MFKDF:
Multiple Factors Knocked Down Flat

Matteo Scarlata [1], Matilda Backendal [1], and Miro Haller [2]

[1] *Department of Computer Science, ETH Zurich, Zurich, Switzerland*
[2] *Department of Computer Science & Engineering, University of California San Diego, USA*

Abstract

Nair and Song (USENIX 2023) introduce the concept of a Multi-Factor Key Derivation Function (MFKDF), along with constructions and a security analysis. MFKDF integrates dynamic authentication factors, such as HOTP and hardware tokens, into password-based key derivation. The aim is to improve the security of password-derived keys, which can then be used for encryption or as an alternative to multi-factor authentication. The authors claim an exponential security improvement compared to traditional password-based key derivation functions (PBKDF).

We show that the MFKDF constructions proposed by Nair and Song fall short of the stated security goals. Underspecified cryptographic primitives and the lack of integrity of the MFKDF state lead to several attacks, ranging from full key recovery when an HOTP factor is compromised, to bypassing factors entirely or severely reducing their entropy. We reflect on the different threat models of key-derivation and authentication, and conclude that MFKDF is always weaker than plain PBKDF and multi-factor authentication in each setting.

1 Introduction

With more and more data stored online or distributed across multiple devices, an increasing number of security-sensitive applications face the challenge of combining data availability with user-friendly key management. The traditional solution is passwords, for both authentication and key derivation. Password-based systems are relieved from some of the difficulties of key management, since users do not have to handle cryptographic keys. In return, allowing (and even encouraging) human-memorable secrets means that passwords often have low entropy, come from a small and predictable "dictionary" and may be highly correlated.

In response to a wide variety of attacks stemming from these issues, password-only authentication to web services is being phased out. Instead, users are offered a two step verification process, where they need to provide a second "factor" in addition to their password. This is known as Two-Factor Authentication (2FA), or more generally, Multi-Factor Authentication (MFA). Examples of such second factors include Time-Based and HMAC-Based One-Time-Passwords (T/HOTP), Out-Of-Band (OOB) authentication and codes generated by hardware tokens such as YubiKeys. Surveys show a sharp increase in the adoption of MFA among users [15], and cybersecurity advisory bodies such as CISA now advise the use of MFA in their recommended best practices [16].

In contrast, passwords are still commonly used as the sole source of entropy for key derivation, using Password-Based Key Derivation Functions (PBKDFs), in systems performing client-side encryption. Examples of this include full disk encryption, end-to-end encrypted cloud storage, password managers and cryptocurrency wallets. In all of these settings, keys derived from passwords remain as (in)secure as the passwords from which they stem.

MFKDF. In an attempt to remedy this, Nair and Song propose the concept of a "Multi-Factor Key Derivation Function" (MFKDF) as a way to integrate the advantages of multi-factor authentication into password-based key derivation [39]. MFKDF extracts entropy from existing authentication factors, such as T/HOTP, and combines them with the password before applying a PBKDF to derive a key. The goal is to derive a key that has the entropy of all the factors and password combined, and to retain security even against partial compromise of the source key material (i.e., the password and the authentication factors). The security guarantees claimed for MFKDF state that attackers should not be able to separately guess individual MFKDF factors, and that the difficulty of correctly guessing all factors simultaneously gives an "exponential security improvement over password-based key derivation" [39].

Nair and Song (NS) propose to use MFKDF-derived keys directly in other applications requiring a symmetric key, such as database encryption in a password manager. We will refer to this setting as the *key derivation* setting. Additionally, the authors state that MFKDF-derived keys can be used to authenticate users, as a replacement for MFA, since an MFKDF

key implicitly verifies the user's authentication factors. We will refer to this setting as the *authentication* setting.

The concept of MFKDF was presented at USENIX 2023 [39], and is accompanied by a Javascript reference implementation [36].

1.1 Our Contributions

We analyzed MFKDF and discovered several issues, ranging from a fundamental problem with the idea of using authentication factors for key derivation (Attack 3.1), to vulnerabilities in the proposed constructions (Attacks in 3.2–3.3). We briefly outline the attacks below.

1. The **Dynamic Factor Attack** (3.1) surfaces a critical mismatch between the goals of key derivation and authentication, and shows that the use of the most common authentication factors is unsuitable for key derivation. By construction, key derivation functions are deterministic; they need to always re-derive the same key on the same input. Hence, MFKDF extracts *static* key material from so called *dynamic* authentication factors, such as T/HOTP. This attack shows that an adversary can recompute this static key material from a single one-time code from the authentication factor, thereby permanently degrading the security of MFKDF. This is in sharp conflict with the original security models of TOTP and HOTP.

The threat models for authentication and key derivation allow for different kinds of leakage of secret values. In particular, "one-timeness" of authentication codes is not reflected in the confidentiality requirement of KDF inputs. So while the idea of using authentication factors to strengthen password-derived keys is praiseworthy, our first attack shows that—because of these disparate threat models—the use of existing authentication factors for key derivation is problematic.

We present many other attacks, showcasing vulnerabilities in the MFKDF construction that undermine the claimed security guarantees. These attacks fall into two main categories:

2. The **State Integrity Attacks** (3.2) leverage the fact that a malicious server can modify the MFKDF state: this allows the adversary to control the output of the key derivation, to bypass factors, and even to recover derived keys when users interact with the MFKDF construction.

3. The **Specification Attacks** (3.3) exploit incorrect or missing assumptions in the specification of MFKDF. In particular, we show that MFKDF as currently built from H/TOTP is insecure. Additionally, MFKDF relies on properties of symmetric cryptography that do not follow from traditional security notions such as IND-CPA, and incorrectly assume information theoretical security of the many-time pad. These mistakes, and other, invalidate the security theorems presented by NS.

Most of the above attacks impact not only the formal specification of MFKDF, but also the reference implementation. This highlights how mistakes at the specification level transform into practically exploitable vulnerabilities in code.

After presenting the attacks, we propose countermeasures for the issues which can be mitigated, and draw wider conclusions from the remaining fundamental problems. We argue that the proposed combination of MFA and PBKDF fails to recognize the distinct threat models for authentication and key derivation, and is therefore irrecoverably flawed.

In the authentication setting, we show that MFKDF is always weaker than corresponding traditional MFA choices. Additionally, we believe that the statement that MFKDF can be implemented with "no noticeable change to the user experience" may lead to careless handling of one-time codes, which are much more security critical when used for MFKDF than for traditional MFA. We hence caution against the use of MFKDF in this setting.

In the key derivation setting, our attacks show that the usage of MFKDF can annul the security of a strong password. We propose mitigations to these issues, but show that even after patching the vulnerabilities, MFKDF does not provide a significant advantage over a naïve PBKDF construction which derives the key from the concatenation of multiple static secrets.

1.2 Related Work

MFA and Key Derivation. The state of the art in multi-factor authentication includes WebAuthn [22], a part of the FIDO2 Project [3]. WebAuthn enables completely passwordless authentication, thus side-stepping the problem of user passwords. WebAuthn credentials, also known as Passkeys [13], leverage public-key cryptography for authentication, and can either be bound to a hardware authenticator (such as a YubiKey [20]), or synchronized across user devices.

For local authentication and key derivation, modern approaches include relying on secure hardware to achieve higher security guarantees. Examples include the use of Trusted Platform Modules (TPMs) [23, 33] for disk encryption and smartphones leveraging secure chips [23,50] to defend against bruteforce attacks on PINs and passcodes.

In the online setting, Oblivious Pseudo-Random Functions (OPRFs) [14, 18, 25, 41] let a client derive a key as a function of its password and a server secret, without the server learning the password or the derived key. Combining the entropy of the password with the server secret increases the cost of offline dictionary attacks on the password for external adversaries, making OPRFs a promising alternative to PBKDFs. However, against an adversary that knows the server secret, the key is no stronger than the password from which it is derived.

KeeChallenge and OtpKeyProv. KeePass [45] is a password manager that supports authentication factors, such as

HOTP and HMAC challenge-response authentication [52], in addition to a password to encrypt the application database. This integration is implemented through two plugins, KeeChallenge [47] and OtpKeyProv [46], which both predate MFKDF [30]. While an in-depth analysis of the security of these plugins is outside of the scope of this paper, we note that these plugins also suffer from some of the same problems of MFKDF; namely, the lack of ephemerality of one-time codes and the HOTP bias.

MF*. MFKDF also appears in other works by NS. The custodial wallet design [37] is based on MFKDF and inherits its vulnerabilities. The MFCHF credential hashing function [38] is vulnerable to our attacks on factor constructions.

Attacks. There is a flourishing variety of attacks on cryptosystems in the wild. Key-overwriting attacks in the style of our OOB attack began with Klima and Rosa [27], and have seen a recent resurgence in modern cryptanalysis [1, 6, 12]. Lack of integrity in metadata has been exploited in practice in recent attacks in messaging applications [2, 42], but its perils have been long known by the community [43].

Non-uniform distributions were exploited in the contexts of biased ECDSA nonces [11], randomness failures in RSA key generation [21], and RC4 [31]. While RFC 4226 [34, A.4] discusses why HOTP biases are negligible in the authentication setting, but to the best of our knowledge, we are the first to construct and evaluate an attack that exploits the bias when HOTP is used for encryption.

PBKDF constructions have been analyzed before [4, 51], but MFKDF aspires to be an entirely novel primitive, warranting its own analysis.

1.3 Ethical Considerations

We disclosed our findings to Nair and Song (NS) on 01/17/2024 and they acknowledged receiving them on 01/22/2024. While NS argue that the underspecification exploited by some of our attacks is intentional for "flexibility and future-proofness" of MFKDF, they confirmed the vulnerabilities in their implementation, as well as that Attack 3.1 is unavoidable. We discussed the mitigations from Section 4 with NS. They intend to modify the MFKDF specifications to highlight the importance of protecting the integrity of the MFKDF state and the use of secure encryption primitives. They clarified that the MFKDF implementation is intended as a proof of concept; they are unaware of any deployment of their code, and they would implement any necessary changes prior to use in any production application.

1.4 Paper Structure

We start by defining a formal syntax for the MFKDF constructions and primitives in Section 2, adapting it slightly from the syntax presented in [39] for the sake of clarity.[1] Section 3 presents our attacks against the MFKDF construction and reference implementation. Section 4 follows with the possible mitigations. Finally, Section 5 discusses the inherent flaws in MFA-based key derivation functions, and the different threat models of key derivation and multi-factor authentication. We conclude and propose future directions in Section 6.

2 Background

Notation and Conventions. We denote assigning value x to variable a by $a \leftarrow x$, where x might be the output of an algorithm. Similarly, $a \leftarrow\!\!\$\, \mathcal{X}$ denotes sampling uniformly at random from the set $\mathcal{X}$ and assigning the result to a. We write $[0, 10^6)$ for the right-open interval of all integers from 0 (inclusive) to 10^6 (exclusive). The symbol $\|$ denotes string concatenation and $\oplus$ denotes XOR. For an encryption scheme $(\mathsf{Enc}, \mathsf{Dec})$, $\mathsf{Enc}(k, m)$ denotes the encryption of message m under key k and $\mathsf{Dec}(k, c)$ decryption of ciphertext c.

2.1 Authentication Factors

Following [39], we define a *factor*, F, to be a two-valued probability distribution (σ, α) generated by an efficient probabilistic algorithm, which we also refer to as F. The *factor material* σ represents the secret part of the factor (for example, a password, or a key for the HOTP algorithm) whereas the *factor parameters* α represent the public knowledge about σ. For example, if the factor material σ is a password, then α could be the distribution from which it is sampled, as well as other information about the user which might have influenced their choice of password. (Note the analogy to a *source of keying material*, as defined in e.g. [29].)

An authentication factor additionally specifies a protocol between a prover and a verifier, consisting of a *witness generation* algorithm and a *verification* algorithm. The witness generation algorithm, run by the prover, takes the factor material σ and some optional additional input *inp* to produce a witness W. Depending on the protocol, the optional input *inp* may be generated by the verifier and sent to the prover (e.g., in challenge-response protocols), or be part of the state kept by both parties (e.g., a counter). The verification algorithm takes as input σ, *inp* (if present) and W, and checks if W is a valid witness.

As an example, in the case of passwords the witness W is often the password σ itself. For T/HOTP, W is a one-time password, and *inp* the current timestamp or counter value used to generate it. The secret factor material σ is assumed to be shared (for example, during a setup phase) between the prover and the verifier, such that at the time of authentication, only the witness must be communicated. We say that an

[1] Note that none of our attacks are affected by this change in syntax; we deviate from the original MFKDF description only in the interest of clearness and internal consistency.

authentication factor is *dynamic* if the witness changes with every use of the factor, and learning W does not allow an adversary to compute σ.

Below, we review the dynamic factors used in MFKDF.

HOTP and TOTP. The HOTP algorithm, defined in RFC4226 [34] and shown in Algorithm 1, takes as input a static symmetric key k_h (the secret factor material σ) and a counter ctr (as the witness input inp). It first generates a 20-byte digest hs by computing HMAC on the key and counter, then truncates the digest to 31 bits using a dynamic truncation algorithm that we call DynTrunc[2]. Finally the truncated digest is transformed to an integer using the function BitsToInt, which on input a bitstring $b_n b_{n-1} \dots b_0$ returns the integer $s_{int} = \sum_{i=0}^{n} b_i \cdot 2^i$. The reduction of s_{int} to d digits is the resulting one-time password $\mathsf{HOTP}(k_h, ctr)$, which constitutes the witness W.

Algorithm 1 HMAC-based One-Time Password

1: **procedure** $\mathsf{HOTP}(k_h, ctr)$ ▷ Witness generation
2: $hs \leftarrow \mathsf{HMAC\text{-}SHA\text{-}1}(k_h, ctr)$
3: $s_{bits} \leftarrow \mathsf{DynTrunc}(hs)$ ▷ $|s_{bits}| = 31$ bits
4: $s_{int} \leftarrow \mathsf{BitsToInt}(s_{bits})$
5: **return** $s_{int} \bmod 10^d$ ▷ Witness W

The verification algorithm simply recomputes the one-time password and checks that $\mathsf{HOTP}(k_h, ctr) = W$.

TOTP [35] is defined as $\mathsf{TOTP} := \mathsf{HOTP}(k_h, T)$[3] where the counter ctr is replaced with T, a coarse-grained integer representation of the current time. Note that for both HOTP and TOTP, the PRF security of HMAC [7, 8] implies that access to old one-time codes does not allow an attacker to produce new ones or recover k_h, making these factors dynamic.

Out-Of-Band Codes. A popular non-cryptographic way to implement a dynamic factor is to send one-time codes over an "out of band" channel (e.g., SMS or email). In this case, there is no shared secret material, i.e., σ is empty. Instead, the server generates a random one-time code as the witness input inp, and sends it to the user over the OOB channel. The user then sends this value back to the server over the main channel. (Hence, the witness is directly the input inp.) The server verifies that $inp = W$, i.e., that the user sent back the same code that the server generated. This kind of factor lacks cryptographic guarantees, and assumes the OOB channel to be confidential and exclusive to the user.

HMAC Hardware Tokens. Some hardware tokens such as YubiKeys offer a HMAC-SHA-1 evaluation for authentication: this is equivalent to HOTP, except that the input to the witness generation algorithm is an arbitrary challenge rather than a counter, and the output is the full-length HMAC tag.

[2]This algorithm is defined on page 7 of RFC4226 [34], but the details do not matter for our purposes.

[3]Additionally, TOTP may use a different hash function than HOTP.

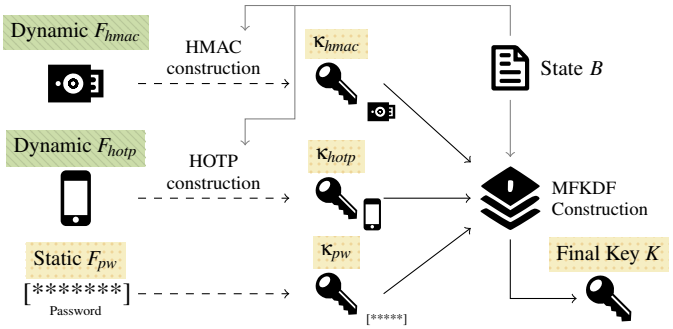

Figure 1: MFKDF combining authentication factors (F_{hotp} and F_{hmac}) with a password to derive a key. Dynamic factors are colored green and striped, static ones yellow and dotted.

2.2 MFKDF

The core idea of MFKDF is to combine MFA with PBKDF. That is, the goal is to use some combination of authentication factors to derive a static symmetric key. The MFKDF paradigm consists of two parts: (1) a factor construction part, which takes the factor material from an authentication factor and turns it into *source key material*, and (2) the KDF part, in which the source key material from multiple authentication factors is turned into a key. See Figure 1 for an illustration.

In the following, we briefly describe the syntax of a factor construction, and the constructions proposed by NS for some dynamic factors. Next, in Section 2.2.2, we review the syntax of a key derivation function and describe how factor constructions are combined to build MFKDF.

2.2.1 Factor Constructions

A factor construction for a factor F consists of algorithms Setup and Derive. Via $(\kappa_F, \beta_0) \leftarrow \mathsf{Setup}(\sigma)$, the randomized setup algorithm on input the factor material σ "constructs" the static source key material κ_F and initial state β_0.[4] Then, the randomized and stateful algorithm Derive on input a witness W_i for F and (public) state β_i, enables continued evaluation by reconstructing the key material κ_F and producing an updated state β_{i+1}. That is, $(\kappa_F, \beta_{i+1}) \leftarrow \mathsf{Derive}(W_i, \beta_i)$.

For dynamic factors, the witnesses change over time, making it non-trivial to derive the same static κ_F from every witness. NS solve this by storing the secret factor material σ in β, thereby allowing future witnesses to be computed within algorithm Derive and the changes offset by updates to the state. To protect σ, it is encrypted under the MFKDF-derived key K. This establishes a "feedback mechanism" where K is

[4]Note that in [39], NS refer to the source key material κ_F as "factor material" and the public parameters β as "factor parameters" (and denote them σ and α, respectively). However, not all factor constructions use the factor material σ as the source key material κ_F, (for example, the HOTP construction uses a fresh static secret as κ_F, not the HMAC key $\sigma = k_h$). Hence, we choose to separate the notation to avoid confusion.

first derived from all factors, and then used to encrypt the factor materials. In order for the feedback mechanism to work, the setup and derive algorithms must be run in parallel for all factors used in MFKDF, and intermediate outputs from each factor construction influence the computations of others. To model this, we deviate from the syntax presented in [39] and split algorithms Setup and Derive into two stages each: Setup_1, Setup_2, Derive_1, and Derive_2. Stage one reconstructs κ_F, which is combined with other factors' key material to derive K. Stage two uses K and κ_F to create β_{i+1} for the next iteration. We thus define a factor construction as follows.

Definition 1 (Factor Construction)**.** A factor construction for factor F with factor material σ is a tuple of algorithms

$$\begin{aligned}\kappa_F &\leftarrow \mathsf{Setup}_1(\sigma),\\ \beta_0 &\leftarrow \mathsf{Setup}_2(K,\sigma,\kappa_F),\\ \kappa_F &\leftarrow \mathsf{Derive}_1(W_i,\beta_i), \text{ and}\\ \beta_{i+1} &\leftarrow \mathsf{Derive}_2(K,\kappa_F,\beta_i),\end{aligned}$$

where K is the final key output by MFKDF (see Section 2.2.2).

For factors that do not employ the feedback mechanism, we will simply refer to the combined algorithms $(\kappa_F,\beta_0) \leftarrow \mathsf{Setup}(\sigma)$ and $(\kappa_F,\beta_{i+1}) \leftarrow \mathsf{Derive}(W_i,\beta_i)$.

In the following paragraphs, we review the factor constructions proposed by NS in [39] relevant to our attacks.

HOTP Factor Construction. Recall that for an HOTP factor F_{hotp}, the private factor material σ is the HMAC key k_h, and the witness is a one-time password $W_i = \mathsf{HOTP}(k_h,i)$, where i is a counter. In the factor construction for MFKDF, shown in Algorithm 2, the source key material κ_{hotp} consists of a d-digit integer (called the "target" in [39]), which is sampled uniformly at random from $[0,10^d)$.[5] κ_{hotp} is encrypted in a stream-cipher fashion by subtracting modulo 10^d the next one-time password W_{i+1}, generating a ciphertext $c_{\kappa,i+1}$ which is stored in the factor state β_{i+1}. This allows κ_{hotp} to be (re-)derived by adding the witness back to $c_{\kappa,i+1}$.

To access the next witness W_{i+1} used to encrypt κ_{hotp}, NS use the feedback mechanism described above: the HMAC key k_h is stored encrypted in the state using the final MFKDF key K. NS do not further specify the encryption scheme used in their HOTP (and TOTP) factor construction. In particular, they do not demand that the scheme should be randomized, nor do they define the expected security properties. As we will see later, this under-specification makes the HOTP (and TOTP) factor construction of MFKDF vulnerable to attacks.

This feedback mechanism adds an element of circular security to the factor construction: the source key material κ_{hotp} is encrypted with the witnesses derived from k_h, and k_h itself is encrypted under the key K derived from the source key material. This is true for all factor constructions proposed by NS that make use of the feedback mechanism. A formal proof of security of MFKDF would hence be limited to what can be achieved in this setting of key-dependent messages [10].

[5] In practice, $d = 6$ is commonly used for HOTP tokens.

Algorithm 2 HOTP Factor Construction for HOTP key k_h

procedure $\mathsf{Setup}_1(k_h)$
  $\kappa_{hotp} \leftarrow\!\!\$\ [0,10^d)$
  return κ_{hotp}
procedure $\mathsf{Setup}_2(K,k_h,\kappa_{hotp})$
  $W_0 \leftarrow \mathsf{HOTP}(k_h,0)$
  $c_{\kappa,0} \leftarrow (\kappa_{hotp} - W_0) \bmod 10^d$
  $c_\sigma \leftarrow \mathsf{SK.Enc}(K,k_h)$
  return $(0,c_{\kappa,0},c_\sigma)$
procedure $\mathsf{Derive}_1(W_i,\beta_i)$
  $(i,c_{\kappa,i},c_\sigma) \leftarrow \beta_i$
  $\kappa_{hotp} \leftarrow (c_{\kappa,i} + W_i) \bmod 10^d$
  return κ_{hotp}
procedure $\mathsf{Derive}_2(K,\kappa_{hotp},\beta_i)$
  $(i,c_{\kappa,i},c_\sigma) \leftarrow \beta_i$
  $k_h \leftarrow \mathsf{SK.Dec}(K,c_\sigma)$
  $W_{i+1} \leftarrow \mathsf{HOTP}(k_h,i+1)$
  $c_{\kappa,i+1} \leftarrow (\kappa_{hotp} - W_{i+1}) \bmod 10^d$
  return $(i+1,c_{\kappa,i+1},c_\sigma)$

TOTP Factor Construction. The TOTP factor construction is analogous to the HOTP construction, except that the ciphertext $c_{\kappa,i}$ is replaced by a vector $\overline{c_{\kappa,i}}$ which contains the encryption of κ_{hotp} under all OTP witnesses for a window of future times, such that decryption is possible using any witness generated during a time in the window. For details, see [39, Appendix A].

HMAC Factor Construction. This is similar in spirit to the HOTP construction, but this time the HMAC key k_h is directly used as the source key material κ_{hmac}. Challenges are random 160-bit values, and the witnesses W_i from the authenticator are used as one-time pads to encrypt κ_{hmac}. Refer to Appendix A for a full pseudocode description.

OOB Factor Construction. Recall that OOB authentication factors rely on a secure channel between the verifier and the prover, such that the verifier can send a one-time password (OTP) to the prover when they want to authenticate. For MFKDF, NS assume a particular instantiation of this channel, based on S/MIME: the users owns a key pair (sk,pk) for an associated public-key encryption scheme $(\mathsf{PK.Enc},\mathsf{PK.Dec})$. The key material κ_{oob} is sampled uniformly at random in $[0,10^6)$, and encrypted in a stream-cipher fashion (like in the HOTP construction) with a random OTP $\in [0,10^6)$ generated by the user. Finally, the user encrypts the OTP to themselves and sends it to the server for storage. At factor derivation time, the user can fetch the OTP and decrypt it. Refer to Appendix A for the pseudocode.

2.2.2 Combining Factors

Here we review how MFKDF uses the source key material from the factor constructions to derive a key. NS present two variants of MFKDF: a plain one, where all the factors are necessary to derive the final MFKDF key K, and a threshold variant, where K can be derived using only t-out-of-n factors. We focus here on the threshold construction in order to provide the necessary background for Attacks 3.2.2 and 3.3.4. The details of the plain construction are given in Appendix B.

The threshold MFKDF presented in [39, App. A] is built from a PBKDF and a Shamir secret sharing scheme [48]. We review the syntax of both as used in MFKDF.

Definition 2 (Password-Based Key Derivation Function)**.** A key derivation function $\mathsf{PBKDF}(\kappa, salt, \ell) \mapsto K$ takes as input source key material κ, a salt $salt \in \{0,1\}^*$ and a length $\ell \in \mathbb{N}$ to produce a key $K \in \{0,1\}^\ell$.

Definition 3 (Shamir Secret Sharing)**.** A Shamir secret sharing scheme consists of algorithms

$$\begin{aligned}(s_1,\ldots,s_n) &\leftarrow \mathsf{Share}(\kappa,t,n),\\ \kappa &\leftarrow \mathsf{Comb}((s_{u_1},\ldots,s_{u_t}),t,n), \text{ and}\\ s_j &\leftarrow \mathsf{Rec}(\kappa,t,n,j).\end{aligned}$$

Share splits secret κ into n shares, out of which t suffice to reconstruct the secret through algorithm Comb. We refer to [48] for the correctness and security properties. Algorithm Rec is introduced by Nair and Song, who do not describe its properties explicitly. From its use, we deduce that given the secret κ and the sharing parameters, it recovers the j^{th} share.

Similarly to factor constructions, both the plain and threshold MFKDF schemes define algorithms Setup and Derive. Threshold MFKDF additionally defines an algorithm Recover, which is used in case a factor is lost or needs to be replaced.

Definition 4 (Threshold MFKDF Construction)**.** Let $F_1, F_2, \ldots, F_n$ be authentication factors and let $\overline{\sigma} = (\sigma_{F_1}, \sigma_{F_2}, \ldots, \sigma_{F_n})$ be a vector with the corresponding factor materials. The t-out-of-n threshold MFKDF construction for factors $F_1, F_2, \ldots, F_n$ consists of algorithms

$$\begin{aligned}(K, B_0) &\leftarrow \mathsf{Setup}(\overline{\sigma},t,\ell),\\ (K, B_{i+1}) &\leftarrow \mathsf{Derive}(\overline{W}_i, B_i), \text{ and}\\ B_{i+1} &\leftarrow \mathsf{Recover}(\overline{W}_i, B_i, x, \sigma_{F'_x}).\end{aligned}$$

Algorithm Setup takes as input $\overline{\sigma}$, the threshold parameter t and $\ell \in \mathbb{N}$. It returns the derived key $K \in \{0,1\}^\ell$ and the initial public state B_0. Derive takes as input a vector of witnesses for a subset of t factors $\overline{W}_i = (W_{i,u_1}, \ldots, W_{i,u_t})$ and the i^{th} public state B_i (containing the state of all n factors). It returns the key K and the updated state. Algorithm Recover takes as input $\overline{W}_i$, the public state B_i, the index x of the factor F_x to be replaced, and the new factor material $\sigma_{F'_x}$ of factor F'_x replacing it.

Appendix B contains the full specification of the algorithms. Here, we recall a high-level overview from [39].

$\mathsf{Setup}((\sigma_{F_1}, \ldots, \sigma_{F_n}), t, \ell)$

1. Sample κ and $salt$ randomly from $\{0,1\}^\ell$.
2. Split κ into n shares using $(s_1, \ldots, s_n) \leftarrow \mathsf{Share}(\kappa, t, n)$.
3. For each factor F_j, compute the key material κ_j using algorithm F_j.Setup associated to the factor construction.
4. Expand each κ_j to pad_j of length ℓ using HKDF-Expand.
5. Encrypt each share s_j with the pad derived from the expanded factor via $c_{s_j} \leftarrow s_j \oplus \text{pad}_j$.
6. Run $K \leftarrow \mathsf{PBKDF}(\kappa, salt, \ell)$ to derive $K \in \{0,1\}^\ell$.
7. Compute $\beta_{0,j}$ for each factor (using K for the factors that use the feedback mechanism).
8. Let $B_0 \leftarrow (t, \ell, salt, (c_{s_1}, \ldots, c_{s_n}), (\beta_{0,1}, \ldots, \beta_{0,n}))$.
9. Return K and the public state B_0.

$\mathsf{Derive}((W_{i,u_1}, \ldots, W_{i,u_t}), B_i)$

1. For each of the t witnesses, derive the factor key material κ_{u_j} from witness W_{i,u_j} using algorithm F_{u_j}.Derive.
2. Expand each κ_{u_j} to pad_{u_j} of length ℓ using HKDF-Expand.
3. Decrypt share s_{u_j} via $s_{u_j} \leftarrow c_{s_{u_j}} \oplus \text{pad}_{u_j}$.
4. Combine the shares into κ using $\mathsf{Comb}((s_{u_1}, \ldots, s_{u_t}), t, n)$.
5. Use PBKDF on κ to produce K.
6. Compute β_{i+1,u_j} for each factor (using K for the factors that use the feedback mechanism) and update B_{i+1}.
7. Return K and the updated public state B_{i+1}.

$\mathsf{Recover}(\overline{W}_i, B_i, x, \sigma_{F'_x})$

1. Compute κ as in steps 1–4 of Derive.
2. Set up the new factor F'_x, computing κ'_x and pad'_x as in steps 3 and 4 of Setup.
3. Recover the share of the lost factor via $s_x \leftarrow \mathsf{Rec}(\kappa, t, n, x)$.
4. Update c_{s_x} to $c'_{s_x} \leftarrow s_x \oplus \text{pad}'_x$.
5. Compute the inital state for F'_x and update the state of each of the t factors used in the recovery.
6. Update B_{i+1} with the new states and c_{s_x} and return it.

3 Attacks

In this section, we present our attacks against the MFKDF construction and its reference implementation (release v1.4.7) [36]. Table 1 provides an overview of our attacks, categorized by fundamental (Section 3.1), attacking unauthenticated state (Section 3.2) and exploiting underspecifications and bad assumptions (Section 3.3). It also shows prerequisites, impact, and if the attack can be mitigated.

Under impact, we highlight the security goals of MFKDF contradicted by our attacks. In particular, all attacks break the "Entropy" goal [39, §3]: *"Attacking the derived key should be as hard as attacking the weakest set of allowed factors."* Some attacks additionally contradict "Safety": *"Providing an invalid set of factors should be highly unlikely to derive the*

Table 1: Summary of the attacks, listing the type of attack (**fund**amental, state **int**egrity, **impl**ementation-related, or caused by bad assumptions or insufficient detail the **spec**ification), and for each attack its prerequisites (*read/write* access to the MFKDF state B; *leak* of a witness W_i or the factor material σ; *user actions* such as key derivation and replacing a factor), and its impact (recover the final key K, the source key material κ_F, or violate security theorems) and whether it can be mitigated (from 'no mitigations possible' ○, to 'mitigations unclear' ◑, and 'mitigations provided' ●).

		Read State	Write State	Leak One Factor Witness	Leak Factor Material	User Action	Recover Final Key	Recover Src. Key Material	Break *'Sdfety'*	Break *'Entropy'*	Break Exp. Security	
Dynamic Factor (3.1)	fund	✓	✗	✓	✗	✗	✗	✓	✗	✓	✓[c]	○
OOB Overwriting (3.2.1)	int	✓	✓	✗	✗	✓	✗	✓	✗	✓	✓[c]	◑
Parameter Tampering (3.2.2)	int	✓	✓	✗	✗	✓	✓	✗	✗	✓	✗	◑
Share Dilution (3.2.3)	int, impl	✓	✓	✗	✗	✓	✓[a]	✗	✓	✓	✗	◑
HOTP Compromise (3.3.1)	spec, impl	✓	✗	✗	✓	✗	✓	–	✗	✓	✓	◑
HOTP Bias (3.3.2)	spec	✓	✗	✗	✗	✗	✗	✓	✗	✓	✓[c]	●
Share Recovery (3.3.3)	spec	✓	✗	✓[b]	✗	✓	✗	✓	✓	✓	✓[c]	●
Share Format (3.3.4)	spec	✓	✗	✗	✗	✗	✓[a]	✓	✗	✓	✓	●
	Type	Prerequisites					Impact					Mitigations

[a] Depending on the minimum entropy of the remaining factors in the target MFKDF construction.
[b] Witness of a replaced factor. No witness needed if the factor has low entropy.
[c] As a consequence of recovering the source key material for the target factor.

correct key." [39, §3]: Together, they invalidate the MFKDF security theorem [40, App. B, Thm 4].

We also falsify the claim of "exponential security" by showing attacks that recover the final key or the source key material of factor constructions: *"[A]ttackers cannot separately guess individual MFKDF factors, and must simultaneously correctly guess all factors to derive a key. [...] Therefore, given n available factors with a mean factor entropy of 2^{-m} bits, the crack time t will be $t \propto 2^m$ for PBKDFs and $t \propto 2^{mn}$ for MFKDF. We thus claim that n-factor MFKDF provides an exponential security improvement over PBKDFs. [39, § 6]"* Many of our attacks additionally lead to practically exploitable vulnerabilities in the reference implementation of MFKDF.

3.1 Dynamic Factor Attack

The H/TOTP, HMAC and OOB factor constructions convert dynamic authentication factors into static source key material. We show that these constructions inherently degrade the security compared to what is expected of a dynamic factor: namely, that access to an old witness does not compromise the security of the factor. When used for MFKDF, leakage of a single witness to an adversary with access to the public factor state instead compromises the factor permanently. As we discuss in Section 5, this represents a fundamental issue with MFKDF, that cannot be mitigated.

Setting. The attack assumes an adversary with access to an authentication witness for the dynamic factor. The witness does not have to be "fresh", in the sense that it would be valid if the factor was used for authentication. For example, an already used or outdated H/TOTP or OOB code suffices.

This threat model matches the standard expectation that "one-time" codes are ephemeral, in the sense that leakage of an old code does not compromise the security of the factor. It is also formally supported by the threat model for HOTP described in RFC 4226 [34]. As a real-world example, an adversary that captures a one-time code as it is being used (e.g., by looking over the shoulder of the victim as they type it in, or through a key-logger on a public computer) can mount this attack, even after the code has been used or expired.

Additionally, the attack assumes read access to the MFKDF state. Recall from Line 8 of the Setup algorithm of Definition 4, that the MFKDF state contains all factor construction states. As described in [39], this state does not have to be kept confidential, and could even "be stored on a public blockchain" [39, p. 4]. This threat model is further supported by the proposed deployments of MFKDF as an online password manager [39, §10.1] or a decentralized cryptocurrency wallet [39, §10.2], where the state is outsourced by the client and available to the adversary.

Attack Overview. Recall from Figure 1 that the source key material κ_F associated to a factor construction F is derived from the i-th witness W_i, and the public factor state β_i via

$$\kappa_F \leftarrow \mathsf{Derive}_1(W_i, \beta_i) \; .$$

This means that an adversary with access to the public state β_i can recover κ_F as soon as a witness is observed, thereby reducing the entropy of the final MFKDF key by the amount provided by factor F. Furthermore, since the adversary can record all past public states $\beta_0, \ldots, \beta_i$, κ_F can be compromised as soon as any previous witness is leaked. As stated above, this removes the security guarantee of dynamic factors which is normally provided by the ephemerality of the witnesses.

Impact. The consequence of this attack is that dynamic factors are no more secure than static factors with the same entropy. For instance, a 6 digit HOTP code from a software authenticator is no stronger than a six digit PIN which the user could store in a password manager or memorize. More troublesome, the gap between the expected security of one-time codes (that is, that exposure of old codes is harmless) likely leads to a false sense of security in users, who, unaware of the implications for MFKDF, might not protect their old one-time codes as they would a PIN or a password, thereby inadvertently risking the security of the source key material. In conclusion: in MFKDF, one-time codes are not one-time.

3.2 State Integrity Attacks

Beside the inherent issues with using dynamic authentication factors, the public state B of MFKDF provides a large attack surface. This state is accessed and updated in both the plain and threshold constructions (see algorithm Derive in Definition 4 and 5). In [39], NS do not consider whether the state needs to be authenticated or protected in any way. We show that the lack of integrity of the state leads to practical attacks against MFKDF. In Section 4, we argue that mitigations to this problem are non-trivial, because of its circularity: the integrity of the state should be verified prior to its usage, but before using the state, an MFKDF user has no key that can be used for this verification.

The attacks presented here assume an adversary with read and write access to the MFKDF construction state. This is the standard threat model for end-to-end encrypted systems where users outsource the public state to the server. For instance, if MFKDF is used in an online password manager with an untrusted server, as proposed by NS, the service provider or an adversary controlling the server could perform this attack.

3.2.1 OOB Overwriting Attack

This attack exploits that users fetch their S/MIME public key from the unauthenticated MFKDF state. Due to the lack of integrity of the outsourced key, an attacker with access to the public state can overwrite the user's public key pk with a malicious key, such that the user encrypts their next OOB witness to the adversary. This allows the attacker to recover the source key material κ_{oob}, since it is stored in the public state "encrypted" under the OOB witness.

Attack Overview. Let $\beta_i = (pk, c_{\kappa,i}, c_{W,i})$ be the (honest) OOB factor state after the i-th use of the OOB factor, consisting of the user's public key pk, the encrypted key material $c_{\kappa,i}$ and the encryption of the current one-time code $c_{W,i}$.

The attacker overwrites the state with $\beta_i^* = (pk^*, c_{\kappa,i}, c_{W,i})$, where pk^* is the public key of an S/MIME key pair for which the adversary knows the corresponding private key sk^*. Upon the next use of the OOB factor, the user fetches β_i^* from the state and runs:

$\underline{\mathsf{Derive}(W_i, \beta_i^*)}$
$(pk^*, c_{\kappa,i}, c_{W,i}) \leftarrow \beta_i$
$\kappa_{oob} \leftarrow (c_{\kappa,i} + W_i) \bmod 10^6$
$W_{i+1} \leftarrow\!\!\$\ [0, 10^6)$
$c_{\kappa,i+1} \leftarrow (\kappa_{oob} - W_{i+1}) \bmod 10^6$
$c^*_{W,i+1} \leftarrow \mathsf{PK.Enc}(pk^*, W_{i+1})$
Return $(\kappa_{oob}, (pk^*, c_{\kappa,i+1}, c^*_{W,i+1}))$

In particular, the key pk^* of the adversary is used to encrypt the next OOB code W_{i+1}. Since the attacker has access to the updated public state $\beta_{i+1} = (pk^*, c_{\kappa,i+1}, c^*_{W,i+1})$, it can recover the code as $W_{i+1} \leftarrow \mathsf{PK.Dec}(sk^*, c^*_{W,i+1})$. After this, it is trivial to recover the source key material as $\kappa_{oob} \leftarrow (c_{\kappa,i+1} + W_{i+1}) \bmod 10^6$.

Impact. A malicious server can use this attack to compromise the OOB source key material as soon as the OOB factor is used, removing the entropy from this factor from the MFKDF key. This compromise is invisible to the user: after obtaining the source key material, the adversary can overwrite β^*_{i+1}, either with the previous honest state β_i, or by replacing $c^*_{W,i+1}$ with $c_{W,i+1} \leftarrow \mathsf{PK.Enc}(pk, W_{i+1})$, ensuring that the user can successfully decrypt the witness ciphertext with their S/MIME private key the next time the OOB factor is used.

3.2.2 Parameter Tampering Attacks

The public MFKDF state B includes information on the key derivation function and its parameters, such as the number of iterations for PBKDF and the length of the derived key. Tampering with this information will, in most cases, result in deriving a bogus key. Nonetheless, the resulting key could be used to infer information about the factors, or can be easy to guess. The attacks we present illustrate the danger of storing this "KDF metadata" unauthenticated in the MFKDF state, rather than having the MFKDF client use a fixed KDF.

For this attack, we additionally assume that the MFKDF key is used to, for example, encrypt data with an AEAD, or to authenticate to a service, and that the adversary can see the resulting ciphertexts or authentication messages. If MFKDF is used to derive a key for a password manager or outsourced file storage, an attacker in the malicious or compromised service provider threat model can perform this attack. Similarly, if MFKDF is used for authentication with the ISO/IEC 9798-2 Unilateral Authentication protocol, as proposed by NS [40, §7], the verifier can perform this attack.

1: Short Key Attack. Consider MFKDF with key length ℓ chosen by the user. (The default length is 32 bytes.) The adversary can overwrite ℓ, picking a new arbitrary length: let's assume 4 bytes. If this key is used to encrypt user data or to produce authentication messages, the adversary can easily guess the 4-bytes key by brute-force.

This trivially violates the confidentiality of the encrypted data, but also has further consequences. Note that some of

the PBKDFs proposed for use in MFKDF (such as PBKDF2) are actually Extendable Output Functions (XOFs), meaning that the 4-bytes derived key will be a prefix of the original key K. The adversary thus learns the first four bytes of K. The attack can then continue iteratively: the adversary sets the key length to 8 bytes, and guesses the remaining unknown 4 bytes. This process can be iterated until the full key is recovered.

2: Weak PBKDF Attack. The recommended choice of PBKDF for MFKDF is Argon2, a memory-hard PBKDF. The use of Argon2 slows down a brute-force attack on the MFKDF factors, by limiting the effectiveness of parallelization [9, 44].

However, since the choice of PBKDF is stored in the public state B, an adversary can overwrite B, replacing Argon2 with a weak KDF such as, e.g., PBKDF2 with a single iteration of HMAC. When MFKDF is evaluated, a different key $K' \neq K$ will be derived from the same factors. If the adversary observes cryptographic operations where K' is used, like encryption or authentication, it can mount a brute-force attack to guess the source key material of the factors used to derive K', with a high degree of parallelism and fast computation for each guess. A successful attack would expose the source key material, enabling the adversary to recover the original key K. Hence, the benefits of using a memory-hard PBKDF are void against an attacker with write access to the state.

Impact. The attacks significantly reduce the computation cost of brute-force attacks on MFKDF. For the weak PBKDF attack, an analysis by Percival [44] estimates the cost of guessing a password with 56 bits entropy to $10M for PBKDF2, compared to $210B for a memory-hard KDF. For the short key attack, a full key search for a 32 byte key requires 2^{256} guesses, compared to only $8 \cdot 2^{32}$ guesses when iterating the attack to recover 4 bytes at a time.

3.2.3 Share Dilution Attack

The t-out-of-n threshold MFKDF construction (Section 2.2.2) uses Shamir secret sharing to allow the MFKDF key to be derived from any subset of at least t out of the n factors. Like for the parameter tampering attacks, the threshold value t is stored in the unauthenticated MFKDF state. This attack uses the lack of integrity of the state, together with the fact that the concrete code implementation of algorithm Recover [36] in the reference implementation deviates from what NS describe in [39], to effectively remove factors from MFKDF.

Attack Overview. Instead of recovering the share of the lost factor with Rec, as specified in [39], the implementation of algorithm Recover runs $\mathsf{Share}(\kappa, t, n)$ to generate fresh shares for all factors, where t is fetched from the MFKDF state B.[6] This is equivalent to running the share setup algorithm again with the same secret. Since B is not integrity-protected, an adversary can overwrite t with an arbitrary value $t' < t$. This change has no effect until the user runs algorithm Recover (e.g., to change their password). At that point, the adversary-chosen threshold t' is used instead of t as input to Share, with the result that t' of the new shares suffice to derive the MFKDF key.

Impact. This attack impacts the security of the t-out-of-n threshold MFKDF construction by reducing the threshold needed to compute the final key, hence "diluting" the entropy provided by multiple factors. As an example, a 3-out-of-4 construction with HOTP, HMAC, password and OOB as factors becomes trivially broken when t' is set to 2: an extensive key search for the ~40 bits of entropy provided by the OOB and HOTP factors would take seconds on a laptop.

Limitations. The MFKDF reference implementation is only vulnerable to our attack if the attacker sets $t' > 1$, as Shamir secret sharing is not used for thresholds of 1.

3.3 Specification Attacks

Underspecification in cryptographic systems can lead to attacks, as showcased by for example JWT [49]. We now turn to attacks on MFKDF which fall into this category. They either stem from a lack of rigor in the specification of cryptographic primitives, or from incorrect assumptions on the security of these primitives. Most of these attacks are also reflected in the reference implementation of MFKDF itself. The attacks in this section all assume read-only access to the public MFKDF state.

3.3.1 HOTP Compromise Attack

This attack allows an adversary who already compromised factor material (i.e. the HMAC key)[7] to recover the MFKDF key if H/TOTP factor constructions are used, bypassing other factors entirely. For both HOTP and TOTP, the HMAC key is stored in encrypted form in the public factor state. This encryption is performed using a "symmetric encryption scheme", but no security requirements of the scheme are specified. In their concrete implementation of the H/TOTP factor constructions, Nair and Song use the one-time pad (i.e., XOR) as the symmetric encryption scheme.

Attack Overview. In the reference implementation of MFKDF, the feedback encryption mechanism using the final MFKDF key K creates the factor material ciphertext on line 7 of Algorithm 2 as $c_\sigma \leftarrow \mathsf{SK.Enc}(K, k_h)$, where

$$\mathsf{SK.Enc}(K, k_h) := K \oplus k_h \,.$$

The ciphertext c_σ is part of β_i, contained in the public MFKDF state. Hence, an attacker that compromises the factor material k_h of a *single* HOTP or TOTP factor can recover the final MFKDF key as $K \leftarrow c_\sigma \oplus k_h$.

[6] See src/classes/MFKDFDerivedKey/reconstitution.js, line 365

[7] This HMAC key can be commonly extracted from authenticator apps for smartphones by using the backup functionality [5, 17, 19].

Impact. This attack violates the MFKDF security guarantee that the derived key can only be recovered after *all* factors have been compromised. While leakage of the HMAC key is a strong prerequisite, the consequence is that this attack completely voids the security of MFKDF in the presence of a *single* compromised factor.

3.3.2 HOTP Bias Attack

This attack exploits a small bias in HOTP witnesses to recover the source key material κ_{hotp} from sufficient observations of the public state in the HOTP factor construction.

The reduction of the 31-bit integer s_{int} modulo 10^d on Line 5 of HOTP in Algorithm 1 adds a small bias to the output, favoring values closer to zero. While this bias is too small to be an issue for authentication [34, A.4.1], it can be exploited for MFKDF because NS use the HOTP witnesses to protect the confidentiality of a static secret. This connects different samples (despite independent witnesses) and allows an adversary to exploit the small bias in each observation to accumulate information about the hidden secret.

Attack Overview. The HOTP factor construction uses a d-digits long witness to hide the static source key material $\kappa_{hotp} \in [0, 10^d)$: it predicts the next witness W_{i+1} and encrypts κ_{hotp} via $c_{\kappa,i+1} = \kappa_{hotp} - W_{i+1} \bmod 10^d$ (line 17 of Algorithm 2). The resulting ciphertext $c_{\kappa,i+1}$ is called the "offset" in [39]. The adversary may observe many such offsets, since they are part of the public state.

Contrary to the assumption of NS, the HOTP values specified in RFC 4226 [34] are not uniformly random in $[0, 10^d)$: they have a slight bias towards smaller values. Algorithm 1 shows that HOTP produces the witness by generating 31 random bits[8] , interpreting them as an integer, and taking the result modulo 10^d to get a d-digit witness. This results in a bias, since the largest number, $2^{31}-1$, is not divisible by 10^d. Hence, for $\gamma = (2^{31}-1) \bmod 10^d$, values in $[0, \gamma]$ are slightly more likely to be output than values in $(\gamma, 10^d)$. While the HOTP factor construction would be provably secure with truly uniformly random witnesses (as alleged by NS in the security theorem for the H/TOTP factor construction [39, §5.2.2]), the actual, biased, witnesses can be exploited to recover κ_{hotp}.

As above, let γ denote the largest witness which occurs more frequently than average. Figure 2 illustrates the bias for a simplified example with 1-digit ($d = 1$) HOTP values.[9] Since the offsets $c_{\kappa,i} = \kappa_{hotp} - W_i \bmod 10$ are generated by subtracting the biased factor witnesses W_i from the secret, offsets in the interval $[\kappa_{hotp} - \gamma, \kappa_{hotp}]$ (modulo 10^d) will appear more frequently than the rest. Hence, κ_{hotp} can be recovered by finding the upper endpoint of this interval.

[8]The bits are selected from the HMAC digest. However, for our purpose and due to the PRF security of HMAC, we can consider the bits to be random.

[9]Our attack targets full-sized HOTP codes of at least 6 digits, as specified in RFC 4226 [34]. Figure 2 is only an illustrative example.

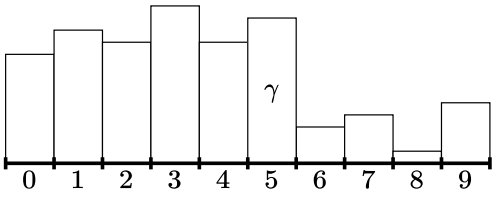

Figure 2: Bias of 1-digit HOTP witnesses for $n = 10$ bins.

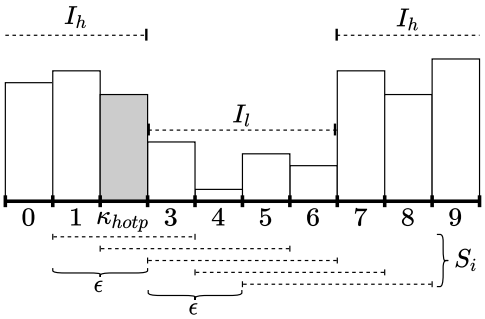

Figure 3: Bias of offsets, shifting the HOTP witness bias depending on the hidden secret κ_{hotp}.

Figure 3 shows a possible distribution of the biased offset for the static secret $\kappa_{hotp} = 2$. By I_h we denote the interval of more frequent offsets, and I_l the less frequent. In the previous example, we have $I_l = [3,6]$ and $I_h = [0,2] \cup [7,9]$.

Given m offsets $c_{\kappa,i}$, the optimal attack strategy to locate interval I_h uses the maximum likelihood estimator. Hence, we identify the consecutive interval (with wraparound at 10^d) that has the maximal sum of offsets. We generalize the problem of finding the exact end point of I_h to finding a value that is at most ε values away from the correct secret κ_{hotp}, as the adversary may have external means to test multiple guesses for κ_{hotp}. For instance, it may be feasible to try some number of HOTP values for every password guess, and perform trial decryption with the resulting MFKDF key to verify the correctness of the guess.

It is instructive to model MFKDF offsets as a "balls into bins" problem with biased probabilities, where samples correspond to balls and integers in $[0, 10^d)$ to bins, to gain a more rigorous understanding of our attack. As discussed in Appendix C, we can express the number of balls in an interval of bins as the sum of Bernoulli random variables and use two Chernoff bounds to estimate the worst case number of samples needed to get within ε distance of κ_{hotp}.

Attack Cost. TOTP factor constructions of MFKDF, which build on HOTP, require the user to precompute many offsets to allow keys to be re-derived in the future. As TOTP witnesses change every 30 seconds, NS suggest storing 86400 offsets, such that users can be inactive for 30 days without losing access to their key material. This is an unusually high number

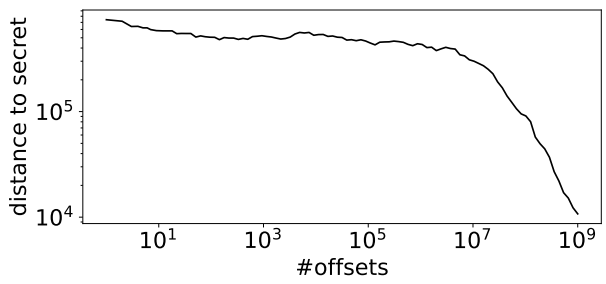

Figure 4: Simulated attack on HOTP factor constructions with up to 10^9 samples, averaged over 100 random targets κ_{hotp}.

of HOTP codes that are available to the adversary "for free" in the public MFKDF state. To understand the practicality of our attack, we simulate it by generating samples of the bias using 6-digit HOTP codes (conforming to RFC 4226 [34]) to hide different targets κ_{hotp}.[10] Figure 4 shows a log-log plot of the attack success, averaged over 100 targets and up to 10^9 samples. We measure the attack success as the position of target κ_{hotp} in the list of possible values i that is sorted in decreasing order by the overall sum of offsets observed in the interval $[i-\gamma, i] \pmod{10^d}$.[11] This distance decreases from around 300,000 for 10^7 samples to 10700 for 10^9 samples.

Impact. This attack recovers the source key material κ_{hotp} from the bias in HOTP witnesses, given enough observations of the offsets stored in the MFKDF state. While our simulation indicates that this HOTP bias attack is not practical[12], it directly invalidates the security theorem for the H/TOTP factor constructions in [39, §5.2.2]. In particular, the claims that the offsets "reveal no further information" about the source key material are clearly false.

Unfortunately, this attack also means MFKDF cannot use T/HOTP codes and would require witnesses that are actually uniformly random. This significanly weakens the contribution of MFKDF, as one of its major advantages is the *"compatibility with existing systems using PBKDFs; systems should not need to be entirely rearchitected to use MFKDF, and the user experience should not be impacted whatsoever. [39, § 4]."* However, T/HOTP codes are widely used for authentication, and we are not aware of any authentication factors with uniform witnesses that are similarly popular.

[10] Running the simulations took 14.5 hours on an Intel® Xeon® CPU E5-2699A v4 @ 2.40GHz with 44 physical cores and 512 GB of memory.

[11] We conservatively measure attack success and count values that have the same overall sum as the target κ_{hotp} to the values in front of κ_{hotp} to get the worst-case distance of the correct answer from the top of the sorted list.

[12] However, it could be dangerous for an application to allow users to be inactive for many years, which leads to it publishing many more samples in the MFKDF state.

3.3.3 Share Recovery Attack

This attack impacts the security of the t-out-of-n threshold MFKDF construction (Section 2.2.2).

Let F_x be a "lost" factor and let F'_x be the new factor that will replace it. Before the recovery process, the public MFKDF state contains $c_{s_x} = s_x \oplus \text{pad}_x$, the encryption of s_x under a pad derived from the source key material of F_x. If the algorithm Recover works as described by NS in the paper[13], then step 3 uses algorithm Rec to recover s_x with help of the other factors. The recovered share is then re-encrypted with the pad corresponding to the new source key material pad'_x, and the ciphertext in the public state is updated to $c'_{s_x} = s_x \oplus \text{pad}'_x$. However, the share itself is not updated.

Attack Overview. The share not being updated when Recover is run has two consequences. First, an adversary who compromises the source key material of the *lost factor* can use it to decrypt c'_{s_x} and recover s_x. Hence key material of a lost factor is as critical for security as that of any still valid factor. Second, the same share s_x is encrypted twice using different pads pad_x and pad'_x, a "two-time pad". This is completely insecure when the pads have small entropy. If the adversary observes the public state before and after the update, it obtains c_{s_x} and c'_{s_x}. It can therefore compute:

$$c_{s_x} \oplus c'_{s_x} = \text{pad}_x \oplus \text{pad}'_x$$

and start guessing values for pad_x and pad'_x. For factors of entropy m, pad can only take $M = 2^m$ possible values. We can efficiently recover the values of the pads in $O(M)$ guesses: first we recompute all the M possible values of $\text{pad}_x \oplus c_{s_x} \oplus c'_{s_x}$, storing them in a hashtable. Then we iterate over the M possible values of pad'_x, checking in the hashtable for matches. This attack succeeds with overwhelming probability if $2^m \ll 2^\ell$, where ℓ is the bit length of the share. We analyse this in more detail in the full version of this paper.

Impact. Our first observation means that the recovery algorithm should not be used as a means of *replacing* a compromised factor (e.g., in the event of a password breach), as witnesses for the lost factor can still be used to compute the share. By our second observation, the recovery process allows an adversary to efficiently recover the share of low-entropy factors when they are replaced, violating the security claim that individual MFKDF factors cannot be guessed in isolation.

3.3.4 Share Format Attack

Recall that in threshold MFKDF, the public MFKDF state B contains a one-time pad encrypted share c_s and factor construction state β_i for each factor F. NS note that the one-time pad can "be replaced by symmetric-key encryption as long as no checksums or integrity mechanisms are included in the

[13] The reference implementation diverges from this description, and is therefore vulnerable to the share dilution attack described in Section 3.2.3.

scheme" [39, A.4]. The following attack shows that this esoteric requirement is vital for the security of MFKDF, but that even using the one-time pad, threshold MFKDF is vulnerable to a theoretical guessing attack against individual factors, leveraging the byte representation of Shamir shares.

Attack Overview. The attack assumes that the byte representation of the shares produced by Shamir secret sharing is non-uniform. Note that in Shamir's original description [48], shares are elements of a prime field; for an l-byte prime $p < 256^l$, each share is a value in $[0, p)$, and hence cannot be represented as an l-byte string uniform in $[0, 256^l)$. Since the distribution of shares is non-uniform, we can construct a distinguisher algorithm $\mathcal{D}$ that, given as input either a share or a random byte string of the same length, outputs 1 if the input was a share and 0 otherwise. We refer to the success probability of this algorithm as:

$$p = |\Pr[\mathcal{D}(\mathrm{s}) = 1 | \mathrm{s \text{ is a share}}] - \Pr[\mathcal{D}(\mathrm{s}) = 1 | \mathrm{s \text{ is random}}]|$$

An attacker can use this distinguisher to guess witnesses for each factor independently of the other factors, as follows. The adversary can check a witness guess W_i^* for F by computing:

$$\kappa_F^* \leftarrow \mathsf{Derive}_1(W_i^*, \beta_i)$$
$$\mathrm{pad}^* \leftarrow \mathsf{HKDF.Exp}(\kappa_F^*, \varepsilon, \ell)$$

It can now use the distinguisher to check if guess was correct by evaluating if $\mathcal{D}(c_\mathrm{s} \oplus \mathrm{pad}^*) = 1$. This confirms a correct guess with probability p.

Note that if the encryption scheme included any "integrity mechanisms", the distinguisher could be replaced by the decryption algorithm. The attack would then succeed with overwhelming probability even if the shares were uniform.

Impact. This attack undermines the claims of exponential security of MFKDF, allowing an adversary to attack each factor separately if shares have a biased representation. For example, in case the distinguisher fails with probability 2^{-20}, i.e. $p = 1 - 2^{-20}$, HOTP and TOTP factors are trivial to brute-force since they individually provide only ~20 bits of entropy.

Limitations. In the reference implementation, the share representation used is such that the bias is always contained in the first byte of each share. Additionally, because of an implementation bug, the first 16 bytes of shares are actually not encrypted. Therefore, the reference implementation inadvertently avoids being vulnerable to this attack. However, the attack would work if the shares were properly encrypted.

4 Mitigations

In this section, we discuss how some of the attacks from Section 3 can be mitigated. Unfortunately, this does not apply to Attack 3.1; due to the static nature of key derivation[14] and the mismatch between the threat models of authentication and key derivation, this attack *cannot be mitigated*. The result is that dynamic authentication factors such as H/TOTP, OOB codes and HMAC hardware tokens—the most common MFA methods in practice—are unsuitable for use with MFKDF. This represents a significant drawback for MFKDF, as one of its main goals is to *"realize the full benefits of multi-factor authentication within the key-derivation process"* with *"no noticeable change to the user experience, and while supporting all of the same 2FA factors that users have already grown accustomed to"* [39, §2]. Section 5 discusses this fundamental problem of MFKDF in more detail; in the rest of this section, we focus on the problems that can be fixed.

State Integrity. It is clear at this point that MFKDF should not be used if its state has no integrity protection: the OOB Overwriting (3.2.1), Share Dilution (3.2.3) and Parameter Tampering (3.2.2) attacks all result from this lack of integrity.

A partial solution could be to give up on some of the flexibility and functionality of MFKDF, such as the support for arbitrary threshold policies and user-specified key length, and embed these parameters in the client software instead of keeping them in the state. While fixing the threshold value, key length, and PBKDF parameters prevents attacks 3.2.3 and 3.2.2, it is not possible to hard code public keys to protect against attack 3.2.1. Moreover, other attacks that rely on the lack of integrity of the remaining state may still be possible; this mitigation only helps against some of the concrete vulnerabilities that we found, but does not fundamentally solve the issue of state integrity.

Unfortunately, verifying the integrity of the state presents a "chicken-and-egg" dilemma: the purpose of MFKDF is to derive a key, so we cannot assume the existence of a key which can be used to protect the state, but without a key, the integrity of the state cannot be verified. The only option is to use the MFKDF key, derived from the unprotected state, to itself ensure the integrity of the state. But this circular approach is vulnerable to attacks where the adversary exerts control over the derived key (in the vein of Attack 3.2.2).

Therefore, as it stands, MFKDF itself cannot achieve state integrity. The two plausible alternatives to instantiate MFKDF in a secure way are to rely on a trusted third party to protect the state, or to come up with a new, stateless construction.

In their response to our disclosure, NS suggest the first option, saying that the state can be stored on a blockchain or on IPFS. This impacts the practicality of MFKDF, and leaves open the problem of updating the state: either the state is immutable, or the user would need to authenticate in order to modify it. An MFKDF construction without state might be of interest for future work, but requires considerable change to the current design.

Cryptographic Primitives. The HOTP Compromise (3.3.1) and Share Format (3.3.4) attacks result from under-specified requirements for the cryptographic primitives used

[14] For functionality, the key must be deterministically deriveable from the factors.

in MFKDF. NS argued in their response that this underspecification is intentional, to ensure that MFKDF is "future proof" and can be adapted and implemented as required by applications. We believe the specification should be more stringent and clarify the security properties of the primitives on which MFKDF relies. In this vein, NS convene with us that symmetric encryption should at least be IND-CPA secure (hence in particular excluding the usage of one-time pad).

Independently from our other proposed mitigations, we advise to use AEAD encryption where appropriate to protect dynamic parts of the state from changing. A notable exception is the encryption of shares, which instead requires a non-authenticated scheme to avoid factor bruteforce attacks through trial-decryption. Avoiding this class of attack requires non-standard assumption on the encryption scheme: decryptions under incorrect keys should yield plausible-looking shares. This has been studied by Jules and Ristenpart in their work on *honey encryption* (HE) [26]: we postulate that a HE scheme could be used to encrypt the shares while withstanding brute-force attacks on low-entropy keys.

HOTP. While the HOTP Bias Attack (3.3.2) can be fixed by discontinuing the use of HOTP codes as an MFKDF factor, the fundamental issue with dynamic factors (including HOTP) remain; they should not be used for key derivation.

Share Recovery. As described in Section 3.3.3, the Share Recovery Attack can be prevented by having Recover generate fresh shares when a factor is lost. If the share threshold t were to be additionally authenticated or constant, this results in a secure factor recovery mechanism (that does not instead expose the user to the Share Dilution Attack 3.2.3).

5 Discussion

In our analysis of MFKDF, we found multiple issues with both the proposed constructions and the implementation. However, the most damaging finding is arguably the dynamic factor attack, which points to a fundamental flaw in the *concept* of MFKDF, and cannot be mitigated. This flaw stems from a gap between the two threat models of password-based systems in which MFKDF operates: key derivation and authentication.

In the following, we discuss this gap and why it arises. We compare MFKDF to the state of the art in authentication and key derivation, respectively. Our analysis shows that for authentication, MFKDF is always weaker than traditional MFA, and that for key derivation, MFKDF is no better than using PBKDF on a longer password—and the latter is arguably the safer option from a usability point-of-view.

Authentication. In the setting of (multi-factor) authentication, the goal is to prevent *outsiders* from impersonating the prover towards the verifier. For instance, consider the case of a server authenticating a user to an application such as e-banking, unencrypted email or cloud storage. In its role as a verifier, the server is trusted: it aims to prevent an external actor—which may partially compromise user secrets and observe the communication between user and server—from gaining access to the user's resources. For MFKDF, this threat model is relevant when the derived key is used for authentication; NS propose this usage, arguing that the final key implicitly verifies the authentication factors used to derive it, and can hence replace traditional MFA.

Next, we compare the security achieved by using the MFKDF key for authentication, to directly using traditional authentication factors for MFA. Note, however, that the state-of-the-art for authentication are protocols like Passkeys [13], which leverage public-key cryptography for completely passwordless authentication. MFKDF does not try to compete with these more modern alternatives; its goal is instead to strengthen the security of password-based systems, making MFA the appropriate baseline.

MFKDF vs. MFA. In traditional MFA, dynamic authentication factors present a significant advantage over static ones: access to a dynamic factor witness is only useful to authenticate *once* (the witness is "one-time"), and sometimes only during a short time interval (cf. TOTP). This is possible thanks to the threat model, which allows the client to put some trust in the server (concretely, in the form of a shared secret such as an HMAC key, or a server-generated OOB code).

This is not the case in MFKDF. Because the final key has to be static, dynamic factors need to be turned into static ones. Consequently, the one-timeness/freshness properties of dynamic factor witnesses are lost; as shown in Attack 3.1, corrupting a single witness allows an adversary to recover the source key material for the corresponding factor.

NS do not discuss the consequences of turning dynamic factors into static ones in their paper, and—in response to our disclosure—dismissed this limitation as inherent to the properties of a key derivation function. However, dynamic factors can make a difference in practice: consider a user authenticating to their e-banking with a password and a hardware token, on a device infected with a keylogger. The adversary learns the password and the token witness, allowing it to derive the MFKDF key. This enables the attacker to repeatedly authenticate as the user, also in the future. In contrast, if traditional MFA was used, the attacker would be unable to authenticate at all since it does not have access to the hardware token, and the captured witness has already been used.

Hence, authenticating with traditional MFA is stronger than using MFKDF. Furthermore, the connotation that e.g. HOTP codes are "one-time" (as alluded to in the name) might convey a false sense of security to users, who should not reasonably be expected to understand that their previously dynamic authentication factors have now become static, and that their "one-time passwords" need to be treated with as much care as their normal passwords.

Key Derivation. In the setting of password-based key derivation, the goal is to extract entropy from a password to produce a cryptographic key, while making brute force and dictionary attacks hard. The resulting key can then be used for cryptographic operations, without requiring the user to memorize or store the key. Notable use-cases include encrypting files for end-to-end encrypted cloud storage (e.g. MEGA [32]), or protecting passwords stored in an online password manager (e.g. Bitwarden [24]): here, the service provider hosting the encrypted data is not trusted. This threat model differs from that of authentication in two ways.

First, if the key is used to protect outsourced data, the server storing that data is considered malicious. As we outlined under *Authentication* above, this deviates from the authentication setting, where the client may share secret key material with the server for the purpose of authentication.

Second, the attacker "trivially" wins if it compromises the device of the user where the key derivation is taking place, since it then learns the key. In the case of MFKDF, this is true even if it has not compromised any of the factors used for the key derivation. In the authentication setting, in contrast, compromise of the device where authentication is taking place is insufficient to impersonate a user thanks to the dynamic factors; all factor material must be corrupted in order for the attacker to win (see the keylogger example above).

One of the explicit goals of MFKDF is to bring the security guarantees of MFA to key derivation, but this is unattainable, as the two threat models are in conflict: trust in the server cannot be leveraged in the key derivation setting, and MFKDF cannot help against compromise of the user's device (as MFA does) if the server is also malicious.

MFKDF vs. PBKDF. In light of the above considerations, we argue that MFKDF is no stronger than using PBKDF on a longer password (or, equivalently, the combination of multiple static secrets). First of all, Attack 3.1 shows that dynamic factors are unsuitable for use with MFKDF; they have to be turned into static source key material to satisfy the requirements for key derivation, and hence lose their advantage over static factors. Since MFKDF internally applies PBKDF to the concatenated source key material of all factors (see lines 5–6 of Algorithm 5), it is clear that this approach is no more secure than applying PBKDF to a longer password of the same entropy as that of the factors combined. More importantly, users are faced with the non-obvious shift in threat model which implies that one-time codes from their dynamic factors need to be treated as secrets lest they permanently degrade the security of their key.

The other claimed advantages of MFKDF, such as threshold key derivation and factor recovery, have to be traded off against their disadvantages. Indeed, our attacks show that the substantial complexity added by MFKDF can easily make it considerably less secure than just PBKDF. Among the reasons for this are the vulnerabilities that arise from having security critical values in unauthenticated state, as well as implementation pitfalls such as the use of biased Shamir shares combined with encryption; a use case for which they were not designed. Such features would require careful formal treatment.

6 Conclusions

MFKDF is promoted as a more secure alternative to PBKDF, bringing the advantages of multi-factor authentication to password-based key derivation. In our analysis, we showed that the security claims made for MFKDF are void: the proposed construction suffers from vulnerabilities that allow an adversary to compromise or bypass the factors used for key derivation, or directly recover the derived key without corrupting all of the factors. Additionally, even with the mitigatable vulnerabilities patched, MFKDF is not more secure than PBKDF with longer passwords of equivalent entropy.

As a concept, MFKDF does propose a way of harvesting the entropy from authentication factors that a user might already have, and use it for key derivation. Indeed, the authors state that MFKDF provides *"a paradigm shift toward direct cryptographic protection of user data using all available authentication factors, with no noticeable change to the user experience"* [39]. However, when proposing to use authentication factors for key derivation, they fail to discuss the disparate threat models of these objects. The result is that the protection provided by dynamic factors against impersonation attacks is lost when MFA is replaced by authentication through MFKDF with the same factors. Additionally, dynamic factor witnesses such as HOTP codes are no longer one-time when used for MFKDF; they lose their advantage over static factors, and they instead represent a security risk for users who do not understand this change.

Future Work. Constructing an alternative to PBKDF which strengthens the security of derived keys by combining the password with additional inputs is a highly relevant quest. If we move away from the authentication threat model, we can design dynamic factors specifically for key derivation purposes which can achieve this goal, and provide security for the KDF even under leakage of all the inputs and outputs of the factor. Think, for instance, of placing an OPRF key on a smartphone, and using the phone as a second factor to interactively evaluate the OPRF on input a static secret such as a password. With such a design, even with access to all the messages of the OPRF flow, an adversary would need to compromise both the password and the OPRF key in order to derive the final key.

References

[1] Martin R. Albrecht, Miro Haller, Lenka Mareková, and Kenneth G. Paterson. Caveat implementor! Key

recovery attacks on MEGA. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 190–218. Springer, Heidelberg, April 2023.

[2] Martin R. Albrecht, Lenka Mareková, Kenneth G. Paterson, and Igors Stepanovs. Four attacks and a proof for Telegram. In *2022 IEEE Symposium on Security and Privacy*, pages 87–106. IEEE Computer Society Press, May 2022.

[3] FIDO Alliance. Fido authentication: A passwordless vision. https://fidoalliance.org/fido2/, 2019. Accessed on 2024-01-25.

[4] Leonardo C. Almeida, Ewerton R. Andrade, Paulo S. L. M. Barreto, and Marcos A. Simplício Jr. Lyra: password-based key derivation with tunable memory and processing costs. *Journal of Cryptographic Engineering*, 4(2):75–89, June 2014.

[5] 2FAS Auth. How to use/sync more devices with 2fas? https://2fas.com/support/2fas-mobile-app/how-to-use-sync-more-devices-with-2fas/, January 2024. Accessed on 2024-01-12.

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

[7] Mihir Bellare. New proofs for NMAC and HMAC: Security without collision-resistance. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 602–619. Springer, Heidelberg, August 2006.

[8] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In Neal Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 1–15. Springer, Heidelberg, August 1996.

[9] Alex Biryukov, Daniel Dinu, and Dmitry Khovratovich. Argon2: New generation of memory-hard functions for password hashing and other applications. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 292–302, 2016.

[10] John Black, Phillip Rogaway, and Thomas Shrimpton. Encryption-scheme security in the presence of key-dependent messages. In Kaisa Nyberg and Howard M. Heys, editors, *SAC 2002*, volume 2595 of *LNCS*, pages 62–75. Springer, Heidelberg, August 2003.

[11] Joachim Breitner and Nadia Heninger. Biased nonce sense: Lattice attacks against weak ECDSA signatures in cryptocurrencies. In Ian Goldberg and Tyler Moore, editors, *FC 2019*, volume 11598 of *LNCS*, pages 3–20. Springer, Heidelberg, February 2019.

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

[13] Tim Cappalli and Matthew Miller. Passkey Developer Resources. https://passkeys.dev/. Accessed on 2023-09-19.

[14] Sílvia Casacuberta, Julia Hesse, and Anja Lehmann. Sok: Oblivious pseudorandom functions. In *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*, pages 625–646, 2022.

[15] Dave Childers. State of the auth 2021. https://duo.com/assets/ebooks/state-of-the-auth-2021.pdf, September 2021. Accessed on 2023-10-19.

[16] Cybersecurity and Infrastructure Security Agency. Cybersecurity best practices. https://www.cisa.gov/topics/cybersecurity-best-practices, October 2023. Accessed on 2023-10-19.

[17] Beem Development. Aegis authenticator. https://getaegis.app/, January 2024. Accessed on 2024-01-12.

[18] Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In Joe Kilian, editor, *TCC 2005*, volume 3378 of *LNCS*, pages 303–324. Springer, Heidelberg, February 2005.

[19] Google. Transfer your google authenticator codes. https://support.google.com/accounts/answer/1066447?sjid=3042048156516579308-EU#zippy=, 2023. Accessed on 2024-01-12.

[20] Christopher Harrell. Yubikeys, passkeys and the future of modern authentication. https://www.yubico.com/blog/passkeys-and-the-future-of-modern-authentication/, 2022. Accessed on 2024-01-25.

[21] Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. Mining your ps and qs: Detection of widespread weak keys in network devices. In Tadayoshi Kohno, editor, *USENIX Security 2012*, pages 205–220. USENIX Association, August 2012.

[22] Jeff Hodges, J.C. Jones, Michael B. Jones, Akshay Kumar, and Emil Lundberg. Web authentication: An api for accessing public key credentials. https://www.w3.org/TR/webauthn/, 2021. Accessed on 2024-01-25.

[23] Apple Inc. Apple platform security. https://manuals.info.apple.com/MANUALS/1000/MA1902/en_US/apple-platform-security-guide.pdf, May 2021. Accessed on 2023-12-08.

[24] Bitwarden Inc. Bitwarden security whitepaper. https://bitwarden.com/help/bitwarden-security-white-paper/, February 2023. Accessed on 2024-01-11.

[25] Stanislaw Jarecki, Hugo Krawczyk, and Jiayu Xu. OPAQUE: An asymmetric PAKE protocol secure against pre-computation attacks. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 456–486. Springer, Heidelberg, April / May 2018.

[26] Ari Juels and Thomas Ristenpart. Honey encryption: Security beyond the brute-force bound. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 293–310. Springer, Heidelberg, May 2014.

[27] Vlastimil Klima and Tomas Rosa. Attack on private signature keys of the OpenPGP format, PGP(TM) programs and other applications compatible with OpenPGP. Cryptology ePrint Archive, Report 2002/076, 2002. https://eprint.iacr.org/2002/076.

[28] Dr. Hugo Krawczyk and Pasi Eronen. HMAC-based Extract-and-Expand Key Derivation Function (HKDF). RFC 5869, May 2010.

[29] Hugo Krawczyk. Cryptographic extraction and key derivation: The HKDF scheme. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 631–648. Springer, Heidelberg, August 2010.

[30] Internet Archive Wayback Machine. Keepass plugins and extensions. https://web.archive.org/web/20150106032133/https://keepass.info/plugins.html, January 2015. Accessed on 2024-01-11.

[31] Itsik Mantin. Predicting and distinguishing attacks on RC4 keystream generator. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 491–506. Springer, Heidelberg, May 2005.

[32] Mega Limited. Mega security white paper – third edition. https://mega.nz/SecurityWhitepaper.pdf, Jun 2022.

[33] Microsoft. Bitlocker overview. https://learn.microsoft.com/en-us/windows/security/operating-system-security/data-protection/bitlocker/, October 2023. Accessed on 2023-10-19.

[34] David M'Raihi, David M'Raihi, Frank Hoornaert, David Naccache, Mihir Bellare, and Ohad Ranen. HOTP: An HMAC-Based One-Time Password Algorithm. RFC 4226, December 2005.

[35] David M'Raihi, Johan Rydell, Mingliang Pei, and Salah Machani. TOTP: Time-Based One-Time Password Algorithm. RFC 6238, May 2011.

[36] Vivek Nair. MFKDF GitHub repository. https://github.com/multifactor/MFKDF, August 2023. Accessed on 2024-01-12.

[37] Vivek Nair and Dawn Song. Decentralizing custodial wallets with mfkdf. In *2023 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 1–9, 2023.

[38] Vivek Nair and Dawn Song. Multi-factor credential hashing for asymmetric brute-force attack resistance. In *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*, pages 56–72, 2023.

[39] Vivek Nair and Dawn Song. Multi-factor key derivation function (MFKDF) for fast, flexible, secure, & practical key management. In Joseph A. Calandrino and Carmela Troncoso, editors, *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*. USENIX Association, 2023.

[40] Vivek Nair and Dawn Song. Multi-factor key derivation function (MFKDF) for fast, flexible, secure, & practical key management (full version). https://arxiv.org/abs/2208.05586v3, 2023.

[41] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *38th FOCS*, pages 458–467. IEEE Computer Society Press, October 1997.

[42] Kenneth G. Paterson, Matteo Scarlata, and Kien Tuong Truong. Three lessons from threema: Analysis of a secure messenger. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 1289–1306, Anaheim, CA, August 2023. USENIX Association.

[43] Kenneth G. Paterson and Arnold K.L. Yau. Cryptography in theory and practice: The case of encryption in IPsec. Cryptology ePrint Archive, Report 2005/416, 2005. https://eprint.iacr.org/2005/416.

[44] Colin Percival. Stronger key derivation via sequential memory-hard functions. https://www.tarsnap.com/scrypt/scrypt.pdf, 2009.

[45] Dominik Reichl. Keepass password safe. https://keepass.info/index.html, October 2023. Accessed on 2024-01-11.

[46] Dominik Reichl. Keepass plugins: Otpkeyprov is a key provider based on one-time passwords. https://keepass.info/plugins.html#otpkeyprov, June 2023. Accessed on 2024-01-11.

[47] Ben Rush. Keechallenge: A plugin for keepass2 to add yubikey challenge-response capability. https://keepass.info/index.html, May 2016. Accessed on 2024-01-11.

[48] Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.

[49] Yaron Sheffer, Dick Hardt, and Michael B. Jones. JSON Web Token Best Current Practices. RFC 8725, February 2020.

[50] Xiaowen Xin. Titan M makes Pixel 3 our most secure phone yet. https://blog.google/products/pixel/titan-m-\makes-pixel-3-our-most-secure-phone-yet/, October 2018. Accessed on 2023-10-19.

[51] Frances F. Yao and Yiqun Lisa Yin. Design and analysis of password-based key derivation functions. In Alfred Menezes, editor, *CT-RSA 2005*, volume 3376 of *LNCS*, pages 245–261. Springer, Heidelberg, February 2005.

[52] Yubico. Using your yubikey with keepass. https://support.yubico.com/hc/en-us/articles/360013779759-Using-Your-YubiKey-with-KeePass, May 2020. Accessed on 2024-01-11.

A MFKDF Factor Constructions

Algorithm 3 HMAC Factor Construction for HMAC key k_h

procedure Setup(k_h)
  $\kappa_{hmac} \leftarrow k_h$; $c_0 \leftarrow\$ \{0,1\}^{160}$
  $W_0 \leftarrow \mathsf{HMAC\text{-}SHA\text{-}1}(\kappa_{hmac}, c_0)$
  return $(\kappa_{hmac}, (c_0, \kappa_{hmac} \oplus W_0))$
procedure Derive(W_i, β_i)
  $(c_i, c_{\kappa,i}) \leftarrow \beta_i$
  $\kappa_{hmac} \leftarrow c_{\kappa,i} \oplus W_i$
  $c_{i+1} \leftarrow\$ \{0,1\}^{160}$
  $W_{i+1} \leftarrow \mathsf{HMAC\text{-}SHA\text{-}1}(\kappa_{hmac}, c_{i+1})$
  return $(\kappa_{hmac}, (c_{i+1}, \kappa_{hmac} \oplus W_{i+1}))$

Algorithm 3 shows the HMAC factor construction used in MFKDF to enable support for hardware authenticators, such as YubiKeys, via the HMAC-SHA-1 challenge-response mechanism. In contrast to in the usual authentication protocol of an HMAC hardware token, where the authenticator is used to compute the response to a freshly sampled challenge, in the factor construction the next challenge c_{i+1} is derived ahead of time and stored in the public state β_{i+1}. The state also contains the XOR of the HMAC key k_h (which is also the source key material κ_{hmac}) with the next witness W_{i+1} computed as response to c_{i+1}. When the client authenticates, it uses the hardware authenticator to re-derive the witness W_{i+1} in response to the stored challenge and can hence recover κ_{hmac}.

Algorithm 4 OOB Factor Construction

Require: Let $(\mathsf{PK.Enc}, \mathsf{PK.Dec})$ be a public-key encryption scheme with associated keys (pk, sk).
procedure Setup()
  $\kappa_{oob} \leftarrow\$ [0, 10^6)$
  $W_0 \leftarrow\$ [0, 10^6)$
  $c_{\kappa,0} \leftarrow (\kappa_{oob} - W_0) \bmod 10^6$
  $c_{W,0} \leftarrow \mathsf{PK.Enc}(pk, W_0)$
  return $(\kappa_{oob}, (pk, c_{\kappa,0}, c_{W,0}))$
procedure Derive(W_i, β_i) ▷ The user has derived W_i using their sk by $W_i \leftarrow \mathsf{PK.Dec}(sk, c_{W,i})$.
  $(pk, c_{\kappa,i}, c_{W,i}) \leftarrow \beta_i$
  $\kappa_{oob} \leftarrow (c_{\kappa,i} + W_i) \bmod 10^6$
  $W_{i+1} \leftarrow\$ [0, 10^6)$
  $c_{\kappa,i+1} \leftarrow (\kappa_{oob} - W_{i+1}) \bmod 10^6$
  $c_{W,i+1} \leftarrow \mathsf{PK.Enc}(pk, W_{i+1})$
  return $(\kappa_{oob}, (pk, c_{\kappa,i+1}, c_{W,i+1}))$

Algorithm 4 shows the setup and derive procedures for the OOB factor construction proposed by Nair and Song. Here, the witness is the OTP from the OOB authentication factor, but generated by the client rather than the server. Note that the decryption of the OTP witness W_i from the outsourced ciphertext $c_{W,i}$ is part of the normal authentication protocol for the OOB authentication factor, and hence assumed to happen outside of algorithm Derive.

B MFKDF Constructions

Definition 5 (Plain MFKDF Construction)**.** Let $F_1, F_2, \ldots, F_n$ be authentication factors and let $\overline{\sigma} = (\sigma_{F_1}, \sigma_{F_2}, \ldots, \sigma_{F_n})$ be a vector with the corresponding factor materials. The MFKDF construction for factors $F_1, F_2, \ldots, F_n$ consists of algorithms

$$(K, B_0) \leftarrow \mathsf{Setup}(\overline{\sigma}, \ell), \text{ and}$$
$$(K, B_{i+1}) \leftarrow \mathsf{Derive}(\overline{W}_i, B_i).$$

Here ℓ is the desired length of the derived key and $\overline{W}_i = (W_{i,F_1}, \ldots, W_{i,F_n})$ is a vector consisting of the i^{th} witness for each factor in the construction. K is the final key output by MFKDF and B_i is the i^{th} public state.

For each factor F_j, let $F_j.\mathsf{Setup}$ and $F_j.\mathsf{Derive}$ be the setup and derive algorithm, respectively, associated to F_j by the factor construction. The "plain" (n-out-of-n) MFKDF construction for factors $F_1, F_2, \ldots, F_n$ is given in Algorithm 5.

Algorithm 5 Plain MFKDF Construction

Require: Let PBKDF be a memory-hard KDF.

procedure $\mathsf{Setup}((\sigma_{F_1},\ldots,\sigma_{F_n}),\ell)$
  for $j=1$ to n **do**
    $\kappa_{F_j} \leftarrow F_j.\mathsf{Setup}_1(\sigma_{F_j})$ ▷ If F_j uses feedback
    $(\kappa_{F_j},\beta_{0,F_j}) \leftarrow F_j.\mathsf{Setup}(\sigma_{F_j})$ ▷ Else
  $\kappa \leftarrow \kappa_{F_1}\|\kappa_{F_2}\|\ldots\|\kappa_{F_n}$; $salt \leftarrow\$ \{0,1\}^\ell$
  $K \leftarrow \mathsf{PBKDF}(\kappa, salt, \ell)$
  for $j=1$ to n **do** ▷ If F_j uses feedback
    $\beta_{0,F_j} \leftarrow F_j.\mathsf{Setup}_2(K,\sigma_{F_j},\kappa_{F_j})$
  $B_0 \leftarrow (\ell, salt, (\beta_{0,F_1},\ldots,\beta_{0,F_n}))$
  return (K,B_0)
procedure $\mathsf{Derive}(\overline{W}_i, B_i)$
  $(W_{i,F_1},\ldots,W_{i,F_n}) \leftarrow \overline{W}_i$
  $(\ell, salt, (\beta_{i,F_1},\ldots,\beta_{i,F_n})) \leftarrow B_i$
  for $j=1$ to n **do**
    $\kappa_{F_j} \leftarrow F_j.\mathsf{Derive}_1(W_{i,F_j},\beta_{i,F_j})$ ▷ If feedback
    $(\kappa_{F_j},\beta_{i+1,F_j}) \leftarrow F_j.\mathsf{Derive}(W_{i,F_j},\beta_{i,F_j})$ ▷ Else
  $\kappa \leftarrow \kappa_{F_1}\|\kappa_{F_2}\|\ldots\|\kappa_{F_n}$
  $K \leftarrow \mathsf{PBKDF}(\kappa, salt, \ell)$
  for $j=1$ to n **do** ▷ If F_j uses feedback
    $\beta_{i+1,F_j} \leftarrow F_j.\mathsf{Derive}_2(K,\kappa_{F_j},\beta_{i,F_j})$
  $B_{i+1} \leftarrow (\ell, salt, (\beta_{i+1,F_1},\ldots,\beta_{i+1,F_n}))$
  return (K,B_{i+1})

The syntax defining the t-out-of-n threshold MFKDF construction is given in Definition 4. We review the details of algorithms Setup and Derive in Algorithm 6, and of algorithm Recover in Algorithm 7. Algorithms $(\mathsf{Share},\mathsf{Comb},\mathsf{Rec})$ are part of a Shamir secret sharing scheme, as defined in Definition 3.

Algorithm 6 t-out-of-n Threshold MFKDF Construction

Require: Let PBKDF be a memory-hard PBKDF, and let HKDF.Exp be HKDF-Expand per RFC 5869 [28].

procedure $\mathsf{Setup}((\sigma_{F_1},\ldots,\sigma_{F_n}),t,\ell)$
  $\kappa \leftarrow\$ \{0,1\}^\ell$; $salt \leftarrow\$ \{0,1\}^\ell$
  $K \leftarrow \mathsf{PBKDF}(\kappa, salt, \ell)$
  $(s_1,\ldots,s_n) \leftarrow \mathsf{Share}(\kappa,t,n)$
  for $j=1$ to n **do**
    $\kappa_j \leftarrow F_j.\mathsf{Setup}_1(\sigma_{F_j})$ ▷ If F_j uses feedback
    $(\beta_{0,j},\kappa_j) \leftarrow F_j.\mathsf{Setup}(\sigma_{F_j})$ ▷ Else
    $\mathrm{pad}_j \leftarrow \mathsf{HKDF.Exp}(\kappa_j,\varepsilon,\ell)$ ▷ Expand κ_j to length ℓ with empty context
    $c_{s_j} \leftarrow \mathrm{pad}_j \oplus s_j$
  for $j=1$ to n **do**
    $\beta_{0,j} \leftarrow F_j.\mathsf{Setup}_2(K,\sigma_{F_j},\kappa_j)$ ▷ If feedback
  $B_0 \leftarrow (t,\ell,salt,(c_{s_1},\ldots,c_{s_n}),(\beta_{0,1},\ldots,\beta_{0,n}))$
  return (K,B_0)

Algorithm 7 Threshold MFKDF Construction Continued

procedure $\mathsf{Derive}(\overline{W}_i, B_i)$ ▷ Using t factors $(F_{u_1},\ldots,F_{u_t})$ of the n original factors
  $(W_{i,u_1},\ldots,W_{i,u_t}) \leftarrow \overline{W}_i$
  $(t,\ell,salt,(c_{s_1},\ldots,c_{s_n}),(\beta_{i,1},\ldots,\beta_{i,n})) \leftarrow B_i$
  for $j=1$ to t **do**
    $\kappa_{u_j} \leftarrow F_{u_j}.\mathsf{Derive}_1(W_{i,u_j},\beta_{i,u_j})$ ▷ If feedback
    $(\kappa_{u_j},\beta_{i+1,u_j}) \leftarrow F_{u_j}.\mathsf{Derive}(W_{i,u_j},\beta_{i,u_j})$ ▷ Else
    $\mathrm{pad}_{u_j} \leftarrow \mathsf{HKDF.Exp}(\kappa_{u_j},\varepsilon,\ell)$
    $s_{u_j} \leftarrow \mathrm{pad}_{u_j} \oplus c_{s_{u_j}}$
  $\kappa \leftarrow \mathsf{Comb}((s_{u_1},\ldots,s_{u_t}),t,n)$
  $K \leftarrow \mathsf{PBKDF}(\kappa, salt, \ell)$
  for $j=1$ to n **do**
    if $j \in \{u_1,\ldots,u_t\}$ **then**
      $\beta_{i+1,j} \leftarrow F_j.\mathsf{Derive}_2(K,\kappa_j,\beta_{i,j})$ ▷ If F_j uses feedback
    else $\beta_{i+1,j} \leftarrow \beta_{i,j}$
  $B_{i+1} \leftarrow (t,\ell,salt,(c_{s_1},\ldots,c_{s_n}),(\beta_{i+1,1},\ldots,\beta_{i+1,n}))$
  return (K,B_{i+1})
procedure $\mathsf{Recover}(\overline{W}_i, B_i, x, \sigma_{F'_x})$ ▷ Recover share x using t factors $(F_{u_1},\ldots,F_{u_t})$ of the n original factors
▷ Recover the secret and update existing factors:
  $(W_{i,u_1},\ldots,W_{i,u_t}) \leftarrow \overline{W}_i$
  $(t,\ell,salt,(c_{s_1},\ldots,c_{s_n}),(\beta_{i,1},\ldots,\beta_{i,n})) \leftarrow B_i$
  for $j=1$ to t **do**
    $\kappa_{u_j} \leftarrow F_{u_j}.\mathsf{Derive}_1(W_{i,u_j},\beta_{i,u_j})$ ▷ If feedback
    $(\kappa_{u_j},\beta_{i+1,u_j}) \leftarrow F_{u_j}.\mathsf{Derive}(W_{i,u_j},\beta_{i,u_j})$ ▷ Else
    $\mathrm{pad}_{u_j} \leftarrow \mathsf{HKDF.Exp}(\kappa_{u_j},\varepsilon,\ell)$
    $s_{u_j} \leftarrow \mathrm{pad}_{u_j} \oplus c_{s_{u_j}}$
  $\kappa \leftarrow \mathsf{Comb}((s_{u_1},\ldots,s_{u_t}),t,n)$
  $K \leftarrow \mathsf{PBKDF}(\kappa, salt, \ell)$
  for $j=1$ to n **do**
    if $j \in \{u_1,\ldots,u_t\}$ **then**
      $\beta_{i+1,j} \leftarrow F_j.\mathsf{Derive}_2(K,\kappa_j,\beta_{i,j})$ ▷ If feedback
    else $\beta_{i+1,j} \leftarrow \beta_{i,j}$
▷ Replace the lost factor:
  $\kappa'_x \leftarrow F'_x.\mathsf{Setup}_1(\sigma_{F'_x})$ ▷ If F'_x uses feedback
  $\beta_{i+1,x} \leftarrow F'_x.\mathsf{Setup}_2(K,\sigma_{F'_x},\kappa'_x)$
  $(\beta_{i+1,x},\kappa'_x) \leftarrow F'_x.\mathsf{Setup}(\sigma_{F'_x})$ ▷ Else
  $\mathrm{pad}'_x \leftarrow \mathsf{HKDF.Exp}(\kappa'_x,\varepsilon,\ell)$
  $s_x \leftarrow \mathsf{Rec}(\kappa,t,n,x)$
  $c_{s_x} \leftarrow \mathrm{pad}'_x \oplus s_x$
  $B_{i+1} \leftarrow (t,\ell,salt,(c_{s_1},\ldots,c_{s_n}),(\beta_{i+1,1},\ldots,\beta_{i+1,n}))$
  return B_{i+1}

C Formal Analysis of the HOTP Bias Attack

To rigorously analyze the HOTP bias attack, we model the MFKDF offsets as a "balls into bins" problem with biased probabilities. There are $n = 10^d$ bins (one for every possible offset) and m balls (corresponding to $c_{\kappa,i}$). The interval I_h of more likely offsets has $\gamma = 2^{31} \bmod 10^d$ elements (e.g., in Figure 3, $\gamma = 6$), and $|I_l| = 10^d - \gamma$. We now compute the required number of balls to recover κ_{hotp} with good probability.

Let X_i^j be a Bernoulli random variable that is 1 if ball i falls into bin j and 0 otherwise. The biased probabilities p_h for likely bins, respectively p_l for unlikely bins, are as follows:

$$p_h := \Pr[X_i^j = 1 | j \in I_h] = \frac{\lfloor 2^{31}/10^d \rfloor + 1}{2^{31}}$$
$$p_l := \Pr[X_i^j = 1 | j \in I_l] = \frac{\lfloor 2^{31}/10^d \rfloor}{2^{31}}$$

Let $B_i = \sum_1^m X_i^j$ be the random variable counting the number of balls in bin j. Since $|I_h| = \gamma$, we define a random variable for the sum of γ consecutive bins as follows, ending with bin i:

$$S_i = \sum_{j=0}^{\gamma-1} B_{i-j \bmod 10^d}$$

Our attack is perfectly successful if the interval I_h (which ends with the bin κ_{hotp}) has the largest overall sum, i.e., $\Pr[S_{\kappa_{hotp}} > \max_{j \neq \kappa_{hotp}} S_j]$.

In the generalized version of the problem, we consider the attack to be successful if any bin with distance at most ε from κ_{hotp} has the maximal sum for $i, j \in [0, 10^d)$:

$$\Pr\left[\max_{|i-\kappa_{hotp}| \leq \varepsilon} S_i > \max_{|j-\kappa_{hotp}| > \varepsilon} S_j\right] = \bigvee_{|i-\kappa_{hotp}| \leq \varepsilon} \bigwedge_{|j-\kappa_{hotp}| > \varepsilon} \Pr[S_i > S_j] \quad (1)$$

We will now compute $\Pr[S_i > S_j]$ for some i, j satisfying $|i - \kappa_{hotp}| \leq \varepsilon$ and $|j - \kappa_{hotp}| > \varepsilon$, by applying two Chernoff bounds for independent Poisson trials for some $0 < \delta_1, \delta_2 < 1$:

$$\Pr[S_i > (1-\delta_1)\mu_i] \geq 1 - e^{-\delta_1^2 \cdot \mu_i/2}$$
$$\Pr[S_j < (1+\delta_2)\mu_j] \geq 1 - e^{-\delta_2^2 \cdot \mu_j/3}$$

where μ_k denotes the expected value of S_k for $k \in \{i, j\}$, computed explicitly below. If we carefully pick δ_1, δ_2 such that $(1-\delta_1)\mu_i > (1+\delta_2)\mu_j$, then we can combine the above probabilities to estimate $\Pr[S_i > S_j]$. Although S_i and S_j are clearly not independent, we find that the empirically measured attack success reasonably matches the values computed here.

Before we can pick δ_1, δ_2, we compute the mean μ_k for some $k \in [0, 10^d)$ with the straightforward computation:

$$\mu_k = \mathbb{E}[S_k] = \sum_{j=0}^{\gamma-1} \sum_{i=1}^{m} \mathbb{E}[X_{k-j \bmod 10^d}^i] = m \cdot ((\gamma - \alpha_k) p_h + \alpha_k p_l)$$

for $\alpha_k = \min(|k - \kappa_{hotp}|, \min(n - \gamma, \gamma))$, the number of unlikely bins among the ones that S_k sums over (which can be at most the total number of unlikely bins $n - \gamma$, and not more than all bins of S_k).

We derive the following limits for δ_1, δ_2:

$$0 < \delta_1 < 1 - (1+\delta_2)\frac{\mu_j}{\mu_i}$$
$$0 < \delta_2 < \frac{\mu_i}{\mu_j} - 1$$

For the optimal bounds, we would set the values δ_1, δ_2 such that they maximize $\Pr[S_i > (1-\delta_1)\mu_i] \cdot \Pr[S_j < (1+\delta_2)\mu_j]$[15]. For simplicity, we just pick the middle of the interval for δ_2 (which is not significantly worse than the optimal choice):

$$\delta_2 = \frac{\mu_i - \mu_j}{2\mu_j}, \delta_1 = \frac{\mu_i - \mu_j}{2\mu_i}$$

We can estimate the worst case number of samples needed to identify the secret (up to an error ε) by picking the desired success probability t for $\Pr[S_i > S_j]$, for a reasonable choice of i, j, and then solve for m. We approximate

$$t = \Pr[S_i > S_j] \geq (\Pr[S_i > (1-\delta_1)\mu_i])^2$$

since we pick δ_1 and δ_2 to roughly balance the two Chernoff bounds. Then, we compute the number of samples m as

$$m \leq \frac{-2\ln(1-\sqrt{t})}{\delta_1^2 \cdot \gamma \cdot p_h}.$$

For example, at most $m \approx 10^{19}$ samples are required to recover κ_{hotp} uniquely for 6-digit HOTP witnesses[16].

[15] To maximize the product of these probabilities, we would need to balance both terms, i.e., have $\delta_1^2 \cdot \mu_i/2 \approx \delta_2^2 \cdot \mu_j/3$. Hence, we would have to pick δ_1, δ_2 such that $\frac{\delta_1}{\delta_2} \approx \sqrt{\frac{2\mu_j}{3\mu_i}}$.

[16] Using $t = 0.81$, $\varepsilon = 0$ (the answer needs to match κ_{hotp}), $i = \kappa_{hotp}$ and $j = \kappa_{hotp} + 1 \bmod 10^d$ since distinguishing S_j from S_i is maximally challenging because they only differ by one element. For these choices, we have the expectations $\mu_i = m \cdot \gamma \cdot p_h$ and $\mu_j = m \cdot ((\gamma - \varepsilon - 1) \cdot p_h + (\varepsilon + 1) \cdot p_l)$.