This work will appear at USENIX Security 2025

Detecting Compromise of Passkey Storage on the Cloud

Mazharul Islam[*◇], Sunpreet S. Arora[‡], Rahul Chatterjee[◇], Ke Coby Wang[†‡]
[◇]*University of Wisconsin—Madison,* [‡]*Visa Research*

Abstract

FIDO synced passkeys address account recovery challenges by enabling users to back up their FIDO2 private signing keys to the cloud storage of passkey management services (PMS). However, it introduces a serious security risk — attackers can steal users' passkeys through breaches of PMS's cloud storage. Unfortunately, existing defenses cannot eliminate this risk without reintroducing account recovery challenges or disrupting users' daily account login routines. In this paper, we present CASPER, the first passkey breach detection framework that enables web service providers to detect the abuse of passkeys leaked from PMS for unauthorized login attempts. Our analysis shows that CASPER provides compelling detection effectiveness, even against knowledgeable attackers who strategically optimize their attacks to evade CASPER's detection. We also show how CASPER can be seamlessly integrated into the existing passkey backup, synchronization, and authentication processes, with only minimal impact on user experience, negligible performance overhead, and minimum deployment and storage complexity for the participating parties.

1 Introduction

FIDO2-based user authentication has compelling security guarantees which allows users to log into their web service accounts with device-bound private keys. While they are promising at (finally) replacing passwords, recovering accounts in the event of device loss remains a significant challenge preventing the adoption of FIDO2. To address this challenge and accelerate the adoption of FIDO2-based passwordless user authentication, Microsoft, Apple, and Google announced *passkeys* [1], which enables cloud backup and multi-device synchronization of FIDO2 private keys. Today, there are many passkey management services (PMS) that allow passkey backup, e.g., iCloud Keychain from Apple [14], Google Password Manager [12], Password Monitor from Microsoft [11], 1Password [5], LastPass [4], and DashLane [3].

While synced passkeys address the account recovery concern, backing up passkeys to a centralized cloud server poses a serious security risk: attackers can steal users' passkeys through PMS breaches and use them to take over users' web service accounts. The breach can happen when an attacker compromises a user's PMS account (possibly by guessing the account password) or when an insider attacker gains access to the cloud backup storage [27, 82]. Unfortunately, existing passkey backup and synchronization implementations lack the capability to detect unauthorized access to users' passkeys stored at the PMS.

In this paper, we address this problem by proposing CASPER[1], which, to the best of our knowledge, is the first framework to detect the abuse of passkeys leaked from PMS. Underlying CASPER is a decoy-based detection technique [30, 50, 75]: it hides the *real passkey* within a list of decoy passkeys that are indistinguishable from the real one. As a result, an attacker who steals the passkeys from PMS and attempts to use them to log into the user's account at a relying party (RP), e.g., a website, inadvertently will end up using a decoy passkey with high probability. These login attempts will trigger breach detection at the RP, indicating the abuse of passkeys stolen from the PMS. However, unlike focusing on allowing relying parties (RP) to detect breaches of *their own password databases*, as explored in prior works [30, 50, 75], CASPER enables the relying parties to detect breaches of PMS — a different service. To do so, there are additional usability challenges that CASPER must overcome. For example, false detections by a denial-of-service attacker must be avoided, as they would undermine the trustworthiness of any breach detection system. Arguably more importantly, even if an additional user secret is needed in such a system, it should be easy for users to manage so it avoids reintroducing the account recovery problem, and it should not disrupt their daily account login routines.

To meet these requirements, we design a new passkey backup and restoration (BnR) protocol and a compromise

*Work partially done during a research internship at Visa Research.
†Corresponding author: coby.wang@visa.com

[1] CASPER is short for Capturing pASskey comPromise by attackER.

detection (CD) algorithm. The BnR protocol protects the real passkeys by encrypting them under a *recovery key* that is retrievable given a user secret (denoted by η, only known to the user). A user with the correct η will be able to recover the real passkey for account logins.

An attacker who could have already obtained users' passkeys from PMS breaches without CASPER deployed must now guess η in order to decrypt and retrieve the user passkeys. Each guess will produce a well-formed passkey, i.e., a well-formed private signing key but not necessarily the *correct* key. As a result, without η, the attacker cannot determine if their guess yields the real passkey or a well-formed decoy unless they test it by attempting to log in with it to the user's RP account. As the decoys are indistinguishable from the real passkey from the attacker's perspective, they will easily end up using a decoy passkey during the login attempt. RPs that implement our CD algorithm will detect such attempts and deem them unauthorized use of passkeys leaked from PMS. This detection will enable RPs to make informed decisions and promptly mitigate such threats.

To show the effectiveness of CASPER, we model sophisticated attackers who strategically optimize their attack strategies to evade CASPER's detection by leveraging useful information leaked from *already compromised* websites where users have accounts and CASPER is deployed. Our detection effectiveness analysis shows that CASPER provides compelling true-detection and negligibly small false-detection probabilities even when the user secret η contains low entropy (e.g., as low as 5 bits) — confirming that CASPER does not impose additional high-entropy secret management burdens on users. Also, CASPER does not impose any extra tasks on users during their daily account login routines. Importantly, as a general framework with η serving as a configurable component, CASPER can be fine-tuned for specific authentication scenarios balancing deployability, usability, and security.

We provide a prototype implementation [47] and show that CASPER can be seamlessly incorporated into FIDO2 authentication protocols (i.e., CTAP 2.0, WebAuthn) without introducing significant storage costs or causing noticeable login delays. As CASPER aligns with FIDO Alliance's ongoing exploration of trusted signals to detect abuse of synced passkeys, we believe our work will spark interest within the authentication community to consider its adoption in practice.

To summarize, our contributions are as follows.

- We propose CASPER, the first framework to detect the abuse of FIDO2 synced passkeys leaked from passkey management service (PMS) providers. Importantly, CASPER can also be easily extended to detect breaches of other cryptographic credentials that are widely used today, such as HMAC-based / time-based one-time passwords (HOTP / TOTP) seeds.

- We demonstrate the detection effectiveness of CASPER systemically against sophisticated attackers who tries to evade detection by leveraging breaches from other already breached websites.

- Through a prototype implementation of CASPER, we confirm that CASPER introduces negligible performance and storage overhead for all parties involved and demonstrate that deploying CASPER requires minimal modifications to PMS and RPs.

2 Background and Related Work

In this section, we provide background on FIDO2 authentication (Section 2.1) and discuss related work (Section 2.2).

2.1 Background

FIDO2 authentication method. FIDO2 consists of a set of sub-protocols (e.g., CTAP 2.0 [36] and WebAuthn [73]) with digital signature schemes serving as its cryptographic heart. Briefly, during registration, the user's authenticator device creates a signature key pair (s, v), where s and v are the private signing key and public verification key, respectively. Only v is sent to the website while s stays private on the authenticator. During the login phase, RP sends a challenge to the authenticator, and the authenticator then uses s to produce a response to the challenge. Finally, the RP can use the corresponding verification key v to verify the response and decide whether to grant the user account access.

FIDO2 authentication is not only resistant to guessing and phishing attacks, which traditional passwords have been long suffering from, but also protects users' accounts against RP data breaches — a breached v reveals only a negligible amount of information about its corresponding s if the underlying digital signature schemes are secure, and importantly, nothing about users' credentials for other RP accounts.

Account recovery concerns. The improved security guarantees of FIDO2 [17, 19, 42] have prompted their adoption either as a single-factor replacement for passwords [1] or itself as multi-factor authentication (MFA) [9]. However, the lack of secure and user-friendly account recovery solutions for the key management required by such cryptographic protocols has been a source of serious user frustration and concerns [18, 35, 52, 57, 59, 61, 68, 81]. This happens when users' FIDO authenticators — the devices to which users' FIDO2 private signing keys are bound — become permanently unavailable due to device loss, reset, or theft. For example, recent studies [18, 81] identify the lack of convenient and secure account recovery options for FIDO2 passwordless authentication as a major obstacle to eliminating passwords on the web in the foreseeable future. A study by Las et al. [52] found that over 60% of the participants expressed serious concerns about secure account recovery in enterprise settings. Some RPs attempt to resolve such user frustration and concerns by offering easy-to-use yet insecure account recovery or backup authentication options, e.g., user-chosen passwords,

email/phone-based one-time codes, or secret questions, for FIDO authentication. This, however, overshadows the security benefits provided by FIDO authentication and degrades the overall security of users' accounts [72]. To tackle the account recovery problem, industry-led efforts have emerged to encourage the adoption of synced passkeys, which we will introduce next.

Synced passkeys. Synced passkeys are rebranded FIDO2 private signing keys, with passkey[2] management services (PMS) synchronizing them between PMS's cloud storage and users' authenticators. Examples of PMS Today include, Apple's iCloud Keychain [14], Google Password Manager [12], Microsoft's Password Monitor [11], 1Password [5], LastPass [4], and DashLane [3]. These services are supported by all major browsers, and used by millions of users [65].

The tension between security vs. recoverability. Due to their centralized nature in storing users' private signing keys, passkeys have inadvertently become a lucrative target for attackers. Many PMS now use another cryptographic key to protect a user's passkeys backup at rest, for example, by encrypting it with another key. This encryption key needs to be consistently available for users; otherwise, users will not be able to decrypt the encrypted passkeys after retrieving them from passkey providers. In other words, if not properly addressed, managing such a cryptographic key, which has long been a daunting task for users [31], inevitably reintroduces the account recovery problem that synced passkeys attempt to solve in the first place.

For this reason, some PMS only require users to maintain a user-chosen secret (e.g., passwords or PINs in most cases), which is presumably more memorable, recallable, and hence recoverable than a high-entropy cryptographic key. In practice, this secret can be used in two flavors of strategies, i.e., key derivation based and key escrow based, with each emphasizing different priorities on the security and recoverability of the encryption key. However, both strategies rely on user-chosen secrets that are subject to guessing attacks (e.g., offline cracking), making them insufficient for protecting passkeys. If PMS is compromised and the user-chosen secrets used to derive or to retrieve the decryption key are guessed, attackers can access users' passkeys in plaintext. Please refer to Appendix A for a more detailed introduction about these two strategies and their limitations.

Need for passkey compromise detection. The foregoing discussion echoes the challenge we are facing today to secure users' passkeys against breaches, revolving around many moving parts. These include, but are not limited to, the security of users' PMS accounts, the resistance of user secrets (for key derivation or escrow) to offline cracking, the correctness of PMS implementations, and PMS's defenses against both remote and insider attackers. The failure of any of these components could expose all of a user's passkeys, leading to users' mistrust in the provider [59] and, more critically, large-scale account takeovers across numerous RPs, which highlights the urgent need for an effective framework to detect passkey breaches from PMS storage, which we provide in this paper.

[2]For conciseness, in this paper, we use "passkey" to refer specifically to a "synced passkey" as opposed to a "device-bound passkey" — a FIDO private key that never leaves the authenticator.

2.2 Related work

Decoy-based credential breach detection. Existing decoy-based detection systems for user-chosen passwords [30,50,75] allow websites to detect the breaches of *their own password storage*. The idea is for the websites to plant a fixed number of decoy passwords, known as *honeywords*, in their password storage alongside the user-chosen password. If an attacker breaches the website's password storage and attempts to exploit the leaked (hashed) passwords to access the user's account at the website, entering a honeyword will trigger a password breach detection.

Traditional honeyword systems [13, 30, 50] rely on the information asymmetry between the defender (e.g., a website or RP) and the attacker — they assume that the attacker cannot access the secrets stored by the website to distinguish between honeywords and the user-chosen passwords. This information asymmetry inevitably requires additional assumptions about what the website knows that the attacker does not in order to enable effective detection. For example, the information asymmetry may arise from a *honeychecker*, which stores an identifier of the user-chosen password that is known only to the website [50]. Alternatively, it may originate from a pseudorandom number generator only accessible by the website [30], or from the assumption that the attacker is unaware of the deployment of a breach detection system facilitated by machine-dependent password hashing [13].

Amnesia [75] is the first *symmetric* design that allows attackers to learn the entire persistent state of a website and enables detection based on probability distribution changes of some password "markings" that can help distinguish between honeywords and the user-chosen one.

Existing honeyword systems mentioned above aim to detect unauthorized logins at a website or an RP — the same party where the password breach occurs. In comparison, CASPER enables the detection of unauthorized login attempts that abuse breached passkeys at an RP when passkey breaches occur at a different party — a PMS provider. For this reason, existing honeyword systems fall short of addressing the problem CASPER aims to solve. To be practical, however, CASPER adopts a symmetric security assumption similar to that of Amnesia and avoids relying on additional security assumptions required by other traditional designs.

Credential keys / password backup storage leakage. Existing work on *preventing* unauthorized access to users' accounts due to breaches of credential backup services can be roughly

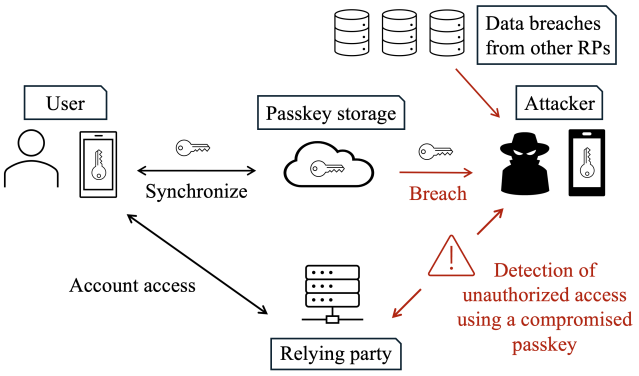


Figure 1: Overview of CASPER and threat model: CASPER allows a relying party to detect the abuse of a compromised passkey for unauthorized account access when the attacker obtains the passkey from a breached passkey storage of a passkey management service (PMS) provider and also has access to a certain number of data breaches of other RPs.

categorized into two camps.

The first line of work [24, 67] aims to prevent the leakage of users' cryptographic keys by distributing them securely across multiple parties on the cloud. In the context of synced passkeys, one might consider distributing users' passkey storage across multiple PMS providers to mitigate the risk of breaches affecting one or a subset of them. However, this approach would pose a significant deployment burden, as it is against the centralized nature of existing major PMS providers, e.g., Apple, Google, and Microsoft, which manage users' passkeys within their own ecosystems and is incompatible with their current implementations for credential synchronization and storage.

The second line of work [21, 25, 26] proposes *honey password vaults* to make attackers difficult to perform efficient offline-cracking on leaked *password vaults / managers*[3]. However, the insufficient rate-limiting defense of most websites against online guessing [37, 60] and vulnerabilities in current honey vault designs against sophisticated attacks [26, 32, 41, 69] have cast doubt on the overall efficacy of these proposals in recent years.

In comparison, CASPER is a *detection* framework to enhance users' account security by enabling RPs to detect the abuse of passkeys leaked from PMS. Notably, CASPER is deployable and user-friendly, while remaining effective even against highly sophisticated attackers.

3 Threat Model

In this section, we detail the participating parties and our threat model.

Participating parties. Following the convention of FIDO2, we refer to the device that generates the private signing keys as the *authenticator*, and the websites as *relying parties* (RP). We assume the authenticator supports synced passkeys, meaning that it has the capability to synchronize the private signing keys with a passkey management service (PMS) *provider* who stores users' passkeys on their cloud storage. The authenticator itself could be dedicated hardware that is platform-independent (e.g., YubiKeys) or a virtual one that is platform-dependent (e.g., phones or laptops). The users authenticate themselves locally to the authenticator (e.g., via PINs, biometrics) and allow a *client* (e.g., browsers, mobile apps, etc.) to complete account registration/authentication at a RP using passkeys.

Introducing `caat`. We consider a credential backup abuse attacker (`caat`) who has access to the data (and associated metadata, if any) stored at the PMS provider. The goal of `caat` is to undetectedly take over user accounts at RPs where CASPER is deployed. We follow the threat model of FIDO2 authentication [2] and assume that the user and her devices (e.g., the authenticator and client involved) are trusted. We also assume that RPs are trusted for honestly detecting attacks by `caat` and have employed other state-of-the-art defenses such as protection against denial-of-service attempts during account registration and login. All communication channels are assumed to be secure and authenticated using a standard secure communication protocol (e.g., SSL/TLS [29]).

Making `caat` realistically stronger. Given the increasing number of data breaches that websites are experiencing today, we also consider the case where `caat` may have access to data breaches (e.g., obtained from black market forums [71]) from breached websites where the target user has accounts. We follow, to the best of our knowledge, the strongest threat model [75] in the literature on credential breach detection (as discussed in Section 2.2) where `caat` is given *read access* to the *entire* persistent storage of a certain number of compromised websites, including all data used for account login or registration and breach detection by CASPER. However, like prior work on breach detection [13, 30, 50, 75], we do not allow `caat` to actively compromise those breached websites, and we assume the transient information that arrives in a login attempt is not stored by a breached website and not available to `caat`. Note that the primary focus of this paper is to detect breaches of PMS storage, and we do not aim to enable breached websites to detect PMS breaches, as we view those data breaches as *static* information that can be leveraged by `caat`. However, our framework can be extended to enable even passively breached RPs to effectively detect PMS breaches by incorporating the probabilistic detection method proposed in [75].

To summarize, as shown in Figure 1, we assume that `caat` has access to users' passkey backups at PMS, as well as to the persistent storage of a certain number of passively breached websites. As we will show in Section 6, CASPER can effectively detect a sophisticated attacker like `caat` when it adopts

[3] Password vaults / managers are cloud backed up synced passwords of the user — encrypted under a single master password.

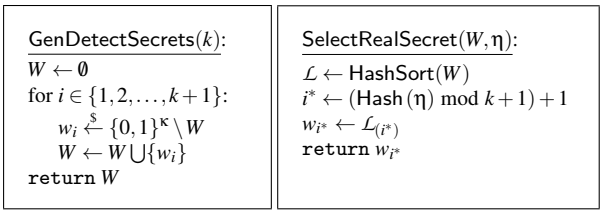


Figure 2: An instantiation example of GenDetectSecrets and SelectRealSecret. Here HashSort sorts all elements in W by their hash values seeded by η, i.e., $\mathsf{Hash}(w_i \| \eta)$ for all $w_i \in W$, and then outputs these elements as an ordered list. Hash is a collision-resistant hash function. Here i^* indicates the index of the *real* detection secret.

its optimal strategy to take over user accounts at *unbreached* RPs and to evade detection by CASPER.

4 Detection Secrets

CASPER relies on a set of detection secrets $W = \{w_1, w_2, \ldots, w_{k+1}\}$ where one of these secrets is the real detection secret and its index is denoted by $i^* \in \{1,2,\ldots,k+1\}$ throughout the paper. Following this notation, the *real* detection secret is denoted by $w_{i^*} \in W$, while the remaining k secrets in $W \setminus \{w_{i^*}\}$ are referred to as *decoy secrets*.

Survivable user secret η. We use η to represent the information needed to identify w_{i^*} from a given W. This η should not be shared with any PMS providers or RPs, and we assume η to be *survivable*; that is, we assume that η is always accessible by the user (even in the case of device loss or failure). For example, η could be instantiated in practice by secrets that can be easily recalled by the user (e.g., PINs) or retrievable from physical objects or trusted parties (e.g., credit card verification codes, bank account numbers), or biometrics. The options of instantiating η with a proper user secret will be discussed further in Appendix E.

Decoy generation. How to generate good decoy credentials [23, 33, 43, 74, 76] has been more of an orthogonal line of research to our work. In this paper, we consider a pair of procedures $\mathcal{G} = \langle \mathsf{GenDetectSecrets}, \mathsf{SelectRealSecret} \rangle$ where GenDetectSecrets is a *randomized* procedure for generating decoy secret candidates and SelectRealSecret a *deterministic* detection secret selection procedure. The procedure GenDetectSecrets produces a set W of size $k+1$ when given an integer k. SelectRealSecret takes W and a user's survivable secret η as input, and outputs $w_{i^*} \in W$ so that w_{i^*} is the real detection secret and other k are decoys. A simple instantiation example of GenDetectSecrets and SelectRealSecret is shown in Figure 2. As such, when the passkey backups are additionally encrypted under an encryption key derived from w_{i^*}, the attacker who attempts to recover users' passkeys from stolen passkey backups, without knowing η, can only make guesses on which in W is w_{i^*}. Such guesses may be undesired in other authentication scenarios, but our framework, as we will show in later sections, leverages such guesses to detect the compromise of users' passkey storage on the cloud.

Symbol	Description
w / W	detection secret / the set of detection secrets
v / V	passkey verifier / the set of auth. verifiers
V'	the set of *active* decoy passkey verifiers $V' \subseteq V$
s / $\tilde{\mathsf{s}}$	passkey / encrypted passkey
u	the recovery key used to encrypt s
w_{i^*}/s_{i^*}/v_{i^*}	the *real* detection secret / passkey / passkey verifier
z	nonce used to encrypt s
k	# of decoys, $\lvert W\rvert = \lvert V\rvert = k+1$
α	fraction of *active* decoy passkey verifiers $\alpha = \frac{\lvert V'\rvert}{\lvert V\rvert - 1}$
n	# of breaches caat has observed
m	# of relying parties (RPs) caat wants to login
sid	unique identifier of the RP
aid / uid	unique user account identifier at PMS / RP

Figure 3: Notations used in this paper.

5 CASPER: A New Detection Framework

To enable a relying party (RP) to detect the abuse of users' passkey leaked from passkey management services (PMS) provider, CASPER introduces a new passkey backup and restoration (BnR) protocol (Section 5.2) and a compromise detection (CD) algorithm executed by the RP (Section 5.3). For brevity, we refer to the attacker who has compromised users' passkey storage at breached PMS provider as caat. Section 5.1 provides a high-level overview of CASPER and its design considerations. Figure 3 presents the key notations used throughout this paper.

5.1 Design considerations and overview

Design considerations. Consider that a user (say Alice) uses a PMS provider to backup and synchronize her passkeys for RP account logins. As usual, Alice is responsible for certain operations already required by PMS provider (e.g., account creation with PMS using a master password and new authenticator setup using passcode) for passkey synchronization or access across her existing devices / authenticators. CASPER is designed to effectively detect malicious login attempts by caat using Alice's passkeys leaked from a PMS provider, while meeting the design requirements as detailed below.

❶ *Easy to deploy.* CASPER does not require PMS providers to modify their protocol designs or implementations for passkey synchronization and storage, and it requires only minimal changes on the RP side. Also, CASPER is compatible with existing two-/multi-factor authentication or risk-based authentication schemes and does not require any additional (trusted) hardware for PMS providers or RPs. Please

see Section 8.3 for further discussion of the deployment considerations for CASPER.

❷ *User-friendly.* To avoid introducing usability challenges, such as the account recovery problem caused by requiring Alice to manage additional high-entropy keys, CASPER is designed to achieve high detection accuracy while allowing Alice to manage only a low-entropy secret that is easy to remember (e.g., a 2-digit numeric PIN) or retrieve (e.g., the last few digits of Alice's bank account number). Note that, after the initial PMS and device setup, CASPER operates transparently in Alice's view — she can use online services provided by RPs and PMS providers without any additional actions required by CASPER during account login or passkey backup and synchronization. Furthermore, to avoid introducing new usability challenges, CASPER does not rely on users to manage multiple devices for new device setup. We will discuss CASPER's usability in more detail in Section 8.1, including ways to further enhance its user experience.

❸ *No security degradation.* CASPER does not degrade users' account security in any way. In other words, Alice's accounts must be at least as secure as they would be without CASPER deployed. This also implies that, when no PMS passkey breach has occurred, a denial-of-service attacker must not be able to disrupt the availability of RPs' services by abusing CASPER to cause false breach detection.

CASPER overview. Behind the scenes, given Alice's η, CASPER produces a set of detection secrets $W = \{w_1, w_2, \ldots, w_{k+1}\}$ with one of them being the real detection secret represented by w_{i^*}. Then CASPER encrypts Alice's passkey, denoted by s, with a key derived from w_{i^*}. If the encrypted passkey is denoted by $\tilde{\mathsf{s}}$, Alice backs up to provider the detection secret set W together with $\tilde{\mathsf{s}}$ *instead of* s. In other words, PMS backs up $\tilde{\mathsf{s}}$ as if it were backing up s, requiring no changes to its implementation for passkey synchronization, storage, and end-to-end encryption (if enabled). If a `caat` has access to $(W, \tilde{\mathsf{s}})$ leaked from PMS, they can attempt to decrypt $\tilde{\mathsf{s}}$ and recover Alice's passkey by guessing which among W is w_{i^*}. In this case, the attacker may guess a wrong detection secret, derive a wrong key to decrypt $\tilde{\mathsf{s}}$, and subsequently obtain a wrong passkey for unauthorized account login attempts. With detection information registered at the RP beforehand, the RP would be able to verify the authentication response with the corresponding decoy verification key, which indicates a potential compromise of users' encrypted passkeys $\tilde{\mathsf{s}}$ leaked from PMS.

In this paper, we introduce CASPER in the context of detecting FIDO2 passkeys leaked from PMS. However, the concept of CASPER can also be applied to detect breaches of PMS that support other system-generated authentication credentials, such as randomly generated passwords and long-term seeds for HMAC / time-based one-time passwords (HOTP / TOTP) widely used for 2FA today. For interested readers, we will explain in Appendix F how CASPER can be extended for OTP as another example of CASPER's application.

5.2 Backup & restoration protocol

As shown in Figure 4, the passkey backup and restoration (BnR) protocol includes the following four steps.

Step 1. Provider account setup. This initialization setup is invoked only once when the user signs up for a new passkey management service (PMS) account. In this step, alongside assigning a unique provider account identifier denoted by aid, the authenticator generates a detection secret set W of size $k+1$ by calling the $\mathsf{GenDetectSecrets}$ procedure described earlier in Section 4. The authenticator then synchronizes (aid, W) to the PMS provider for backup and saves a local copy optionally.

Step 2. Authenticator setup. Upon successful authentication, PMS typically requires an existing PMS user to set up a new passkey authenticator to enable it for passkey synchronization via PMS. For CASPER, in addition to the usual authenticator setup processes specified by PMS, the authenticator also retrieves (aid, W) from PMS and prompts the user to provide η. The authenticator executes $\mathsf{SelectRealSecret}(W, \eta)$, recovers the real detection secret w_{i^*}, and saves $(\mathsf{aid}, w_{i^*}, W)$. Then the user can start using this authenticator to register new RP accounts (see Step 3) or to log into RP accounts with passkeys restored from PMS passkey backups (see Step 4). Crucially for the security, w_{i^*} is kept private locally from any other parties including the PMS provider or RPs. The user secret η is never stored at any participating parties.

Step 3. Passkey registration. When the user registers a new account at a RP identified by sid, the authenticator first runs KeyGen to generate a passkey pair $(\mathsf{s}_{i^*}, \mathsf{v}_{i^*})$ including the passkey s_{i^*} and its corresponding public verification key v_{i^*}. We can identify this newly registered account by uid and implicitly follow the implementation of KeyGen as defined by FIDO2 standard signature schemes (e.g., ECDSA [48]).

Next the authenticator calls two procedures Π_{EncCred} and $\Pi_{\mathsf{GenVerifierSet}}$ (as shown on the right side of Figure 4). Π_{EncCred} returns $\tilde{\mathsf{s}}$ — an encrypted version of the user's passkey s_{i^*} for later passkey synchronization and backup via PMS. Given s_{i^*}, $\Pi_{\mathsf{GenVerifierSet}}$ returns a set of verification keys V, which will be provided to the RP for the compromise detection (CD) algorithm we will detail in Section 5.3.

Under the hood to encrypt the s_{i^*} as $\tilde{\mathsf{s}}$, the Π_{EncCred} procedure first samples a nonce uniformly at random z and then invokes a key derivation function (KDF) with w_{i^*} and z as its input. KDF returns a *recovery key* u_{i^*} which serves as the encryption key to encrypt s_{i^*} in a "one-time pad" manner. Considering what lies ahead for the CD protocol, however, CASPER needs to ensure that the attacker gets an incorrect but *well-formed* valid passkey s when running Π_{DecCred} with a decoy detection secret $w \in W \setminus \{w_{i^*}\}$. Thus the authentica-

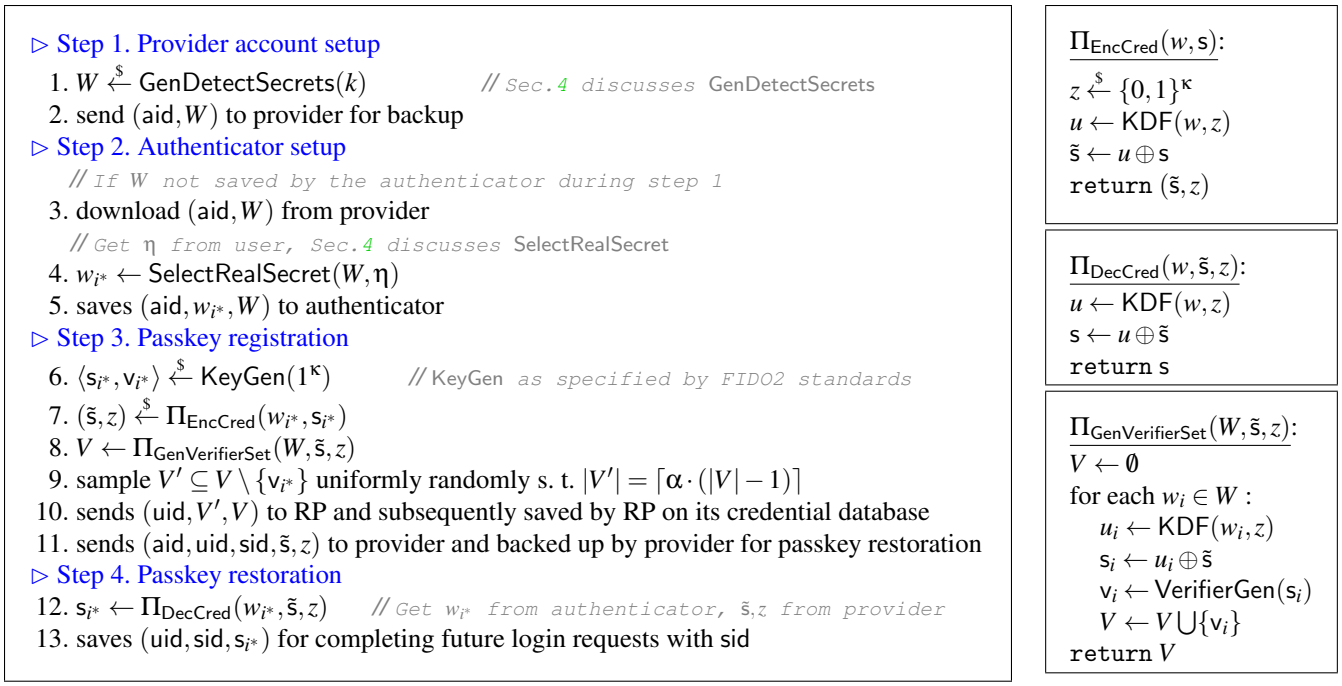


Figure 4: (Left) Passkey Backup and Restoration (BnR) protocol as described in Section 5.2. (Right) Building blocks introduced by CASPER and used by the BnR protocol. Notations used are explained briefly in Figure 3.

tor should perform additional passkey validity tests. These tests include for all $w \in W \setminus \{w_{i^*}\}$, running $\Pi_{\mathsf{DecCred}}(w, \tilde{\mathsf{s}}, z)$ to get s and checking if s is a well-formed passkey. Otherwise, the authenticator should repeat running KeyGen and Π_{EncCred} until all resulting s are well-formed passkeys. This process is efficient because the probability of yielding a well-formed (decoy) private key when $\tilde{\mathsf{s}}$ is decrypted with an incorrect w is overwhelmingly high (see Appendix D for more details).

The $\Pi_{\mathsf{GenVerifierSet}}$ procedure takes $(W, \tilde{\mathsf{s}}, z)$ as input and generates $k+1$ verification keys with each corresponding to a detection secret $w_i \in W$. To do this, for each $w_i \in W$, the authenticator invokes $\mathsf{KDF}(w_i, z)$ to get a recovery key u_i and uses it to decrypt $\tilde{\mathsf{s}}$ by running $\mathsf{s}_i \leftarrow u_i \oplus \tilde{\mathsf{s}}$. Then the authenticator assembles the set of verification keys $V \leftarrow \bigcup_{i=1}^{k+1} \{\mathsf{v}_i\}$ by deriving v_i from s_i. Afterward, the authenticator marks a certain fraction (denoted by α) of $V \setminus \{\mathsf{v}_{i^*}\}$ uniformly at random as a new subset decoy verification keys, denoted by V', and thus $\alpha = |V'|/(|V| - 1)$. Both (V', V) are sent to RP to complete passkey registration[4], and these two sets are used by RP to run the compromise detection algorithm in Section 5.3.

Looking ahead, we refer to V' as the set of *active* decoy verification keys for the RP because, as we will show later in Figure 5, an authentication response successfully verified by any of these *active* decoy verification keys will trigger a detection alert. In contrast, other decoy verification keys in $V \setminus (\{\mathsf{v}_{i^*}\} \cup V')$ are *inactive* and, together with the real verification key, v_{i^*}, will be viewed as valid verification keys since successful verification by these verification keys will lead to successful logins.

Finally, after receiving the successful registration confirmation from RP, the authenticator synchronizes $(\mathsf{aid}, \mathsf{uid}, \mathsf{sid}, \tilde{\mathsf{s}}, z)$ to PMS. Now PMS backs up the $\tilde{\mathsf{s}}$ alongside the corresponding $(\mathsf{uid}, \mathsf{sid}, z)$ for the user's PMS account identified by aid. Note that PMS now manages $\tilde{\mathsf{s}}$ instead of s_{i^*}. As a result, after a PMS breach, a `caat` cannot accurately derive s_{i^*} from $\tilde{\mathsf{s}}$ without η.

Step 4. Passkey restoration. As usual, PMS allows the user to synchronize their passkeys back to a registered authenticator. The registered authenticator first retrieves from PMS a copy of the encrypted passkey $\tilde{\mathsf{s}}$ alongside $(\mathsf{uid}, \mathsf{sid}, z)$. It then executes the procedure $\Pi_{\mathsf{DecCred}}(w_{i^*}, \tilde{\mathsf{s}}, z)$ to retrieve s from $\tilde{\mathsf{s}}$. This procedure, as shown on the right side of Figure 4, decrypts $\tilde{\mathsf{s}}$ with the recovery key u_{i^*} to obtain s_{i^*}. Finally, the user's authenticator saves $(\mathsf{uid}, \mathsf{sid}, \mathsf{s}_{i^*})$ so the user can use the restored passkey s_{i^*} to log into their account with the identifier uid at RP identified by sid. For correctness, given a $(\tilde{\mathsf{s}}, z)$ pair, observe that $\exists i \in [k+1]$ s.t. $w_i = w_{i^*}$, and thus $u_i = u_{i^*}$. Then we have $\mathsf{s}_i = \mathsf{s}_{i^*}$ since $\mathsf{s}_i = u_i \oplus \tilde{\mathsf{s}} = u_{i^*} \oplus \mathsf{s}_{i^*} \oplus u_{i^*} = \mathsf{s}_{i^*}$.

An alternative design. Instead of backing up a single encrypted passkey $\tilde{\mathsf{s}}$ in Step 3, CASPER could alternatively derive and back up all $k+1$ passkeys, i.e., $\mathsf{s}_1, \ldots, \mathsf{s}_{k+1}$, to PMS. This alternative design could provide the same level of de-

[4]Here, the authenticator sends V' together with V for better clarity. In practice, instead of the entire V', it suffices for the authenticator to send the RP only a set of identifiers or indices indicating which verification keys among V are in V', which can slightly lower the communication and storage costs.

- For a login request for an account uid, the RP checks if uid exists and then returns a challenge to the user.
- Let v, rsp, and γ denote the public verification key, the response produced by the user's authenticator, and the signature generated with the user's passkey, respectively. Upon receiving $(\mathsf{v}, \mathsf{rsp}, \gamma)$ from the user, the RP performs the following tests and actions:
  - retrieves (V, V') corresponding to uid
  - `if` $\mathsf{Verify}(\mathsf{v}, \mathsf{rsp}, \gamma) = \mathit{false}$ OR $\mathsf{v} \notin V$, RP rejects this login request.
  - `else if` $\mathsf{v} \in V'$, RP raises a detection alarm.
  - `else` RP accepts this login request.

Figure 5: The Compromise Detection (CD) algorithm of CASPER used by the RP to detect passkey compromise.

tection effectiveness but would introduce two deployment limitations: (i) PMS storage and communication overhead would increase from $O(1)$ to $O(k)$ for each user account; (ii) it would become incompatible with existing PMS designs as existing PMS synchronization and storage implementations are designed to handle one passkey for one credential registration at RPs. We therefore take a different approach by drawing inspiration from honey encryption [49]: instead of directly backing up all possible decoy plaintexts (i.e., passkeys s_i), our design ensures that decrypting the backed up ciphertext (e.g., the encrypted passkey $\tilde{\mathsf{s}}$) with incorrect decryption keys yields those decoys. This approach optimizes PMS compatibility while minimizing PMS communication and storage overhead.

5.3 Compromise detection algorithm

The passkey compromise detection (CD) algorithm (Figure 5) is run by the relying party (RP) upon each login request it receives. It allows the RP to leverage the set of *active* decoy verification keys to detect the `caat`'s account login attempts with corresponding decoy passkeys (see Step 3 in the BnR protocol for producing active decoy verification keys). Next, we describe in detail how the detection is performed for FIDO2.

In FIDO2 (as shown in Figure 5), when the RP receives a login request with an account identifier uid, it produces a random challenge and sends it together with other information specified by the WebAuthn protocol [73]. Upon receiving the challenge, the authenticator produces an authentication response rsp and generates a signature γ using the private signing key s by running $\gamma \leftarrow \mathsf{Sign}(\mathsf{s}, \mathsf{rsp})$ on the authenticator[5]. Finally, the user device sends $(\mathsf{v}, \mathsf{rsp}, \gamma)$ as a response back to the RP for login.

Once the RP receives the response $(\mathsf{v}, \mathsf{rsp}, \gamma)$, it will perform a set membership test for $\mathsf{v} \stackrel{?}{\in} V$ and also run $\mathsf{Verify}(\mathsf{v}, \mathsf{rsp}, \gamma)$ to check if the received γ is a valid signature of rsp under the verification key v. If either of these two tests outputs *false*, this login attempt fails. If both tests output *true*, the RP can further check whether v is in the active verification key set V' to determine whether the current login attempt triggers a passkey breach detection or results in a successful login.

[5]The definitions and implementations of Verify and Sign adhere to the FIDO2 standards, which we omit here for simplicity.

6 Detection Effectiveness

In this section, we show the detection effectiveness of CASPER. First, we show that CASPER does not make it any easier for a `caat` to distinguish the real detection secret from the decoys (Section 6.1). We then estimate true detection probabilities and the overall security benefits CASPER provides through probabilistic model checking (Section 6.2), followed by a false detection analysis (Section 6.3).

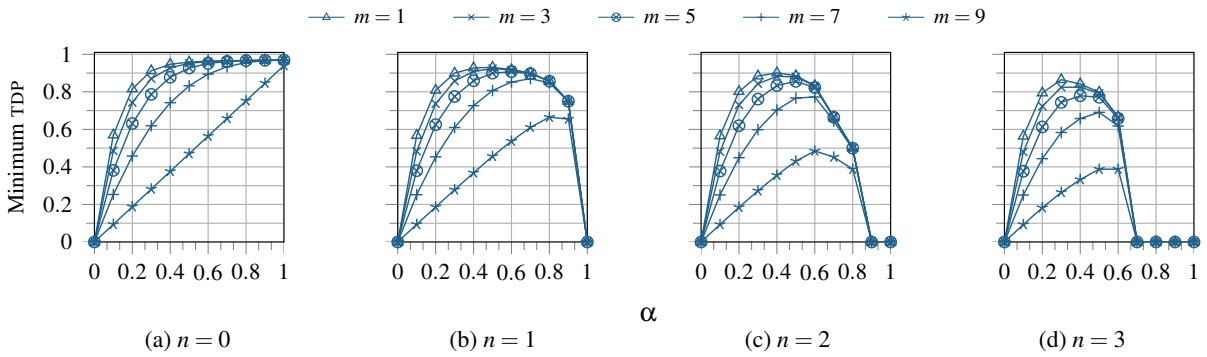
6.1 Flatness preservation

Following the informal definition of *flatness* by related work on decoy passwords [33, Sec. 2.1], [43, Sec. 3.1], here we informally define *flatness* as the probability that the `caat` outputs the real detection secret w_{i^*} as its guess *on the first try* given the set of detection secrets W and denote it by $\mathsf{flt}(W, w_{i^*})$. We also say W to be perfectly flat if $\mathsf{flt}(W, w_{i^*}) = \frac{1}{|W|}$ (or equivalently, $\frac{1}{k+1}$). Recall that in Section 4, we specified that CASPER relies on $\mathcal{G} = \langle \mathsf{GenDetectSecrets}, \mathsf{SelectRealSecret} \rangle$ that given (η, k) outputs w_{i^*} and W of size $k+1$. We show that CASPER provides a critical security property, *flatness preservation*. We formally define and prove this property in Appendix C.

The flatness preservation property of CASPER ensures that given multiple $(\tilde{\mathsf{s}}, z)$ pairs from the PMS credential backup compromise for the same user, the `caat` does no better in identifying the real detection secrets, w_{i^*}, from W with the knowledge of multiple $(\tilde{\mathsf{s}}, z)$ pairs than without. Briefly, to show this, as further explained in Appendix C, we consider a simulator without multiple $(\tilde{\mathsf{s}}, z)$ pairs but can by itself generate multiple *simulated* $(\tilde{\mathsf{s}}', z')$ pairs by choosing all $\tilde{\mathsf{s}}'$ and z' uniformly at random from $\{0,1\}^{\kappa}$. Since simulated $(\tilde{\mathsf{s}}', z')$ pairs are indistinguishable from $(\tilde{\mathsf{s}}, z)$, there is no useful information in $(\tilde{\mathsf{s}}, z)$ about which among W is w_{i^*} for the `caat` to improve its guessing on w_{i^*}.

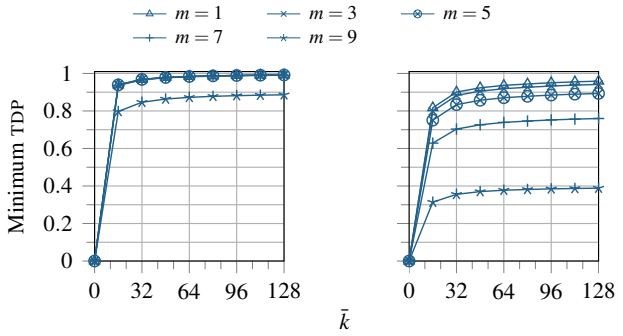
6.2 True detection and efficacy of CASPER

We provide a comprehensive perspective by estimating both true detection probabilities (TDP) and the overall security efficacy of CASPER. The former provides a simple yet clear picture of how likely CASPER can detect the attack by the `caat` *if the attack occurs*. However, TDP alone falls short of capturing the security efficacy provided by CASPER to users' account security completely in a more realistic setting where the `caat`'s attack strategy has already been influenced by the deployment of CASPER; it may decide to attack later or stop



(a) $n = 0$ (b) $n = 1$ (c) $n = 2$ (d) $n = 3$

Figure 6: Minimum expected true detection probabilities as a function of α with varying m and n, where $\bar{k} = 32$. Here α is the fraction of *active* decoy verifiers V' present in V (i.e. $\alpha \leftarrow \frac{|V'|}{|V|-1}$), n is the number of breached websites `caat` observes, m is the number of websites `caat` wants to compromise.



(a) $n = 0, \alpha = 0.9$ (b) $n = 2, \alpha = 0.4$

Figure 7: Minimum true detection probabilities as a function of $\bar{k}$ with varying m, n, and α.

attacking early. For example, to avoid detection by CASPER, a cautious `caat` may postpone the attacks until a later time to obtain more information about the user. However, PMS may have already discovered its credential backup compromise and asked the user to reset their credentials prior to the `caat` starting such a deliberately delayed attack. In this case, the deployment of CASPER would already add efficacy to the user's account security. To capture this nuanced scenario besides TDP, we measure the efficacy of CASPER by measuring how comparatively well CASPER can reduce the `caat`'s ability to consistently take over user accounts unnoticeably with compromised credentials leaked from PMS.

Modeling detection secrets. Recall that η is the survivable user secret that determines which among W is w_{i^*}. The level of flatness provided by (W, w_{i^*}) and measured by $\mathsf{flt}(W, w_{i^*})$ is in fact difficult to estimate since we do not know (1) the probability distribution of η or (2) how well the `caat`'s strategy together with its knowledge about η can improve its guessing results. Therefore, to have a better generality for our true detection and efficacy analysis, we follow the similar formal treatments of password guessing from prior work (e.g., [51]). Specifically, instead of providing the `caat` with the original detection secret set output by $\mathcal{G}$ with flatness $\mathsf{flt}(W, w_{i^*})$, we give an *equivalent* detection secret set $\bar{W}$ of size $\bar{k}$ that is *perfectly flat* and provides approximately the same flatness as $\mathsf{flt}(W, w_{i^*})$. This implies the `caat` can do no better in guessing the real detection secret in $\bar{W}$ with probability $\frac{1}{\bar{k}+1}$, where $\bar{k} = \lfloor \frac{1}{\mathsf{flt}(W, w_{i^*})} - 1 \rfloor$, i.e., the largest integer $\bar{k}$ such that $\frac{1}{\bar{k}+1} \leq \mathsf{flt}(W, w_{i^*})$.

This abstraction helps us to model the `caat`'s ability to guess the real detection secret in our analysis as a function of $\bar{k}$, avoiding additional (and possibly inaccurate) assumptions on the `caat`'s knowledge about η and its strategies, as well as making our analysis applicable for different types of probability distributions and flatness that η as instantiated in practice may have. To see this consider the case when η that is generated uniformly at random, e.g., a random (numerical) PIN with x digits where $10^x > k+1$, then (W, w_{i^*}), produced by $\mathcal{G}$ given k and η here, provides perfect flatness, i.e., $\mathsf{flt}(W, w_{i^*}) = \frac{1}{k+1}$ and so $\bar{k} = k$. However, when η is a *user-chosen* password or PIN, the min-entropy [22] of these secrets must be at least $\log_2(\bar{k}+1)$ bits for a given $\bar{k}$. For example, for $\bar{k} = 32$, if instantiated with a random (numerical) PIN, η should include at least 2 digits such that $10^2 > 32+1$; whereas if η is a user-chosen PIN, the min-entropy of η should be at least $\log_2(32+1) \approx 5$ bits.

Following Section 3, we allow the `caat` to learn from PMS the detection secret set W sent to PMS during PMS account setup together with $(\tilde{\mathsf{s}}, z)$ pairs sent during the RP account registration step by the authenticator. Additionally, the `caat` observes breaches from n other already *compromised* RPs. These breaches include both the verification key set V and the *active* decoy verification key set V' from each n compromised RPs. Considering that, given a $(\tilde{\mathsf{s}}, z)$ pair, for each $\mathsf{v} \in V$, there exists a $w \in W$ such that v can verify the authentication response produced by its corresponding s, which is recoverable by $\Pi_{\mathsf{DecCred}}(w, \tilde{\mathsf{s}}, z)$, the `caat` can rule out exactly $|V'|$ decoy detection secrets in W by observing V' from the first breach snapshot and rule out $\leq |V'|$ for each of the rest of the $n-1$

breach snapshots due to that overlapping may exist. Here we use $\bar{W}^{(n)}$ to denote the subset of detection secrets corresponding to verification keys that always fall into $V \setminus V'$ across these n breaches.

The goal of the caat is to compromise the user's accounts at a specified number of *unbreached* RP (denoted by m). Note that the probability of the caat triggering a true alarm when compromising a user's account at an *unbreached* RP is the probability of the event that the caat picks a detection secret from $\bar{W}^{(n)}$, uses it together with $\tilde{s}$ to derive a (decoy) passkey s, and produces an authentication response verifiable by an *active* decoy verification key $v \in V'$ at the unbreached RP (see Figure 5).

True detection probability. To capture the caat's best strategy for minimizing the true detection probability (TDP) while achieving its goal, we model the caat as a Markov decision process (MDP). Conceptually an MDP consists of a set of states and potential transitions between them. When the MDP is in a specific state, it can select one of several available actions, which leads to a specific probability distribution over the possible next states. The MDP attacker enters the final state when it triggers a breach alarm raised by CASPER while attempting to compromise the users' accounts at least one of the m *unbreached* RPs. In our experiments, without loss of generality, we consider that the caat is targeting *one user* with a web account at each of the m *unbreached* and n *breached* RP with CASPER deployed. Specifically, using the PRISM model checker [56], we construct an MDP to model a caat who does the following:

- The caat randomly selects a detection secret w from $\bar{W}^{(n)}$, derives a passkey through $\Pi_{\mathsf{DecCred}}(w, \tilde{s}, z)$ and attempts to authenticate at the first RP.
- If the selected detection secret w corresponds to a verification key $v \in V'$ at the current RP, the caat enters the "detection" state and the experiment is over. Otherwise, this RP is considered compromised without detection. If $m > 1$, the caat can choose to either 1) reuse the same detection secret used in the last attack or 2) randomly select a *different* detection secret w', i.e., $w' \xleftarrow{\$} \bar{W}^{(n)} \setminus \{w\}$, and then attack the next RP.
- The caat repeats the above attacks until it either 1) moves to the final "detection" state and triggers an alarm or 2) completes attacking and compromising all m RPs without detection.

The experiment outputs the minimum probability of the best-strategy caat entering the final state over all possible action paths, as reported in Figure 6 and Figure 7.

Results of TDP experiments. Figure 6a-6d correspond to four different $n \in \{0,1,2,3\}$ respectively for a fixed $\bar{k} = 32$. When $n = 0$ (Figure 6a), a larger α results in a higher probability of the caat triggering a detection alarm because it is more likely that the verification key corresponding to the selected detection secret falls into V'. Also, attacking more RPs exposes the caat to greater risks in triggering a detection alarm *more than one RPs*. However, as n is increased to 1, 2, or 3 as shown in Figure 6b, Figure 6c, and Figure 6d respectively, the detection probabilities for a relatively high α, e.g., 0.9, drop significantly but remain almost unaffected for a relatively low α, 0.1. This is because a higher α leads to a smaller $\left|\bar{W}^{(n)}\right|$ given a fixed n, and, as a result, increases the possibility of the caat selecting the real detection secret at the very first attack.

Figure 7 shows the minimum true detection probabilities as a function of $\bar{k}$ for a given fixed α and the number of RPs n where the caat has observed breaches. One interesting observation here is that when $\bar{k}$ reaches a certain level, e.g., 32, increasing $\bar{k}$ further can hardly give a significant boost in the minimum probability. We believe this observation provides an informative insight for determining a proper $\bar{k}$ that provides a good balance between CASPER's detection accuracy and performance—additional detection power provided by an unnecessarily larger $\bar{k}$ would be limited, but it would negatively affect the overall performance and storage costs of CASPER (see Section 7).

Estimating security efficacy. We implement another model-checking experiment to investigate the efficacy of CASPER when the caat decides to wait for τ time intervals before starting the attack or stop attacking early after successfully taking over m' ($\leq m$) accounts to minimize detection probability. Specifically, once the caat have access to a user's compromised credential backups from PMS, the caat decides whether to attack immediately (i.e., $\tau = 0$) risking detection with a higher probability (see Figure 6, particularly Figure 6a) or wait τ time intervals, hoping to observe more RP breaches to lower the detection probability. In that latter case, however, the caat has to accept the risk that, over time, the PMS provider will discover the compromise of the user's credential backup itself and notify the user. We also consider that the caat may choose to stop the attack early after successfully taking over m' ($\leq m$) accounts if the caat "believes" compromising accounts at more RPs will lead to a high detection probability.

To understand CASPER's detection efficacy under this realistic scenario, we allow the caat to have access to snapshots of a certain number of compromised RPs' persistent storage, which are breached at a Poisson arrival rate with mean λ per time interval. We use a random variable T_{PMS} to represent the time interval soon after which the PMS (provider) identifies by itself the compromise of the user's credential backup *without* CASPER. We define another random variable T'_{PMS}, that is, the time interval soon after which the PMS (provider), if it has failed to detect its breach for the first τ interval, identifies the compromise of the user's credential backup by either itself or CASPER.

Given the public knowledge of λ and T_{PMS}, we build an MDP attacker similar to the one specified in Section 6.2 but with two additional options for the `caat`.

- The `caat` can either choose to immediately start attacking at $\tau = 0$ by attempting to log into each RP (one at a time for simplicity) or to wait for $\tau > 0$ time intervals before starting the attack. At the end of the τ-th interval, if PMS has not identified the compromise by itself in the first τ intervals, the `caat` can start its attack with persistent storage snapshots of RPs that are breached at a Poisson arrival rate with mean λ.
- After successfully taking over a certain number of accounts, the `caat` can choose to continue to attack the next or stop its attack early. In the former case, if it triggers detection by CASPER when attacking the next, the experiment ends immediately with $T'_{\mathsf{PMS}} = T_{\mathsf{PMS}}$. In the latter case, the number of taken-over accounts is fixed, denoted by m' ($\leq m$) and we assume that the `caat` can persistently access m' taken-over accounts until T'_{PMS} when the PMS detects the compromise by itself.

We measure the security efficacy of CASPER by observing how CASPER reduces the expected overall takeover duration of the user's RP accounts and define it as follows.

$$\mathsf{eff} = \frac{\mathbb{E}\left(T_{\mathsf{PMS}} \times m\right) - \mathbb{E}\left((T'_{\mathsf{PMS}} - \tau) \times m'\right)}{\mathbb{E}\left(T_{\mathsf{PMS}} \times m\right)} \tag{1}$$

Here $\mathbb{E}\left(T_{\mathsf{PMS}} \times m\right)$ is the expected overall takeover duration of the user's m RP accounts until when PMS discovers the credential compromise *without* CASPER. $\mathbb{E}\left((T'_{\mathsf{PMS}} - \tau) \times m'\right)$ represents the expected overall takeover duration of m' ($\leq m$) accounts that are taken over by the `caat` with the deployment of CASPER. The MDP attacker's goal is to minimize eff (and maximize $\mathbb{E}\left((T'_{\mathsf{PMS}} - \tau) \times m'\right)$) by adopting the best available strategies.

In our experiment, for a reasonable realistic estimate, we see each time interval as one month. Thus assuming a nine-month average compromise discovery delay as reported in [45], we set T_{PMS} following a normal distribution with mean $\mu = 9$ with a varying standard deviation σ. To interpret that the `caat` can observe one additional RP breach snapshot after waiting for every 2 and 4 additional months, we set $\lambda = 0.5$ and $\lambda = 0.125$, respectively.

Results of efficacy experiments. To understand the efficacy of CASPER, we explore the effects on eff from (1) under different parameterization settings of $(\lambda, \bar{k}, \sigma)$ as reported in Figure 8. Figure 8a represents the baseline setting where $\lambda = 0.5, \bar{k} = 32,$ and $\sigma = 2$. We modify λ, $\bar{k}$, and σ respectively in Figure 8b, Figure 8c, and Figure 8d from the baseline setting to observe the improvement in efficacy.

We observe a boost in the minimum eff when λ is decreased from 0.5 (8a) to 0.125 (8b) and $\bar{k}$ increased from 32 (8a) to 128 (8c) as shown in Figure 8b and Figure 8c respectively for $\alpha \in [0.6, 0.9]$. The main reason behind these boosts is that lower λ (and thus smaller n) or higher $\bar{k}$ increase the probability of the `caat` triggering detection by CASPER, particularly for a relatively large α (see Figure 6 and Figure 7). So the `caat` chooses either to bear such an increased risk or to attack later, hoping to observe more breached RPs. This results in shorter account takeover duration and is reflected by the boost of the minimum eff. The effect of modifying σ on the minimum eff is slightly less pronounced as shown in Figure 8a ($\sigma = 2$) and Figure 8d ($\sigma = 1$) respectively. This is because a smaller σ results in a more tightly centered normal distribution — a stronger detection power of the PMS provider.

One interesting observation is that m does not affect the results for a relatively large α, e.g., when $\alpha > 0.6$ in Figure 8a and Figure 8d, and $\alpha > 0.8$ in Figure 8c. This is because larger α motivate the best-strategy `caat` to adapt its strategy to increase τ, that is, to wait longer (i.e., with a larger τ) for more breached RP snapshots for a more accurate guess on the real detection secret. With it, the `caat` can derive real authentication credentials and log into all $m(= m')$ RP accounts without being detected by CASPER. This, however, shortens the overall account takeover duration from $\mathbb{E}\left(T_{\mathsf{PMS}} \times m\right)$ to $\mathbb{E}\left((T'_{\mathsf{PMS}} - \tau) \times m\right)$. As such, eff from (1) can be simplified to $\frac{\mathbb{E}(T_{\mathsf{PMS}}) - \mathbb{E}(T'_{\mathsf{PMS}} - \tau)}{\mathbb{E}(T_{\mathsf{PMS}})}$. The elimination of m in this simplified expression explains why m has no effects on eff for these cases.

6.3 False detection by a false-alarm attacker

CASPER may raise false alarms if a false-alarm attacker (also known as a denial-of-service attacker in literature, e.g., [50]), without compromising the PMS provider (and thus without knowledge of $\tilde{\mathsf{s}}$), successfully derives a passkey s that corresponds to an *active* decoy verification key, that is, $\mathsf{v} \in V'$. However, without knowledge of $\tilde{\mathsf{s}}$, a guess by a false-alarm attacker can hit one of the *active* decoy verification keys with a probability no greater than $\alpha k / 2^{\kappa}$ which is negligible in terms of the credential length κ. For example, for 256-bit private signing keys with k set to 32 and α set to 0.6, the probability of the false-alarm attacker guessing one decoy key that can trigger a false alarm is no greater than $0.6 \times 32 / 2^{256} < 2^{-251}$ which is negligibly small.

7 Experimental Evaluation

Implementation details. We developed a prototype implementation of CASPER [47] in Go language using the open source library virtualwebauthn [28] — which itself is built on top of another WebAuthn library [34]. We first implemented the three building blocks Π_{EncCred}, Π_{DecCred}, $\Pi_{\mathsf{GenVerifierSet}}$ required during the four steps of the BnR protocol as shown in Figure 4. The key derivation function (KDF) used by these three blocks is instantiated using the Password-Based Key

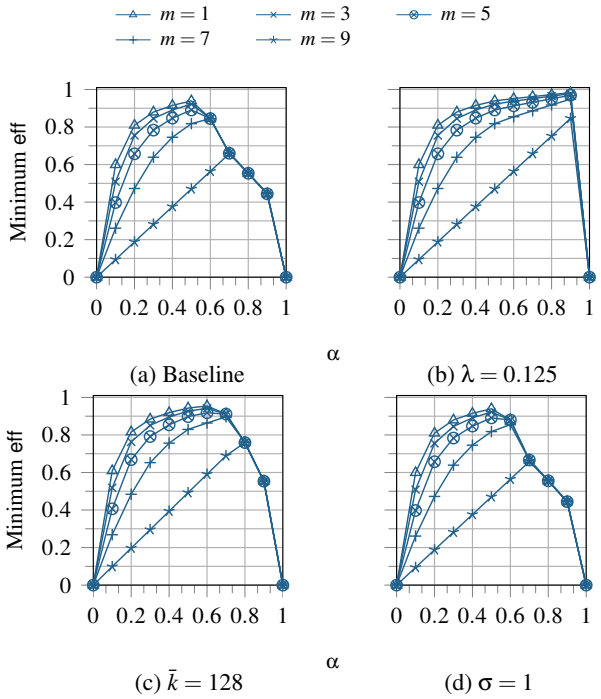


Figure 8: eff as a function of α with varying m. Subfigures (8b), (8c), and (8d) show the effects of strengthening security by on parameter, i.e., λ, $\bar{k}$, and σ respectively, from the baseline (8a) where $\lambda = 0.5, \bar{k} = 32,$ and $\sigma = 2$.

Derivation Function (PBKDF2) with SHA-256 and 600,000 iterations. For all cryptographic operations, we select the elliptic curve group secp256k1 and set $\kappa = 128$.

To demonstrate the implementation feasibility of CASPER, we instantiated the user secret η with 2-digit system-generated PINs for completeness and set $k = 32$ and $\alpha = 0.6$. We perceive this as a reasonable setting based on our detection analysis presented in Section 6, it will have a minimum true detection rate of 0.92 (for $m = n = 1$) and 0.83 (for $m = n = 3$) as observed from Figure 6. The virtual authenticator marks $\lfloor \alpha \cdot k \rfloor = \lfloor 0.6 \cdot 32 \rfloor = 19$ of the 32 decoy verification keys as active and sends to the RP all $32+1$ verification keys including all decoys and the real, as well as 19 indices indicating which are active decoys.

We implemented the RP and PMS on two separate server nodes also written in Go language. The RP node allows a WebAuthn-supported user account registration and a login end point for the virtual client. For simplicity, instead of using existing APIs PMS from providers such as Apple, Google, or Microsoft, we emulated the same for our PMS node that exposed standard APIs for passkey sync and restoration support. These APIs were then consumed by our virtual authenticator and client. The current prototype implementation did not use TLS to secure the communication channel of the virtual client and authenticator with PMS and RP but is strongly recommended when CASPER is deployed in practice.

Measuring performance overhead. For latency and performance comparisons, we run the PMS (t2.medium) and the RP (t2.micro) on two different AWS EC2 instances running on Ubuntu 20.04 LTS and located in two different regions — US-East and US-West respectively. We run the virtual authenticator and the client on commodity hardware (MacBook Pro M2 with 16 GB of memory). Finally, we registered 10 user accounts at the RP and attempted 25 login attempts on each of the accounts and measured the time delay during account registration and login as experienced by users for CASPER.

When $k = 32$, and $\alpha = 0.6$, our evaluation shows that CASPER introduced an additional delay of 325 ($\pm$ 29) ms to account registration. The majority of the computation time during account registration is taken by the KDF used by Π_{EncCred}, Π_{DecCred}, $\Pi_{\mathsf{GenVerifierSet}}$ procedures of around 224 ($\pm$ 3) ms. Increasing (k, α) would increase the number of invocations of KDF, and thus result in higher delays in user registration and authenticator setup time (step 1 and 2). However, we remark that user registration is an infrequent operation, and will not affect user experience significantly. Importantly, the compromise detection algorithm, specifically the additional membership tests required by CASPER in the algorithm, adds only an average delay of 36 ($\pm$ 8) ms to users' daily login time, which we argue would be hardly noticeable by users.

Benchmarking storage overhead. CASPER adds only minimal storage costs to participating parties. The PMS additionally stores a random value z for each credential entry and a detection secret set W for each user. Each authenticator also needs to store the real detection secret w_{i^*} and a set of detection secrets W. An RP needs to store V and V' for each user account, assuming that there is only one valid credential per account. If z, w, and v are of size 256 bits, and $k = 32$, as a rough estimate for 1 million users with each having 200 RP accounts, the deployment of CASPER would cost the PMS approximately 13.86 Gigabytes and each RP no more than 2.08 Gigabytes in storage only.

8 Discussion

We design CASPER to enable RPs to detect attackers' attempts to log into users' accounts using passkeys stolen from a PMS. In Section 6, we demonstrate the detection effectiveness of CASPER. In this section, we discuss the usability and deployment considerations of CASPER.

8.1 Usability analysis

We propose CASPER as a general detection framework, with the user secret η serving as a flexible component. The usability of CASPER depends on that of η. Specifically, from an end-user's perspective, CASPER may involve user participation in two operations:[6] (1) registration of η during setup

[6] As we will detail later, certain design options can eliminate the need for user involvement in one or both of the operations. However, for the sake of examining CASPER's usability, we conservatively aim to list all possible

of the first authenticator following PMS *account creation*, and (2) providing the same η to a new authenticator during *authenticator setup* for retrieving passkeys from the PMS.

The user secret η can be a memorized secret selected by the user or assigned by the system, or derived from biometrics or other (long-term) secrets (e.g., bank account numbers or device unlock PINs) of the user. Here we present a brief usability analysis for the case where η is instantiated with 2-digit system-generated PINs [7] to identify the usability caveats and potential mitigation for this instantiation.

PIN-based CASPER: a usability case study. Since CASPER is mostly invisible to users in their daily lives, we focus solely on the following user tasks required by this "PIN-based CASPER": (1) *PIN registration*: During setting up the first device after creating an account with the PMS, CASPER generates a 2-digit random PIN and displays it to the user, instructing them to memorize or note it down. The user is then required to confirm receipt by entering the PIN twice. The last step is repeated if either entry fails to match the PIN displayed. (2) *PIN management*: The user needs to memorize or note down the 2-digit PIN and recall or retrieve it when needed. (3) *PIN entry*: Later, for each subsequent authenticator device setup, the interface will prompt the user to enter the 2-digit PIN twice.

For PIN-based CASPER, the key usability issue is difficulty in memorizing or noting down the PIN, and recalling or retrieving the PIN when setting up subsequent devices / authenticators. This challenge is not particular to CASPER—similar usability challenges also arise in everyday scenarios where such secrets are commonly used for device unlocking or end-to-end encryption in most cloud-based credential management systems [31]. In these cases, users must effectively manage their PINs to avoid being locked out of devices or losing access to their credential backups. We argue that with PIN-based CASPER, the likelihood of such events is lower, as prior user studies have shown that 2-digit PINs are relatively easier for users to manage [44]. Other usability issues such as PIN input errors during device setup could be avoided by leveraging existing typo-reducing techniques, e.g., [53] or by requesting repetitive PIN entry and confirmation.

While system-generated PINs offer quantifiable security and involve lower deployment complexity, other options to instantiate η, e.g., based on biometrics, can relieve users of memory burdens and address most of the usability issues identified above. For further discussion on alternative options to instantiate η, see Appendix E.

Future user studies. To thoroughly evaluate CASPER's usability — including user adoption, perceptions, ease of use, and action accuracy — comprehensive user studies tailored to *specific* η *instantiations and UI designs* are necessary prior to deployment. This paper does not include such studies, leaving them as future work to better understand CASPER's real-world usability.

8.2 Enhancing CASPER's usability

Here we provide suggestions and recommendations to enhance CASPER's overall usability, such as by setting up accurate user perceptions of CASPER, and reducing or eliminating users' efforts additionally required by CASPER.

User perceptions of CASPER. We believe CASPER is appealing to users who: 1) wish to use PMS, 2) have concerns about the security of their PMS passkey storage, as highlighted in a recent user study [59], and 3) prefer not to adopt less user-friendly mitigations for account security that affects daily login routines (e.g., 2FA / MFA). However, establishing an accurate perception and mental model is a necessary step towards a usable CASPER. To do so, the user interface of CASPER should first clearly detail its goal — to improve users' account security by detecting abuse of passkeys leaked from PMS providers. Second, the interface should clarify that CASPER aims to provide "an additional detection layer" without disrupting users' daily account login routines.

Reducing users' efforts. After η registration, CASPER only requires additional user efforts for η input during authenticator setup, and such instances will be rare. Assuming users set up a new device as often as they replace smartphones (approximately every 40 months [8]), a user would enter η only 32 times over 106 years.

Users' efforts could be even further minimized for the majority of cases when users have at least one registered authenticator available when setting up a new one — for example, when setting up a new smartphone while a previously registered smartphone, tablet, or laptop is available. In such cases, secure device-to-device communication (e.g., similar to Apple's AirDrop) can be used between a new authenticator and a registered one to synchronize w_{i^*} derived from η, eliminating the need for users to input η.

Making CASPER user-invisible. When usability requirements dictate that no additional user tasks should be performed, one may consider extracting η from existing secrets that users already have and use daily, e.g., from PINs / passwords, pattern locks, or biometrics locally for device unlock. In this scenario, it is possible to hide CASPER entirely from users to improve CASPER's usability. However, we caution readers that such a design could introduce additional security implications, particularly if the entropy source is a user-chosen secret without sufficient entropy, which could render CASPER less effective.

8.3 Deployment considerations

Deployment requirements. CASPER is by design well-suited for real-world deployment because it is compatible

user involvement here.

[7] A 2-digit system-generated PIN has approximately $\log_2(100) \approx 6.64$ bits of entropy, whereas η with around 5 bits already provides compelling detection accuracy according to our analysis in Section 6.

with PMS providers' existing credential synchronization and storage implementations and requires only minimal changes on RPs' side. Implementation-wise, CASPER only requires PMS to additionally store the constant size detection secret set W. For RPs, CASPER requires them to store public verification keys (i.e., V' and V) as specified in the BnR protocol shown in Figure 4 and to implement the simple detection algorithm prescribed in Figure 5.

However, a potential deployment challenge may arise from the need to modify authenticators to support the BnR protocol and its underlying algorithms. Therefore, to promote the deployment of CASPER, authenticator compatibility would need to be provided by vendors and supported by standardization bodies such as the FIDO Alliance.

Again, instead of a specific instantiation and implementation, we propose CASPER as a general framework to detect centralized PMS. Beyond direct deployment, we hope that our work will inspire future efforts to explore instantiations of CASPER— exploring options such as η instantiations and input methods and alternative methods for generating decoy credentials. These choices should be closely aligned with the real-world requirements for security, usability, and deployment. Moreover, as cryptographic credentials like passkeys and HOTP / TOTP seeds as well as their backup and recovery become increasingly important, we hope our work encourages further research on breach detection for credential backup systems with the goal of improving users' account security.

Risk-based authentication and CASPER. To improve account security, one might seek to compare Risk-Based Authentication (RBA) [39, 78, 79] with CASPER to guide deployment decisions. To help with this process, we provide a detailed comparison of RBA and CASPER to highlight their differences and complementary characteristics.

RBA aims to detect likely unauthorized login attempts by profiling login behaviors and identifying malicious behaviors, without requiring users to manage additional secrets. In contrast, given only a low-entropy η, CASPER provides RPs with a reliable signal indicating whether a user's passkey may have been leaked from PMS. This signal offers deeper insight into the *cause* of the unauthorized account access and allows RP to make more informed decisions, such as requesting users to reset their passkeys to protect their accounts from risks resulting from breaches of PMS. In contrast, RBA aims to flag unauthorized login attempts based on users' login patterns to help RPs to decide if further authentication is needed [78].

RBA relies on login information, e.g., IP addresses, user-agent strings, and user login patterns, which is *independent* of the deployed authentication schemes. This makes RBA compatible with a wide range of authentication schemes, although it is mainly deployed to complement password-based authentication today. CASPER, on the other hand, is a detection framework specifically designed to identify PMS breaches that leak synchronized cryptographic credentials like passkeys and HOTP / TOTP seeds.

In terms of security assumptions, the effectiveness of RBA depends on whether the login information used to identify unauthorized login attempts could be stolen and spoofed. However, existing RBA mostly relies on non-private login information like geolocations and user-agent strings [39], which can be easily obtained and spoofed by a sophisticated attacker to effectively reduce RBA's accuracy [38]. In contrast, CASPER remains effective against the same sophisticated attacker. As shown in Section 6, CASPER achieves high detection accuracy even when the attacker is allowed to breach multiple RPs to gather information for attacking other RPs.

Social engineering attacks against CASPER. Attackers could use social engineering techniques like phishing [54, 80] or pretexting [80] to steal the user secret η. User secrets required by existing PMS such as iCloud and Google backup are also susceptible to similar social engineering attacks. We note that the leak of a user's η only renders the detection ineffective — as if CASPER had never been enabled for this user — but would not degrade the security of their passkeys or RP accounts.

Detection notifications. RP may choose to notify the victim user of the potential compromise of their passkeys from the provider. We discuss two ways the RP could notify the victim user: actively or passively. For active notification, RP can promptly send detection notifications to users via established communication channels (e.g., email, app notifications, SMS). When all communication channels between the user and the RP are unavailable, the RP could also resort to a passive notification such as notifying the user on its webpage or application when they log into their account next time. With such notification, the user can decide to reset their passkeys, better secure their provider account, or even consider switching to a different provider.[8]

Handling spoofed detection alerts. In this paper, we make no effort to address the possibility that participating RPs might withhold detection alerts from users when detection happens, given that these RPs could equally well do so by simply not participating. However, we have to take into consideration that participating RPs may attempt to send spoofed detection notifications to users when detection does not happen (e.g., just to wrong an unbreached PMS provider). CASPER can provide a strong guarantee for identifying such spoofed notifications by additionally requiring the RP to return to the user the authentication response $(\mathsf{v}', \mathsf{rsp}', \gamma')$ that triggers the breach alert as a "proof of detection".

Specifically, for the FIDO2 case, the user (device), after receiving $(\mathsf{v}', \mathsf{rsp}', \gamma')$ as the proof of detection, first performs

[8] We note that recovering from such an identified compromise, as required by other account recovery needs as they exist today [62], requires users to register at the RP for a secure secondary authentication method (or sometimes termed backup/fallback authentication) in advance to ensure the user's legitimacy for resetting credentials.

$\mathsf{Verify}(\mathsf{v}', \mathsf{rsp}', \gamma')$ to check that the detection proof is well-formed. Given that γ' is a valid signature of rsp' under v', the user device then runs $\Pi_{\mathsf{GenVerifierSet}}$ to re-generate V, and checks if $\mathsf{v}' = \mathsf{v}_{i^*}$ OR $\mathsf{v}' \notin V$. If this check is true, the user device can confirm this is a spoofed notification because only a (rsp', γ') pair verifiable by one of the verifiers in $V \setminus \{\mathsf{v}_{i^*}\}$ can trigger a detection alarm. Informally, without the knowledge of the user's private signing key s_{i^*} or a `caat`'s presence (i.e., compromise of users' credential backup at provider), it would be difficult for an RP to forge a valid detection proof if the underlying digital signature scheme is secure.

9 Conclusion

In this paper, we present CASPER, the first framework to detect the abuse of users' passkeys leaked from passkey management services (PMS). CASPER can be seamlessly integrated into the existing FIDO2 authentication protocols without disrupting users' daily account login routines. Additionally, CASPER is compatible with existing PMS implementations and introduces minimum storage and computation overhead to participating parties. We demonstrate that CASPER provides compelling detection effectiveness, even against attackers who exploit information from website breaches to optimize their strategies to avoid detection. We believe that the widespread deployment of CASPER will enhance users' account security particularly in scenarios where a PMS provider fails to protect users' passkey storage from compromise.

Acknowledgments

We thank the anonymous reviewers and the shepherd of our paper for their insightful comments and suggestions. This research is supported in part by the University of Wisconsin—Madison Office of the Vice Chancellor for Research and Graduate Education and NSF award #2339679, and US Department of Commerce award #70NANB21H043.

Ethics and Open Science Policy

Ethics considerations. Our work aims to improve account security by detecting the abuse of passkeys leaked from the cloud storage of passkey management services or providers. Thus we believe our work does not pose any potential harm to users or raise any ethical concerns.

Open science. We released the code base of CASPER as described in Section 7 as open source software [47] under the GPL-3.0 license. Additionally, we released the model checking scripts (in PRISM [56] model checking language) to evaluate the detection effectiveness of CASPER in Section 6.2.

References

[1] CNBC: Why passkeys from Apple, Google, Microsoft may soon replace your passwords. https://fidoalliance.org/cnbc-why-passkeys-from-apple-google-microsoft-may-soon-replace-your-passwords/, 2024.

[2] FIDO Alliance. The FIDO ("Fast IDentity Online") Alliance – Industry Association to Promote Authentication Standards. https://fidoalliance.org/fido2/, 2024.

[3] Passkeys in Dashlane. https://support.dashlane.com/hc/en-us/articles/7888558064274-Passkeys-in-Dashlane, 2024.

[4] Start Your Passwordless Authentication Journey - LastPass. https://www.lastpass.com/features/passwordless-authentication, 2024.

[5] Unlock 1Password with a passkey (beta). https://support.1password.com/passkeys/, 2024.

[6] Experts Fear Crooks are Cracking Keys Stolen in LastPass Breach. https://krebsonsecurity.com/2023/09/experts-fear-crooks-are-cracking-keys-stolen-in-lastpass-breach/, Available on: 2024-03-01.

[7] LastPass' latest data breach exposed some customer information. https://www.theverge.com/2022/11/30/23486902/lastpass-hackers-customer-information-breach, Available on: 2024-03-01.

[8] Smartphone providers await 2H smartphone rebound. https://www.bloomberg.com/professional/blog/smartphone-providers-await-2h-smartphone-rebound, Available on: 2024-03-01.

[9] Two-Factor Authentication Market. https://www.marketresearchfuture.com/reports/two-factor-authentication-market-3772, Available on: 2024-03-01.

[10] Apple Plaftorm Security, iCloud Keychain security overview. https://support.apple.com/guide/security/icloud-keychain-security-overview-sec1c89c6f3b/1/web/1, Available on: 2025-01-30.

[11] Passkeys in Windows. https://support.microsoft.com/en-us/windows/passkeys-in-windows-301c8944-5ea2-452b-9886-97e4d2ef4422, Available on: 2025-01-30.

[12] Passwordless login with passkeys. https://developers.google.com/identity/passkeys, June 2023.

[13] M. H. Almeshekah, C. N. Gutierrez, M. J. Atallah, and E. H. Spafford. Ersatzpasswords: Ending password cracking and detecting password leakage. In *Proceedings of the 31st Annual Computer Security Applications Conference*, pages 311–320, 2015.

[14] Apple. About the security of passkeys. 2024. https://support.apple.com/en-us/HT213305l.

[15] Apple. Escrow security for iCloud Keychain. https://support.apple.com/guide/security/escrow-security-for-icloud-keychain-sec3e341e75d/1/web/1, 2024.

[16] L. Ballard, S. Kamara, F. Monrose, and M. K. Reiter. Towards practical biometric key generation with randomized biometric templates. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 235–244, 2008.

[17] M. Barbosa, A. Boldyreva, S. Chen, and B. Warinschi. Provable security analysis of FIDO2. In *Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part III 41*, pages 125–156. Springer, 2021.

[18] K. Bicakci and Y. Uzunay. Is FIDO2 Passwordless Authentication a Hype or for Real?: A Position Paper. In *2022 15th International Conference on Information Security and Cryptography (ISCTURKEY)*, pages 68–73. IEEE, 2022.

[19] N. Bindel, C. Cremers, and M. Zhao. FIDO2, CTAP 2.1, and WebAuthn 2: Provable security and post-quantum instantiation. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 1471–1490. IEEE, 2023.

[20] A. Birgisson. Security of Passkeys in the Google Password Manager. 2023. https://security.googleblog.com/2022/10/SecurityofPasskeysintheGooglePasswordManager.html.

[21] H. Bojinov, E. Bursztein, X. Boyen, and D. Boneh. Kamouflage: Loss-resistant password management. In *Computer Security–ESORICS 2010: 15th European Symposium on Research in Computer Security, Athens, Greece, September 20-22, 2010. Proceedings 15*, pages 286–302. Springer, 2010.

[22] J. Bonneau. The science of guessing: analyzing an anonymized corpus of 70 million passwords. In *2012 IEEE symposium on security and privacy*, pages 538–552. IEEE, 2012.

[23] D. Chang, A. Goel, S. Mishra, and S. K. Sanadhya. Generation of secure and reliable honeywords, preventing false detection. *IEEE Transactions on Dependable and Secure Computing*, 16(5):757–769, 2019.

[24] M. Chase, H. Davis, E. Ghosh, and K. Laine. Acsesor: A new framework for auditable custodial secret storage and recovery. *Cryptology ePrint Archive*, 2022.

[25] R. Chatterjee, J. Bonneau, A. Juels, and T. Ristenpart. Cracking-resistant password vaults using natural language encoders. In *2015 IEEE Symposium on Security and Privacy*, pages 481–498. IEEE, 2015.

[26] H. Cheng, W. Li, P. Wang, C.-H. Chu, and K. Liang. Incrementally updateable honey password vaults. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 857–874, 2021.

[27] E. Dauterman, H. Corrigan-Gibbs, and D. Mazières. SafetyPin: Encrypted backups with Human-Memorable secrets. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 1121–1138, 2020.

[28] Descope. virtualwebauthn. https://github.com/descope/virtualwebauthn, 2024.

[29] T. Dierks and E. Rescorla. The transport layer security (tls) protocol version 1.2. Technical report, 2008.

[30] A. Dionysiou and E. Athanasopoulos. Lethe: Practical data breach detection with zero persistent secret state. In *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*, pages 223–235. IEEE, 2022.

[31] J. Doolani, M. Wright, R. Setty, and S. M. Haque. Locimotion: Towards learning a strong authentication secret in a single session. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–13, 2021.

[32] F. Duan, D. Wang, and C. Jia. A security analysis of honey vaults. In *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024.

[33] I. Erguler. Achieving flatness: Selecting the honeywords from existing user passwords. *IEEE Transactions on Dependable and Secure Computing*, 13(2):284–295, 2015.

[34] F. Amacker. WebAuthn server library (Go/Golang). https://github.com/fxamacker/webauthn, 2024.

[35] F. M. Farke, L. Lorenz, T. Schnitzler, P. Markert, and M. Dürmuth. "You still use the password after all"—exploring FIDO2 security keys in a small company. In *Sixteenth Symposium on Usable Privacy and Security (SOUPS 2020)*, pages 19–35, 2020.

[36] FIDO Alliance. Client to authenticator protocol (ctap). https://fidoalliance.org/specs/fido-v2.2-rd-20230321/fido-client-to-authenticator-protocol-v2.2-rd-20230321.html, 2023.

[37] D. Florêncio, C. Herley, and P. C. Van Oorschot. An administrator's guide to internet password research. In *28th large installation system administration conference (LISA14)*, pages 44–61, 2014.

[38] D. Freeman, S. Jain, M. Dürmuth, B. Biggio, and G. Giacinto. Who are you? a statistical approach to measuring user authenticity. In *NDSS*, volume 16, pages 21–24, 2016.

[39] A. Gavazzi, R. Williams, E. Kirda, L. Lu, A. King, A. Davis, and T. Leek. A study of Multi-Factor and Risk-Based authentication availability. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 2043–2060, 2023.

[40] C. Gilsenan, F. Shakir, N. Alomar, and S. Egelman. Security and privacy failures in popular 2FA apps. In *32nd USENIX Security Symposium (USENIX Security 23)*, 2023.

[41] M. Golla, B. Beuscher, and M. Dürmuth. On the security of cracking-resistant password vaults. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 1230–1241, 2016.

[42] J. Guan, H. Li, H. Ye, and Z. Zhao. A formal analysis of the FIDO2 protocols. In *European Symposium on Research in Computer Security*, pages 3–21. Springer, 2022.

[43] Z. Huang, L. Bauer, and M. K. Reiter. The impact of exposed passwords on honeyword efficacy. *33rd USENIX Security Symposium (USENIX Security 24)*, 2024.

[44] J. H. Huh, H. Kim, R. B. Bobba, M. N. Bashir, and K. Beznosov. On the memorability of system-generated PINs: Can chunking help? In *eleventh symposium on usable privacy and security (SOUPS 2015)*, pages 197–209, 2015.

[45] IBM Security. Cost of a data breach report 2023. https://www.ibm.com/security/digital-assets/cost-data-breach-report/, 2023.

[46] M. Islam, M. S. Bohuk, P. Chung, T. Ristenpart, and R. Chatterjee. Araña: Discovering and characterizing password guessing attacks in practice. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 1019–1036, Anaheim, CA, August 2023. USENIX Association.

[47] M. Islam and K. C. Wang. CASPER. https://archive.softwareheritage.org/browse/origin/directory/?origin_url=https://github.com/islamazhar/CASPER, 2024.

[48] D. Johnson, A. Menezes, and S. Vanstone. The elliptic curve digital signature algorithm (ecdsa). *International journal of information security*, 1:36–63, 2001.

[49] A. Juels and T. Ristenpart. Honey encryption: Security beyond the brute-force bound. In *Advances in Cryptology–EUROCRYPT 2014: 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings 33*, pages 293–310. Springer, 2014.

[50] A. Juels and R. L. Rivest. Honeywords: Making password-cracking detectable. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 145–160, 2013.

[51] J. Katz, R. Ostrovsky, and M. Yung. Efficient and secure authenticated key exchange using weak passwords. *Journal of the ACM (JACM)*, 57(1):1–39, 2009.

[52] M. Kepkowski, M. Machulak, I. Wood, and D. Kaafar. Challenges with passwordless fido2 in an enterprise setting: A usability study. In *2023 IEEE Secure Development Conference (SecDev)*, pages 37–48. IEEE, 2023.

[53] M. Khamis, T. Seitz, L. Mertl, A. Nguyen, M. Schneller, and Z. Li. Passquerade: Improving error correction of text passwords on mobile devices by using graphic filters for password masking. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–8, 2019.

[54] M. Khonji, Y. Iraqi, and A. Jones. Phishing detection: a literature survey. *IEEE Communications Surveys & Tutorials*, 15(4):2091–2121, 2013.

[55] H. Krawczyk. Cryptographic extraction and key derivation: The HKDF scheme. In *Annual Cryptology Conference*, pages 631–648. Springer, 2010.

[56] M. Kwiatkowska, G. Norman, and D. Parker. Prism 4.0: Verification of probabilistic real-time systems. In *Computer Aided Verification: 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings 23*, pages 585–591. Springer, 2011.

[57] L. Lassak, A. Hildebrandt, M. Golla, and B. Ur. " it's stored, hopefully, on an encrypted server": Mitigating users' misconceptions about FIDO2 biometric WebAuthn. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 91–108, 2021.

[58] L. Lassak, P. Markert, M. Golla, E. Stobert, and M. Dürmuth. A comparative long-term study of fallback authentication schemes. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, 2024.

[59] L. Lassak, E. Pan, B. Ur, and M. Golla. Why Aren't We Using Passkeys? Obstacles Companies Face Deploying FIDO2 Passwordless Authentication. In *Proceedings of the 33rd USENIX Security Symposium. Philadelphia, PA, August 2024*, 2024.

[60] B. Lu, X. Zhang, Z. Ling, Y. Zhang, and Z. Lin. A measurement study of authentication rate-limiting mechanisms of modern websites. In *Proceedings of the 34th annual computer security applications conference*, pages 89–100, 2018.

[61] S. G. Lyastani, M. Schilling, M. Neumayr, M. Backes, and S. Bugiel. Is FIDO2 the kingslayer of user authentication? A comparative usability study of FIDO2 passwordless authentication. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 268–285. IEEE, 2020.

[62] P. Markert, A. Adhikari, and S. Das. A transcontinental analysis of account remediation protocols of popular websites. *arXiv preprint arXiv:2302.01401*, 2023.

[63] D. M'Raihi, M. Bellare, F. Hoornaert, D. Naccache, and O. Ranen. RFC 4226: HOTP: An HMAC-based one-time password algorithm, 2005.

[64] D. M'Raihi, S. Machani, M. Pei, and J. Rydell. RFC 6238: TOTP: Time-based one-time password algorithm, 2011.

[65] E. Naprys. Third of Americans use password managers. https://cybernews.com/security/third-of-americans-use-password-managers/, 2024.

[66] National Vulnerability Database. CVE-2023-42847. https://nvd.nist.gov/vuln/detail/CVE-2023-42847, Published on: 2023-10-25.

[67] C. Orsini, A. Scafuro, and T. Verber. How to recover a cryptographic secret from the cloud. *Cryptology ePrint Archive*, 2023.

[68] K. Owens, O. Anise, A. Krauss, and B. Ur. User Perceptions of the Usability and Security of Smartphones as FIDO2 Roaming Authenticators. In *SOUPS USENIX Security Symposium*, pages 57–76, 2021.

[69] T. Rao, Y. Su, P. Xu, Y. Zheng, W. Wang, and H. Jin. You reset i attack! a master password guessing attack against honey password vaults. In *European Symposium on Research in Computer Security*, pages 141–161. Springer, 2023.

[70] Certicom Research. SEC 2: Recommended Elliptic Curve Domain Parameters. https://www.secg.org/sec2-v2.pdf, 2010.

[71] K. Thomas, F. Li, A. Zand, J. Barrett, J. Ranieri, L. Invernizzi, Y. Markov, O. Comanescu, V. Eranti, A. Moscicki, D. Margolis, V. Paxson, and E. Bursztein. Data breaches, phishing, or malware? understanding the risks of stolen credentials. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, pages 1421–1434, 2017.

[72] E. Ulqinaku, H. Assal, A. AbdelRahman, S. Chiasson, and S. Capkun. Is Real-time Phishing Eliminated with FIDO? Social Engineering Downgrade Attacks against FIDO Protocols. In *Proceedings of the 30th USENIX Security Symposium (USENIX Security 21)*, pages 3811–3828. USENIX Association, 2021.

[73] W3C. Web authentication: An api for accessing public key credentials level 3. https://www.w3.org/TR/webauthn-3/, 2023.

[74] D. Wang, Y. Zou, Q. Dong, Y. Song, and X. Huang. How to attack and generate honeywords. In *43rd IEEE Symposium on Security and Privacy*, pages 966–983. IEEE, May 2022.

[75] K. C. Wang and M. K. Reiter. Using amnesia to detect credential database breaches. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 839–855, 2021.

[76] K. C. Wang and M. K. Reiter. Bernoulli honeywords. In *31st ISOC Network and Distributed System Security Symposium*, February 2024.

[77] A. Weinert. How it works: Backup and restore for Microsoft Authenticator. https://techcommunity.microsoft.com/t5/microsoft-entra-blog/how-it-works-backup-and-restore-for-microsoft-authenticator/ba-p/1006678, 2019.

[78] S. Wiefling, M. Dürmuth, and L. Lo Iacono. More than just good passwords? a study on usability and security perceptions of risk-based authentication. In *Proceedings of the 36th Annual Computer Security Applications Conference*, ACSAC '20, page 203–218, New York, NY, USA, 2020. Association for Computing Machinery.

[79] S. Wiefling, L. Lo Iacono, and M. Dürmuth. Is this really you? an empirical study on risk-based authentication applied in the wild. In *ICT Systems Security and Privacy Protection: 34th IFIP TC 11 International Conference, SEC 2019, Lisbon, Portugal, June 25-27, 2019, Proceedings 34*, pages 134–148. Springer, 2019.

[80] M. Workman. Wisecrackers: A theory-grounded investigation of phishing and pretext social engineering threats to information security. *Journal of the American society for information science and technology*, 59(4):662–674, 2008.

[81] L. Würsching, F. Putz, S. Haesler, and M. Hollick. FIDO2 the Rescue? Platform vs. Roaming Authentication on Smartphones. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pages 1–16, 2023.

[82] M. Zinkus, T. M. Jois, and M. Green. SoK: Cryptographic confidentiality of data on mobile devices. *In Proceedings on Privacy Enhancing Technologies (PoPETS)*, 2022.

A Two Common Strategies of PMS for Credential Backup Protection

As discussed in Section 2, PMS providers typically offer users two strategies to secure their credential backups with user secrets. One is based on *key derivation* and the other is based on *key escrow*.

Key derivation. In the first strategy, a user's credential backup is encrypted using a key that only the user can access. The encryption key can be directly derived from a user-chosen secret (e.g., a "master password"), e.g., via key derivation functions (KDF). This approach provides end-to-end encryption to secure users' passkeys — encryption and decryption of a user's PMS passkey backup are performed locally on the user's authenticator, avoiding giving PMS access to the user-chosen secret and the derived encryption key. This strategy is adopted by popular credential managers such as LastPass [4], 1Password [5]. While this strategy gives the user some security control over their passkeys backup at PMS provider, it also enables attackers to perform offline cracking on the user's passkey backup when it gets leaked from the provider. In particular, prior research has observed that user-chosen secrets are easily guessable, especially those that users can recall consistently [46]. Many applications utilize the KDF defined in PKCS#12 which is not appropriate to withstand offline attacks leveraging modern hardware [40]. Thus, ensuring the protection of users' passkeys with this strategy has long been

a significant challenge. For example, serious concerns have been raised regarding the impact of offline cracking attacks following the breach of LastPass's cloud storage [6, 7].

Key escrow. The second strategy requires the user secret (e.g., a PIN or passcode, or a screen-lock pattern) to be independent of the encryption key generation and serves solely as a "verification secret" that the PMS provider or its components use to verify the user's identity. Specifically, provider independently generates an encryption key to encrypt a user's passkey backup at rest and stores the key on its own key management service (KMS). When the user wishes to retrieve their credential backup, the PMS provider retrieves their key from KMS securely only if the user presents a valid "verification secret". This strategy is adopted by iCloud KeyChain from Apple, Google Password Manager, and Password Monitor from Microsoft [15, 20, 77]. Compared to the former strategy, this strategy requires a user to give up their control over their credential backups and trust that the provider can implement their cloud storage and KMS correctly and securely as expected, even though this may not always be the case [66].

Furthermore, even if PMS implementation is secure, insider attackers within the PMS provider could potentially obtain query access to the KMS and retrieve the key after successfully guessing the low-entropy user verification secret [27, 82]. This remains possible even under the provider's rate-limiting policies [10, 20], which could not only fail to prevent such attacks but also undesirably introduce denial-of-service concerns for users. What is more concerning is that many PMS providers adopt one or more non-cryptographic authentication methods such as passwords or secret questions for account access, recovery, or verification secret reset [58]. In this case, the security of the user's passkey backups against a remote attacker may eventually fall back to that of those weaker authentication methods.

B Definition of KDF

A key derivation function (KDF) is a fundamental cryptographic primitive that produces cryptographic keys from a private input, such as a user password. When used as encryption keys for data storage or transmission, it is crucial that the keys generated by a KDF are computationally indistinguishable from random strings [55] in order to prevent an attacker from obtaining useful information of the private input string. In other words, an attacker should not be able to determine whether a given binary string is a cryptographic key produced by a KDF or just a random string of equivalent length. In this paper, we follow standard assumptions by requiring a KDF to be a pseudorandom function (PRF) [55] and consider the following definition of a KDF:

Definition 1 (Key Derivation Function)**.** *A key derivation function (KDF), denoted by* $\mathsf{KDF}(w,z)$*, is a pseudorandom function* F*:* $\{0,1\}^\kappa \times \{0,1\}^\kappa \to \{0,1\}^\kappa$ *that takes as input a user detection secret* w *and a randomness* z *uniformly chosen from* $\{0,1\}^\kappa$ *and outputs a user key* u*.*

C Flatness Preservation

Here we show that the flatness of detection secrets used in CASPER is *preserved* despite the compromise of PMS storage. To capture how accurately a distinguishing attacker can identify w_{i^*} from W output by $\mathcal{G} = \langle \mathsf{GenDetectSecrets}, \mathsf{SelectRealSecret} \rangle$, we consider a *flatness* experiment $\mathsf{Expt}^{\mathsf{flt},\mathcal{G}}_{\eta,k}$ defined as follows:

$$
\begin{array}{l}
\texttt{Experiment } \mathsf{Expt}^{\mathsf{flt},\mathcal{G}}_{\eta,k}(\mathcal{D}) \\
\quad W \xleftarrow{\$} \mathsf{GenDetectSecrets}(k) \\
\quad w_{i^*} \leftarrow \mathsf{SelectRealSecret}(W,\eta) \\
\quad \hat{w} \leftarrow \mathcal{D}_{\mathcal{G}}(W) \\
\quad \texttt{if } \hat{w} = w_{i^*} \\
\qquad \texttt{then return } 1 \\
\quad \texttt{else return } 0
\end{array}
$$

We define the advantage of $\mathcal{D}$, given η and k, as:

$$
\mathsf{Adv}^{\mathsf{flt},\mathcal{G}}_{\eta,k}(\mathcal{D}) \stackrel{\text{def}}{=} \mathbb{P}\left(\mathsf{Expt}^{\mathsf{flt},\mathcal{G}}_{\eta,k}(\mathcal{D}) = 1\right) - \frac{1}{k+1},
$$
$$
\mathsf{Adv}^{\mathsf{flt},\mathcal{G}}_{\eta,k} \stackrel{\text{def}}{=} \max_{\mathcal{D}} \{\mathbb{P}\left(\mathsf{Expt}^{\mathsf{flt},\mathcal{G}}_{\eta,k}(\mathcal{D}) = 1\right)\},
$$

where the maximum is taken over all distinguishing attackers $\mathcal{D}$.

We then consider the attacker's ability to get a user's passkey backup entries for *multiple* accounts leaked from PMS by defining the following PMS oracle O_{cbs} that takes w_{i^*} as input and outputs a pair $(\tilde{\mathsf{s}}, z)$ for each oracle query. Then we consider the following experiment to characterize how much better a decoy distinguishing adversary $\mathcal{A}$ can distinguish w_{i^*} when it additionally has access to such a PMS oracle O_{cbs}, where aid, uid and sid are ignored due to the independence of W and w_{i^*} on them:

$$
\begin{array}{l}
\text{Oracle } O_{\mathsf{cbs}}(w_{i^*}) \\
\quad \mathsf{s}_{i^*} \xleftarrow{\$} \{0,1\}^\kappa \\
\quad (\tilde{\mathsf{s}}, z) \leftarrow \Pi_{\mathsf{EncCred}}(w_{i^*}, \mathsf{s}_{i^*}) \\
\quad \texttt{return } (\tilde{\mathsf{s}}, z)
\end{array}
$$

$$
\begin{array}{l}
\texttt{Experiment } \mathsf{Expt}^{\mathsf{flt},\mathsf{cbs}}_{\eta,k}(\mathcal{A}) \\
\quad W \xleftarrow{\$} \mathsf{GenDetectSecrets}(k) \\
\quad w_{i^*} \leftarrow \mathsf{SelectRealSecret}(W,\eta) \\
\quad \hat{w} \leftarrow \mathcal{A}_{\mathcal{G},O_{\mathsf{cbs}}(w_{i^*})}(W) \\
\quad \texttt{if } \hat{w} = w_{i^*} \\
\qquad \texttt{then return } 1 \\
\quad \texttt{else return } 0
\end{array}
$$

We define the advantage of $\mathcal{A}$ as

$$\mathsf{Adv}^{\mathsf{flt,cbs}}_{\eta,k}(\mathcal{A}) \stackrel{\text{def}}{=} \mathbb{P}\left(\mathsf{Expt}^{\mathsf{flt},\mathcal{G}}_{\eta,k}(\mathcal{A})=1\right) - \frac{1}{k+1}$$
$$\mathsf{Adv}^{\mathsf{flt,cbs}}_{\eta,k} \stackrel{\text{def}}{=} \max_{\mathcal{A}}\{\mathsf{Adv}^{\mathsf{flt,cbs}}_{\eta,k}(\mathcal{A})\},$$

where the maximum is taken over all flatness adversaries $\mathcal{A}$.

Proposition C.1.

$$\mathsf{Adv}^{\mathsf{flt},\mathit{cbs}}_{\eta,k} = \mathsf{Adv}^{\mathsf{flt},\mathcal{G}}_{\eta,k}.$$

Proof. Given $\mathcal{A}$ for the experiment $\mathsf{Expt}^{\mathsf{flt,cbs}}_{\eta,k}$, we construct a decoy distinguisher $\mathcal{D}$ for the experiment $\mathsf{Expt}^{\mathsf{flt},\mathcal{G}}_{\eta,k}$ defined in Section 4. $\mathcal{D}$ provides $\mathcal{A}$ with W it receives from the experiment. $\mathcal{D}$ responds to $\mathcal{A}$'s $\mathcal{G}$ query by its own $\mathcal{G}$ query response. For each $\mathcal{O}_{\mathsf{cbs}}$ oracle query made by $\mathcal{A}$, $\mathcal{D}$ does the following:

- $\mathcal{D}$ chooses $\tilde{\mathsf{s}}'$ and z' from $\{0,1\}^{\kappa}$ uniformly at random.
- For all $w_i \in W$, $\mathcal{D}$ runs $\mathsf{s}'_i \leftarrow \Pi_{\mathsf{DecCred}}(w_i, \tilde{\mathsf{s}}', z')$ and check if s'_i is a valid passkey, i.e., $\mathsf{s}'_i \stackrel{?}{\in} \mathcal{S}$, where $\mathcal{S}$ is the passkey space.
  - If there exists $w_i \in W$ such that $\mathsf{s}'_i \notin \mathcal{S}$, $\mathcal{D}$ restarts the whole process by re-choosing a fresh $\tilde{\mathsf{s}}'$ uniformly at random.
  - If $\mathsf{s}'_i \in \mathcal{S}$ for all $w_i \in W$, $\mathcal{D}$ returns $(\tilde{\mathsf{s}}', z')$ to $\mathcal{A}$

Note that this process is efficient because it is almost unlikely that an invalid passkey will be produced (see Appendix D). Finally $\mathcal{D}$ outputs 1 if $\mathcal{A}$ outputs 1. Considering that, given a uniformly randomly chosen z', for each $w_i \in W$, there exists a corresponding s_i uniformly distributed in $\mathcal{S}$ such that $(\tilde{\mathsf{s}}', z') = \mathsf{KDF}(w_i, z') \oplus \mathsf{s}_i$, and so $(\tilde{\mathsf{s}}', z')$ and $(\tilde{\mathsf{s}}, z)$ are distributed identically for each $\mathcal{O}_{\mathsf{cbs}}$ oracle query simulated by $\mathcal{D}$, we have:

$$\mathbb{P}\left(\mathsf{Expt}^{\mathsf{flt},\mathcal{G}}_{\eta,k}(\mathcal{D})=1\right) \geq \mathbb{P}\left(\mathsf{Expt}^{\mathsf{flt,cbs}}_{\eta,k}(\mathcal{A})=1\right),$$
$$\mathsf{Adv}^{\mathsf{flt},\mathcal{G}}_{\eta,k}(\mathcal{D}) \geq \mathsf{Adv}^{\mathsf{flt,cbs}}_{\eta,k}(\mathcal{A}).$$

Also, we can construct a $\mathcal{A}$ for the experiment who runs a decoy distinguisher $\mathcal{D}$ for the experiment $\mathsf{Expt}^{\mathsf{flt},\mathcal{G}}_{\eta,k}$ as a subroutine. Upon receiving W from the experiment, $\mathcal{A}$ directly provides W as $\mathcal{D}$'s input and responds to $\mathcal{D}$'s $\mathcal{G}$ query by its own $\mathcal{G}$ query response. Finally $\mathcal{A}$ outputs 1 if $\mathcal{D}$ outputs 1 so:

$$\mathbb{P}\left(\mathsf{Expt}^{\mathsf{flt,cbs}}_{\eta,k}(\mathcal{A})=1\right) \geq \mathbb{P}\left(\mathsf{Expt}^{\mathsf{flt},\mathcal{G}}_{\eta,k}(\mathcal{D})=1\right),$$
$$\mathsf{Adv}^{\mathsf{flt,cbs}}_{\eta,k}(\mathcal{A}) \geq \mathsf{Adv}^{\mathsf{flt},\mathcal{G}}_{\eta,k}(\mathcal{D}).$$

□

D Well-Formed ECDSA Keys from Decrypting $\tilde{\mathsf{s}}$ with an Incorrect w

Here we show that decrypting $\tilde{\mathsf{s}}$ with an incorrect detection secret w, where $w \neq w_{i^*}$, almost certainly results in a well-formed private key for ECDSA. For elliptic curves supported by FIDO2 for ECDSA, e.g., secp256r1 and secp256k1, n is the order of the base point of the curve and is a large prime number. Specifically, n is equal to `0xFFFFFFFF00000000FFFFFFFFFFFFFFFFBCE6FAADA7179E84F3B9CAC2FC632551` and `0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEBAAEDCE6AF48A03BBFD25E8CD0364141` for secp256r1 and secp256k1 respectively [70]. Decrypting $\tilde{\mathsf{s}}$ with an incorrect w will yield s $(\neq \mathsf{s}_{i^*})$ that is uniformly distributed in the range of $[0, 2^{256}-1]$ (except for s_{i^*}), given w is uniformly sampled by GenDetectSecrets (see Figure 2). For s $(\neq \mathsf{s}_{i^*})$ to be a valid private key of ECDSA on secp256r1 or secp256k1, s needs to be within the range of $[1, n-1]$. Thus, the probability that decrypting $\tilde{\mathsf{s}}$ with an incorrect detection secret w results in a well-formed private key is $\frac{n-2}{2^{256}-1} > 0.999999999767$ for ECDSA on secp256r1 and $\frac{n-2}{2^{256}-1} > 0.999999999999$ on secp256k1. On another note, if decrypting $\tilde{\mathsf{s}}$ with an incorrect key is non-negligibly unlikely to produce a well-formed private key for a given signature scheme, an alternative design could be adopted, as mentioned in Section 5. Specifically, instead of producing and syncing a single $\tilde{\mathsf{s}}$, one could randomly generate k additional key pairs as decoys and synchronize $k+1$ private keys (passkeys) to the PMS directly.

E More about User Secret η

Here, we first explore alternative methods for instantiating the user secret η without imposing a memory burden on users. We then discuss potential privacy concerns related to η.

User memory-independent η. In practice, there are other desirable ways to instantiate η, making it user memory-independent but retrievable by users from physical objects (e.g., using credit cards CVVs as η retrievable from credit cards) or from third parties trusted for maintaining the secrecy and availability of the secrets (e.g., using several digits of bank account numbers as η retrievable from banks). Additionally, η can be derived from users' biometric data (e.g., fingerprints or faces) [16] . While such instantiation of η adds little memory burden on users and raises negligible privacy concerns as discussed next when w_{i^*} is stolen, the user interface should clearly explain how CASPER uses these secrets and, more importantly, that η will never be stored at any participating party in CASPER.

Privacy of η A `caat` who has compromised W and luckily guessed the correct w_{i^*} might be tempted to deduce η from W and w_{i^*}. However, this is in fact not easy because the modulo operation in SelectRealSecret restricts the information leakage about η to only $\log_2(k+1)$ bits of entropy. For example,

- Upon receiving a login request for uid, RP retrieves $\mathsf{v}(=\mathsf{s})$ for uid and requests an OTP, if the uid exists.
- User device runs $\mathsf{pwd} \leftarrow \mathsf{GenOTP}(\mathsf{s})$ and sends pwd to RP.
- Upon receiving pwd from the user, RP performs the following tests and actions:
  - `if` $\forall \mathsf{v} \in V : \mathsf{pwd} \neq \mathsf{GenOTP}(\mathsf{v})$: RP rejects this login request.
  - `else if` $\mathsf{v} \in V'$, RP raises a detection alarm.
  - `else` RP accepts this login request.

Figure 9: The compromise detection algorithm of CASPER for OTP. The BnR protocol remains the same for OTP.

considering the case where a user's η is the last four digits of their bank account number and k is set to 32, η would be hidden from the attacker among $10000 \times \frac{1}{32+1} \approx 303$ different 4-digit numbers that, if taken with W by SelectRealSecret as input, would yield the same w_{i^*}.

An alternative design for generating W given η. Recall that in Figure 2, we provided an example of generating W and determining the real secret index based on a user input η. Here, we introduce an alternative design for generating W that reduces the probability of a manual η input error leading to a false detection. The concept is straightforward: CASPER can assemble a set (denoted as N) of size k, which includes decoy user secrets randomly sampled from the same secret space of η. For example, when $k = 32$ and η is an x-digit system-generated PIN where $2 \leq x < \log_{10} 2^{\kappa}$, CASPER can randomly select 32 distinct PINs as the decoy user secrets from the remaining $10^x - 1$ (incorrect) PINs. CASPER then directly assembles W by including $\mathsf{Hash}(\eta)$ as the real detection secret and the hashes of the k decoy user secrets in N as decoys. This design provides better resistance against false detection triggered by manual η user input errors. In the x-digit PIN example here, a randomly entered incorrect η' will be mapped to a decoy passkey with a probability of at most $\frac{32}{10^x}$. This design is more suitable when η input errors are a larger concern than the privacy of η, as the `caat` can more easily determine η if it identifies the real detection secret $(= \mathsf{Hash}(\eta))$ within W.

F Extending CASPER for One Time Passwords

Besides passkeys, the concept of CASPER can also be easily extended to authentication methods based on other cryptographic credentials such as HMAC or time-based one-time passwords (OTP [63, 64]).

Background on OTP. Unlike FIDO2, OTP authentication is based on a shared secret between a user device (or authenticator) and an RP (the authentication server). During registration, the user device and the RP agree on a shared cryptographic seed s sampled uniformly at random. During the login phase, the user device generates a new OTP from s as $\mathsf{pwd} \leftarrow \mathsf{GenOTP}(\mathsf{s})$. Here GenOTP takes s as its input and outputs an OTP that is a truncated output of $\mathsf{HMAC}(\mathsf{s}, c)$ (as defined in [63, Sec. 5.2] and [64, Sec. 1.2]) where c is a counter used to maintain the freshness of a new OTP. Thus, c is updated either incrementally for each login (i.e., HMAC-based OTP [63]) or periodically based on the current time (i.e., timing-based OTP [64]). When the user submits pwd to the RP for authentication, the RP also generates an OTP pwd' by invoking GenOTP with the shared s and the same counter c, and verifies the user's identity by checking $\mathsf{pwd} \stackrel{?}{=} \mathsf{pwd}'$.

Extending CASPER for OTP. The BnR protocol for OTP is similar to the one for FIDO2 as shown in Figure 4. As shown in Figure 9, the compromise detection algorithm here is largely similar to the one used for FIDO2-based authentication, with only slight variations due to differences in their authentication flows. During a login, the user device generates an OTP via $\mathsf{pwd} \leftarrow \mathsf{GenOTP}(\mathsf{s}_{i^*})$, and sends the OTP pwd to the RP. Note that for OTP, there is no privacy and public key pair, and in a credential pair $\langle s, v \rangle$, $s = v$ because they are the same random seed shared and kept private by both the user device and RP. However, we keep the notations of s and v just to follow the specification of our generic BnR protocol. Therefore, to authenticate the login request, the RP will also generate an OTP pwd' on their own for each v in V from the list of verifiers , and check if $\mathsf{pwd}' = \mathsf{pwd}$. Similar to the detection algorithm for FIDO2 as described above, if there exists $\mathsf{v} \in V$ such that $\mathsf{pwd}' = \mathsf{pwd}$, then whether v is in the active verifier set V' or not determines if this login attempt will trigger a PMS detection or result in a successful login.

Handling spoofed false alert notifications from RP for OTP. Unlike FIDO2, detecting spoofed false alerts with a verifiable proof is challenging for OTP-based authentication methods, or, more broadly, authentication methods based on shared secrets between the user and the RP. This is because both parties can generate the same authentication response from their shared secret, making it difficult to determine whether a triggering authentication response is produced by a PMS breaching attacker or a misbehaving RP itself.

While future work can investigate further how to handle this challenge, for now, we propose an alternative approach that instead provides a *probabilistic* guarantee against such alerts. This approach requires the user device to select a subset of W at random, denoted by W', while still using W as the input of $\Pi_{\mathsf{GenVerifierSet}}$ to generate V for the RP. Meanwhile, CASPER requires the RP to send back the triggering OTP pwd as a probabilistic detection proof. With this setup, the probability of a RP producing a pwd with an OTP seed, s_i, that corresponds to a detection secret $w \in W \setminus W'$ is $1 - \frac{|W'|}{|W|}$. When this happens, the user can learn that the RP intentionally raised a false detection.