MFDPG: Multi-Factor Authenticated Password Management With Zero Stored Secrets

Vivek Nair
UC Berkeley
vcn@berkeley.edu

Dawn Song
UC Berkeley
dawnsong@berkeley.edu

***Abstract*—While password managers are a vital tool for internet security, they can also create a massive central point of failure, as evidenced by several major recent data breaches. For over 20 years, deterministic password generators (DPGs) have been proposed, and largely rejected, as a viable alternative to password management tools. In this paper, we survey 45 existing DPGs to asses the main security, privacy, and usability issues hindering their adoption. We then present a new multi-factor deterministic password generator (MFDPG) design that aims to address these shortcomings. The result not only achieves strong, practical password management with zero credential storage, but also effectively serves as a progressive client-side upgrade of weak password-only websites to strong multi-factor authentication.**

I. Introduction

Passwords constitute the primary authentication factor for the vast majority of currently deployed web applications. The prevalence of attacks like credential stuffing [2] has made safe password management amongst the most critical security tasks that an average user faces today. Accordingly, a variety of popular password management tools have emerged to help users address this issue in a secure and convenient manner.

The most commonly used password management solutions are cloud-based applications like LastPass [34], Dashlane [15], and 1Password [1]. These systems store user credentials in a centralized vault, typically encrypted using a key derived from the user's master password via PBKDF2 [30]. In doing so, they afford users the convenience of accessing their accounts from any device, but also create a central point of failure that, if compromised, could reveal all of a user's passwords. Indeed, LastPass alone has experienced 8 major security incidents [45], including a recent total data breach of stored credentials [76].

Alternatively, open-source password management software like Bitwarden [11] and KeePassX [32] has emerged in part to eliminate the central point of failure created by cloud-based password managers. While popular amongst experienced users, these solutions have failed to achieve widespread adoption due their relative lack of usability. In particular, synchronizing data between devices is often a manual and cumbersome process.

Network and Distributed System Security (NDSS) Symposium 2024
26 February - 1 March 2024, San Diego, CA, USA
ISBN 1-891562-93-2
https://dx.doi.org/10.14722/ndss.2024.23xxx
www.ndss-symposium.org

For over two decades, deterministic password generators (DPGs) have been proposed as a substitute for all forms of password management [31]. DPGs like PwdHash [71] apply a cryptographic hash function to a user's master password and the domain name of a website to generate a unique pseudorandom password for each domain. As such, DPGs theoretically work seamlessly across devices without the need for synchronization, offering the security of a client-side password manager with the flexibility of a cloud-based solution.

In practice, DPGs have been widely criticized for having a variety of security, privacy, and usability flaws [7] that have thus far seriously hindered their adoption. In this paper, we present a detailed analysis of 45 existing DPGs and summarize the main issues hindering their adoption. In particular, current DPG schemes allow a user's master password to be directly attacked in the event that any of a user's generated passwords are compromised. Most DPGs also lack the flexibility to support password rotation or complex password policies.

Based on these findings, we present a new multi-factor deterministic password generator (MFDPG), which aims to rectify the shortcomings of existing DPGs by incorporating multi-factor key derivation [43] into the password management process. In doing so, MFDPG effectively allows users to unilaterally upgrade password-only websites to support strong multi-factor authentication like TOTP [82] and YubiKey [85]. We further propose novel algorithmic solutions that facilitate password rotation and complex password policy compliance without leaking service usage patterns. By presenting a truly secure and practical design, we hope to revive the notion of DPGs as a viable alternative to password management.

Contributions:

1) We analyze 45 existing DPGs to asses the security, privacy, and usability issues hindering their adoption (§II-B).
2) We present MFDPG, a practical multi-factor authenticated DPG with zero client or server-side secret storage (§IV).
   a) MFDPG uses multi-factor key derivation to harden the system against attacks on the master password (§IV-A).
   b) We use Cuckoo filters to solve the revocation problem without leaking private account information (§IV-B).
   c) Our novel password generation algorithm supports any regular password policy using DFA traversal (§IV-C).
3) We evaluated MFDPG to verify its compatibility with all of the 100 most popular existing web applications (§V).
4) MFDPG has the further effect of progressively upgrading any password-based website to support strong MFA (§VI).

II. Background and Related Work

Passwords are notoriously weak as a sole authentication factor [22], [20], with attacks such as password spraying [67] and credential stuffing [2] affecting millions of online accounts each year. Nevertheless, a combination of circumstances has led passwords to remain the dominant authentication factor for online accounts today, making secure password management of paramount importance to one's overall security posture. An effective password management solution will allow users to have unique, high-entropy passwords for every website while maintaining security, portability, and ease of use.

In this section, we aim to motivate the need for a multi-factor deterministic password generator by discussing the range of existing password management and deterministic password generation solutions and their associated pitfalls.

A. Password Managers

The primary function of password managers is to provide an encrypted vault for the secure and portable storage of passwords. Today, users are faced with a difficult choice between the convenience of cloud-based password managers and the security of open-source self-managed solutions.

Cloud-based password managers, such as LastPass [34], Dashlane [15], and 1Password [1], are generally amongst the most popular password management solutions due to their perceived ease of use. The core technology used to secure these platforms is password-based key derivation.

Password-based key derivation functions like PBKDF2 [30], bcrypt [61], scrypt [58], and Argon2 [10] are one-way functions that convert a user's password into a cryptographic key that can be used for encryption. Password-based key derivation functions are built upon cryptographic hash functions like SHA-256 [23], but also incorporate a degree of intentional computational inefficiency that increases the relative difficulty of brute-force attacks. For example, the PBKDF2 configuration used by LastPass uses 100,000 sequential rounds of SHA-256 to increase its computational difficulty.

Most cloud-based password managers use a password-based key derivation function such as PBKDF2 to derive a key from the user's password upon login. A symmetric encryption function like AES-256 [44] is then used to encrypt all of a user's secrets on the client side prior to their storage in a centralized database. In theory, even an adversary with full access to the database will not be able to derive the key needed to decrypt the user's secrets without knowing their password.

Data breaches associated with major password managers are surprisingly common; LastPass alone has experienced at least 8 major security incidents [45], including a recent total data breach of encrypted credentials [76]. As no cloud-based service is totally immune from such a compromise, the security of these applications reduce, in practice, to the security of their users' master passwords. Given the aforementioned weakness of passwords as a sole authentication factor, there is a serious risk of attackers compromising the credentials stored in these services by performing offline brute-force attacks. As such, cloud-based password managers may even constitute a net liability for some users by creating a central point of failure.

Self-hosted open-source password managers like Bitwarden [11], KeePass [69], and KeePassX [32] use a similar architecture to cloud-based password managers to encrypt and store credentials using password-derived keys. However, these tools aim to address the vulnerability of cloud-based password managers to massive centralized data breaches by only storing encrypted passwords locally on a user's device, or on a server directly owned and maintained by the end user. In doing so, they avoid creating a concentrated high-value attack target as is inevitably the case with cloud-based password managers.

Still, self-hosted password managers are not immune to attack, with threats such as malware posing a risk to the database of encrypted credentials. As with cloud-based password managers, the user's security in the event of a breach ultimately reduces to that of their master password.

Moreover, open-source password management solutions have failed to achieve widespread adoption due their perceived impracticality. The lack of a centralized database to store credentials requires users to either host and maintain their own servers, or to manually synchronize data between all of their devices. Further, while cloud-based password managers are usually designed around resilient and highly-available architectures, self-hosted solutions can be susceptible to a total loss of data if a single device is lost or rendered inoperable. As such, most users have gravitated toward centralized password managers despite their relative security drawbacks.

B. Deterministic Password Generators

Deterministic password generators (DPGs) represent an interesting alternative to both cloud-based and self-hosted password management applications. Instead of storing encrypted passwords in any vault, DPGs work by deriving site-specific passwords using a cryptographic hash of the site's domain name and the user's master password. The deterministic nature of the underlying hash function ensures that the DPG will always generate the same password for a given site as long as the user remembers their master password, without needing to synchronize additional values between devices. This basic idea was originally proposed by HP in 2002 [31], and was popularized by Stanford's PwdHash in 2003 [63]. When implemented as a browser extension that automatically determines a site's URL, it has the additional benefit of resisting phishing attacks.

Since the advent of DPGs in the early 2000s, dozens of DPG tools have been developed and released as open-source websites, mobile applications, and browser extensions. Despite this, DPGs have seen even less mainstream adoption than self-hosted password managers, with the 20 most popular DPG extensions having less than 10,000 combined downloads.

Consumers' unwillingness to adopt DPGs may stem from a variety of security, privacy, and usability flaws [7], [46] that researchers have discovered with open-source DPGs. However, the relative obscurity of DPGs as a mainstream tool has resulted in limited literature being generated on this topic. To gain a better understanding of the current landscape of DPGs and their associated flaws, we performed a large-scale analysis of existing DPG implementations. By searching GitHub, the Chrome Web Store, and the Firefox Extensions website, we identified 45 free software packages implementing some form of deterministic password generation algorithm.

We explored the source code of these 45 DPG extensions to identify whether there were any particular threats to security, privacy, or usability that may limit their use as an alternative to traditional password managers. Our findings for each DPG system are detailed in Table I. Overall, we found four key issues affecting a large portion of the studied implementations:

1) **Brute-Force Susceptibility**. By far the largest concern with current DPG implementations is the ability to attack a user's master password. Because of the lack of stored ciphertexts in DPGs, exposure of the master password is sufficient to reveal all of a user's site passwords. However, as it stands, obtaining any of a user's site-specific passwords allows an attacker to perform an offline brute-force attack on that user's master password by checking which master password would, when combined with the site's known URL, have resulted in the correct site password being generated.

   This threat would be somewhat mitigated by the use of a strong progressive password-based key derivation function to increase the computational difficulty of a brute-force attack. Unfortunately, of the 45 applications surveyed, 28 only used a standard cryptographic hash, with 11 using deprecated function (MD5 or SHA1). A further 13 applications did use a progressive hash function, but supplied a cost parameter too low to effectively deter modern hardware. Only four of the 45 applications implemented a progressive hash function with well-chosen cost parameters.

2) **No Policy Support**. A second issue that affects most DPGs is the lack of support for applications with complex password policies. Websites often enforce a number of password strength and complexity requirements, with length, capitalization, character sets, etc. varying greatly from one service to another. No single static generator can hope to accomodate a wide range password policies without tuning the generation algorithm for each site.

   Of the 45 DPGs surveyed, 27 provide no settings at all for customizing password generation; users of these applications would likely face scenarios in which the DPG cannot generate a password compatible with a service they wish to use. The remaining 18 provide limited customization, which in most cases only allows the output length to be tweaked. Users of these applications must also remember these settings and configure them correctly on every login.

3) **No Revocation Support**. Similarly, websites often impose password rotation policies that require users to periodically change their password. However, the deterministic nature of DPGs usually does not allow users to have more than one password for a given website. Only four of the DPGs we analyzed provided a way to revoke a password and generate a new one for the same service. Of these, two required users to remember a revocation counter value for each website, while the other two stored this counter value in a file.

4) **Multi-Factor Authentication**. Finally, a fundamental limitation of all existing DPGs is the solitary reliance on a master password and the lack of support for multi-factor authentication. Most centralized password managers are enhanced by their support for multi-factor authentication at the login stage, though password vaults are still ultimately only encrypted with password-derived keys.

In addition to the common issues noted above, we found specific vulnerabilities in a six implementations: two schemes designed their own insecure hash function, two incorrectly deployed progressive hash functions, and two had obvious network-related vulnerabilities. More information about these issues is given in the footnotes of Table I.

As a result of many of the discussed issues, DPGs have remained a relatively obscure technology and have largely been dismissed as a viable alternative to conventional password managers. Thus, DPG projects have largely been abandoned, with most open-source DPGs receiving no updates in the past five years, as illustrated in Table I. We hope, in this paper, to revive the field of deterministic password generation, aided by recent cryptographic advances that enable the incorporation of multiple authentication factors into derived keys.

C. Multi-Factor Authentication

The most popular form of multi-factor authentication in use today is "out-of-band authentication" (OOBA) [29], which includes email and SMS-based authentication. However, these channels are not trustless and fundamentally require a server; thus, they do not receive significant attention in this paper.

Aside from OOBA, popular MFA options include "soft tokens" like HMAC-based one-time password (HOTP) [81] and time-based one-time password (TOTP) [82], as implemented in applications like Google Authenticator. These factors use a counter or timestamp to generate one-time passwords based on a pre-shared secret. "Hard tokens" such as YubiKeys [85] are also a popular option that requires using specialized hardware.

D. Multi-Factor Key Derivation

The Multi-Factor Key Derivation Function (MFKDF) [43] is a recent improvement over PBKDFs that incorporates multiple authentication factors into the key derivation process. It is a trustless cryptographic operation that can handle many popular authentication factors like HOTP, TOTP, and YubiKeys on the client side without the need for a trusted server.

The MFKDF specification contains two major architectural components. The first component is the set of so-called "factor constructions," which convert a dynamic *factor witness*[1] and public parameters into static key material. The public parameters require no security assumptions and can safely be stored in a database without concern for revealing information about the factors to potential adversaries. Constructions are given for a variety of popular authentication factors.

The second major component of MFKDF is the key derivation function itself, which adds a secret sharing layer to provide functionality such as threshold-based key derivation, advanced policy enforcement, and factor recovery. Together, these components convert multiple authentication factors into a cryptographic key, and serve as a replacement for PBKDFs.

This paper proposes using MFKDF to enhance the security and utility of classical DPGs. While DPGs have been around in some form since 2002, we believe the recent introduction of MFKDF has provided a critical tool for their practical use.

[1] In this case, the *witness* refers to the message used to authenticate (e.g., a 6-digit OTP), which is often not the same as the underlying shared secret.

Name	Last Updated	Hash Function (Cost)	Policies?	Revocation?	Flaws?
PasswordMaker [53]	08/2010	SHA1/MD5	Limited	No	
PasswordProtect [54]	07/2011	PBKDF2-SHA1 (10000)	No	No	
Password Hasher [49]	01/2012	SHA1	Limited	No	
Vault [80]	07/2012	PBKDF2-SHA1 (8)	Limited	No	
RndPhrase [70]	10/2012	CubeHash	No	No	
Magic [36]	12/2012	Custom	No	No	Yes[2]
BPasswd [12]	01/2013	bcrypt (64)	No	No	
My Password [40]	06/2013	MD5	No	No	
Hash Password [24]	07/2014	Custom	Limited	No	Yes[3]
SecPassGen [72]	08/2014	PBKDF2-SHA1 (10000)	No	No	
pastor [56]	08/2014	PBKDF2-SHA256 (1000)	No	No	
Passera [48]	09/2014	SHA512	Limited	No	
pwgen [64]	10/2014	MD5	No	No	
PswGen Toolbar [62]	11/2014	SHA512	No	No	
Extrasafe [18]	04/2015	SHA3	Limited	No	
determ-pwgen [16]	05/2015	SHA512	Limited	No	
HashPass [27]	06/2015	SHA1/MD5	No	No	
hash0 [25]	07/2015	PBKDF2-SHA256 (100000)	No	No	
vPass [83]	08/2015	TEA (10)	No	No	
Recall my password [68]	09/2015	SHA512	No	No	Yes[4]
MS [39]	10/2015	SHA1	Limited	No	
BPasswd2 [13]	11/2015	bcrypt (64)	Limited	No	
Domain [17]	01/2016	SHA1	No	No	
Password Maker X [50]	02/2016	SHA1/MD5	No	No	
UniPass [78]	03/2016	PBKDF2-SHA256 (4096)	No	No	
passwordgen [52]	04/2016	SHA256	No	No	
uPassword [79]	05/2016	SHA1	Limited	Local	
Pegasus [57]	11/2016	PBKDF2-SHA512 (555000)	Limited	Manual	Yes[5]
strongpass [73]	11/2016	scrypt (16384)	No	Manual	
Persistent Generator [59]	03/2017	MurmurHash	Limited	No	
PwdHash [63]	05/2017	HMAC-MD5	No	No	Yes[6]
Phashword [60]	10/2017	SHA1	Limited	No	
python-dpg [65]	10/2017	SHA256	No	No	
LastWord [35]	12/2017	SHA1	No	No	
MasterPassX [38]	02/2018	HMAC-SHA256	No	No	
Art [8]	03/2018	PBKDF2-SHA256 (100)	No	No	
Tresor [77]	12/2018	PBKDF2-SHA1 (8)	Limited	No	
mypass [41]	02/2019	Skein	Limited	Local	
PasswordShaker [55]	05/2019	scrypt (32768)	Limited	No	
Aprico [5]	06/2019	scrypt (16384)	No	No	
PasswordBuilder [51]	06/2019	scrypt (1024)	No	No	
CCTOO [14]	09/2019	scrypt (16384)	No	No	
Hasher Plus [26]	10/2019	HMAC-SHA1	Limited	No	
masterpassword [37]	03/2022	scrypt (32768)	No	No	Yes[7]
Passcrambler [47]	01/2023	MD5	Limited	No	

TABLE I. SURVEY OF 45 DETERMINISTIC PASSWORD GENERATION TOOLS AND EXTENSIONS.

[2]Magic Password Generator uses a custom hash function that is seemingly not a true one-way function.
[3]Hash Password Generator uses a custom hash function that is seemingly not a true one-way function.
[4]Recall my password sends the user's password to the author's website to perform the hashing on the backend.
[5]The PBKDF2 hash only uses the master password. That key is then combined with the site name via SHA512. This makes breaking PBKDF2 unnecessary.
[6]PwdHash requires you to enter the master password into a web page that loads external scripts and contains a basic XSS vulnerability.
[7]The scrypt hash only uses the username and master password. That key is then combined with the URL via SHA256. This makes breaking scrypt unnecessary.

III. Problem Statement

Our goal is to present a deterministic password generator (DPG) design that remedies the security, privacy, and usability issues of existing DPGs. In this section, we detail the specific goals and assumptions of the presented MFDPG construction.

A. Threat Model

Consider a DPG implemented as a client-side application or browser extension with optional server-side storage of settings, salts, and other trustless public parameters. We consider security under a *total data breach* threat model; i.e., at some instant, an adversary receives a snapshot of all materials stored on the client or server. The adversary may use all such data to attempt to violate one or more of the below security goals.

B. Security Goals

Our design must first and foremost satisfy the following standard properties of DPGs. Though existing DPGs use a master password as a sole input factor, we generalize these definitions to support a collection of input factors for forward-compatibility with our scheme. Let one such set of input factors (and corresponding output) be defined as "correct" for a given user. The desired properties are then as follows:

1) **Correctness** (Determinism). Given a static set of "correct" input factors, the scheme always outputs the same "correct" password for any target service; i.e., it is deterministic.

2) **Safety** (Pseudorandomness). Given an "incorrect" set of input factors, the scheme outputs an "incorrect" password for any target service except with negligible probability.

3) **Secretless**. While password managers can achieve the above properties through stateful storage of ciphertexts, a DPG should not store site-specific ciphertexts in any location, i.e., it is stateless other than public parameters.

Because the goal of this paper is to significantly improve upon the current state-of-the-art DPGs, we impose a few additional requirements on our MFDPG scheme:

4) **Brute-Force Resistance**. A brute-force attack on the master password should be insufficient to generate site-specific passwords. Adversaries that obtain a site-specific password should not easily be able to attack the master password.

5) **Compatibility**. The DPG should be compatible with a wide variety of password policies. Namely, the DPG should be able to generate passwords compliant with any regular password policy (i.e., representable by a regular expression).

6) **Revocability**. Users should, at any time, be able to revoke a password for a given service and generate a new one for the same service, without changing their input factors, and without manually remembering a counter value.

7) **Privacy**. In implementing the above properties, a DPG should not store site-specific values that allow any outside adversary to identify the services utilized by the DPG user.

IV. Proposed Approach

We now present our proposed design for a multi-factor deterministic password generator (MFDPG). We begin with an overview of our construction and then specifically address the issues of password policy support and revocation in dedicated sections. Our design places a focus on modularity, with any of these components being replaceable in future iterations.

A. Multi-Factor Password Generation

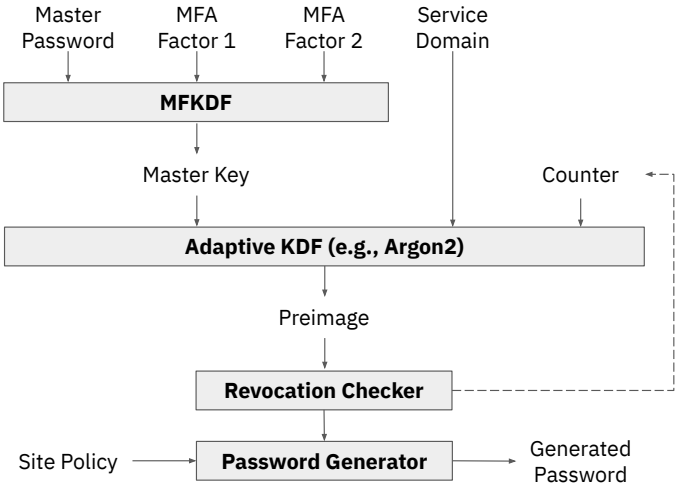

Fig. 1. Overview of multi-factor password generation architecture.

Fig. 1 shows an overview of our general approach for deriving a site-specific password from multiple authentication factors. The derivation algorithm (Alg. 1) follows 4 general steps (the detailed algorithms for revocation and generation in steps 3 and 4 are discussed in §IV-B and §IV-C, respectively):

1) Use MFKDF to derive a master key from multiple factors.
2) Use an adaptive KDF to derive a site-specific preimage.
3) Check if the preimage is revoked, and iterate until clear.
4) Convert preimage into a site policy-compliant password.

The most obvious difference with this approach in comparison with traditional DPGs is the use of MFKDF to support multiple authentication factors, including YubiKeys, HOTP, TOTP, etc., in addition to passwords. In some cases, doing so requires the storage of public parameters (e.g., salts or one-time pads) either locally or in the cloud. Importantly, these values are entirely trustless; for example, MFKDF-based cryptocurrency wallets [42] store them openly on blockchains.

Algorithm 1 MFDPG Algorithm

Require: M is MFKDF per Nair and Song [43]
Require: K is an adaptive KDF (e.g., Argon2 [10])
function MFDPG(factors, service, policy)
  master_key $\leftarrow \mathsf{M}$(factors)
  counter $\leftarrow 0$
  repeat
    counter $\leftarrow$ counter $+ 1$
    preimage $\leftarrow \mathsf{K}$(master_key $\odot$ service $\odot$ counter)
  until not CHECK(preimage)
  return GENERATE(preimage, policy)
end function

Using MFKDF rather than a PBKDF addresses the most significant issue with current PRGs: brute-force attack susceptibility. Because of the "exponential entropy" property of MFKDF, adversaries can no longer use a site-specific password to attack the user's master password. Instead, they would have to simultaneously crack all of a user's factors, a task that is significantly harder than guessing their master password alone.

An additional advantage of using MFKDF in this context is its support for factor recovery. MFKDF presents a threshold variant that allows 3 authentication factors to be established, any 2 of which can be used to derive the key. When using this variant, the resulting MFDPG construction could allow users to recover from the loss of a single authentication factor without losing access to all of their generated passwords, as is the case in present password-only DPGs.

B. Revocation Algorithm

Another key difficulty with current PRGs is the lack of revocation support. As discussed in §II-B, websites often implement password rotation policies that require users to update their password occasionally. Our goal is to support revocation in a way that does not store information about the services a user has accounts on, but also does not require users to remember and manually enter a unique revocation counter value for each service. To achieve this, we suggest using a Cuckoo filter [19] or a similar set membership data structure.

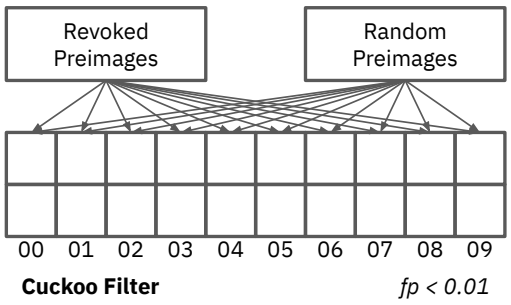

Fig. 2. Fixed-capacity Cuckoo filter for private credential revocation support.

Our suggested approach, illustrated in Fig. 2, involves filling a Cuckoo filter with both true, revoked preimages and deterministic fictitious preimages, such that the total number of entries remains static. These true and fictitious entries should be indistinguishable, such that the number of true revoked items in the set should not be discernable to an adversary with access to the Cuckoo filter but not the user's master key.

In addition to its time and space efficiency, the use of a probabilistic set membership data structure has the advantage of not revealing its exact members. When correctly tuned, this approach is likely to further frustrate brute-force attacks, with a high false-positive rate for attackers, while still posing sufficiently low friction for users. As a result, users can revoke credentials as needed. Unlike current DPGs, they may do so without storing information that reveals the services they use, or even the number of revocations that they have performed. The exact revocation algorithm is detailed in Alg. 2.

Algorithm 2 MFDPG Revocation Algorithms

Require: S is a Cuckoo filter per Fan et al. [19]
Require: H is a cryptographic hash function (e.g., SHA256)
Require: N is the maximum number of allowed revocations

function SETUP(master_key)
  for i **in** $[0 \dots \mathsf{N}]$ **do**
    S.add(H(master_key $\odot$ i))
  end for
end function
function REVOKE(master_key, preimage)
  S.add(preimage)
  for i **in** $[0 \dots \mathsf{N}]$ **do**
    if S.has(H(master_key $\odot$ i)) **then**
      S.remove(H(master_key $\odot$ i))
      return
    end if
  end for
end function
function CHECK(preimage)
  return S.has(preimage)
end function

C. Password Generation Algorithm

Finally, we discuss the issue of password policy support. As services enforce a wide array of seemingly arbitrary requirements on password length, complexity, character sets, etc., the final step of converting a site-specific preimage into a generated password must be sufficiently flexible to ensure compatibility with a variety of services.

Rather than implementing an obscure password policy notation standard such as the Password Policy Markup Language (PPML) [28] or NIST 800-63-3 [21], we chose to support any regular password policy (i.e., any policy that can be expressed as a regular expression), as this includes all instances of the former, as well as most conceivable realistic password policies.

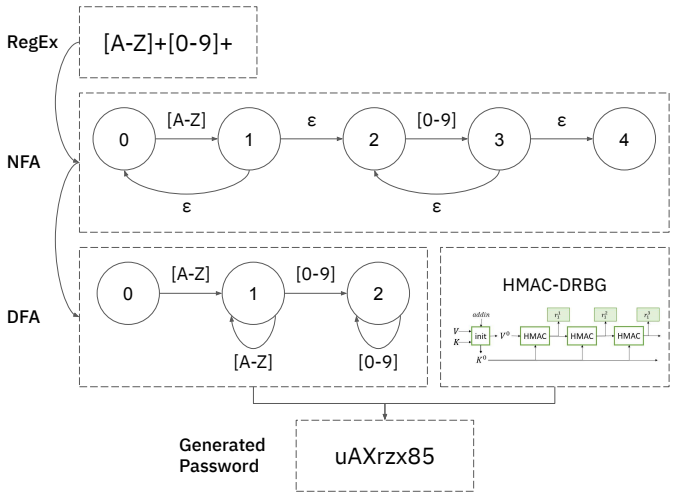

Fig. 3. Algorithm for generating random instances of regular password policy.

To support all regular password policies, we suggest using a random regular DFA traversal algorithm such as Xeger [74] as shown in Fig. 3. First, a password policy, stored as a regular expression, is converted into an NFA using the

McNaughton–Yamada–Thompson algorithm [75]. That NFA is, in turn, converted into a DFA using the subset construction algorithm [66]. Finally, a DFA traversal algorithm, such as Xeger [74], performs a random walk of the generated DFA. The source of randomness is overridden to use a cryptographically secure PRNG, such as HMAC-DRBG [9], seeded using the preimage for a given service. Thus, for a given service, using the same preimage and password policy, the same password will always be generated.

The composition of these methods is shown in Alg. 3. According to our survey of 45 existing DPGs in §II-B, we believe we are the first to propose this general approach.

Algorithm 3 MFDPG Password Generation Algorithm

Require: G is a random regex DFA traverser (e.g., Xeger [74])
Require: R is HMAC-DRBG per NIST SP 800-90 [9].
1: **function** GENERATE(preimage, policy)
2: **return** G(policy, R(preimage))
3: **end function**

V. EVALUATION

We evaluate the proposal on three grounds. First, we implement and benchmark MFDPG in a typical use case. Next, we systematically verify its compatibility with a large number of line services. Finally, we present semi-formal security arguments to demonstrate that our scheme should, in theory, satisfy the desired security properties of §III.

A. Implementation

To demonstrate the immediate practical utility of MFDPG and provide a blueprint for its deployment, we implemented a fully-featured open-source MFDPG JavaScript library, which is offered under a BSD license. Our implementation supports all of the previously discussed features, including multi-factor authentication, portability, revocation, and password generation based on regular expressions. During our evaluation, we interacted with MFDPG using a simple command-line interface. However, the library could also readily be used to produce a website, browser extension, mobile application, desktop program, or other means of accessing MFDPG.

B. Performance

To evaluate the performance of MFDPG in a practical setting, we benchmarked our JavaScript implementation using Node.js v16.15.0 on Windows 10 v21H2.

We used Argon2id as the underlying KDF, with p=1, t=2, and m=24576. Our test device contains an AMD Ryzen 9 5950X (16-core, 3.4 GHz) processor and 128 GB of system memory. However, only single-thread performance is relevant with the chosen KDF parameters, and significantly less than 1 GB of system memory is ever utilized.

We chose to set the maximum allowed revocations (labeled N in Alg. 2) to 4096, and configured the underlying Cuckoo filter to have a false positive rate of 0.0001.

To benchmark the performance of MFDPG, we performed the following operations, in sequence:

1) Create a new MFDPG instance with three factors.
2) Export the public parameters as a string.
3) Reload the MFDPG instance using the same three factors.
4) Generate a password for a service using a regular policy.
5) Revoke the newly generated password.

This process was then repeated 100 times. The time taken to perform each step is shown as a box plot in Fig. 4. The results show that no individual operation took more than 100 ms on average. Thus, the latency of MFDPG is well within the bounds that most users will comfortably tolerate [6].

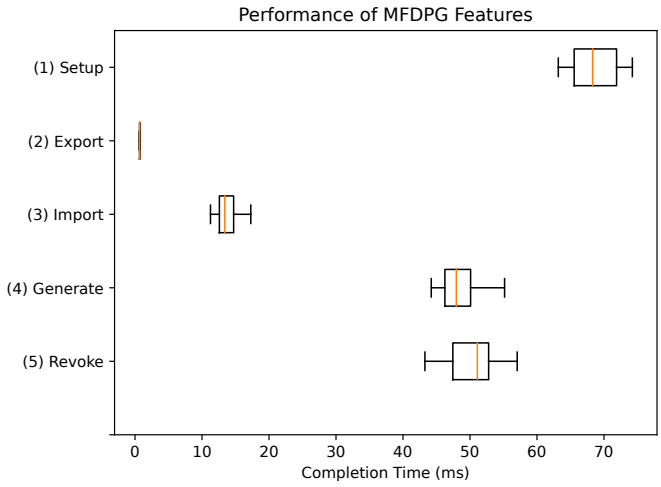

Fig. 4. Benchmarking results of each tested MFDPG feature (box plot).

C. Compatibility

Next, we evaluated our MFDPG implementation from the perspective of compatibility with popular websites and their respective password policies. As the additional flexibility granted by support for all regular password policies is amongst our claimed contributions, we felt it necessary to demonstrate that this approach does indeed enhance the practicality of the resulting DPG. Thus, we tested the process of creating an account for each of the 100 most popular websites on the internet, according to their Alexa rank as of Feb 1, 2023 [3].

We found that MFDPG was indeed flexible enough to support the password policies of all 100 evaluated services. For instance, Apple's password policy does not allow three or more consecutive identical characters [4], a constraint that MFDPG readily handled but that no existing DPG of the 45 surveyed in §II-B could correctly enforce.

The ease of representing all encountered password policies as regular expressions suggests that a practical deployment of MFDPG could simply utilize a central database of password policies to ensure that generated passwords are automatically compliant with all popular services without requiring any manual intervention of the end user.

D. Security Arguments

We conclude our evaluation with brief arguments for why the MFDPG scheme of §IV, as implemented herein, satisfies our original security goals in §III-B. While a formal proof framework for multi-factor key derivation exists [33], [43], there is no clear equivalent for deterministic password generation. As such, our arguments are instead based on a semi-formal reduction to the security properties of the underlying cryptographic primitives, such as MFKDF and Argon2.

1) **Correctness**. We first argue for the correctness (determinism) of the password generation algorithm. The Thompson algorithm [75] and subset construction algorithm [66] are both completely deterministic, i.e., they will always generate the same DFA given the same regular expression. The Xeger algorithm [74] performs a walk of the DFA, with each decision made according to the output of a PRG. When using HMAC-DRBG [9] as the underlying PRG, the output is deterministic when using a static HMAC key. Thus, given the same HMAC key (preimage) as input, Xeger always performs the same traversal of the DFA and thus generates the same password as the output string.

   Now, given the correctness of the password generation scheme, the correctness of MFDPG reduces to the ability to generate the same preimage for a given service using a fixed set of factors. This, in turn, is guaranteed by the "correctness" property of MFKDF and by the determinism of the underlying KDF such as Argon2.

2) **Safety**. Similarly, we begin by arguing the safety (pseudorandomness) of the password generation algorithm. We note that HMAC-DBRG [9] is a cryptographically secure PRNG; it has been proven, with machine-checked proofs, that its output is indistinguishable from random [84]. It follows that HMAC-DBRG inherits the avalanche property of HMAC: if the preimage is changed slightly (e.g, flipping a single bit), the generated bits change significantly (e.g., half the bits flip). Thus, any change to the preimage will result in a significantly different walk of the DFA, and will thus generate a significantly different password (except with negligible probability w.r.t. the policy search space).

   Given the safety of the password generation scheme, the safety of MFDPG depends on generating a different preimage for a given service when using different factors. This, in turn, is guaranteed by the "safety" property of MFKDF to generate a different master key if the factors change, and by the "collision resistance" of the underlying KDF to generate a different preimage if the master key changes.

3) **Secretless**. We note that the MFDPG scheme does not store site-specific ciphertexts in any location. In fact, the scheme only suggests two persisted values: a Cuckoo filter for revocation, and the public parameters for dynamic MFKDF factors, such as HOTP, TOTP, and YubiKey.

   The former contains hashes of revoked credentials, but cannot easily be reversed to reveal the actual credentials due to the one-way nature of cryptographic hash functions. Even if somehow reversed, it would not be possible to test which service the credential belongs to without knowing the master key due to the one-way nature of the KDF used. The latter is trustless (and in fact, can be stored on a public [42]), and in any case, is not site-specific.

   In our construction, the cardinality of the Cuckoo filter is static regardless of the number of revoked credentials. Thus, there are no persisted ciphertexts or other values that reveal any information about the services utilized by the user or even the number of services ever utilized or revoked.

4) **Brute-Force Resistance**. First, we note that adversaries cannot directly reverse a site-specific password to obtain the master key or authentication factors, due in part to the one-way nature of the KDF, such as Argon2 [10].

   Moreover, adversaries can no longer use a site-specific password to perform a brute-force attack on the master password. Due to the "exponential security" of MFKDF, attackers must now simultaneously guess every possible combination of input factors, and cannot attack any input factor individually (see Nair and Song Thm. 2 [43]).

5) **Compatibility**. Next, we argue that MFDPG is able to generate passwords compliant with any regular password policy. First, the Thompson algorithm [75] can convert any regular expression to an NFA, and any NFA can be converted to its equivalent DFA [66]. When performing a random walk of that DFA, the Xeger [74] algorithm will always end on an "accept" state, and will thus always produce a string that matches the original regular expression.

6) **Revocability**. When revoking a credential in MFDPG, the preimage corresponding to that credential is added to the Cuckoo filter. When generating that credential, the preimage will be found to be in the Cuckoo filter (with $p = 1$), and a new preimage will be generated. As discussed in relation to the safety property, changing the preimage will result in a different password being generated (except with negligible probability). Thus, revoking a password guarantees that the next password generated will be different (except with negligible probability), and MFDPG achieves this without requiring users to change their input factors or manually remember a counter value.

7) **Privacy**. In implementing the above properties, MFDPG does not store any site-specific values, as discussed in relation to the "secretless" property. Thus, MFDPG does not store site-specific values that allow any outside adversary to identify the services utilized by its user.

VI. Discussion

Since their introduction in 2002, deterministic password generators have been categorically rejected by consumers and researchers alike as being a viable alternative to conventional password managers, with even the most popular DPGs having a negligible market share of overall password management usage. Our survey of 45 existing DPGs shows that this decision is not entirely without merit, with current DPGs demonstrating an array of issues hindering their security, privacy, and usability.

It does not, however, follow from this that DPGs should be entirely discarded as a meritorious field of study. In light of new cryptographic primitives, we hope to revive academic interest in DPGs as a password management approach.

While the MFDPG does not necessarily rise to the level of a production-ready system, it nevertheless constitutes a significant improvement over existing DPG proposals. By strategically combining relevant data structures, such as Cuckoo filters, algorithms, such as DFA traversal, and cryptographic primitives, such as MFKDF, we have presented a serious attempt at building a modern-day DPG, and hope to demonstrate that most of the issues surrounding DPGs are not intrinsic, and can in fact be addressed by diligent engineering.

In light of the recent massive data breach associated with LastPass [76] and previous string of incidents associated with major password managers [45], there has been an uptick in academic research and discussion around secure cryptography and system designs for password management. Due to the fundamental appeal of not storing passwords, neither in plaintext nor encrypted, in any location, we hope DPGs, such as MFDPG, remain a part of this conversation.

One final advantage of MFDPG worthy of highlighting is its ability to progressively upgrade weak password-only websites to effectively use strong MFA, simply by using an MFDPG-generated password on the client side. When configured with authentication factors like HOTP, TOTP, or YubiKeys, the master key, and thus the site-specific password, cannot be derived without using multiple strong authentication factors, even if the site itself does not accept factors other than the generated password. Moreover, MFDPG links the ability to derive the site-specific password to the correctness of these multiple factors in a strong cryptographic sense, unlike password managers that may support MFA, but ultimately encrypt credentials with a password-derived key. This, on its own, constitutes a significant advantage of MFDPG, particularly for those who frequently interact with legacy systems with limited multi-factor authentication support, but still require trustless cryptographic enforcement of multiple strong factors.

A. Limitations

There are a number of trade-offs associated with MFDPG, both in comparison with prior DPGs, as well as in comparison with conventional password managers.

First, unlike some DPGs, MFDPG is not completely stateless, storing both a Cuckoo filter for revocation and public parameters for certain MFKDF factors. This is a necessary concession to support multi-factor authentication and password revocation, and because these parameters require no trust assumptions, they can be stored locally or in the cloud without weakening the security of the scheme. Still, the need to store any information at all may be viewed as a drawback compared to the simplest forms of deterministic password generation. Also, while not vulnerable to a breach of the stored materials, the system may still be vulnerable to active spyware that can obtain the master key from system memory; however, this limitation likely applies to DPGs and password managers alike.

With respect to revocation, the use of Cuckoo hashing also has potential drawbacks, being a probabilistic data structure. Namely, there is a potential for a credential revocation to incorrectly revoke another credential that is actively in use. Correctly configuring the parameters can make the probability of this small, but non-zero. Alternatively, the Cuckoo filter could be replaced with another construction that supports set addition, subtraction, and membership testing.

Furthermore, if a credential has indeed been revoked, the current construction suggests repeating the KDF invocation to create a new preimage. While this is advantageous from a security perspective, in the event that a credential has been revoked many times, generating that credential may be very slow due to the need to repeatedly invoke the adaptive KDF.

Finally, the MFDPG scheme depends on the security properties of MFKDF, and thus also inherits many of its limitations. In particular, MFKDF supports a limited set of authentication factors, and implementing new factors requires a specific, purpose-built factor constriction. While the supported factors include popular factors like HOTP, TOTP, YubiKey, etc., schemes relying on MFKDF may not be able to support as many authentication factors as a typical centralized password manager. Further, these factors must be entered and verified simultaneously, rather than sequentially, in order to achieve the "exponential security" of MFKDF. While this provides a significant security advantage in resisting brute-force attacks, it may somewhat degrade the usability of the system.

Some of the limitations presented in this section are intrinsically implicated in deterministic password generation, but many can be rectified, at least in part, with further cryptographic or system design improvements over time.

B. Future Work

Currently, the generation algorithm requires a password policy to be specified as a regular expression. While these policies can be saved for popular services in a central database, the significant effort would be required to update and maintain such a database, and less popular services would inevitably be excluded. In the future, researchers could investigate an approach that automatically determines the password policy for a given service, such as through the use of language models.

The revocation component of the system could also potentially be improved. For example, some form of multi-party computation, such as a private set intersection algorithm, could perhaps be used to communicate with a central server to check whether a credential has been revoked on any device without revealing the identity of the credential being checked.

One major criticism of DPG schemes not discussed thus far is the lack of support for existing credentials. Currently, DPG systems, including MFDPG, have a high upfront adoption cost, as they essentially require a user to change every password for all of their existing services to the DPG-generated password. Instead of this, a hybrid implementation, supporting both deterministic password generation and conventional password storage may be more user-friendly in the short term.

Finally, we hope to see usable security or HCI research that evaluates the usability of DPGs, such as MFDPG, in comparison with conventional password managers, via a controlled user study. As usability is likely as significant a factor in hindering DPG adoption as are security and privacy concerns, further advancements in this area are greatly encouraged.

VII. Conclusion

In our brief survey covering over 20 years of DPG history, we identified a number of key issues hindering the mainstream adoption of DPGs, despite their theoretically strong advantage of not storing any site-related secrets. In this paper, we have chosen to view these problems as opportunities for improvement rather than disqualifying the field of DPG writ large.

MFDPG completely solves some of the major drawbacks of existing DPGs, such as password policy compatibility and multi-factor authentication support, and at a minimum, makes significant progress towards remedying other flaws, such as revocability and brute-force susceptibility. By cryptographically incorporating the entropy of multiple strong authentication factors into the password generation process, it also has the effect of turning any password-based authentication flow into a secure multi-factor authenticated login process.

In light of recent major security incidents with popular password managers, researchers are rightly taking a second look at the way we handle password management. While there might be a temptation to ignore DPGs as an outdated approach, we hope this work serves as a launching point for further research into secure, usable DPGs, and believe DPGs should remain an important part of this conversation moving forward.

Availability

Our fully-functional JavaScript implementation of MFDPG, with support for multi-factor authentication, portability, revocation, and password generation based on regular expressions, is available here under a BSD license:

https://github.com/multifactor/mfdpg

Acknowledgments

We appreciate the advice of Conor Gilsenan. This work was supported by the National Science Foundation, the National Physical Science Consortium, the Fannie and John Hertz Foundation, and the Center for Responsible, Decentralized Intelligence. Any opinions, findings, and recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the supporting entities.

References

[1] 1Password, "1Password Security Design," 2021. [Online]. Available: https://1passwordstatic.com/files/security/1password-white-paper.pdf

[2] Akamai, "2020 state of the internet." [Online]. Available: https://www.akamai.com/site/en/documents/state-of-the-internet/soti-security-credential-stuffing-in-the-media-industry-report-2020.pdf

[3] "Alexa Top Websites | Last Save before it was closed." [Online]. Available: https://www.expireddomains.net/alexa-top-websites/

[4] "Apple - Dumb Password Rules." [Online]. Available: https://dumbpasswordrules.com/sites/apple/

[5] "Aprico." [Online]. Available: https://addons.mozilla.org/en-US/firefox/addon/aprico-free-password-manager//

[6] I. Arapakis, S. Park, and M. Pielot, "Impact of Response Latency on User Behaviour in Mobile Web Search," in *Proceedings of the 2021 Conference on Human Information Interaction and Retrieval*, Mar. 2021, pp. 279–283, arXiv:2101.09086 [cs]. [Online]. Available: http://arxiv.org/abs/2101.09086

[7] T. Arcieri, "4 fatal flaws in deterministic password managers," Nov. 2016. [Online]. Available: http://tonyarcieri.com/4-fatal-flaws-in-deterministic-password-managers

[8] "Art." [Online]. Available: https://addons.mozilla.org/en-US/firefox/addon/art-password/

[9] E. Barker and J. Kelsey, "Recommendation for Random Number Generation Using Deterministic Random Bit Generators," National Institute of Standards and Technology, Tech. Rep. NIST Special Publication (SP) 800-90A Rev. 1, Jun. 2015. [Online]. Available: https://csrc.nist.gov/publications/detail/sp/800-90a/rev-1/final

[10] A. Biryukov, D. Dinu, and D. Khovratovich, "Argon2: New generation of memory-hard functions for password hashing and other applications," in *IEEE EuroS&P*, 2016, pp. 292–302.

[11] "Bitwarden open source password manager — bitwarden." [Online]. Available: https://bitwarden.com

[12] "Bpasswd." [Online]. Available: https://web.archive.org/web/20181102000510/https://addons.mozilla.org/en-US/firefox/addon/bpasswd/

[13] "Bpasswd2." [Online]. Available: https://web.archive.org/web/20181102202022/https://addons.mozilla.org/en-US/firefox/addon/bpasswd2//

[14] "Cctoo." [Online]. Available: https://addons.mozilla.org/en-GB/firefox/addon/cctoo/

[15] Dashlane, "Security white paper," Mar. 2021. [Online]. Available: https://www.dashlane.com/download/whitepaper-en.pdf

[16] "determ-pwgen." [Online]. Available: https://github.com/I3ck/determ-pwgen

[17] "Domain." [Online]. Available: https://web.archive.org/web/20181102054736/https://addons.mozilla.org/en-US/firefox/addon/domain-password-generator/

[18] "Extrasafe." [Online]. Available: https://web.archive.org/web/20181102205616/https://addons.mozilla.org/en-US/firefox/addon/extrasafe/

[19] B. Fan, D. G. Andersen, M. Kaminsky, and M. D. Mitzenmacher, "Cuckoo filter: Practically better than bloom," in *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, 2014, pp. 75–88.

[20] D. Florencio and C. Herley, "A Large Scale Study of Web Password Habits," Microsoft, Tech. Rep. MSR-TR-2006-166, Nov. 2006. [Online]. Available: https://www.microsoft.com/en-us/research/publication/a-large-scale-study-of-web-password-habits/

[21] P. A. Grassi, M. E. Garcia, and J. L. Fenton, "Draft nist special publication 800-63-3 digital identity guidelines," *National Institute of Standards and Technology, Los Altos, CA*, 2017.

[22] A. Hanamsagar, S. S. Woo, C. Kanich, and J. Mirkovic, "How Users Choose and Reuse Passwords," p. 16.

[23] T. Hansen and D. E. Eastlake 3rd, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)," Internet Engineering Task Force, Request for Comments RFC 6234, May 2011. [Online]. Available: https://datatracker.ietf.org/doc/rfc6234

[24] "Hash password." [Online]. Available: https://web.archive.org/web/20181102030655/https://addons.mozilla.org/en-US/firefox/addon/hash-password-generator/

[25] "hash0." [Online]. Available: https://web.archive.org/web/20181103000530/https://addons.mozilla.org/en-US/firefox/addon/hash0/

[26] "Hasher plus." [Online]. Available: https://addons.mozilla.org/en-US/firefox/addon/password-hasher-plus/

[27] "Hashpass." [Online]. Available: https://web.archive.org/web/20181101222025/https://addons.mozilla.org/en-US/firefox/addon/hashpass-firefox/

[28] M. Horsch, M. Schlipf, S. Haas, J. Braun, and J. Buchmann, "Password policy markup language," 2016.

[29] P. Identity, "What is Out-of-Band Authentication (OOBA)?" [Online]. Available: https://www.pingidentity.com/en/resources/blog/post/what-is-out-of-band-authentication-ooba.html

[30] B. Kaliski, "PKCS #5: Password-Based Cryptography Specification Version 2.0," Internet Engineering Task Force, Request for Comments RFC 2898, Sep. 2000, num Pages: 34. [Online]. Available: https://datatracker.ietf.org/doc/rfc2898

[31] A. H. Karp, "Site-specific passwords," *HP Labs Technical Report*, 2002.

[32] "Keepassx." [Online]. Available: https://www.keepassx.org/

[33] H. Krawczyk, "Cryptographic Extraction and Key Derivation: The HKDF Scheme." [Online]. Available: https://eprint.iacr.org/2010/264

[34] LastPass, "Our Zero-Knowledge Security Model." [Online]. Available: https://www.lastpass.com/security/zero-knowledge-security

[35] "Lastword." [Online]. Available: https://github.com/LoneFry/LastWord

[36] "Magic." [Online]. Available: https://github.com/arantius/magic-password-generator

[37] "masterpassword." [Online]. Available: https://addons.mozilla.org/en-GB/firefox/addon/masterpassword-firefox/

[38] "Masterpassx." [Online]. Available: https://chrome.google.com/webstore/detail/masterpassx/acocljodaoecblhjggkadfhnbjcfgbbb

[39] "Ms." [Online]. Available: https://web.archive.org/web/20171013225154/https://addons.mozilla.org/en-US/firefox/addon/strong-password-generator//

[40] "My password." [Online]. Available: https://web.archive.org/web/20181101221606/https://addons.mozilla.org/en-US/firefox/addon/my-password/

[41] "mypass." [Online]. Available: https://github.com/anoma-/mypass

[42] V. Nair and D. Song, "Decentralizing custodial wallets with mfkdf," 2023.

[43] ——, "Multi-factor key derivation function (mfkdf) for fast, flexible, secure, & practical key management," *USENIX Security '23*, 2023.

[44] NIST, "FIPS 197, Advanced Encryption Standard (AES)," p. 51.

[45] K. Norton, "LastPass: Is it a Safe Password Manager?" Jul. 2021. [Online]. Available: https://www.esecurityplanet.com/products/lastpass-review/

[46] W. Palant, "Security considerations for password generators," Apr. 2016, section: articles. [Online]. Available: https://palant.info/2016/04/20/security-considerations-for-password-generators/

[47] "Passcrambler." [Online]. Available: https://github.com/hasherezade/password_scrambler

[48] "Passera." [Online]. Available: https://web.archive.org/web/20181102003104/https://addons.mozilla.org/en-US/firefox/addon/passera/

[49] "Password hasher." [Online]. Available: https://web.archive.org/web/20181002124942/https://addons.mozilla.org/en-US/firefox/addon/password-hasher/

[50] "Password maker x." [Online]. Available: https://web.archive.org/web/20181102214310/https://addons.mozilla.org/en-US/android/addon/password-maker-x/

[51] "Passwordbuilder." [Online]. Available: https://addons.mozilla.org/en-GB/firefox/addon/uniquepasswordbuilder-addon/

[52] "passwordgen." [Online]. Available: https://addons.mozilla.org/en-US/firefox/addon/passwordgen-for-firefox-1/

[53] "Passwordmaker." [Online]. Available: https://web.archive.org/web/20181002125051/https://addons.mozilla.org/en-US/firefox/addon/passwordmaker/

[54] "Passwordprotect." [Online]. Available: https://web.archive.org/web/20181102194105/https://addons.mozilla.org/en-US/firefox/addon/passwordprotect/

[55] "Passwordshaker." [Online]. Available: https://addons.mozilla.org/en-US/firefox/addon/passwordshaker/

[56] "pastor." [Online]. Available: https://github.com/appnician/pastor

[57] "Pegasus." [Online]. Available: https://addons.mozilla.org/en-US/firefox/addon/pegasus-password-generator/

[58] C. Percival, "Stronger Key Derivation via Sequential Memory-Hard Functions," p. 16. [Online]. Available: https://www.researchgate.net/publication/252853607_Stronger_key_derivation_via_sequential_memory-hard_functions

[59] "Persistent generator." [Online]. Available: https://web.archive.org/web/20181101232611/https://addons.mozilla.org/en-US/firefox/addon/sebres-pwd-hasher//

[60] "Phashword." [Online]. Available: https://web.archive.org/web/20181102233200/https://addons.mozilla.org/en-US/firefox/addon/phashword/

[61] N. Provos and D. Mazières, "A Future-Adaptable password scheme," in *1999 USENIX Annual Technical Conference (USENIX ATC 99)*. Monterey, CA: USENIX Association, Jun. 1999. [Online]. Available: https://www.usenix.org/conference/1999-usenix-annual-technical-conference/future-adaptable-password-scheme

[62] "Pswgen toolbar." [Online]. Available: https://web.archive.org/web/20181102012502/https://addons.mozilla.org/en-US/firefox/addon/pswgen-toolbar/

[63] "Pwdhash." [Online]. Available: https://addons.mozilla.org/en-US/firefox/addon/pwdhash/

[64] "pwgen." [Online]. Available: https://github.com/quintopia/pwgen

[65] "python-dpg." [Online]. Available: https://github.com/psmiraglia/python-dpg

[66] M. O. Rabin and D. Scott, "Finite automata and their decision problems," *IBM journal of research and development*, vol. 3, no. 2, pp. 114–125, 1959.

[67] R. Ranjan, "Password Spraying Attack | OWASP Foundation." [Online]. Available: https://owasp.org/www-community/attacks/Password_Spraying_Attack

[68] "Recall my password." [Online]. Available: https://web.archive.org/web/20181002161104/https://addons.mozilla.org/en-US/firefox/addon/recall-my-password

[69] D. Reichl, "KeePass Password Safe." [Online]. Available: https://keepass.info/

[70] "Rndphrase." [Online]. Available: https://web.archive.org/web/20181102013645/https://addons.mozilla.org/en-US/firefox/addon/rndphrase/

[71] B. Ross, C. Jackson, N. Miyake, D. Boneh, and J. C. Mitchell, "Stronger password authentication using browser extensions." in *USENIX Security Symposium*, vol. 17. Baltimore, MD, USA, 2005, p. 32.

[72] "Secpassgen." [Online]. Available: https://web.archive.org/web/20181103002640/https://addons.mozilla.org/en-US/firefox/addon/secpassgen//

[73] "strongpass." [Online]. Available: https://github.com/grempe/strongpass

[74] X. Team, "Xeger string generator," 2009.

[75] K. Thompson, "Programming techniques: Regular expression search algorithm," *Communications of the ACM*, vol. 11, no. 6, pp. 419–422, 1968.

[76] K. Toubba, "Notice of Recent Security Incident," Dec. 2022. [Online]. Available: https://blog.lastpass.com/2022/12/notice-of-recent-security-incident/

[77] "Tresor." [Online]. Available: https://github.com/mstum/TresorLib/

[78] "Unipass." [Online]. Available: https://web.archive.org/web/20181103011439/https://addons.mozilla.org/en-US/firefox/addon/unipass/

[79] "upassword." [Online]. Available: https://web.archive.org/web/20181102223643/https://addons.mozilla.org/en-US/firefox/addon/upassword/

[80] "Vault." [Online]. Available: https://web.archive.org/web/20181102032353/https://addons.mozilla.org/en-US/firefox/addon/vault/

[81] M. View, D. M'Raihi, F. Hoornaert, D. Naccache, M. Bellare, and O. Ranen, "HOTP: An HMAC-Based One-Time Password Algorithm," Internet Engineering Task Force, Request for Comments RFC 4226, Dec. 2005, num Pages: 37. [Online]. Available: https://datatracker.ietf.org/doc/rfc4226

[82] M. View, J. Rydell, M. Pei, and S. Machani, "TOTP: Time-Based One-Time Password Algorithm," Internet Engineering Task Force, Request for Comments RFC 6238, May 2011. [Online]. Available: https://datatracker.ietf.org/doc/rfc6238

[83] "vpass." [Online]. Available: https://web.archive.org/web/20181102055008/https://addons.mozilla.org/en-US/firefox/addon/vpass-password-generator/

[84] K. Ye, "The notorious prg: Formal verification of the hmac-drbg pseudorandom number generator," 2016.

[85] Yubico, "Yubikey: Strong two-factor authentication." [Online]. Available: https://www.yubico.com/