the data collection as a whole. It should be infeasible for an adversary to modify existing data or inject new data, undetectably delete or modify any data, or change the semantics of any data in a vault, e.g., by altering metadata. In particular, the integrity of any user-specific settings should be maintained.

3. Secure sharing and account recovery. If these features are enabled, then only the set of other users with whom a given user intends to share some or all of their vault data (or grant an account recovery capability) should be able to access that data (respectively, recover the account).

[1] The apparent inconsistency between number of users and market share arises due to different sources: vendors' self-reported claims vs market study. Dashlane disputes the accuracy of this source due to the small sample size.

Several subsidiary security properties are needed to achieve these goals. For example, it should be hard for a malicious server to weaken the cryptography used by clients (e.g., reducing the iteration count used in password hashing).

More advanced goals can also be envisaged, e.g. vault indistinguishability, oblivious client operations, device privacy, etc. Such goals are far from being achieved by today's password managers and thus out of scope for our analysis; however, see [11] for a recent study of possible security properties.

Our Results. In Section 4, we give a detailed analysis of Bitwarden, Dashlane and LastPass, presenting a cornucopia of practical attacks. In the artefacts that accompany our paper, we give Proof of Concept (PoC) implementations of all of these attacks, demonstrating their feasibility. The attacks allow us to downgrade security guarantees, violate security expectations, and even fully compromise users' accounts. Table 1 lists the attacks and their impacts. Worryingly, the majority of the attacks allow recovery of passwords – the very thing that the password managers are meant to protect.

We group the attacks into four categories: attacks exploiting the key escrow features used for account recovery and SSO login, attacks based on lack of integrity of the vault as a whole, attacks enabled by the sharing features, and, finally, attacks exploiting backwards compatibility features.

These attacks reveal common design anti-patterns and cryptographic misconceptions. *Lack of authentication* of public keys is widespread. When combined with key escrow and sharing features, this results in the adversary being able to fully compromise vaults. Another recurring failure mode is (wrongly) assuming origin-authentication of public key ciphertexts, leading to *key substitution* attacks against Bitwarden in the style of [5, 45]. LastPass stands out for lacking any form of *ciphertext integrity*, using AES-CBC as its main encryption mode. Thanks to legacy code and backwards compatibility exploits, we can downgrade Bitwarden and Dashlane to similarly hazardous states. We also show that integrity is only achieved for single fields in individual items, instead of at the vault level. This enables cut and paste attacks within items and across the vault. Such attacks can often be chained to compromise the confidentiality of the vault as well. These attacks work even when proper authenticated encryption is used. They are possible because of insufficient key separation in vaults with complex structures and/or a lack of cryptographic binding between data and metadata.

Target Selection. We selected the three providers as targets based on their historical and current market share, the availability of unobfuscated client source code (available for Bitwarden and, partially, for Dashlane and LastPass), the richness of the offered feature set, and the diversity of approaches. We additionally performed an initial analysis of 1Password, see Appendix D for the results. As noted above, Apple and Google now dominate the password manager market. However, the unavailability of source code precludes easy analysis

Class	Ref	Name	Cause	Impact	Interaction	Mitigations
Key Escrow	BW01	Malicious Auto-Enrolment	Lack of Key Auth, Key Substitution	Full vault compromise †	1 join	PKA, SC
	BW02	Malicious Key Rotation	Key Substitution	Full vault compromise †	1 rotation	SC
	BW03	Malicious KC Conversion	Lack of Key Auth, Key Substitution	Full vault compromise †	1 dialog	PKA, SC
Item-Level Vault Encryption	LP01	Malicious Password Reset	Lack of Key Auth	Full vault compromise †	1 login	PKA, SC
	BW04	Unprotected Item Metadata	Lack of Auth Enc	Read and modify metadata	-	AE, AD
	BW05	Item Field Swapping	Lack of Key Sep	Field and item swapping	-	KS
	BW06	Icon URL Item Decryption	Lack of Key Sep	Loss of confidentiality †	1 open	KS
	BW07	Remove KDF Iterations	Lack of Auth Enc	No brute-force protection	1 login	AD
	LP02	Item Field Swapping	Lack of Auth Enc	Field and item swapping	-	KS, AE
	LP03	Icon URL Item Decryption	Lack of Key Sep	Loss of confidentiality †	1 open	KS, AE
	LP04	Remove KDF Iterations	Lack of Auth Enc	No brute-force protection	1 login	AD
	LP05	Malleable Vault	Lack of Auth Enc	Loss of vault integrity	-	AE
	LP06	Unprotected Item Metadata	Lack of Auth Enc	Read and modify metadata	-	AD, AE
	DL01	Transaction Replay	Lack of Key Sep	Loss of vault integrity	-	KS
Sharing	BW08	Organisation Injection	Lack of Key Auth	Add users to arbitrary orgs	1 sync	SC
	BW09	Organisation Overwrite	Lack of Key Auth, Key Substitution	Organisation compromise †	1 join	PKA, SC
	LP07	Sharing Key Overwrite	Lack of Key Auth	Shared vault compromise †	1 join	PKA
	DL02	Sharing Key Overwrite	Lack of Key Auth	Shared vault compromise †	1 join	PKA
Backwards Compatibility	BW10	Disable Per-Item Keys	Lack of Auth Enc	Downgrade key hierarchy	-	KS, AD
	BW11	User Key Overwriting	CBC Support	Loss of confidentiality †	2 logins	AE
	BW12	Downgrade to Legacy	CBC Support	Full vault compromise †	2 logins	AE
	DL03	Item Injection	CBC Support	Loss of vault integrity	10^4 syncs	AE
	DL04	Remove KDF Iterations	CBC Support	No brute-force protection	10^4 syncs	AE
	DL05	CBC-Only Downgrade	CBC Support	Loss of confidentiality †	10^5 syncs	AE
	DL06	Lucky 64	CBC Support	No brute-force protection	10^4 syncs	AE

Table 1: Summary of attacks. The attacks are grouped in four categories, depending on the password manager feature they exploit. The attack ref(erence) also indicates the product: BW for Bitwarden, LP for LastPass, and DL for Dashlane. We highlight the main causes for the attack, its high-level impact († denotes recovery of encrypted passwords) and the client interaction required: periodic or user-triggered synchronisation (sync); the user logging in (login), opening the vault (open), joining an organisation (join), sharing a vault (share) or clicking on a misleading dialog (dialog). Finally, we report the mitigations from Section 5.

of these systems. Additionally, E2EE is an opt-in feature for Google's password manager, so users do not enjoy security against malicious servers by default. Finally, while numerous other cloud-based password managers could also have been studied, our deep examination of the three selected products – representing over 60 million users and 23% of the market – was already sufficient to surface severe design flaws and common misunderstandings.

Mitigations and Disclosure. We describe potential mitigations for each attack in Section 5. We also discuss general strategies to avoid the common pitfalls we discovered. We have shared these mitigations as part of a detailed, vendor-specific disclosure document with Bitwarden, LastPass, and Dashlane. In each case we have followed an industry-standard coordinated vulnerability disclosure approach, giving vendors at least 90 days to deploy countermeasures. For more details, see our ethics analysis in Appendix A.

Broader Impact and Contributions. The immediate impact of our work is to help the affected vendors patch their vulnerabilities, thereby providing stronger security for very sensitive data of millions of users. But the impact of our findings also goes beyond the specific products they affect to illustrate some of the frontiers in the space of end-to-end encrypted applications. The importance of the integrity of vaults is a new finding, but parallels can be drawn with similar, recent notions of integrity for cloud storage [12]. Building systems that support key escrow without opening up attack avenues for malicious actors is considered a hard problem, one that has arisen repeatedly in the arena of lawful access and the Crypto Wars [3]. The problem of reliably authenticating public keys that (as we show) plagues password managers also comes up in the messaging space, where it is being tackled by the introduction of Auditable Key Directories, cf. [56]. Section 6 considers these and other issues at greater length.

Beyond categorizing the common pitfalls of E2EE cloud-based password managers and drawing parallels to where these arise in other E2EE systems, a main contribution of our work is to firmly establish the need for formal study of these products and their security. The number of severe vulnerabilities we uncover among a diverse set of vendors shows that getting end-to-end encryption right in this setting is not easy. Building on initial notions for password manager security in [42], we propose as future work to formally define the goals of E2EE against a malicious server, akin to what was recently done for cloud storage [12]. Analyzing cloud-based password managers in this threat model is a novel aspect of our work, and we have seen from our interactions with vendors that our attacks provide a strong motivation to work towards achieving theoretically well-founded guarantees in this setting, promoting the benefits of formal analysis and provable security in practice.

Related Work. There was substantial prior analysis of password managers shortly after they first appeared on the market [14, 40, 42, 57, 68, 69, 73]. However, none of this work considered the malicious server threat model as we do. In 2020, Carr and Shahandashti [23] and Oesch and Ruoti [61] revisited this early work concluding that password managers have improved their security. More recently, Fábrega et al. [38] developed injection attacks, their threat model assuming a malicious client in combination with a passive server (or a network adversary); Duan et al. [36] studied 43 password managers and their vulnerability to offline guessing attacks.

Expanding on the above, early analysis started more than a decade ago. Li et al. [57] examined five browser-based password managers (LastPass and four others that are no longer extant), uncovering vulnerabilities in four categories: bookmarklet issues, classic web issues, authorization issues, and UI issues. The threat model used in [57] is that of a "web attacker". In the same timeframe, Stock and Johns [69] studied the impact of XSS-based attacks on password managers. In contrast to [57, 69], our vulnerabilities are of markedly different type, stemming more from cryptographic issues, and in some cases are more devastating. Meanwhile, Silver et al. [68] examined the dangers of autofill while Fahl et al. [40] studied the impact of malicious Android apps on password managers. Gasti and Rasmussen [42] provided a theoretical treatment of a range of password manager database formats, focussed on a local storage model and assuming the adversary has either read-only or read-write access to the encrypted database. Such capabilities are available to our malicious server adversary for the user vaults that it stores. Gasti and Rasmussen [42] found only one format offering both integrity and secrecy in their attack model, indicating developers' lack of understanding of the need to provide appropriate cryptographic protection. In contrast to our work, [42] did not explore the practical consequences of their findings. Passive (rather than active) attacks on password manager applications running on iOS and Blackberry were presented by Belenko and Sklyarov in [14]; this work highlighted the use of weak PIN protections or lack of suitably strong password-based key derivation functions. LastPass was one of the targeted products; in 2012 when [14] was published, it used only one iteration of SHA256 to derive the vault master key from the user password. This has improved since. Zhao et al. [73] also considered the security of LastPass, but focussed mostly on the complexity of brute-force attacks on its password hashing mechanism; this paper also observed that a passive server adversary could observe sensitive data in the RoboForm product because of lack of client-side encryption, a valid attack in their insider attack model which allowed for server-side monitoring and data theft (but not an actively malicious server).

In 2020, Oesch and Ruoti [61] revisited some of this early work to consider the full password manager lifecycle for seven different standalone managers (including the three that we study) and five browser-provided managers. Their consideration of password storage documented vault encryption mechanisms and briefly studied metadata privacy. Their main takeaway is that password managers have improved their security compared to how they performed in earlier studies, though still with significant weaknesses in the areas of unencrypted metadata, insecure defaults, and vulnerability to clickjacking attacks. A similar study was carried out by Carr and Shahandashti in [23], focussed on phishing, clipboard and PIN brute-forcing vulnerabilities. A further five years later, our work challenges the findings of [23, 61] in the malicious server threat model that we have justified to be appropriate for cloud-based password managers. Fábrega et al. [38] considered injection attacks against ten different password managers, wherein an adversary (only) controls their own application client, which they use to "inject" chosen payloads to a victim's client via, for example, sharing credentials with them. The adversary is also able to observe the protected vaults in some form. The threat model in [38] is different to ours – roughly speaking it considers a malicious client in combination with a passive server (or a network adversary) whereas we consider a malicious server. Finally, Duan et al. [36] provided a categorisation and formal models in the UC framework, in addition to studying offline guessing attacks against 43 password managers.

Parallel to our work, Avoine et al. [11] study password managers in a threat model where the server remains "covert" in any attack it mounts. This preprint classifies cloud-based password managers into three types and sketches attacks by type. The attacks violate vault integrity, as well as several weaker properties not currently targeted by vendors. The attacks' impacts are not fully explored, and there are no PoCs. Some of them rely on weak passwords and/or low PBKDF iteration counts.

A raft of work has studied password managers from human and usability perspectives [24, 26, 39, 47, 50, 58, 59, 62–64]. In addition, studies on general password usability, password

policies, technical measures for improving password security, and password breaches abound.

Close in spirit to our work is a recent line of work analysing the security of E2EE cloud storage providers in the malicious server setting [5,6,13,44,46]. Many vendors in this space also use the term Zero Knowledge Encryption in a similar way to vendors of password managers. Our findings are broadly analogous to the ones of these papers, though significantly different in the technical details of attacks. We compare our work with this line of work in more detail in Section 6.

2 Threat Model

As we laid out in our discussion of related work, there has been a significant amount of security analysis of password managers over the last decade (and more). At the same time, the landscape of password managers has changed, with cloud-based solutions becoming significantly more common. This makes it timely to revisit the question: what is the appropriate threat model in this setting?

Before going further, we note that there is a trivial attack against end-to-end encrypted applications that we wish to take out of consideration from the outset. Namely, a user can be supplied with a malicious client which, for example, simply returns the user's master password to the server in the clear. Such an attack would be detectable through code audits and – though possible via an obfuscated functionality – would be very risky for a vendor to conduct. This is especially true for signed software releases (e.g., desktop and mobile clients, browser extensions), as the existence of a signed malicious client would constitute proof of vendor misconduct.

Setting this trivial attack aside, we argue here that it is appropriate to consider a threat model in which the server is considered fully malicious, meaning that it can deviate arbitrarily from its expected behaviour. Such deviations are hard to detect, especially when targeted at selected users. We argue that this threat model is motivated by the advertised security guarantees by the vendors, the sensitivity of vault data, and the fact that malicious-server security is by now the norm for outsourced data in other, related settings. We expand on each of these below, and end by briefly discussing the role of client interaction in this threat model.

Other E2EE Domains. Security in the face of a malicious server is by now the *de facto* standard in related settings such as messaging systems and cloud storage, where sensitive user data (in the form of private messages or files) is also relayed or stored by service providers. In particular, the malicious server threat model is the accepted interpretation of the security guarantees provided by end-to-end encryption in these settings. (We note that some vendors of E2EE cloud storage also use "Zero Knowledge Encryption" to refer to E2EE. Thus, also in this setting, this term is taken to mean data confidentiality and integrity in the face of a malicious or compromised server.) For example, recent analyses of cloud storage systems [5,6,13,44,46] have forced vendors in this space to upgrade their approach to provide security against a malicious server, while major messaging systems like Signal, WhatsApp and iMessage have long featured E2EE in the standard sense (Telegram remains a notable and regrettable exception in this area, with E2EE only being optionally available on a per conversation basis rather than as a default, and not available at all for groups). We think it is reasonable to expect these terms (zero knowledge and E2E encryption) to have the same meaning in the context of password managers.

Passwords have High Value. Passwords are among the most sensitive kind of user data, as they often grant access to accounts and services which may in turn reveal more data, or allow an attacker to impersonate the victim. Since password manager servers host high concentrations of passwords from thousands of users, they thus provide an attractive target for hackers. Attacks on the provider server infrastructure can be prevented by carefully designed operational security measures, but it is well within the bounds of reason to assume that these services are targeted by sophisticated nation-state-level adversaries, for example via software supply-chain attacks [25] or spear-phishing. Moreover, some of the service providers have a history of being breached – for example LastPass suffered breaches in 2015 and 2022, and another serious security incident in 2021. In the 2022 breach, the attacker gained access to the development environment, source code, and technical information, used that information to compromise the computer of a senior DevOps engineer, then gained access to an encrypted corporate vault, and finally to the keys of the Amazon S3 "buckets" of the backups to customer files [43,70]. In a self-hosted deployment involving a less technically advanced customer, it seems highly likely that the server hosting user vaults could not resist such a determined attack. While none of the breaches we are aware off involved reprogramming the server to make it undertake malicious actions, this goes just one step beyond attacks on password manager service providers that have been documented. Active attacks more broadly have been documented in the wild [48,65].

Vendors' Statements. Additionally, we provide example statements made by the providers which we believe justify this threat model here (taken from their web sites and white papers where they explain their "Zero Knowledge Encryption" claims).

> *"Zero knowledge encryption: Bitwarden team members cannot see your passwords. Your data remains end-to-end encrypted with your individual email and master password. [...] Since it's fully encrypted before it ever leaves your device, only you have access to your data. Not even the team at Bitwarden can read your data (even if we wanted to)." [21]*

> *"Dashlane Password Manager is designed using zero-*

knowledge architecture, with the data encrypted locally on the user's device. Only the user can access the data by using a password or another form of authentication. Since Dashlane doesn't have access to the user's vault and doesn't store the user's Master Password, malicious actors can't steal the information, even if Dashlane's servers are compromised." [32]

"With a zero-knowledge approach, you can rest easy knowing that no one else but you, not even your password manager vendor, has the keys to the kingdom. [...] For example, zero-knowledge means that no one has access to your master password for LastPass or the data stored in your LastPass vault, except you (not even LastPass)." [34]

While none of the vendors quoted here explicitly describe the threat model under which they expect their product to remain secure in terms of attacker capabilities, we argue that – based on statements like the ones above – their customers could reasonably deduce that their vault data remains protected even against a malicious server. In addition to these general statements, the vendors to whom we have disclosed so far (Bitwarden, LastPass and Dashlane) have not questioned our adoption of this threat model for our analysis.[2]

Client Interaction. While our threat model concerns a malicious server, some of our attacks rely on interaction with the (honest) client victim. We treat the amount of client interaction needed for an attack to succeed as a resource, and for each attack, we list the expected client interactions required.

The majority of our attacks require simple interactions which users or their clients perform routinely as part of their usage of the product, such as logging in to their account, opening the vault and viewing the items, or performing periodic synchronization of data. We also present attacks that require more complex user actions, such as key rotations, joining an organization, sharing credentials, or even clicking on a misleading dialog. Although assessing the probability of these actions is challenging, we believe that, within a vast user base, many users will likely perform them. Table 1 summarises the interaction requirements for each attack.

3 The Password Managers

3.1 Bitwarden

The key hierarchy of a Bitwarden user, illustrated in Figure 1, is rooted in the user's master password and email. These are used respectively as key and salt of a KDF, which can either be PBKDF2 or Argon2id, in order to generate a 32-byte-long *master key* (k_m). The master key is used for authentication by sending the result (h_m) of a single iteration of PBKDF2 on the master key with the user's password as salt to the server, which compares it to the value of h_m stored at registration. The master key is additionally used to derive two 32-byte keys via HKDF-Expand with two different input labels: `"enc"` and `"mac"`. The resulting keys ($k_{em}^{enc}, k_{em}^{mac}$) are concatenated to form the 64-byte *extended master key* (k_{em}).

After deriving the extended master key, a 64-byte *user key* (k_u) is sampled uniformly at random. Similarly to k_{em}, this key can be interpreted as consisting of an encryption (k_u^{enc}) and a MAC key (k_u^{mac}), each 32-bytes. The role of the user key is crucial: it protects all the data inside the user's vault, either by directly encrypting vault items, or indirectly (if so-called "per-item keys" are enabled, as explained below). Once the user key is generated, it is encrypted under k_{em} and stored on the server to be served at the next login.

Additionally, the client generates an RSA keypair (with a 2048-bit-long modulus) consisting of a public key pk and a private key sk. This keypair is used in the context of Bitwarden's organisation as described below. Finally, the private key is also encrypted under the user key and sent to the server.

Bitwarden allows users and businesses to self-host the server, in order to have complete control of their data. For users of self-hosted Bitwarden instances, the adversary in our threat model will thus be the self-hosted server.

Key and item encryption. Bitwarden uses AES-CBC-HMAC for authenticated encryption of keys and vault items, using the "enc" and "mac" parts of the key for encryption and MACing, respectively. For backwards compatibility, the HMAC is omitted if the "mac" key part is missing. That is, if the client attempts to encrypt using a key which is only 32 bytes, the client falls back to using AES-CBC.

In vault items, each item data field is encrypted individually. Item fields used to be encrypted directly with the user key k_u, but from version `2024.2.0` new items are encrypted with individually sampled *per-item* keys, which are in turn encrypted by the user key. Old items remain directly under k_u.

Organisations. Credential sharing is implemented in Bitwarden through organisations. When a user creates an organisation, their client samples a new 64-byte *organisation symmetric key* k_{org}, and encrypts this key under the user's public key, yielding c_{org}^{user}. Additionally, the client also generates an *organisation* RSA *keypair* (pk_{org}, sk_{org}) (used for account recovery purposes), and encrypts the private key using k_{org} to create c_{skOrg} (see Figure 1). It then sends c_{org}^{user}, pk_{org}, and the encrypted c_{skOrg} to the server. Items in an organisation are individually encrypted using the organisation symmetric key.

The creator of an organisation can add new members by encrypting the organisation symmetric key for the public key of the invitee. For each user $user'$, the resulting ciphertext $c_{org}^{user'}$ is sent to the server for storage, and served to $user'$ when syncing. The client of $user'$ can then decrypt $c_{org}^{user'}$ using their private key to obtain k_{org}. A client determines that the user is a member of an organisation if it can obtain k_{org}.

[2] LastPass have communicated that they don't adopt a generic malicious server threat model, but apply compensating controls to mitigate server risks.

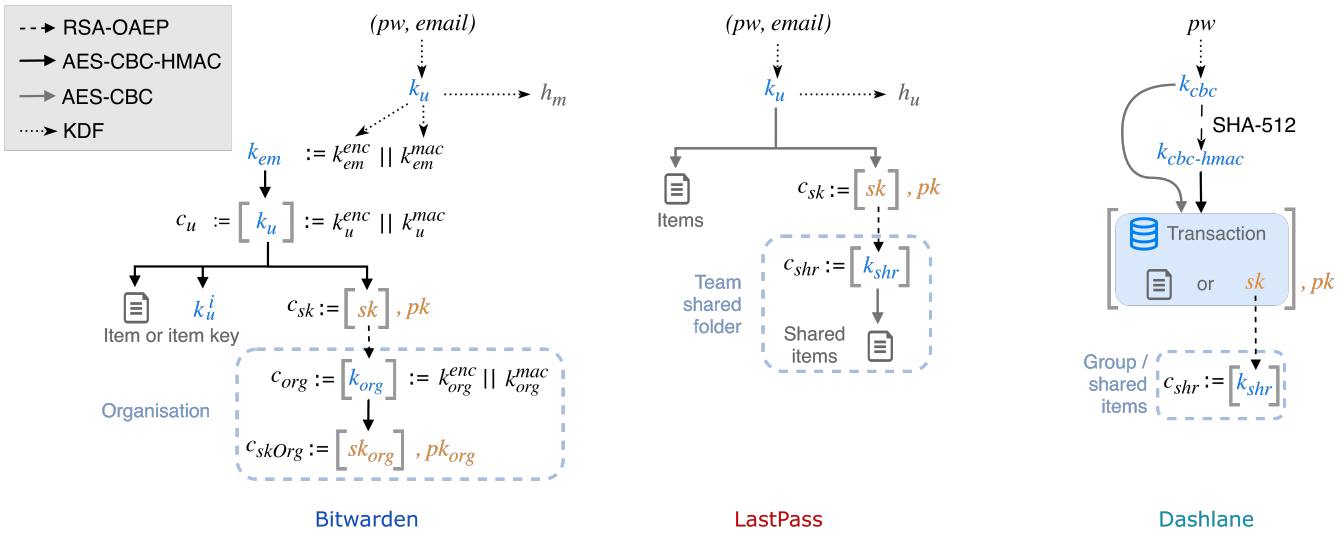

Figure 1: Key hierarchies for Bitwarden (left), LastPass (center) and Dashlane (right).

Further organisation data, such as the organisation name and policies regarding account recovery, for example, are stored unencrypted on the server and sent to the client whenever as sync operation is performed.

3.2 LastPass

Similarly to in Bitwarden, the key hierarchy in LastPass (illustrated in Figure 1) begins with the user's password, which is used to derive a 32-byte *user key* k_u using PBKDF2 with the user's email address as salt. The default KDF iteration count is $600,000$. To authenticate, the client sends h_u, the result of one more iteration of PBKDF2 over k_u with the password as salt, to the server; the server checks if the received h_u matches the value received at registration.

The user key is used to directly encrypt the items in the user's vault using AES-CBC (with no integrity protection). Each user also has an RSA key pair (sk, pk); the secret key is encrypted under k_u using AES-CBC and stored at the server.

Sharing among LastPass users is implemented through shared folders. When a user first shares an item, their client samples a 32-byte shared folder key k_{shr}. This is subsequently used to encrypt the items in the shared folder using AES-CBC. The folder key is itself encrypted using RSA-OAEP under the public key of every user with whom the folder is shared.

3.3 Dashlane

Dashlane's vault is maintained in the form of a transactional database. Each transaction can create, modify, or delete an item in the database: the transaction type and timestamp are unencrypted metadata fields, while the transaction content, consisting of compressed XML data with information about items or user settings, is encrypted.

Content is encrypted using a key derived from the user's master password. The exact details for these operations are specified in the first bytes of the content by a *payload* string: this defines the password-based key derivation function (either PBKDF2, Argon2d, or noderivation), its parameters (e.g., salt length, hash method, and iterations for PBKDF2), the cipher (by default AES-256), and the cipher mode (by default CBC with HMAC). First, an intermediate 32-byte key k_{cbc} is derived from the password. If the cipher mode is CBC with HMAC, then an additional SHA512 is used to derive the 64-byte $k_{cbc\text{-}hmac}$. When the user modifies their vault (e.g., by adding an item), a new transaction is created and sent to the server. See the illustration in Figure 1.

Each user also has a 2048-bit modulus RSA key pair (sk, pk); the secret key is stored encrypted (as a transaction) on the server, together with the public key. The RSA keys are used for sharing: when a user shares an item with another user (either directly or in a group), a symmetric encryption key k_{shr} is sampled and encrypted for the recipient(s) public keys (which the sharer retrieves from the server).

NOTE. The only fully-available open-source client is the CLI client, which does not support encryption nor the credential sharing feature, so the description we provide here is based in part on reverse engineering and may be incomplete. We focus on the parts of the system relevant to our attacks, for which the functionality we describe here is verified by the PoCs.

4 Attacks

We now present our attacks on Bitwarden, LastPass and Dashlane, grouped into four categories based on cause, as follows:

1. In E2EE password managers, forgetting the master password means that a user irrecoverably loses access to their vault. Thus, many password managers implement some form of account recovery mechanism through **key escrow**, such that an administrator can help users recover their vault contents. In Section 4.1, we show attacks against vault confidentiality in Bitwarden and LastPass resulting from vulnerabilities in their key escrow designs. We also discuss a weakness in Dashlane's design.

2. None of the password managers we analyse treat the password vault as a monolithic object. Rather, all data items (as well as sensitive user settings) are encrypted as separate objects, and often combined with unencrypted or unauthenticated metadata. In Section 4.2, we show how such an **item-level vault encryption** design is often in tension with the *vault integrity* security goal. We find attacks against all of Bitwarden, LastPass and Dashlane. Notably, many of these integrity violations also lead to concrete attacks impacting confidentiality.

3. Many password managers implement **sharing** features (targeting for example families and businesses) which allow multiple people to access common shared credentials. In Section 4.3, we show attacks against the sharing features of all three of our targets. These impact both vault integrity and confidentiality.

4. All three password managers that we analyse have undergone cryptographic evolution over time. However, they all maintain some form of **backwards compatibility** to support older client versions. In Section 4.4, we show that such "legacy code" creates cryptographic hazards. Specifically, a malicious server can trigger legacy code paths to be executed in clients, resulting in downgrade attacks in Bitwarden and Dashlane. This holds even if a user intends to use the latest version of the clients. Through such attacks, an adversary can for example downgrade clients to use unauthenticated encryption, opening up avenues for further attacks that violate both vault integrity and confidentiality.

Table 1 gives a complete summary of our attacks. Throughout this section, we refer to "the adversary" as an entity that has full control of the server. Table 2 summarises the analysed software versions and the availability of source code. Unless otherwise specified, all of the attacks affect all the clients mentioned in the table.[3]

Analysis Method. Our approach to analysing password managers follows the "attack" branch of the "Cryptography in the Wild" methodology that is described in detail in [7].

[3] Due to the considerable engineering effort required for testing all the clients, we limit our PoC testing to Bitwarden's web client and the browser extensions for LastPass and Dashlane. For most of the attacks, it is evident that they are design issues, independent of specific client implementations.

	Source	Version	Features
Bitwarden			
Web Client	● [20]	`v2024.10.2`	■
CLI Client	● [20]	`v2024.10.0`	◨
Server	● [20]	`v2024.10.2`	-
LastPass			
Web Client	○	Unknown	◨
Browser Ext.	○	`4.140.0`	■
CLI Client	● [53]	`v.1.6.1`	◨
Dashlane			
Web Client	○	Unknown	◨
Browser Ext.	◑ [30]	`6.2513.1`	■
CLI Client	● [31]	`v6.2447.2`	◨
Android Client	◑ [29]	`018f827f2a01`	■

Table 2: Summary of the software distributions of the password managers we study, detailing source code availability ranging from fully open source (●), open source releases omitting fundamental cryptographic libraries (◑), to closed source (○); the software version; and whether the client supports the full core feature set of the password manager (■) or just a small subset of operations (◨).

We explain here how we deployed this methodology in the specific context of password managers.

Our process began with reviewing white-papers and client code (and server code, when available). We initially focused on Bitwarden, where the availability of source code eased our analysis and enabled us to more easily build our mental model of architecture, protocols, and system features (cf. *object selection* in [7]). From the outset, our goal was to carry out an analysis in the face of a malicious or compromised server: based on the functionality of the target systems, we refined this to a concrete *adversarial model*. We developed pseudo-code models for the main cryptographic actions of each feature of interest (*ingestion*). We then analysed these models to find *attacks*. Here we relied on our own experience in finding cryptographic vulnerabilities in systems, along with a list of potential attack vectors gained from the by-now-extensive literature on cryptographic attacks. For password managers, we drew particular inspiration from recent literature on E2EE cloud storage providers in the malicious server setting [5,6,13,44,46] as well as common attack vectors such as padding oracle attacks [72], key overwriting in various forms, cf. [22], and exploitation of backwards compatibility features, cf. [49]. Finally, we built *PoC* implementations for the attacks, implementing malicious server functionality where needed, and testing against the relevant client. This step is vital for ensuring that attacks developed "on paper" actually work in practice.

Following our initial study of Bitwarden, we iterated the same process to determine whether our findings were applicable to other products. For Dashlane and LastPass, we carried

out some reverse engineering of minified client Javascript and of the Android applications. Having already studied Bitwarden helped us in developing our understanding of how these products operate. We confirmed that our attacks were applicable to the newly selected targets. Additionally, we manually classified the different attacks and grouped them into four categories, providing a starting point for follow-up work analysing further password manager products. This extends the methodology from [7].

Note that absent formal analyses, the set of attacks we found must be considered incomplete. Indeed, in some cases, we stopped looking for attacks after accumulating sufficiently many severe enough vulnerabilities to demonstrate that the given target was vulnerable in our threat model. Future work may involve employing formal tools to uncover other vulnerabilities, or conclusively demonstrate the lack thereof.

4.1 Key Escrow Attacks

These attacks all concern various forms of key escrow functionalities provided by the password managers to enable account recovery in the case of master password loss. Successful attacks on these recovery mechanisms have severe consequences. In most cases, the adversary can recover the full vault of the user, therefore compromising all of the data that it contains. Some recovery mechanisms are less powerful than others, and rely on storing recovery data on the user's devices. In these cases, our attacks are also less impactful.

4.1.1 Bitwarden

Bitwarden supports two flavours of key escrow: organisation account recovery and "Key Connector".

Organisation Account Recovery. Organisation account recovery allows the admins of organisations to reset the master password of all members who enrol in recovery.[4] Enrolling in recovery can either be a manual step requiring user interaction (so-called *self-enrolment*), or happen automatically when the user joins the organisation (*auto-enrolment*). When a user enrols (by either method) in account recovery, the client fetches the organisation public key pk_{org} from the server, encrypts the user key k_u under pk_{org}, and sends the resulting *account recovery ciphertext* c_{rec} to the server. To reset the master password of an enrolled user, an admin of the organisation can fetch the encrypted k_u from the server, decrypt it using the organisation's private key sk_{org}, choose a new master password (and consequently derive a new master key), and finally re-encrypt k_u under the new master key. We describe two key recovery attacks caused by the account recovery feature, BW01 and BW02, below.

[4] While the recovery feature is restricted to "enterprise" organisations (with a paid subscription), all Bitwarden clients include this feature's code, and our attacks succeed independently of user subscription status.

BW01: Malicious Auto-Enrolment. Recall that when a Bitwarden user joins an organisation, the client fetches the organisation's data, such as name, key material, and organisational policies, from the server. This data is not integrity protected. As a consequence, an adversary controlling the server can compromise users as soon as they accept an invitation to join *any* organisation, as follows.

When a user accepts an invitation, the client asks the server for the account recovery policy and the public-key of the organisation. The adversary replaces the organisation's real data, setting auto-enrolment to true in the policy, and replacing the public key pk_{org} with a malicious pk^{adv}_{org} for which they know the secret key sk^{adv}_{org}. Since account recovery is enabled, the client encrypts the user key k_u under the organisation public key pk^{adv}_{org}, and sends the resulting account recovery ciphertext c_{rec} to the server. The adversary decrypts c_{rec} with sk^{adv}_{org} and recovers k_u.

IMPACT. With possession of the user key, the adversary is able to read and modify all vault data of the targeted user.

REQUIREMENTS. The user joins any organisation. Note that the adversary does **not** have to control the organisation; it can be an honest invitation from a trusted enterprise or family member, for example.

NOTE. The account recovery feature can also be used by an attacker to amplify the reach of a compromise from one user to whole organisations, as follows. An organisation's private key sk_{org} is known to all the members of the organisation. It follows that once the adversary has compromised one user, they learn the private key sk_{org} of all the organisations that user is a member of. If any of these organisations enabled account recovery, the adversary then also compromises the user keys of all members of these organisations through their account recovery ciphertexts. This process can be repeated, infecting all organisations that have key recovery enabled and have overlapping members.

BW02: Malicious Key Rotation. Bitwarden allows its users to change their master password, and optionally rotate their keys. If a user decides to rotate their keys, the client samples a new user key k'_u and encrypts it under the new extended master key k'_{em} (cf. Figure 1).

If a user is enrolled in account recovery when they rotate their keys, the client will also have to regenerate their account recovery ciphertext. To do so, it encrypts the new user key k'_u under the organisation public key pk_{org}, and sends the new account recovery ciphertext c'_{rec} to the server.

The key point here is that pk_{org} is not retrieved from the user's vault; rather, the client performs a sync operation with the server to obtain it. Crucially, the organisation data provided by this sync operation is not authenticated in any way (see Attack BW01). This thus provides the adversary with another opportunity to obtain a victim's user key, by supplying a new public key pk^{adv}_{org} for which they know sk^{adv}_{org} and setting the account recovery enrolment to true. The client will

then send an account recovery ciphertext c'_{rec} containing the new user key, which the adversary can decrypt to obtain k'_u.

Note that this attack works whether or not the user is a member of an organisation at the time of key rotation: the adversary can either use Attack BW08 or BW09 to, respectively, add the user to a newly created organisation or overwrite an existing organisation's keys. This attack is described in detail in Appendix C.1.

IMPACT. With possession of the user key, the adversary is able to read and modify all vault data of the target user.

REQUIREMENTS. The user rotates their encryption keys.

Key Connector. The second flavour of key escrow in Bitwarden is the so-called "Key Connector" feature, offered to enterprise organisations that run self-hosted Bitwarden servers as part of the support for single sign-on (SSO).[5] When an organisation starts using SSO, it delegates the authentication of users to an external identity provider (e.g., Google or Microsoft). The purpose of using SSO is to allow users to access multiple different applications with a single set of credentials. That is, it removes the need for a dedicated account (username and password) for every service. However, in the case of end-to-end encrypted applications like Bitwarden, outsourcing the *authentication* to an external identity provider is not sufficient to remove the need for dedicated Bitwarden credentials. Indeed, recall that the master key k_m, derived from the master password and email address as shown in Figure 1, is not just used to authenticate to the server, but also to protect the user key, which in turn is used for client-side encryption of vault items. Thus, in order to allow users of Bitwarden organisations with SSO to entirely remove their Bitwarden password, the user key must also be recoverable without the password. This is the purpose of the Key Connector (KC) application. KC is a self-hosted application which allows Bitwarden organisations using SSO to store users' user keys (k_u in Figure 1) such that they do not need a Bitwarden master password or master key. When an organisation starts using KC, its users are redirected to a "remove master password" page. Each user is then prompted to confirm the change; the client subsequently sends the user's master key k_m to the KC service (henceforth referred to as the "connector"), which uses k_m to decrypt k_u and stores it. We refer to this procedure as *conversion*. In attack BW03, we show that a malicious Bitwarden server can trick any client (regardless of whether the user is a member of any organisation) into starting a conversion flow which, if completed, reveals the user's master to the adversary.

BW03: Malicious KC Conversion. The attack works as follows. The adversary tricks the client into believing that the user is part of an organisation which is enabling KC (as described below), and supplies the client with a (hidden) malicious connector URL. The client will then display the "remove master password" page to the user (see Figure 2 for an example), the content of which is also partially controlled by the adversary. If the user clicks the "remove master password" button, the client sends the master key k_m to the adversarially controlled URL in plain.

We now explain how the adversary can force the client into taking the user to this conversion page. Each time a Bitwarden client performs a sync operation with the server, it checks if the user is logged in via SSO and a member of an organisation which has enabled KC. If so, the client determines that the user needs to perform a conversion. The attack leverages that all of these attributes can be forged by the adversary.

As we saw in Attack BW02, the organisation data retrieved in sync operations is not integrity protected. This includes the organisation symmetric key, flags determining whether KC is enabled, and the connector URL. This lack of integrity allows the adversary to make the user a member of a newly created fictive organisation by sampling a symmetric key k'_{org} and other organisation data and supplying the client with k'_{org} encrypted under the user's public RSA key (as in Attack BW02). The adversary can also set the connector URL (to which the master key will be sent upon conversion) to an arbitrary URL of its choice.

Furthermore, clients infer whether SSO is enabled by looking at some attributes set by the server at login time. Specifically, the client believes itself to be logged in using SSO if it can find an authentication methods reference with the value "external" inside the JSON WEB Token (JWT) access token which it receives from the server when logging in. It is trivial for the adversary to forge this JWT, thus making *all* Bitwarden users vulnerable to this attack.[6]

IMPACT. The user's master key is sent to an adversarially controlled connector URL. With possession of the user's master key, the adversary is able to read and modify all vault data of the target user.

REQUIREMENTS. This attack requires an element of social engineering, since the user always needs to confirm the conversion. This can be achieved in several ways. First of all, the conversion page does not contain any warning about what the process entails, and does not show the URL of the Key Connector being used. Furthermore, the adversary can control the text shown in the conversion page by injecting arbitrary data in the "organisation name" field: Figure 2 shows an example of how the conversion page could look during an attack. Additionally, the adversary can disable the *"Leave organisation"* button in Figure 2, leaving the user no choice but to click on *"Remove master password"* in order to access their vault, as the client will persist the prompt until the user accepts the conversion.

[5] Just like for the account recovery feature, Key Connector is restricted to "enterprise" organisations (with a paid Bitwarden subscription), but since all Bitwarden clients contain the code implementing this feature, our attacks affect all users, regardless of subscription status.

[6] With the exception of users who are members of self-hosted Bitwarden organisations and have already performed KC conversion in the past; these users no longer have master keys, and are thus not vulnerable to the attack.

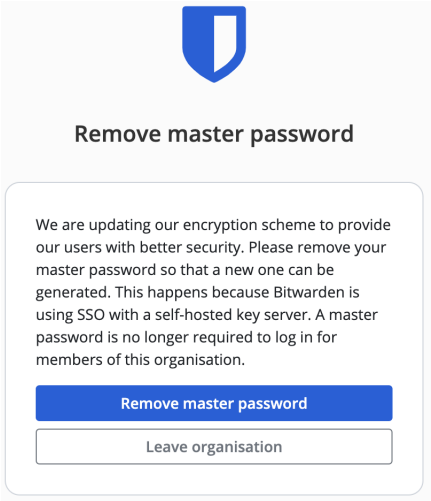

Figure 2: Key Connector conversion page with malicious message injected by the adversary.

4.1.2 LastPass

LastPass also provides two forms of key escrow: account recovery and admin *password reset*. We focus here on the second. The master password reset is a feature for LastPass' enterprise products "business" and "teams".[7] Within a team, privileged user accounts called *super admins* have the ability to reset the master password of users in the team. To this end, each admin account is associated with an RSA keypair (sk_{adm}, pk_{adm}). After a super admin chooses to reset a user's password, the client of the selected user will – upon the next login in the LastPass browser extension – retrieve the list of admins and their public keys from the server, encrypt its user key k_u under each admin's public key pk^i_{adm}, and send the resulting ciphertexts back to the server.[8]

LP01: Malicious Password Reset. Master password reset relies on admin public keys pk^i_{adm} being retrieved from the server. The LastPass client does not authenticate these keys in any way. It follows that an adversary can trivially inject their own pk^{adv}_{adm}, for which they know sk^{adv}_{adm}. They can then recover k_u from the ciphertext sent by the client.

In theory, only users in teams where password reset is enabled and who are selected for reset should be affected by this vulnerability. In practice, however, LastPass clients query the server at each login and fetch a list of admin keys. They then send the account recovery ciphertexts independently of enrolment status. See Appendix C.5 for attack details.

IMPACT. With possession of the user key, the adversary is able to read and modify all vault data of the target user.

REQUIREMENTS. The user logs in to LastPass in the browser extension client.[9]

4.2 Item-Level Vault Encryption

As we have mentioned, even though the user interacts with the abstraction of a "secure vault", the individual items and the user settings that comprise a vault are managed and (typically) encrypted independently. Hence, vaults are not monolithic cryptographic objects.

If the vault were monolithic, vault integrity would be easy to achieve by using authenticated encryption (AE). However, every minor change would necessitate re-encrypting the entire vault, making its maintenance and synchronisation with the server costly. Thus, a common solution is instead to encrypt each item separately with AE under a single key – but this does not necessarily result in integrity guarantees for the whole vault. For example, items could be swapped around or removed entirely without the client being able to detect it.

The attacks that follow all exploit the lack of *vault integrity* for the vendors we study. Not only can the adversary swap and delete items, they can also change metadata, settings such as the number of KDF iterations, or the organisations to which a user belongs.

4.2.1 Bitwarden

In Bitwarden, items and user settings are not uniformly protected. Rather, vault items are encrypted individually, and many settings are not encrypted at all. This also holds true for items in organisations and organisation-wide settings.

Each item is an object with various fields, such as item name, password, username, and URLs. All of these fields are individually encrypted using AES-CBC-HMAC. Up until version `2024.2.0` of Bitwarden, the fields of all items are encrypted directly with the user key k_u. As of `v.2024.2.0`, "per-item keys" are used instead: a fresh key k^i_u is sampled and used for each new vault item. The item key is itself encrypted with the user key k_u, and added as a field to the item. Note that old items are not migrated to this new format.

The client retrieves vault items from the server via a sync request; the server will return the item encoded in a JSON object, with the (unencrypted) field names as keys, and the respective ciphertexts as values.

BW04: Unprotected Item Metadata. A surprising number of item metadata fields are neither encrypted nor integrity protected. These include the type of items (e.g. login, card, secure note, etc.), their creation date and whether they require a password reprompt.

An adversary can arbitrarily read and modify these fields. This can leak information about the content of the vault. This attack has no requirements.

[7] This feature is only available in the paid versions of LastPass [52], but, as with Bitwarden, the code implementing this feature is present in the client independent of their subscription status.

[8] This process is described as a "Key Exchange" by LastPass [54]. This does not correspond to the conventional meaning of the term.

[9] Only the browser extension client supports the "key exchange" procedure used for password reset, thus the attack works only against this client.

BW05: Item Field Swapping. Nothing binds a ciphertext to a particular item field. Hence, an adversary can swap around ciphertexts from different fields within an item.

For all items created before the per-item keys feature was introduced, an adversary can even swap ciphertexts around among different items, as well as create new items by merging fields from other items.

This attack has no requirements, and represents a clear violation of vault integrity. It also functions as a building block for other attacks.

BW06: Icon URL Item Decryption. Items can include a URL field, which is used to autofill the credentials and display an icon on the client. The client decrypts the URL and fetches the icon from the server, including in its request the domain and top-level domain of the URL. For instance, if the URL is "https://host.tld/path", the client request includes "host.tld".

This means that the adversary can learn (part of) the contents of URL fields. Using Attack BW05, an adversary can place the ciphertext of sensitive item fields, such as a username or a password, in the encrypted URL field. After fetching the item, the client will then decrypt the ciphertext, confusing it for a URL. If the plaintext satisfies some conditions (i.e. containing a '.' and no !), it will be leaked to the adversary.

A URL checksum feature was deployed in July 2024, making the clients store a hash of the URL in another encrypted item field, therefore providing a rudimentary integrity check and preventing this attack. Note that old items are never updated to add such a checksum: this feature only protects items created after its introduction. Furthermore, URL checksums are only checked if a per-item key is present for the item. As we will see, an adversary can prevent per-item keys from being enabled with Attack BW10.

IMPACT. The adversary can recover selected target ciphertexts in the item, such as the username or the password.

REQUIREMENTS. The user opens a vault containing items that do not use per-item keys (i.e., items created before July 2024, or after Attack BW10 is run). The target plaintext must satisfy some additional conditions, detailed in Appendix C.3.

BW07: Remove KDF Iterations. Client authentication to Bitwarden is performed by checking if a hash of the master key h_m, derived from the master password input by the user at login time, matches a hash stored on Bitwarden's servers. In order to protect the master password from brute-force attacks, computation-intensive Password-Based Key Derivation Functions (PBKDFs) are used to compute this hash. The derivation speed is determined by the parameters of the PBKDF.

For instance, when PBKDF2 is used with the default 600,000 iterations, the hash is obtained as follows:

$$k_m \leftarrow \mathsf{PBKDF2}(pw, salt = email, i = 600000)$$
$$h_m \leftarrow \mathsf{PBKDF2}(k_m, salt = pw, i = 1)\ .$$

The KDF settings are stored unauthenticated and unencrypted on the server. When a target user logs in, the adversary can choose an arbitrary PBKDF configuration, including one with a minimal iteration count. The user will then send a master key hash that is efficient to compute during authentication. In our example, this means that the adversary can set the first iteration count to 1, thereby reducing the number of PBKDF2 iterations from 600,001 to 2 and achieving a 300,000x speed-up in a brute-force attack.

IMPACT. The adversary can reduce the cost of a brute-force attack on a target user's master password; successfully recovering the password enables full control of the user's account.

REQUIREMENTS. The user logs in.

4.2.2 LastPass

Similarly to Bitwarden, vault items in LastPass are composed of various fields such as item name, username, URL, and password, and all of these fields are individually encrypted under the same user key k_u. Similar attacks therefore apply.

LP02: Item Field Swapping. There is no associated data binding each ciphertext to its field. Hence, an adversary can swap around ciphertexts from different fields within an item, as well as between different items, and create new items by merging fields from other items.

LP03: Icon URL Item Decryption. LastPass login items can include a URL. This is used to autofill the credentials and display an icon on the client. In order to fetch these icons, the client makes a query to LastPass's icon server, including in the request the URL that was stored inside the item. Just like in Bitwarden, an adversary can use the item-field swapping attack (Attack LP02) and place the ciphertext of sensitive item fields, such as the username or the password, in the URL field. The plaintext will be leaked to the adversary via the query to the icon server. Note that, unlike Bitwarden, there are **no conditions** on the plaintext formatting.

IMPACT. The adversary can recover (parts of) selected target ciphertexts in the item, such as the username or the password.

REQUIREMENTS. The client opens the modified vault.

LP04: Remove KDF Iterations. Client authentication to LastPass is performed by checking if a hash of the user key – derived from the master password input by the user at login time – matches a hash stored on LastPass's servers. Just like in Bitwarden, 2 invocation of PBKDFs are used to derive this hash, with a default of $600,000 + 1$ iterations for PBKDF2.

Similarly to Bitwarden, the KDF parameters are unencrypted and unauthenticated, enabling an adversary to choose them arbitrarily. The PBKDF2 iteration count can then be lowered down to 3 (a client-enforced minimum of 2 for the inner invocation, while the outer is fixed to 1). The adversary thus achieves a speed-up of 200,000x in a brute-force attack.

IMPACT. The adversary can reduce the cost of a brute-force attack on a target user's master password; successfully recovering the password enables full control of the user's account.

REQUIREMENTS. The user logs in.

LP05: Malleable Vault. All field items are encrypted separately under a single user key using AES-256-CBC (without integrity protection). An adversary can manipulate the content of the encrypted fields using standard CBC-mode bit-flipping techniques. This attack has no requirements.

LP06: Unprotected Item Metadata. As in Bitwarden, many item metadata fields are unencrypted and not integrity protected: this includes creation timestamp, modification timestamp, and master password reprompt fields. An adversary can change the metadata for an item, including whether or not to reprompt the user for their master password, the item creation and modification timestamp, its folder, and whether the item is in the favourites. They can also enumerate the vault items and their type, such as login credentials, credit card, secure note etc., and observe the victim's activity by looking at the item creation and modification time. This attack has no requirements.

4.2.3 Dashlane

Recall that items in Dashlane are treated as transactions and encrypted with a key derived from the master key based on transaction parameters. For all transactions sharing the same parameters (in particular, KDF and cipher mode; the salt is constant for all transactions) the same derived key is used.

DL01: Transaction Replay. Since the transaction key does not depend uniquely on the transaction, the server can duplicate, reorder or drop transactions without the client noticing.

This attack trivially violates vault integrity. It also serves as a building block for other attacks, see Section 4.4.2.

4.3 Sharing

These attacks target the *sharing* of items. All of the password managers we study implement some kind of sharing, and they fall victim to attacks due to a lack of user authentication.

The attacks in this category range in severity: while the vulnerabilities in LastPass and Dashlane only affect the items being shared, the ones in Bitwarden enable some of our Key Escrow attacks and have further-reaching consequences.

4.3.1 Bitwarden

Recall that in Bitwarden, users can share items with other users in organisations; each organisation has a symmetric key k_{org}, which is stored on the server encrypted under the public RSA key of each member of the organisation.

BW08: Organisation Injection. There is no cryptographic binding between a user's vault (or even the vault abstraction) and the organisations a user is part of. Users learn the list of the organisations they are part of when their client performs a sync to fetch the latest data from the server. Furthermore, copies of the encrypted organisation symmetric key are not authenticated in any way: anyone can encrypt a chosen symmetric key under the public key of any user.

It follows that adding a user to an organisation only requires access to their public key, which is known by the server. Thus, an adversary can create a new organisation and trivially add users to it: they just need to encrypt the organisation symmetric key k_{org} under the target user's public key. The client will silently accept the new organisation at the next sync.

As a corollary, the adversary can also add arbitrary users to any organisation whose symmetric key they know. This attack is described in full detail in Appendix C.2.

IMPACT. The adversary can add users to arbitrary organisations, thus violating vault integrity by adding organisation items to the user's vault. This can be used in framing attacks, for example, by planting incriminating material. Furthermore, it enables several other attacks (e.g., BW03 and BW09).

REQUIREMENTS. The client performs a sync operation. This is done automatically by the client at startup, when the vault page is loaded, in response to some user actions, and upon server-generated events.

BW09: Organisation Overwrite. Immediately after an organisation is created, the client of the creator performs a sync to fetch the latest data from the server. As part of this, the keys of the newly created organisation are fetched. After the sync is completed, the creator of the organisation is able to decrypt the organisation symmetric key using their private RSA key and use it to decrypt all organisation data.

By applying the same method as in Attack BW08 above, the adversary can overwrite the newly created organisation: they can swap the encrypted symmetric key c_{org} and the organisation data with values they control. Upon sync, the client has no way of distinguishing such a malicious key from the legitimate organisation key.

In contrast to Attack BW08 (which creates a new organisation), this attack is entirely undetectable. Note that it is also possible for an adversary to overwrite an organisation after it is created, but in doing so they would delete any pre-existing organisation items.

IMPACT. The adversary can read and modify all the items in the newly created organisation.

REQUIREMENTS. The user creates an organisation.

4.3.2 LastPass

Recall that LastPass implements item sharing by means of shared folders. Items in a shared folder are encrypted with the symmetric shared folder key, which is in turn encrypted under the public key(s) of the recipient(s) of the folder.

LP07: Sharing Key Overwrite. During an item-sharing operation initiated by a user, the server, acting as an adver-

sary, sends an adversary-controlled recipient RSA public key instead of the intended recipient key.

The client (and the user) cannot distinguish the malicious key from the intended one: the recipient RSA public keys are not authenticated in any way, the user interface does not allow for out-of-band verification of key fingerprints, and no key transparency is implemented.

IMPACT. The adversary can read and modify all the items in a shared folder.

REQUIREMENTS. The user shares an item with any recipient.

4.3.3 Dashlane

Like LastPass, Dashlane implements item sharing using public key encryption. The problem of unauthenticated public keys described in Attack LP07 also affects Dashlane.

When a client shares an item with a user or in a group, a fresh symmetric sharing key (k_{shr}) is sampled. This key is either used to directly encrypt the shared item (in the case of user-to-user sharing), or to encrypt group keys which in turn protect shared items (for group sharing). Regardless of setting, the sharing client retrieves the RSA public keys of the recipient(s) from the server and, without verifying their authenticity, encrypts k_{shr} under each recipient's public key, yielding c_{shr}^{user} for each recipient $user$. The client sends each c_{shr}^{user} to the server for storage.

DL02: Sharing Key Overwrite. When the sharing client requests a recipient public key, an adversary can overwrite it with an RSA key pk^{adv} it controls, causing k_{shr} to be encrypted under the malicious public key. The resulting ciphertext c_{shr}^{adv} is then sent to the adversary, allowing them to decrypt it with sk^{adv} and recover k_{shr}. This in turn gives the adversary access to the shared item(s) protected by k_{shr}.

IMPACT. The adversary can read and modify all shared items.

REQUIREMENTS. The user shares an item with any recipient.

4.4 Backwards Compatibility

These attacks exploit backwards compatibility features of Bitwarden and Dashlane. Both password managers have evolved the cryptographic primitives they support over the years, shifting from AES in CBC mode for encryption and PBKDF2 for key derivation to AE schemes such as AES-CBC-HMAC and memory hard PBKDFs like Argon2d. Both also show changes in the key derivation trees, with Bitwarden migrating to better key separation with the introduction of per-item keys.

The current client versions only use modern cryptography by default, but they still offer "legacy" support. The attacks described below enable the adversary to downgrade, sometimes permanently, the primitives used by the clients, reducing the security of the affected password managers to the security of the weakest supported cryptographic algorithm.

4.4.1 Bitwarden

Bitwarden offers several backwards compatibility features, which can all be exploited by a malicious server.

Per-Item Keys. Recall from Section 4.2.1 that Bitwarden version `2024.2.0` enables "per-item keys". When creating a new item, per-item keys are only used if the client enables this feature, and if the server version is recent enough to support per-item keys. In order to determine this, the client fetches the server version at login time by sending a request to the `config` endpoint of the server. This result in the attack BW10 below.

BW10: Disable Per-Item Keys. An adversary can perform a downgrade attack by sending a lower server version such that the client does not enable per-item keys. This is trivial, because the server configuration is not authenticated by the client. As a consequence, all new items will be encrypted with the same key (the user key).

IMPACT. This attack disables a critical security feature, downgrading Bitwarden's key hierarchy and exposing unprotected vault items to numerous other attacks. Recall that Bitwarden does not implement a migration path for legacy items, thus indefinitely exposing affected items even after the attack is concluded.

REQUIREMENTS. None.

Legacy encryption. The latest Bitwarden clients use authenticated encryption (AE) for all symmetric encryption operations by default, thereby ensuring both confidentiality and integrity of the encrypted data. The AE scheme used is AES-CBC with HMAC, combined via the encrypt-then-MAC paradigm. In particular, this encryption scheme is used to decrypt the user key received from the server at login time and to encrypt and decrypt the items inside the vault.

However, older versions of Bitwarden used AES-CBC (without HMAC), which does not offer any integrity protection. To accommodate both formats, each ciphertext has an encryption type attribute (type *0* for AE, type *1* for plain AES-CBC). The encryption type is prepended to the ciphertext and is not integrity-protected.

Furthermore, older versions of Bitwarden did not use the current key hierarchy. To support the old hierarchy, a modern client will generate a new RSA keypair for the user (pk', sk') if one was not provided by the server, and use k_m to encrypt sk' if no user key ciphertext c_u was provided by the server. Both of these features can be seen as backwards compatibility measures, allowing a modern client to support old user vaults and to migrate the user vault to the current feature set.

We exploit these features in to build the attack gadget BWb1, resulting in the attacks BW11 and BW12 below.

BWb1: Known Plaintext Oracle Gadget. An adversary can leverage the generation of RSA keypairs and the support

for encryption using the master key k_m to obtain AES-CBC encryptions of known plaintexts under an (unknown) k_m.

At login time, the adversary responds to the login request with a null c_u and a null private RSA key ciphertext c_{sk}. The absence of c_{sk} will force the user to generate a new RSA keypair. However, since c_u was also set to null, the client will use k_m to encrypt the new secret key using AES-CBC, yielding c'_{sk}, and will then send pk', c'_{sk} to the server.

Notably, sk' also contains an encoding of pk'. Thus, by observing the encrypted c'_{sk} and pk', the adversary learns a 5-block-long plaintext-ciphertext pair.

BW11: User Key Overwriting. The adversary can forge a ciphertext c_u^{adv}, such that the client will decrypt it to a user key $k_u = k_u^{enc} \| k_u^{mac}$ for which the adversary knows k_u^{enc}. Recall that k_u^{enc} and k_u^{mac} are 32 bytes each, spanning two blocks of CBC plaintext.

The adversary constructs c_u^{adv} by using the 5-block plaintext-ciphertext pair from BWb1. Via bit-flipping, the adversary sets the last block of ciphertext to a block P_5 of padding, obtaining a garbled block P_4, and known blocks P_1, P_2, P_3. Finally, the adversary marks this ciphertext as type *0*.

Upon receiving c_u^{adv}, the client will accept it as a valid user key encryption: the ciphertext is encrypted under the master key k_m, the type is *0* so CBC decryption is used, and unpadding succeeds. The client then parses $k_u = k_u^{enc} \| k_u^{mac}$. Since k_u^{enc} consists of blocks P_1 and P_2, the adversary knows k_u^{enc} fully, while k_u^{mac} is unknown to the adversary because of the garbled block P_4.

IMPACT. The adversary learns k_u^{enc}, used to encrypt all the vault items. Therefore, the confidentiality of all items encrypted after the attack is mounted is forfeit. Integrity is preserved, as the adversary does not learn k_u^{mac}.

REQUIREMENTS. The user logs in. Due to the attack, this initial login attempt fails, and the user is redirected to a migration page. Regardless of the action taken by the user after the migration page is displayed, the attack completes successfully during a subsequent login attempt.

BW12: Downgrade to Legacy. The adversary can forge a ciphertext c_u^{adv}, such that the client decrypts it to a user key $k_u = k_u^{enc} \| k_u^{mac}$, where k_u^{mac} is empty.

This can be achieved by following the same steps as in Attack BW11, but using only three blocks of ciphertext: in this case, P_3 is set to a block of padding, and P_2 is garbled. As previously discussed, the client will accept this c_u^{adv} as valid. The ciphertext being too short triggers another backwards compatibility feature: the client parses $k_u = k_u^{enc}$ without the "mac" key part, causing all subsequent encryptions to be performed using CBC-only mode. Note that if per-item keys are enabled, item encryption still uses AES-CBC with HMAC. But the adversary can easily downgrade the client to not use per-item keys, see BW10.

IMPACT. The attack downgrades encryption to CBC-only mode, trivially compromising the integrity of all items encrypted after the attack is executed. In turn, this enables a padding oracle attack [72], leading to a complete compromise of vault confidentiality. We did not develop a PoC for this last aspect of the attack, but the developers of Bitwarden acknowledged its feasibility. We also note that the adversary knows P_1, and therefore learns half of the encryption key k_u^{enc}.

REQUIREMENTS. The same as BW11.

4.4.2 Dashlane

Recall from Section 3.3 that every transaction has a *payload*, which defines the key derivation function, cipher, and mode to use to decrypt the transaction content. Two payload formats are relevant for our attacks: KWC3, using PBKDF2-SHA1 and AES-256-CBC; and Flexible, with a format string which is parsed to select the KDF (PBKDF2-SHA256 or SHA1, Argon2i, or the default Argon2d), its parameters, the cipher (only AES-256), and the cipher mode (the default CBC-HMAC and GCM being the only documented modes).

Modern Dashlane clients default to Flexible mode; upon detecting a KWC3 payload they automatically migrate the transaction to Flexible. Support for the legacy format varies across clients: we verified that the web extension supports it, while the CLI client does not. All of the following attacks only affect clients that support the legacy format. We produced PoCs against the web extension client.

DLb1: CBC Padding Oracle. Unexpectedly, CBC-only encryption is still accepted as an encryption mode in the Flexible payload format of Dashlane and does not trigger a migration. Upon receiving a malformed transaction content ciphertext, the client will emit distinguishable error messages, depending on whether the padding removal fails or if the padding removal succeeds but parsing of the transaction content fails at the application level. This means that an adversary can mount a padding oracle attack [72] using transactions with the modern payload format and CBC-only encryption.

For the attack to work, two conditions must be satisfied. First, due to the error handling code in the client, the transaction type must be Setting. Second, the transaction must use the Flexible payload format with CBC-only set as the cipher mode. Note that Dashlane clients use a separte key k_{cbc} for CBC-only transactions: this means that the padding oracle attack cannot be used to decrypt any modern CBC-HMAC transactions, since they use key $k_{cbc\text{-}hmac}$ instead. Nonetheless, it can be used as a building block for other attacks.

REQUIREMENTS. This attack requires 256 oracle queries per byte of plaintext recovered. The client fetches new transactions (sync) every 5 minutes, meaning that adversary can decrypt one full block in 14 days. Note that this attack can be significantly sped up by more frequent syncs: this can be achieved for instance in the presence of multiple clients.

DLb2: Encryption Oracle. While modern Dashlane clients do not themselves produce transactions vulnerable

to the padding oracle attack described above, the adversary can now use the encryption oracle technique of Rizzo and Duong [37, §3] to create a ciphertext that decrypts to a plaintext of their choosing under the key k_{cbc}.

REQUIREMENTS. This gadget requires running Attack DLb1 once per block of plaintext encrypted.

DL03: Item Injection. Using DLb2, an adversary can encrypt arbitrary plaintext under the key k_{cbc}.

An adversary can then forge arbitrary transactions with Flexible payload, and CBC-only cipher mode. Since the type is a plaintext field of the transaction and does not influence key derivation, the adversary can pick an arbitrary transaction type. The client will accept this transaction as authentic, and correctly derive key k_{cbc} to decrypt it.

IMPACT. The adversary can forge arbitrary transactions. This means that the adversary can now plant arbitrary credentials and notes into the vault, thus violating vault integrity, and change arbitrary settings, enabling furhter attacks.

REQUIREMENTS. Execution of DLb2. Recall that each block of transaction content needs 14 days to be encrypted.

DL04: Remove KDF Iterations. User settings in Dashlane control many client behaviours. Among these, they define what KDF to use for all new transaction payloads, and the respective parameters.

Using Attack DL01, the adversary can remove any existing transaction of type Setting. This moves the user settings to a known clean state. Then the adversary uses Attack DL03 to forge a new transaction of type Setting. When the user creates a new item, the client will derive a key from the master password with the adversary-chosen KDF parameters.

For instance, the adversary can change the number of PBKDF2 iterations from the default of 200,000 iterations to 1, achieving a 200,000x speed-up in a brute-force attack.

IMPACT. The adversary reduces the cost of a brute-force attack on a target user's master password; success enables full control of the target user account.

REQUIREMENTS. Execution of DL01 and DL03.

DL05: CBC-Only Downgrade. The setting transaction also controls which cipher and mode of operation the client uses for future transactions. Similarly to Attack DL04, the adversary can forge settings instructing the client to use CBC-only encryption. When the user creates new items, the client will then always use CBC-only mode. The resulting transaction will be vulnerable to the padding oracle attack in DLb1.

IMPACT. The adversary can decrypt all the transactions produced after the attack is completed.

REQUIREMENTS. Execution of Attacks DL01 and DL03, plus one execution of DLb1 for each decrypted transaction.

Attack timing. In our PoCs, we successfully forge a minimal transaction of type Setting which is only 9 blocks long, enabling us to test Attack DL04 and DL05. We artificially sped up the execution for testing, but we estimate this would take more than 126 days at the normal attack speed. This is a long time, but it is reasonable for a targeted attack.

DL06: Lucky 64. Another Flexible payload setting allows for the client to completely skip key derivation (noderivation), and use the master password directly, if and only if the master password is exactly 64 bytes long. We presume that the intended use of the noderivation mode is for the setting where the user authenticates via SSO, and the master password contains a 64-byte cryptographic key.

In the rare instance of a user actually having picked a master password that is 64 bytes long, the adversary can use Attack DL04 to pick noderivation as a PBKDF and AES-CBC-HMAC as a cipher mode of operation. Then the master password will be split in two: the first 32 bytes will be used as the AES-256 key, and the next 32 bytes will be used as the HMAC key.

IMPACT. The adversary can reduce the cost of a brute-force attack on a target user's master password, via separate brute-force attacks on the two 32-byte halves of the user's master password. For instance, a 64-byte password composed of 10 words has an estimated entropy of 120 bits, but would require on average 2^{60} guesses to recover after the attack, compared to 2^{119} otherwise: a speed-up of a factor 2^{59}.

REQUIREMENTS. Execution of Attack DL04, plus the user selects a (memorable, low entropy) master password of exactly 64 bytes.

5 Mitigations

Our attacks can all be mitigated using a combination of authenticated encryption (AE), key separation (KS), plaintext authentication (AD), public key authentication (PKA), and ciphertext authentication (SC). We detail how in Section 5.1; Table 1 additionally maps attacks to mitigations.

In general, these techniques can be used to design the core of a secure E2EE password manager: in Section 5.2 we briefly sketch such a high-level design.

5.1 Immediate Remediations

In our ongoing discussions with the vendors, the introduction of breaking changes emerges as the major obstacle to deploying our countermeasures. As we argue in Section 6, breaking changes are necessary, and even inevitable. To this aim, we propose the use of specialized password manager clients, with no functionality besides implementing a forced migration to the new vault format. This would prevent any user from losing access to their data, while preserving security for the entire user base. Below, we detail the changes we suggest.

Only use AE (AE). Disabling legacy support for encryption schemes that provide no integrity immediately remediates

most of our Backwards Compatibility attacks on Bitwarden (BWb1, BW11, BW12) and Dashlane (DLb1, DLb2, DL03, DL05, DL06), since they exploit the support of AES-CBC. Encrypting item metadata with AE would remediate BW04 (Unprotected Item Metadata). LastPass would need to introduce AE from scratch. This would remediate many of its Vault Encryption issues (LP05, LP06), and, together with the KS mitigation (see below), also attacks LP02 (Item Field Swapping) and LP03 (Icon URL Item Decryption).

Additionally using authenticated data (AD) to protect metadata fields in the items and user settings in the vault would mitigate the Vault Integrity issues in Bitwarden (BW05, BW04, BW06), LastPass (LP02, LP03, LP06), and Dashlane (DL01).

Full key separation (KS). Most vault integrity issues can be mitigated by using proper key separation: each key should be used only to encrypt a single field of a single vault item, with all keys being created from the master key using a suitable key derivation function and per-item and per-field context strings. Recall that, in contrast to encrypting the whole vault as a monolith, this approach allows efficient synchronisation of changes: each item can be encrypted and synchronised independently, while still protecting the overall integrity.

In Bitwarden, item keys provide a good starting point. Extending this approach to the field level would both make URL checksums redundant, and remediate BW05 (Item Field Swapping) and BW06 (Icon URL Item Decryption). Item keys should also be made mandatory, mitigating BW10 (Disable Per-Item Keys). Together with the AE mitigation, KS would also prevent issues like BWb1 (Known Plaintext Oracle Gadget) from arising. LastPass currently lacks any form of key separation within the vault. Introducing it along with an AE scheme would mitigate LP02 and LP03. In Dashlane, each transaction could be encrypted with a different key, derived according to a unique transaction identifier, to resolve DL01.

Authenticate all plaintext (AD). All settings and metadata that cannot be encrypted should at least be authenticated (i.e., integrity protected). This can be achieved using Authenticated Encryption with Authenticated Data (AEAD) as an extension of AE. This would mitigate the metadata malleability issues (BW04, LP06). Further, authenticating security-critical user settings like PBKDF parameters (such as the iteration count) would mitigate the KDF attacks (BW07, LP04). The client can use the server-provided KDF parameters to derive the authentication key, use it to verify the integrity of the parameters themselves, and – in case of a mismatch – abort before any further communication with the server. On the one hand, this thwarts our immediate attacks: an adversary tampering with the KDF parameters would need to predict the authentication key for the client to accept the modified parameters. On the other hand, as pointed out in [66, §3.2], the security in this setting is circular and relies on non-standard properties of MACs. This measure should therefore be combined with allowlisting of safe sets of KDF parameters in clients.

Similarly, authenticating the last known server version seen by the client by adding it to the vault additionally mitigates BW10 (Disable Per-Item Keys). An attacker can still replay an old version of the vault (with the corresponding server version), but all newly created items are protected.

Public key authentication (PKA). Introducing proper authentication of public keys is non-trivial: a stop-gap mitigation would be storing known public keys in the vault after an (optional) out-of-band verification. This would allow the users to detect public key changes, preventing active attacks. Even when no out-of-band verification is performed, this would still prevent some attack scenarios: the attacker can then only substitute public keys on their first use.

An alternative is to introduce a PKI where a Certificate Authority (CA) signs all public keys. This CA should not be under the control of the same entity that controls the password manager's server, so that it can truly act as a trusted third party. This is particularly relevant in a business setting, given that many companies that use password managers may already operate their own CA infrastructure.

These measures would remediate most of the Sharing issues (BW09, LP07, DL02). They would also allow the user to authenticate the public keys of admins and organisations used for Key Escrow. Additionally, assuming that a correctly signed list of admins is published, this would remediate several Key Escrow issues (BW01, BW03, LP01).

Ciphertext authentication (SC). In Bitwarden, the only cryptographic check of organisation membership is through the decryption of organisation data, encrypted by the user under their own public key. However, as we have noted, anyone can produce such ciphertexts. This results in BW02 (Malicious Key Rotation), BW08 (Organisation Injection), and BW03 (Malicious KC Conversion). These attacks can be prevented by switching to signcryption [74]. By signing plaintexts before encryption, clients can verify that the user themself is the sender. An alternative would be to forgo public-key encryption entirely in favour of symmetric (authenticated) encryption. This is possible since the user is both the sender and the recipient of the affected data. An added benefit of this approach is efficiency gains from removing unnecessary public-key cryptography. However, it might require a larger re-design compared to simply adding signatures. Either approach mitigates all of the above attacks.

A combination of signcryption and PKA is needed when receiving public key ciphertext encrypted by other users, such as organisation invites in Bitwarden, or the list of admins for a team in LastPass. This would be needed to remediate BW01 (Malicious Auto-Enrolment), BW09 (Organisation Overwrite) and LP01 (Malicious Password Reset).

5.2 Future Designs and Limitations

The mitigations we propose can serve as a roadmap for a provably secure password manager design, with minimal performance overhead compared to existing designs, as follows. Starting from a PBKDF-derived key computed from the user's master password, standardized and efficient key derivation functions are used to derive separate symmetric keys for all vault items, thus providing proper key separation. With these keys, vault data is encrypted using a modern AEAD scheme. Using symmetric cryptography with built-in authentication both provides ciphertext integrity and has low performance cost compared to using bespoke mechanisms for achieving vault integrity. All user data is both encrypted and authenticated, with only unavoidable metadata, such as PBKDF parameters, left as authenticated-only additional data. Wherever public-key cryptography is necessary (e.g., for sharing), measures such as certificates are taken to authenticate public keys, and signatures are used to authenticate ciphertexts.

Caveat. We have presented mitigations for all the attacks we found, but there may be other cryptographic attacks yet to be discovered for which our mitigations would be ineffective. Formally defining the security goal of a password manager in the face of a malicious server is a necessary first step toward provable security. The same applies to our high-level sketch of a secure design: only a security analysis backed by formal models and proofs can properly inform a more detailed construction; such analysis is beyond the scope of this paper.

6 Discussion

Password managers deal with critically sensitive data – yet, we have exposed severe attacks against three important vendors in this space. This section explores the manifold reasons for those attacks, discusses broader impacts of our findings, and provides directions for future work.

A burdensome legacy. Password managers were early entrants in the world of end-to-end encrypted applications: they predate all modern end-to-end encrypted messaging applications, with the initial release of LastPass dating back to 2008. Encryption of vaults has always been a focus of these products, but cryptographic best practices have changed a lot in the last years. Dashlane and Bitwarden are more recent, with first releases in 2012 and 2016 respectively, but they were entering a market with quite low cryptographic standards. The early releases of both show a taste for early-2000s cryptography, with lack of integrity protection for encryption and unprincipled designs being prevalent.

Motivated by backwards compatibility, residues of this legacy cryptography have remained across codebase changes and redesigns. In our communication with vendors, they expressed deep concern about leaving users with impossible-to-decrypt vaults because of cryptographic changes. This motivated the extreme lengths to which some of them go in order to support old formats. In the meantime, the security community had learned, through the long saga of attacks on SSL and TLS [4, 8, 9, 15–18, 41, 71], to put aside backwards compatibility and orient itself towards *cryptographic alacrity*: introducing sharp, but clean, breaking changes to keep the ghosts of past cryptographic attacks at bay.

Deceptively simple. Unlike secure channel protocols and end-to-end encrypted messaging, password managers have arguably escaped deep academic scrutiny before now.[10] Perhaps this is because the problem they address looks, on the surface, like a simple matter of key derivation and then encryption, without any interesting problems to solve. After a closer look, we have seen that password managers are far from simple: they have evolved to include complicated protocols for key synchronisation, recovery and rotation, sharing of encrypted elements, and migration between different cryptographic primitives. The complexity they have reached is comparable to that of modern E2EE cloud storage systems.

Even though password managers have evolved, with cryptographic best practices trickling in over time (e.g. AEAD and memory-hard PBKDFs being eventually integrated in Dashlane and Bitwarden), there was never a gold standard to match (like TLS or Signal), or even precise security notions to aim for. This is similar to the situation for E2EE cloud storage, where a recent rash of research [5, 6, 13, 44, 46] has exposed the frailties of deployed systems, and formal foundations closely linked to practice are only now emerging. Indeed, the two applications are close enough that the recent foundational work of [12] could provide a good starting point for password managers too.

Integrity in motion. To draw another parallel with cloud storage, password managers also face the challenge of efficiently synchronizing encrypted data. The naïve approach of storing all user files in a single AE-protected blob is clearly unviable for large amounts of data. Modern password managers, supporting large numbers of items, consequently structure their vaults to enable individual item synchronisation.

As we suggest in Section 5, the apparent tension between integrity and incremental updates can be solved by clean key separation. Preventing deletion remains an open problem; Merkle tree techniques can prevent selective deletion attacks by authenticating the vault structure, but complete roll-back of vaults cannot be ruled out without outsourcing versioning information to a trusted third party or local cryptographic state. The latter approach is problematic for password managers, since users expect access to their vault upon presenting only their master password, even on a stateless client.

[10] They are frequently subjected to code audits and security reviews, but not to *cryptographic* reviews.

Share with care. Many of the vulnerabilities we identify rely, in one form or another, on substituting public keys with adversary-controlled ones. Such attacks have been known since the dawn of public key cryptography [35, §III].

When it comes to verifying public keys of other users, administrators or organisations, password managers run into a very well-known problem: there is no way around the need to provide some form of verification of public keys. In short, PKI (of some form) cannot be avoided.

In our mitigations, we propose relying on PKIs or on Trust-On-First-Use (TOFU) systems. Secure messaging applications are currently exploring the use of Verified Key Directories (VKDs) to overcome this problem. While VKDs would not prevent a malicious server from associating the wrong public key with a user, they make this kind of tampering evident, so that users are alerted when their providers misbehave.

Authenticating public keys is only half the battle. It seems to be a common misunderstanding both here and for E2EE cloud storage [5, 46] that public key encryption (PKE) somehow produces authenticated ciphertexts. In fact, *no* PKE scheme can offer data origin authentication (because, by definition, any party can encrypt). This is why we suggest the usage of primitives like signcryption [74] to correctly bind ciphertexts to their senders.

Securing a "backdoor". Password managers attempt to implement mechanisms for recovery of the vault content in the case the user forgets their master password. There is a clear tension here between security and usability: not having any recovery mechanism provides for the best possible security, but might be untenable for large populations of non-technical users. Admin-assisted recovery can be implemented safely, but success hinges on the key authentication problem discussed above being solved first. Again, we can look at the secure messaging space for state-of-the-art solutions to this problem: messengers like Signal and WhatsApp [27, 33] are converging to the usage of hardware security modules (HSM) in the cloud for protecting account recovery and backups. Similarly, Apple's Advanced Data Protection uses HSMs to allow user to recover their key material from just their PIN [10].

Implementing part of the Key Escrow functionality on an HSM can offer at least some form of guarantee that the server is behaving honestly. Hybrid solutions could include using the HSMs to distribute the keys of the admin, without having to rely on other forms of PKI or out-of-band verification. This is an interesting future direction in the context of formal security guarantees for password managers.

7 Conclusions

We have argued that the malicious server threat model is the correct one for cloud-based password managers: security in this setting is both claimed in vendor advertisement and represents the gold standard for modern end-to-end encrypted systems. We then showed that three popular password managers with more than 60 million users in total are badly broken in this threat model. The vulnerabilities that we describe are numerous but mostly not deep in a technical sense. Yet they were apparently not found before, despite more than a decade of academic research on password managers and the existence of multiple audits of the three products we studied. This motivates further work, both in theory and in practice.

Beyond showcasing the relevance of the malicious server threat model in practice, our attacks show that these systems would benefit from theoretically well-founded designs. Vendors need to update their products to use modern cryptographic primitives and best practices – but to ensure solid foundations, novel definitions to capture security in this setting are also needed. Furthermore, the common issues with features such as key escrow, item-level vault encryption, password sharing, and backward compatibility represent interesting questions for the academic community on how to achieve security while still providing these functionalities.

Working together with vendors to fix the vulnerabilities, our research has contributed to improved security in practice for millions of affected users. It has also increased the vendor-driven interest in formal guarantees of security. We hope that this paper serves to help raise the relevant industry sector's knowledge about encryption further above zero. At the same time, we hope our work spurs the academic community to further research in this area – as well as to consider what more it can do to convert its research artefacts into meaningful impacts on industrial practice.

References

[1] 1Password Surpasses $400M ARR and Expands Executive Team to Advance the Next Era in Identity Security. https://1password.com/press/2025/nov/1password-strengthens-leadership-amid-growth-milestone. Accessed: 2026-01-09.

[2] 1Password. 1Password Whitepaper. https://1passwordstatic.com/files/security/1password-white-paper.pdf. Accessed: 2025-03-25.

[3] Harold Abelson, Ross J. Anderson, Steven M. Bellovin, Josh Benaloh, Matt Blaze, Whitfield Diffie, John Gilmore, Matthew Green, Susan Landau, Peter G. Neumann, Ronald L. Rivest, Jeffrey I. Schiller, Bruce Schneier, Michael A. Specter, and Daniel J. Weitzner. Keys under doormats. *Commun. ACM*, 58(10):24–26, 2015.

[4] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J. Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel

Thomé, Luke Valenta, Benjamin VanderSloot, Eric Wustrow, Santiago Zanella-Béguelin, and Paul Zimmermann. Imperfect forward secrecy: How Diffie-Hellman fails in practice. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015*, pages 5–17. ACM Press, October 2015.

[5] Martin R. Albrecht, Matilda Backendal, Daniele Coppola, and Kenneth G. Paterson. Share with care: Breaking E2EE in Nextcloud. In *2024 IEEE European Symposium on Security and Privacy*, pages 828–840. IEEE Computer Society Press, July 2024.

[6] Martin R. Albrecht, Miro Haller, Lenka Mareková, and Kenneth G. Paterson. Caveat implementor! Key recovery attacks on MEGA. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 190–218. Springer, Cham, April 2023.

[7] Martin R. Albrecht and Kenneth G. Paterson. Analyzing cryptography in the wild: A retrospective. *IEEE Secur. Priv.*, 22(6):12–18, 2024.

[8] Nadhem J. AlFardan, Daniel J. Bernstein, Kenneth G. Paterson, Bertram Poettering, and Jacob C. N. Schuldt. On the security of RC4 in TLS. In Samuel T. King, editor, *USENIX Security 2013*, pages 305–320. USENIX Association, August 2013.

[9] Nadhem J. AlFardan and Kenneth G. Paterson. Lucky thirteen: Breaking the TLS and DTLS record protocols. In *2013 IEEE Symposium on Security and Privacy*, pages 526–540. IEEE Computer Society Press, May 2013.

[10] Apple Support. Advanced Data Protection for iCloud. https://support.apple.com/guide/security/advanced-data-protection-for-icloud-sec973254c5f/web, September 2024. Accessed: 2025-08-25.

[11] Gildas Avoine, Xavier Carpent, and Diane Leblanc-Albarel. In the vault, but not safe: Exploring the threat of covert password manager providers. Cryptology ePrint Archive, Report 2025/1278, 2025.

[12] Matilda Backendal, Hannah Davis, Felix Günther, Miro Haller, and Kenneth G. Paterson. A formal treatment of end-to-end encrypted cloud storage. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part II*, volume 14921 of *LNCS*, pages 40–74. Springer, Cham, August 2024.

[13] Matilda Backendal, Miro Haller, and Kenneth G. Paterson. MEGA: Malleable encryption goes awry. In *2023 IEEE Symposium on Security and Privacy*, pages 146–163. IEEE Computer Society Press, May 2023.

[14] A. Belenko and D Sklyarov. "Secure password managers" and "military-grade encryption" on smartphones: Oh, really? Technical report, Elcomsoft Co. Ltd., 2012.

[15] Benjamin Beurdouche, Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre-Yves Strub, and Jean Karim Zinzindohoue. A messy state of the union: Taming the composite state machines of TLS. In *2015 IEEE Symposium on Security and Privacy*, pages 535–552. IEEE Computer Society Press, May 2015.

[16] Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Alfredo Pironti, and Pierre-Yves Strub. Triple handshakes and cookie cutters: Breaking and fixing authentication over TLS. In *2014 IEEE Symposium on Security and Privacy*, pages 98–113. IEEE Computer Society Press, May 2014.

[17] Karthikeyan Bhargavan and Gaëtan Leurent. On the practical (in-)security of 64-bit block ciphers: Collision attacks on HTTP over TLS and OpenVPN. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 456–467. ACM Press, October 2016.

[18] Karthikeyan Bhargavan and Gaëtan Leurent. Transcript collision attacks: Breaking authentication in TLS, IKE and SSH. In *NDSS 2016*. The Internet Society, February 2016.

[19] Bitwarden. Bitwarden: About Us. https://bitwarden.com/about/. Accessed: 2025-08-26.

[20] Bitwarden. Bitwarden Clients Source Code. https://github.com/bitwarden/clients. Accessed: 2025-03-25.

[21] Bitwarden. How End-to-End Encryption Paves the Way for Zero Knowledge - White Paper. https://bitwarden.com/pdf/resources-zero-knowledge-encryption-white-paper.pdf, 2025. Accessed: 2025-03-25.

[22] Lara Bruseghini, Daniel Huigens, and Kenneth G. Paterson. Victory by KO: Attacking OpenPGP using key overwriting. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 411–423. ACM Press, November 2022.

[23] Michael Carr and Siamak F. Shahandashti. Revisiting security vulnerabilities in commercial password managers. In Marko Hölbl, Kai Rannenberg, and Tatjana Welzer, editors, *ICT Systems Security and Privacy Protection - 35th IFIP TC 11 International Conference, SEC 2020, Maribor, Slovenia, September 21-23, 2020,*

Proceedings, volume 580 of *IFIP Advances in Information and Communication Technology*, pages 265–279. Springer, 2020.

[24] Sunil Chaudhary, Tiina Schafeitel-Tähtinen, Marko Helenius, and Eleni Berki. Usability, security and trust in password managers: A quest for user-centric properties and features. *Comput. Sci. Rev.*, 33:69–90, 2019.

[25] Stephen Checkoway, Jacob Maskiewicz, Christina Garman, Joshua Fried, Shaanan Cohney, Matthew Green, Nadia Heninger, Ralf-Philipp Weinmann, Eric Rescorla, and Hovav Shacham. A systematic analysis of the juniper dual EC incident. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 468–479. ACM Press, October 2016.

[26] Sonia Chiasson, P. C. van Oorschot, and Robert Biddle. A usability study and critique of two password managers. In *Proceedings of the 15th Conference on USENIX Security Symposium - Volume 15*, USENIX-SS'06, USA, 2006. USENIX Association.

[27] Graeme Connell, Vivian Fang, Rolfe Schmidt, Emma Dauterman, and Raluca Ada Popa. Secret key recovery in a Global-Scale End-to-End encryption system. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 703–719, Santa Clara, CA, July 2024. USENIX Association.

[28] Dashlane. 2023 Report - Password Health Scores Around the World. https://www.dashlane.com/uploads/2023/10/2023-Global-Password-Health-Score-Report-1.pdf. Accessed: 2025-08-26.

[29] Dashlane. Dashlane Android Client Source Code. https://github.com/Dashlane/android-apps. Accessed: 2025-03-25.

[30] Dashlane. Dashlane Browser Extension Client Source Code. https://github.com/Dashlane/dashlane-web-extension. Accessed: 2025-03-25.

[31] Dashlane. Dashlane CLI Client Source Code. https://github.com/Dashlane/dashlane-cli. Accessed: 2025-03-25.

[32] Dashlane. Dashlane's Security Principles & Architecture. https://www.dashlane.com/download/whitepaper-en.pdf. Accessed: 2025-03-25.

[33] Gareth T. Davies, Sebastian H. Faller, Kai Gellert, Tobias Handirk, Julia Hesse, Máté Horváth, and Tibor Jager. Security analysis of the WhatsApp end-to-end encrypted backup protocol. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part IV*, volume 14084 of *LNCS*, pages 330–361. Springer, Cham, August 2023.

[34] Rose de Fremery. How Zero Knowledge Keeps Passwords Safe. LastPass Blog, 2023.

[35] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[36] Yihe Duan, Ding Wang, and Yanduo Fu. Security analysis of master-password-protected password management protocols. In Marina Blanton, William Enck, and Cristina Nita-Rotaru, editors, *2025 IEEE Symposium on Security and Privacy*, pages 701–719. IEEE Computer Society Press, May 2025.

[37] Thai Duong and Juliano Rizzo. Cryptography in the web: The case of cryptographic design flaws in asp.net. In *2011 IEEE Symposium on Security and Privacy*, pages 481–489. IEEE Computer Society Press, May 2011.

[38] Andrés Fábrega, Armin Namavari, Rachit Agarwal, Ben Nassi, and Thomas Ristenpart. Exploiting leakage in password managers via injection attacks. In Davide Balzarotti and Wenyuan Xu, editors, *USENIX Security 2024*. USENIX Association, August 2024.

[39] Michael Fagan, Yusuf Albayram, Mohammad Maifi Hasan Khan, and Ross Buck. An investigation into users' considerations towards using password managers. *Hum. centric Comput. Inf. Sci.*, 7:12, 2017.

[40] Sascha Fahl, Marian Harbach, Marten Oltrogge, Thomas Muders, and Matthew Smith. Hey, you, get off of my clipboard - on how usability trumps security in android password managers. In Ahmad-Reza Sadeghi, editor, *FC 2013*, volume 7859 of *LNCS*, pages 144–161. Springer, Berlin, Heidelberg, April 2013.

[41] Christina Garman, Kenneth G. Paterson, and Thyla van der Merwe. Attacks only get better: Password recovery attacks against RC4 in TLS. In Jaeyeon Jung and Thorsten Holz, editors, *USENIX Security 2015*, pages 113–128. USENIX Association, August 2015.

[42] Paolo Gasti and Kasper Bonne Rasmussen. On the security of password manager database formats. In Sara Foresti, Moti Yung, and Fabio Martinelli, editors, *ESORICS 2012*, volume 7459 of *LNCS*, pages 770–787. Springer, Berlin, Heidelberg, September 2012.

[43] Jessica Gentles, Mason Fields, and Garrett Goodman annd Suman Bhunia. Breaking the Vault: A Case Study of the 2022 LastPass Data Breach. http://www.arxiv.org/pdf/2502.04287. Accessed: 2025-03-25.

[44] Nadia Heninger and Keegan Ryan. The hidden number problem with small unknown multipliers: Cryptanalyzing MEGA in six queries and other applications. In

Alexandra Boldyreva and Vladimir Kolesnikov, editors, *PKC 2023, Part I*, volume 13940 of *LNCS*, pages 147–176. Springer, Cham, May 2023.

[45] Jonas Hofmann and Kien Tuong Truong. End-to-end encrypted cloud storage in the wild: A broken ecosystem. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, CCS '24, page 3988–4001, New York, NY, USA, 2024. Association for Computing Machinery.

[46] Jonas Hofmann and Kien Tuong Truong. End-to-end encrypted cloud storage in the wild: A broken ecosystem. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024*, pages 3988–4001. ACM Press, October 2024.

[47] Nicolas Huaman, Sabrina Amft, Marten Oltrogge, Yasemin Acar, and Sascha Fahl. They would do better if they worked together: The case of interaction problems between password managers and websites. In *2021 IEEE Symposium on Security and Privacy*, pages 1367–1381. IEEE Computer Society Press, May 2021.

[48] Microsoft Threat Intelligence. Frozen in transit: Secret blizzard's aitm campaign against diplomats. https://www.microsoft.com/en-us/security/blog/2025/07/31/frozen-in-transit-secret-blizzards-aitm-campaign-against-diplomats/, 2025.

[49] Tibor Jager, Kenneth G. Paterson, and Juraj Somorovsky. One bad apple: Backwards compatibility attacks on state-of-the-art cryptography. In *NDSS 2013*. The Internet Society, February 2013.

[50] Ambarish Karole, Nitesh Saxena, and Nicolas Christin. A comparative usability evaluation of traditional password managers. In Kyung Hyune Rhee and DaeHun Nyang, editors, *Information Security and Cryptology - ICISC 2010 - 13th International Conference, Seoul, Korea, December 1-3, 2010, Revised Selected Papers*, volume 6829 of *Lecture Notes in Computer Science*, pages 233–251. Springer, 2010.

[51] Brian Krebs. Password manager LastPass warns of breach. *Krebs on Security*, June 2015. Accessed: 2025-08-25.

[52] LastPass. LastPass Business. https://www.lastpass.com/products/team-password-manager. Accessed: 2025-03-25.

[53] LastPass. LastPass CLI Client Source Code. https://github.com/lastpass/lastpass-cli. Accessed: 2025-03-25.

[54] LastPass. LastPass Whitepaper. https://support.lastpass.com/s/document-item?language=en_US&bundleId=lastpass&topicId=LastPass/lastpass_technical_whitepaper.html&_LANG=enus. Accessed: 2025-03-25.

[55] LastPass. Knowledge workers have a false sense of password security. https://www.lastpass.com/-/media/7e2b8b715dc44d23ae21c5cf94dc4876.pdf, 2025. LastPass Business Report; Accessed: August 25, 2025.

[56] Sean Lawlor and Kevin Lewi. Deploying key transparency at WhatsApp. https://engineering.fb.com/2023/04/13/security/whatsapp-key-transparency/, 2023.

[57] Zhiwei Li, Warren He, Devdatta Akhawe, and Dawn Song. The emperor's new password manager: Security analysis of web-based password managers. In Kevin Fu and Jaeyeon Jung, editors, *USENIX Security 2014*, pages 465–479. USENIX Association, August 2014.

[58] Peter Mayer, Collins W. Munyendo, Michelle L. Mazurek, and Adam J. Aviv. Why users (don't) use password managers at a large educational institution. In Kevin R. B. Butler and Kurt Thomas, editors, *USENIX Security 2022*, pages 1849–1866. USENIX Association, August 2022.

[59] Daniel McCarney, David Barrera, Jeremy Clark, Sonia Chiasson, and Paul C. van Oorschot. Tapas: design, implementation, and usability evaluation of a password manager. In *Proceedings of the 28th Annual Computer Security Applications Conference*, ACSAC '12, page 89–98, New York, NY, USA, 2012. Association for Computing Machinery.

[60] NordPass. People have around 170 passwords on average, study shows. https://www.globenewswire.com/news-release/2024/05/21/2885556/0/en/People-have-around-170-passwords-on-average-study-shows.html, May 2024. Press Release, Commissioned by NordPass, Conducted by Cint. Accessed: 2025-08-25.

[61] Sean Oesch and Scott Ruoti. That was then, this is now: A security evaluation of password generation, storage, and autofill in browser-based password managers. In Srdjan Capkun and Franziska Roesner, editors, *USENIX Security 2020*, pages 2165–2182. USENIX Association, August 2020.

[62] Sean Oesch, Scott Ruoti, James Simmons, and Anuj Gautam. "It Basically Started Using Me:" An Observational Study of Password Manager Usage. In Simone D. J. Barbosa, Cliff Lampe, Caroline Appert, David A.

Shamma, Steven Mark Drucker, Julie R. Williamson, and Koji Yatani, editors, *CHI '22: CHI Conference on Human Factors in Computing Systems, New Orleans, LA, USA, 29 April 2022 - 5 May 2022*, pages 33:1–33:23. ACM, 2022.

[63] Sarah Pearman, Shikun Aerin Zhang, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. Why people (don't) use password managers effectively. In *Fifteenth Symposium on Usable Privacy and Security (SOUPS 2019)*, pages 319–338, Santa Clara, CA, August 2019. USENIX Association.

[64] Hirak Ray, Flynn Wolf, Ravi Kuber, and Adam J. Aviv. Why older adults (don't) use password managers. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021*, pages 73–90. USENIX Association, August 2021.

[65] Glenn Greenwald Ryan Gallagher. How the NSA plans to infect 'millions' of computers with malware. https://theintercept.com/2014/03/12/nsa-plans-infect-millions-computers-malware/, 2014.

[66] Matteo Scarlata, Matilda Backendal, and Miro Haller. MFKDF: Multiple factors knocked down flat. In Davide Balzarotti and Wenyuan Xu, editors, *USENIX Security 2024*. USENIX Association, August 2024.

[67] Security.org Team. Password Manager Annual Report. https://www.security.org/digital-safety/password-manager-annual-report/, 2024. Also titled: Password Manager Industry Report and Market Outlook (2023-2024). Accessed: 2024-11-14.

[68] David Silver, Suman Jana, Dan Boneh, Eric Yawei Chen, and Collin Jackson. Password managers: Attacks and defenses. In Kevin Fu and Jaeyeon Jung, editors, *USENIX Security 2014*, pages 449–464. USENIX Association, August 2014.

[69] Ben Stock and Martin Johns. Protecting users against XSS-based password manager abuse. In Shiho Moriai, Trent Jaeger, and Kouichi Sakurai, editors, *ASIACCS 14*, pages 183–194. ACM Press, June 2014.

[70] Karim Toubba. Security incident update and recommended actions, 2023.

[71] Mathy Vanhoef and Frank Piessens. All your biases belong to us: Breaking RC4 in WPA-TKIP and TLS. In Jaeyeon Jung and Thorsten Holz, editors, *USENIX Security 2015*, pages 97–112. USENIX Association, August 2015.

[72] Serge Vaudenay. Security flaws induced by CBC padding - applications to SSL, IPSEC, WTLS... In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 534–546. Springer, Berlin, Heidelberg, April / May 2002.

[73] Rui Zhao, Chuan Yue, and Kun Sun. A security analysis of two commercial browser and cloud based password managers. In *International Conference on Social Computing, SocialCom 2013, SocialCom/PASSAT/BigData/EconCom/BioMedCom 2013, Washington, DC, USA, 8-14 September, 2013*, pages 448–453. IEEE Computer Society, 2013.

[74] Yuliang Zheng. Digital signcryption or how to achieve cost(signature & encryption) $\ll$ cost(signature) + cost(encryption). In Burton S. Kaliski, Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 165–179. Springer, Berlin, Heidelberg, August 1997.

A Ethical Considerations

Our work concerns the security analysis of three products claimed to have tens of millions of users in a threat model that we have argued as being worthy of serious consideration. Our analysis has shown these products to be badly broken in our chosen model. This situation certainly calls for a detailed statement concerning possible harms arising from our research, how these are mitigated in our approach, and the extent to which they are balanced by other outcomes. We formulate this in terms of a stakeholder-based analysis. We take a consequentialist viewpoint to conduct our analysis.

The main stakeholders related to our work, as we see them, are: us, the researchers; the vendors of the affected products (Bitwarden, Dashlane, LastPass); vendors of related products; customers using the affected products (both individuals and organisations); society at large.

We eschew further discussion of ourselves as stakeholders, since we cannot readily set aside our own feelings on the matter to provide a rational analysis of how we might be affected by the publication (or otherwise) of our work.

Concerning the vendors of the affected products: they may suffer reputation and thence commercial damage as a result of our work being published. For example, it has been reported that LastPass has lost many customers due to repeated breaches [43, 67]. This potential damage is mitigated in our approach by providing the affected vendors with a 90-day disclosure window during which we did not talk in public about the research. During this window, we supported the vendors, initially with very detailed descriptions of the security issues we have found, then repeatedly offering support to address the issues (video-conferences, review of patches). We believed that 90 days should be sufficient time to address the identified issues, but we always remained open to extending this time

period upon request, and actually did so for one vendor. We consider this approach to be relatively standard for the field of software vulnerability disclosure, except perhaps for the level of our engagement in the triage process. For example it is endorsed by ENISA, an EU body.[11]

We disclosed to Bitwarden on 27.01.2025, to LastPass on 04.06.2025, and to Dashlane on 29.08.2025. We had a video-conference and numerous email exchanges with Bitwarden. At the time of writing, they are well advanced in deploying mitigations for our attacks: BW01, BW03, BW11, BW12 were addressed, the minimum KDF iteration count for BW07 is now 5000, and their roadmap includes completely removing CBC-only encryption, enforcing per-item keys and changing the vault format for integrity. On 22.12.25 they shared with us a draft for a signed organisation membership scheme, which would resolve BW08 and BW09. At our request, to maintain anonymity, they have not yet credited us publicly for the disclosure, but plan to do so. In the case of LastPass, our initial contact via the official e-mail channel on 04.06.2025 did not receive a response. One of us reached out to the CTO over social media, after which our original e-mail was found in the organisation's spam filter on 26.06.2025. We mutually agreed with LastPass to reset the 90-day clock. They provided us with regular updates and we also met for a video-conference to clarify our findings and discuss their mitigation plans. They accepted our request to work with them directly rather than with their preferred bug bounty program because the latter would impose publication restrictions that were not acceptable to us; on 7.10.25 they awarded us two bug bounty payments. At the time of writing, LP03 has been addressed. Our initial email to Dashlane was also lost, but our second attempt at contacting them on 05.09.2025 was successful. Since then, we worked with their engineers to replicate our attacks, and had several discussions over email as well as one video-conferencing call. At the time of writing, DL03, DL04, DL05 and DL06 have been mitigated by disallowing CBC-only mode in Flexible payloads; no remediation is planned for DL01 and DL02.

Upon agreement with all involved vendors, we shared a preprint of our research paper with them in a joint thread, allowing them to also discuss the findings and take-aways among themselves. We also coordinated with them on holding back on public statements concerning the vulnerabilities until the mutually agreed date of 16.02.2026, significantly later than the end of any of the 90-day disclosure periods.

Concerning vendors of related products: there is a clear risk that the issues we have found in the three vendors we have studied may extend to other vendors, either *mutatis mutandis* or with more substantial yet simple changes. In mitigation to the possible harms, we are aware that there is an informal industry forum where our work was already discussed as a result of the disclosure to Bitwarden. While this formally broke any agreement concerning coordinated disclosure (and arguably relieved us of the burden to maintain confidentiality), we also found it understandable and saw it as having a positive effect in alerting other vendors to the possibility of problems in their products. Vendors who have a mature security mindset could already start to analyse their own systems. We will also publicise our work widely after the end of the disclosure periods, including by sharing it (by request from Dashlane) with a group of CTOs of other password managers on the market.

We turn now to customers of this collection of vendors. Although we consider the malicious threat model to be realistic, we also consider that the majority of users would not be targeted by attacks of the type we have presented because of the stringent conditions needed to mount them. Higher risk individuals and organisations, however, could be at substantial risk of harm. Unfortunately, we cannot exclude the possibility that our attacks were already known to advanced threat actors – after all, we have learned from the Snowden revelations that national security agencies are routinely tasked with penetrating systems like the ones we analyse and are willing to conduct active attacks on targets. The best mitigation for these parties is to trust that vendors will rapidly and effectively patch their systems, and here we have made real effort to engage with the affected vendors to assist them in this process.

Finally, we see society at large as benefitting from our work, in that our work should help to raise the security bar for password managers, pushing their vendors to either improve security or make clearer statements about what security their systems actually provide, so that customers can judge (perhaps with the help of expert guides) whether the products meet their requirements or not. In the longer term, we see our work as contributing to reducing the possible harms from using password managers by making them more secure.

Given the overall balance of benefits versus harms, and the extensive mitigations to limit those harms, we took the decision to proceed with the research and to publish it.

B Open Science

Our artefacts consist of proof-of-concept code for all the attacks in the paper. While some of the attacks are in the process of being mitigated, others remain exploitable due to the need for longer-term mitigations (e.g. public key authentication). Due to the sensitivity of the PoCs, we came to the decision that we should also not make these PoCs generally available before all of the agreed embargo periods were complete. However, we made all the artefacts available to the program committee upon request. At the precise time of writing, all the 90-day disclosure periods have ended, but we have agreed a common public disclosure date with the affected vendors of 16.02.2026. We will make PoCs for mitigated attacks publicly available at that date at the following location: https://doi.org/10.5281/zenodo.17977565.

[11] https://www.enisa.europa.eu/publications/coordinated-vulnerability-disclosure-policies-in-the-eu

C Detailed Attack Descriptions

This section contains attack details that were omitted in the main body.

C.1 Details of BW02: Malicious Key Rotation

We describe here the version of the attack where the adversary tricks the client into believing that the user is enrolled in account recovery for a new, fake organisation at the time of key rotation.

Concretely, the adversary will create an organisation org', such that $UserResetPassword$ is set to true and $ResetPasswordKey$ is set to a non-null value. The adversary then generates a 64-byte symmetric key to be used as the organisation symmetric key, and an RSA keypair.

$$k'_{org} \leftarrow\$ \mathsf{AES\text{-}CBC\text{-}HMAC.KeyGen}(512)$$
$$(pk', sk') \leftarrow\$ \mathsf{RSA.KeyGen}(2048)$$

The adversary then encrypts the organisation symmetric key under the victim's public key (pk), and the organisation's private key sk'_{org} under k'_{org}.

$$c'_{org} \leftarrow\$ \mathsf{RSA.Enc}(pk, k'_{org})$$
$$c'_{skOrg} \leftarrow\$ \mathsf{AES\text{-}CBC\text{-}HMAC.Enc}(k'_{org}, sk'_{org})$$

This is the resulting new organisation data org':

```
org' = {
  OrganizationId = Guid.Empty,
  UserId = user.Id, // victim's user ID
  Name = "",
  UseResetPassword = true,
  ResetPasswordKey = "Any non-null value",
  Key = c'_org,
  PublicKey = pk'_org,
  PrivateKey = c'_skOrg,
  [...]
};
```

Using this organisation data in the sync response will trick the client into believing that they are a member of org' and are enrolled in account recovery. (See the details of the organisation injection attack BW08 in the next section, Appendix C.2, for more information about the format of the sync response.) As a consequence, the client will create an account recovery ciphertext for the new user key and send it to the server; the adversary can then decrypt it to obtain k'_u.

C.2 BW08: Organisation Injection

Here we describe in detail how an adversary can create a new, fake organisation and trick a Bitwarden client into acting as if the user is a member of this organisation.

The first step of the attack is to generate a 64-byte symmetric key to be used as the organisation symmetric key, and an RSA keypair.

$$k'_{org} \leftarrow\$ \mathsf{AES\text{-}CBC\text{-}HMAC.KeyGen}(512)$$
$$(pk'_{org}, sk'_{org}) \leftarrow\$ \mathsf{RSA.KeyGen}(2048)$$

Once the malicious organisation keys have been generated, the adversary needs to encrypt the organisation symmetric key under the victim's public key (pk), the organisation's private key sk'_{org} under k'_{org}, and the default collection name under k'_{org}, as follows:

$$c'_{org} \leftarrow\$ \mathsf{RSA.Enc}(pk, k'_{org})$$
$$c'_{skOrg} \leftarrow\$ \mathsf{AES\text{-}CBC\text{-}HMAC.Enc}(k'_{org}, sk'_{org})$$
$$c'_{collName} \leftarrow\$ \mathsf{AES\text{-}CBC\text{-}HMAC.Enc}(k'_{org}, t(defaultCollection)) .$$

The function t simply returns the default collection name used by the client depending on the default language of the client. For instance, if the client language is English, then $t(defaultCollection) =$ "Default Collection".

The second step is to create a sync response that contains the malicious organisation data.

$$syncRes' \leftarrow syncRes$$
$$syncRes'.\mathsf{newOrg}.c_{org} \leftarrow c'_{org}$$
$$syncRes'.\mathsf{newOrg}.pk \leftarrow pk'_{org}$$
$$syncRes'.\mathsf{newOrg}.c_{sk} \leftarrow c'_{skOrg}$$
$$syncRes'.\mathsf{collections.newColl}.name \leftarrow c'_{collName}$$

Here $syncRes'.\mathsf{newOrg}$ represents the injected organization.

Finally, to complete the attack, two more HTTP GET responses need to be forged: the responses to requests to endpoints "organizations/orgId/collections" and "organizations/orgId/collections/details", respectively.

For the endpoint "organizations/orgId/collections", we can fake $\mathsf{collRes}'$ by modifying the legitimate collRes as follows:

$$\mathsf{collRes}' \leftarrow \mathsf{collRes}$$
$$\mathsf{newColl}.name \leftarrow c'_{collName}$$

In a similar manner, for the endpoint "organizations/orgId/collections/details", we can create $\mathsf{collDetailRes}'$ by modifying the legitimate collDetailRes as follows:

$$\mathsf{collDetailRes}' \leftarrow \mathsf{collDetailRes}$$
$$\mathsf{newColl}.name \leftarrow c'_{collName}$$

Having created appropriate responses to all client requests, the adversary has now successfully forged a new organisation of which the client will think it is a member.

C.3 BW06: Icon URL Item Decryption

We describe the details of the Icon URL Item Decryption attack on Bitwarden and list the requirements on the data field in

order for the attack to succeed. (See 'practical considerations' below.)

To mount the attack, the adversary targets one or more items that do not have per-item keys. As mentioned in the main body, encrypted items are stored on the server as JSON files where the JSON keys are in plaintext, but the JSON values are encrypted. An example of an encrypted item c_{item} is shown below:

```
c_item = {
  "Name": "2.4nAqS...Po3Q8ng="
  "Username": "2.a/6Nn...spvAnXI=",
  "Password": "2.B3s29...AkWRY+0="
}
```

Since the JSON keys are unencrypted, an adversary can associate the ciphertexts with the information they encrypt without having to perform decryption. For instance, an adversary can infer that the ciphertext relative to the JSON key "$Password$" contains the encryption of the password for the login item.

Given the above, an attacker can target an encrypted item c_{item} and tamper with it to obtain $c_{item}{}'$ with the following steps:

1. Let $c_{item}{}' \leftarrow c_{item}$.
2. Let c_{secret} be a ciphertext from c_{item} containing a value of interest (e.g., the ciphertext associated with the "Password" JSON key)..
3. Craft a new JSON URL entry $url\_entry \leftarrow$ "{"Uri" : c_{secret}}"
4. Add a new URL field to $c_{item}{}'$ by adding url_entry to the JSON list under the "$Uris$" JSON key. Ensure that url_entry is the first item in the list, as only the first URL will be queried to the icon server.

   If the item does not have URLs, create a new JSON key "$Uris$" containing a list and add url_entry to the list.

For instance, starting with the c_{item} above, the adversary can insert the password ciphertext in the URL and obtain the following encrypted item:

```
c_item = {
  "Name": "2.4nAqS...Po3Q8ng="
  "Username": "2.a/6Nn...spvAnXI=",
  "Password": "2.B3s29...AkWRY+0=",
  "Uris": [{"Uri": "2.B3s29...AkWRY+0="}],
}
```

Once $c_{item}{}'$ is created, it is sufficient to serve it to the client in place of c_{item} in a sync response of the server.

PRACTICAL CONSIDERATION. The sensitive data in $c_{item}{}'$ will only be queried to the icon server if all the conditions below are satisfied.

1. (It does not contain the string "$://$" and contains the character '.') or (it starts with the string "$http$" and it contains the character '.' and it does not start with either of the following strings "$data:$", "$about:$" and "$file:$").
2. It does not contain the character '!'.

Lastly, the sensitive data inserted will be passed to the getHostname function of the tldts library. As a consequence, some parts of sensitive data might be removed from the HTTP GET query.

C.4 BW07: Remove KDF Iterations

The adversary forges a pre-login response p' for the victim containing a low number of iterations. This is trivial as the data in the pre-login response is not authenticated. The adversary should aim for the lowest number of iterations possible, which is one. A value of zero or less will trigger a client error, and the master key hash will not be sent to the server. The forged pre-login response p' is a JSON object containing the following fields:

```
{
  "Kdf": "PBKDF2",
  "KdfIterations": 1,
  // "KdfMemory": <KdfMemory>,
  // "KdfParallelism": <KdfParallelism>
}
```

KdfMemory and KdfParallelism are commented out because only relevant to iArgon2.

Once p' is sent to the client, it will compute and send to the server h'_m:

$$k'_m \leftarrow \mathsf{PBKDF2}(pw, salt = email, i = 1)$$
$$h'_m \leftarrow \mathsf{PBKDF2}(k'_m, salt = pw, i = 1)$$

The adversary observes h'_m and can then mount a brute-force attack.

C.5 LP01: Malicious Password Reset

At each login, the LastPass client checks whether they are required by their organisation to enrol into super-admin-assisted account recovery. This is done by sending an HTTP POST request to the endpoint "`login.php`". The XML response to this request specifies whether the client should enrol into account recovery: if the response contains a list of super-admins, this signals the client that they must enrol into account recovery; otherwise, if the response does not contain such a list, no action is performed.

More concretely, the list is encoded in the response by specifying two XML attributes for each super-admin: the super-admin ID `sauid` and their public key `sakey`.

$$\mathsf{sauid} \,\|\, \mathsf{str}(i) \leftarrow \text{"}\mathsf{str}(id^i_{adm})\text{"}$$
$$\mathsf{sakey} \,\|\, \mathsf{str}(i) \leftarrow \text{"}\mathsf{hex}(pk^i_{adm})\text{"}$$

where id^i_{adm} is the user ID of the i-th super-admin, pk^i_{adm} is the RSA public key of the i-th super-admin, for each $i \in [0, N_{adm}]$ where N_{adm} is the total number of super-admins designated for account recovery.

As a concrete example, an intercepted login XML response specifying one super-admin designated for account recovery is as follows:

```
<?xml version="1.0" encoding="UTF-8"?><response>
    ...
    sauid0="380365424"
    sakey0="30820120300D...BB1F89020111"
    ...
</response>
```

To mount the attack, the adversary modifies the login response by adding a malicious super-admin having an adversary-controlled public key. More precisely, the adversary forges the response res' as follows:

$$res' \leftarrow res$$
$$res'[\texttt{sauid} \,\|\, \texttt{str}(j)] \leftarrow \text{“}\texttt{str}(id'_{adm})\text{”}$$
$$res'[\texttt{sakey} \,\|\, \texttt{str}(j)] \leftarrow \text{“}\texttt{hex}(pk'_{adm})\text{”}$$

where id'_{adm} is an arbitrary integer chosen by the adversary, pk'_{adm} is the public key the adversary controls, and $j = N_{adm} + 1$. In practice, the super-admin ID is not validated by the client, so in the attack this ID is can be set to to an arbitrary value.

D Attacks on 1Password

1Password is the fifth most popular password manager (after Google, Apple, LastPass, and Bitwarden) according to a recent market survey [67]. According to their own reported numbers, 1Password have 180,000 business customers and “millions” of users, including one million developers [1]. Similar to the other password managers we study, the root of security in 1Password is the user's master password; but unlike all other three products, 1Password also includes a high entropy cryptographic key in the key derivation, the so-called *secret key*. This means that users cannot access their vaults using just their master password: they also need access to the secret key. 1Password therefore starts with a security advantage compared to its competitors (at the cost of some loss of usability): thanks to the involvement of the secret key, brute-force attacks should be out of reach.

The key derived from the password and the secret key is referred to as the *kek*, or key-encryption key.[12] The *kek* is used to encrypt (using AES-GCM) the secret part of an RSA keypair, or *keyset*, in 1Password terminology. The keyset is then in turn used to encrypt the *vault key* k_v using RSA-OAEP. Finally, the vault key is used to encrypt individual items. Figure 3 depicts this key derivation process.

Sharing Key Overwrite. 1Password supports sharing vaults. This is implemented by encrypting a vault key under the public key of the user with whom the vault is shared.

[12] More details of this key derivation process, which uses PBKDF2 and HKDF internally, are available in the 1Password whitepaper [2].

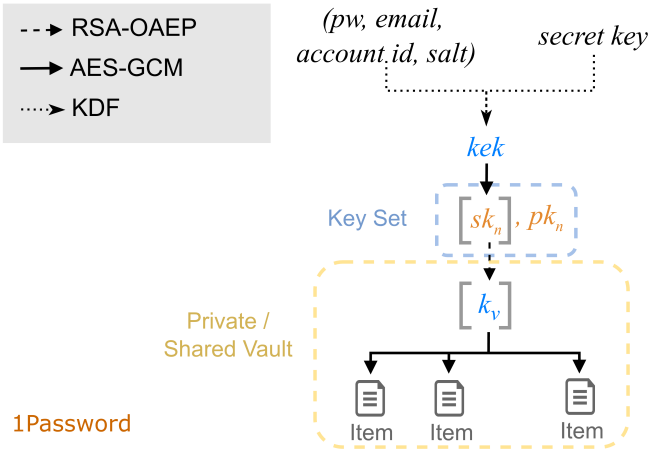

Figure 3: Key hierarchy for 1Password.

Much like the other products we analyse, 1Password lacks authentication of public keys. This trivially enables sharing attacks similar to BW09, LP07 and DL02, something that the 1Password whitepaper [2, Appx. C] openly acknowledges:

> *At present, there's no robust method for a user to verify the public key they're encrypting data to belongs to their intended recipient. As a consequence, it would be possible for a malicious or compromised 1Password server to provide dishonest public keys to the user and run a successful attack.*

While it is true that authenticating public keys of other users is non-trivial, it is easy for a user to verify whether a public key ciphertext has been produced by themselves: signatures in general (and signcryption in particular) solve this problem. 1Password not only lacks authentication of public keys, but also of public-key ciphertexts. This affects not only the security of the credential-sharing feature, but also the confidentiality of the entire vault. This is showcased by the vault substitution attack described below.

Vault Substitution Attack. 1Password uses RSA-OAEP to encrypt the vault key with the user's keyset. While this protects the confidentiality of vault keys, it does not authenticate them, creating an opportunity for a ciphertext substitution attack. In particular, anybody can **encrypt** to the public key of a given keyset, since the public part is not protected.

For instance, a malicious server can execute the following attack at the time of vault creation:

1. The user initiates vault creation, generating a keyset (sk, pk) and a vault key k_v, and saves the encrypted vault key $c = \text{RSA-OAEP.Enc}(pk, k_v)$. The user logs out.
2. The server samples a fresh vault key k'_v, and encrypts k'_v under the public RSA key in the keyset: $c' = \text{RSA-OAEP.Enc}(pk, k'_v)$.
3. The user logs in again, and is offered c' instead of c from the server. The user decrypts c' with their private key

and uses k'_v to encrypt all new vault items. Since the server knows k'_v, it can recover all newly encrypted items as well as inject new items into the vault.

The root cause of this attack is the lack of cryptographic binding between the vault key k_v and the user's key encryption key derived from the password. More specifically, there is no mechanism allowing the client to verify that a vault was genuinely created by the user themselves rather than injected by the server.

IMPACT. Complete compromise of vault confidentiality and integrity. The adversary can read and decrypt all vault contents encrypted after the attack, including passwords, credit card information, secure notes, and other sensitive data stored in the vault. Similarly, they can inject new items into the vault after the attack.

REQUIREMENTS. The client fetches key material from the server, for example due to the user logging in on a new device. If executed on a non-empty vault, the attack results in the client losing access to all items already in their vault, while leaking any new items added to the vault after the attack took place. If the attack is executed at the time of vault creation, the attack is effectively undetectable by the client, since it cannot distinguish between a ciphertext it created and the ciphertext created by the server during the attack.

PROPOSED MITIGATION. A straightforward mitigation is to have the client sign vault keys using the RSA private key in the keyset before encrypting them with the RSA public key. Ideally, two different key pairs would be used for the signing and encryption operations, to avoid any possible key reuse vulnerability. Upon retrieval, the client decrypts c and verifies the signature using the public key in the keyset, confirming that the vault key originated from the user.

This self-authentication prevents server-injected vault key ciphertext from being accepted while requiring no additional public key infrastructure, user interaction, or key distribution mechanisms. This mitigation could also apply to shared vaults: upon receiving a shared vault ciphertext, the client can prompt the user to accept the vault and then sign it, reducing the problem of public key authentication to only the first time a vault is shared.

KDF Parameter Downgrade. The KDF parameters that control the computational cost of deriving the key lncryption key (*kek*) from the master password and secret key are provided by the server and trusted without client-side verification. A malicious server can reduce the iteration count from the default 650,000 iterations to a minimal value of 10,000 iterations.

IMPACT. Although this attacks weakens the bruteforce protection provided by slow hashing, the secret key still contributes enough entropy to make a bruteforce attack infeasible.

PROPOSED MITIGATION. The attack has limited impact, but it would be easy for 1Password to prevent it entirely: the secret key can be used (with proper key derivation) to authenticate the KDF parameters with a cryptographic MAC.

Disclosure. We disclosed our findings to 1Password. Their response was that they regard them as arising from already known architectural limitations. They did not request an embargo period.