Passwords and Cryptwords

The Final Limits on Lengths

Michael Clark
Brigham Young University
Provo, UT, USA
clark.michael.c@gmail.com

Kent Seamons
Brigham Young University
Provo, UT, USA
seamons@cs.byu.edu

ABSTRACT

Computers get faster every year; brains don't. Passwords and other memorized credentials have unique usability advantages over tokens and biometrics, so we desire to design secure systems that maintain lengths that users can memorize. Some passwords are subject primarily to online attacks, and are simple to defend with rate limits and lockouts. Others, used to generate encryption keys, must be secure against offline attacks. We coin the term "cryptword" to distinguish these from passwords subject primarily to online attacks.

Authentication passwords do not need to get longer as computers get faster, if protected by rate limits and lockouts. Using password key derivation functions (pwKDFs) — a class of preexisting cryptographic algorithms — we show that cryptwords can also remain the same length and maintain their security strength despite advances in computation. We achieve this by regularly updating the pwKDF parameters and regenerating the derived key from the cryptword. In cases where it is not possible to meaningfully regenerate the derived key, such as archival data or public verifiers, cryptword lengths should be chosen to last the lifetime of the data.

We provide simple equations that end users and system administrators can use to determine minimal assigned password and cryptword lengths based on personal threat models. We also show how to use the capabilities of cloud computing providers to estimate attacker costs. These same equations give a timeframe for cryptword and secret rotation once the encrypted data leaks. Because these equations do not rely on the current date or current hardware capabilities, they show that if regularly used, password and cryptword lengths can remain constant despite improvements in hardware.

CCS CONCEPTS

• **Security and privacy** → **Authentication**; **Usability in security and privacy**; **Key management**.

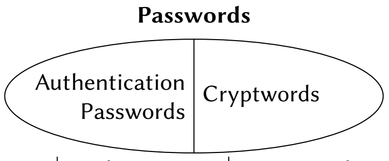
KEYWORDS

passwords, cryptwords, authentication

ACM Reference Format:
Michael Clark and Kent Seamons. 2022. Passwords and Cryptwords: The Final Limits on Lengths. In *New Security Paradigms Workshop (NSPW '22), October 24–27, 2022, North Conway, NH, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3584318.3584324

1 INTRODUCTION

There are two kinds of passwords — those that secure against online attacks and are suitable for authentication, and those that secure against offline attacks and are suitable for encrypting data [29]. For this second kind we coin the term *cryptword* (see Section 6.2) to distinguish them from passwords that enable you to authenticate to another party ("pass"). Figure 1 highlights some key differences in purpose, attack types, and sufficient entropy between authentication password and cryptwords.



	Authentication Passwords	Cryptwords
Purpose	Authentication	Encryption
Attacks	Online	Offline
Entropy	13–47 bits	Typically > 48 bits

Figure 1: Two Kinds of Passwords

Users often pick their own passwords via an internal process. An *assigned password* is any password the user accepts which they did not choose. A software system or organization may assign a password to a user or a user may generate a random password and accept it. While there are examples of non-random assigned passwords (i.e., some cases of password sharing in an organization), our focus is on the security properties of randomly generated passwords. As such, in this paper we treat assigned passwords as synonymous with randomly generated passwords.

To provide security guarantees, both authentication passwords and cryptwords should be generated randomly with equal probability from a pool of possible passwords. When they are, the best chance of a successful guess is the inverse of the number of possible passwords. The Shannon entropy of a randomly generated password measures the guessability, and throughout this paper we will refer to bits (of Shannon entropy) as a measure of security.

When we use the term "password" in this paper, we also refer more generally to passphrases, PINs, and similar keyboard-entered "something you know" tokens. We don't address user-chosen passwords, which follow a Zipf's law distribution [64].

NSPW '22, October 24–27, 2022, North Conway, NH, USA

A *cryptword* is any password that is used to generate a cryptographic key. Cryptwords should[1] be randomly generated as discussed in the preceding paragraph. People use cryptwords in at least the following settings:

- Data at rest
  - Full disk encryption
  - File encryption, like password managers, zip files, or SSH keys stored on disk
- Data in use: Shared verifier
  - Cryptocurrency "Brain Wallet"s, which generate all wallet encryption keys from a user-entered secret
  - Stateless Password Managers (see Section 6.7)
- Data in motion: Communicating cryptographic keys[2]
  - Wi-Fi passwords, specifically WPA-PSK and WPA2-PSK[3]

In effect, cryptwords turn "something you have" (an encryption key) into "something you know" (the cryptword). Cryptwords enable key portability with nothing to carry, and as an alternate means of secure key management potentially remove the need for a Hardware Security Module (HSM).

Attackers commonly use computers to generate numerous password guesses and to check if those guesses are correct. For an *online attack*, an attacker must check their guess by communicating it to another party. Defenders who receive these guesses can use rate limiting and lockouts to guard against online attacks [36].

In an *offline attack*, the attacker's computer can check the password without communicating with any other party. Offline attacks are typically used against password databases where the passwords have been replaced by password verifiers (see Section 2.2.1). When attacking cryptwords, the attacker (1) generates guesses, (2) converts these into trial encryption keys, and (3) attempts decrypting secret information serving as a verifier. Successful decryption means the cryptword guess was correct. A difference from attacking password verifiers is that the encrypted data serves as a verifier instead. These are *known ciphertext* attacks, since the attacker only has access to the encrypted data.

1.1 The Problem

Computers get faster every year; brains don't. In order to resist online and offline attacks, passwords and cryptwords must get longer. As a result, usability decreases while security remains constant. Eventually, minimum password lengths will inevitably grow beyond the point the average person is wiling to tolerate. When this happens, the notion of passwords and comparable "something you know" authentication will expire.

Computers used defensively are the solution. Defending computers can proactively perform extra mandatory work when creating the password hash or encrypting using a cryptword. Absent cryptanalytic attacks, attacker computers will have to perform the same amount of work. It is known [53] that, if the password hash is regularly updated until it is leaked, the attacker has no inherent advantage over the defender; security remains constant for a given security/usability tradeoff.

Our criteria for usable randomly generated passwords are the following. They should:

- be kept short, to ease entering, memorization, and verbal and written communication
- be easy to enter on anticipated devices
- not require frequent changes, to ease memorization
- resist guessing attacks, up to a level of security users are comfortable with

Our research agenda addresses the following issues:

(1) **How can the lengths of authentication passwords and cryptwords remain constant despite computational advances?** If authentication passwords and cryptwords must get longer, they will eventually exceed usable lengths, and memorability research will expire. Conversely, practical upper limits support research, such as shown by Florêncio et al. [29]. Upper limits for offline attack resistance should similarly benefit the research community and end users. As "something you know" tokens have unique advantages [18, 58], it is important to preserve their usability in encryption.

(2) **Do cryptwords need to be treated differently than authentication passwords? If so, how and why?** If cryptwords are different from authentication passwords, it is important to know how to treat them differently, and possibly use a different term to refer to them. Presently cryptwords are simply called passwords, conflating security properties of both. In general cryptwords must be significantly longer than authentication passwords, but the same defenses apply. We find distinctive uses of cryptwords in protocols and to protect data at rest, though these may not be presented clearly to end users. Because protocols provide attackers with cryptword verifiers, these must be treated differently.

(3) **How often do cryptwords need to change, and why?** Where it is possible to perform key rotation, cryptwords may be sized to the key rotation window and kept strong indefinitely. In cases where a verifier leaks, the cryptword itself must be rotated, and any secrets protected by the cryptword should be changed as well if possible.

(4) **How can we determine appropriate lengths for authentication passwords and cryptwords?** We provide simple equations to calculate required random password bit strength based on personalized tradeoffs between security and usability. We also provide some general use suggested lengths for authentication passwords and cryptwords. Where it is not possible to re-encrypt or replace sensitive information, cryptword sizes must be chosen based on prior work regarding encryption key lengths [40].

We don't address attacks such as phishing, credential stuffing, observation attacks, client or server compromise, rubber-hose cryptanalysis, or attacks specific to non-password "something you know" authentication. Password managers are a solution to the credential stuffing problem and can sometimes help with phishing.

If the suggestions in this paper are adopted, we expect to see simpler password advice for users, focused around online and credential stuffing attacks, leaving offline attacks to cryptwords. Users

[1] "This word … mean[s] that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course." [20]

[2] We recommend using Password Authenticated Key Exchange (PAKE) instead of communicating a key between humans using cryptwords.

[3] Superseded by WPA3-SAE, which uses PAKE

will be able to memorize a single strong cryptword to protect their password manager, and keep that cryptword until their password manager database leaks. If these suggestions are not adopted, cryptwords will continue to get longer as shown in Figure 2, until they are no longer feasible for human use. For example, the creator of Diceware [55] judged a 5 word diceware passphrase to be adequate in 1995, since 2014 recommends a 6 word diceware passphrase to attain the same level of security [54].

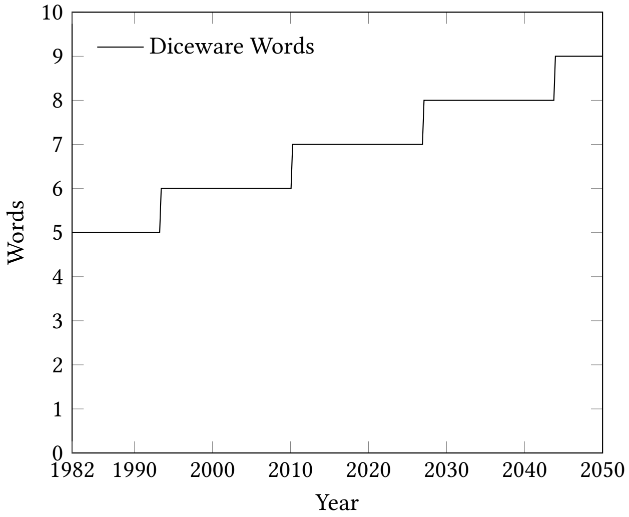


Figure 2: Minimum diceware cryptword lengths derived from Lenstra and Verhuel's [40] equations

1.2 Why Random Passwords?

We accept Kerckhoffs's principle that "the enemy knows the system". An attacker's best strategy against random selection is parallel randomized enumeration [24], finding a random password on average after enumerating half the search space. Any departure from random generation gives the attacker an advantage — they can start enumerating from passwords with a higher chance of being generated.

For small pool sizes such as 4-digit PINs (10^4 possibilities) or other scenarios where user-chosen passwords predominate, an average attacker's best strategy may remain a Zipf's law distribution. In these cases it would be suitable to add a blocklist to the random password generator [45]. In this manner we defend against two kinds of attackers; one where we know their system, and the other where they know ours. While this slightly reduces security against an attacker which knows our system by reducing the password space, it may substantially increase security against a Zipf's law attacker.

2 BACKGROUND AND RELATED WORK

There has been extensive research on usable passwords. We highlight here a few which are particularly pertinent.

Provos and Mazières anticipated this work in 1999 in their seminal paper on `bcrypt` [53]. Similarly, Bonneau has suggested [16, 18] that by using a pwKDF, passwords can have constant protection against attacks at a constant real-world cost. Finally, Polášek gives an equation [52] to model attacker costs on random passwords protected by Argon2.

2.1 Motivation

2.1.1 Usability. Shay et al. found random passwords of comparable strength in various forms to be roughly equally memorable [60]. However, random passphrases took longer to enter and resulted in more typos than random passwords of equivalent strength.

Greene et al. found random passwords with multiple character classes harder to type on mobile phones [31]. The paper suggests adapting password requirements to the device used. An alternative may be to use strong random passwords with a minimal length and number of character classes for all devices.

2.1.2 Random Password Memorability. Schneier suggests generating secure passwords, then writing them down and carrying them with you in your wallet until memorized [59]. This provides a "something you have" recovery method for the "something you know" token, and provides for it the same physical protections given to payment cards, which are also "something you have" tokens. This recovery process naturally strengthens memorability, and the "something you have" recovery token may be safely disposed of once the random password is learned.

Brumen found [22] that when preventing within-subjects participants from writing down their random passwords, one week later only 4 out of 40 remembered a 63-bit random Psychopass password, and none remembered their 50-bit random password or 48-bit random passphrase, after 3 attempts. Each participant was given all three random passwords, and retyped the password and passphrase for 2 minutes each, and the Psychopass password for 5 minutes.

Leonhard and Venkatakrishnan compare [41] Diceware (38.8 bits), alphanumeric (35.7 bits), and pronounceable (30.8 bits) random passwords in a between subjects study. Two weeks later, 2 of 7 Diceware participants and of 1 of 6 of both alphanumeric and pronounceable password participants recalled their random password successfully.

Yan et al. [67] found that users assigned an 8 character password of letters and numbers ($\lessapprox$ 48 bits) found it harder to remember than user-chosen passwords, and carried a password reminder for a mean of 4.8 weeks. The paper also reports that many still carried the written random password 4 months later when surveyed.

However, three recent research papers have shown [18, 27, 34] that participants can successfully memorize 56-bit random passwords. Each paper focused on helping the user memorize the random password, instead of comparing different representations. Future research may build on these findings, showing more efficient memory techniques, or different representations using similar techniques with useful properties and similar or superior memorability.

Notably, users *can* memorize long random secrets. One author has used Schneier's technique [59] to memorize a variety of 5 to 8 word Diceware passphrases.

2.2 Methods

2.2.1 Password Key Derivation Functions. A Key Derivation Function (KDF) converts a secret input into one or more cryptographic keys. Though inadequate for passwords, a simple KDF is to hash the input with SHA256 and truncate the output to the desired length. A secure hashing algorithm has the effect of distributing the true entropy of the passphrase throughout the bits of the hash. However, should the true entropy of the passphrase exceed that of the truncated hash, it is simpler for an attacker to guess the generated hash. To illustrate, a hash truncated to a single bit would require an attacker to guess only two values, instead of guessing the hash input.

Password Key Derivation Functions (pwKDFs) have an adjustable amount of effort to compute the key from the password. This work is balanced between the needs of user-serving devices processing many login requests, and the increased cost to an attacker to test password guesses. All pwKDFs also require a *salt*—a random value long enough to make precomputation attacks infeasible, stored alongside the derived key. The extra effort per guess multiplies an attacker's costs, meaning the attacker has to perform the same amount of computation as though guessing an input from a larger pool. This is sometimes referred to as "key stretching". Popular pwKDFs include `argon2`, `scrypt`, `bcrypt`, and `pbkdf2`.

In 2015, `argon2` [13] won the Password Hashing Competition [7]. `argon2` allows an adjustable work factor which includes access to substantial amounts of RAM, multiple cores, as well as computation effort. This RAM requirement is intended to make custom hardware for guessing passwords no cheaper than buying server time. `argon2` also has a mode resistant against side channel attacks.

Though there are current attacks against various `argon2` algorithms [5], these do not amount to the algorithm being broken. A fivefold attacker speedup can be mitigated by adding an extra 2.3 bits to the minimum size of a generated password.

2.2.2 Password Guess Resistance. There are two major classes of passwords as shown by Florêncio, Herley, and van Oorschot — those that can defend against online attacks and those that can defend against offline attacks [29]. Their work is a key source of inspiration for our research. While user chosen passwords have a Zipf's law distribution [64], our focus is on keyspace guess resistance for random cryptwords.

Wheeler provides `zxcvbn` [65, 66], a system which estimates:

- 100 guesses per hour for a throttled online attack
- 10 guesses per second for an unthrottled online attack
- 10k (10^4) guesses per second for an offline attack against a pwKDF encoded password
- 10B (10^{10}) guesses per second for an offline attack against a fast hashing function

However, these numbers are not peer reviewed and may be optimistic [25]; they might underestimate an attacker's capabilities.

2.2.3 Computational Limits. It is possible to establish a minimal length for cryptwords based on fundamental limits of computation, but this produces relatively high limits, on the order of 256 bits. Although such cryptwords will be strong indefinitely, they are long enough to make them infeasible to remember, communicate, and enter. For example, such a cryptword may be represented as 20 words from the Diceware [55] wordlist.

A simple approach is to use practical limits on computation imposed by physics [43]. A kilogram of perfect computation material filling a volume of one liter can compute roughly 5.4×10^{50} logical operations per second. The planet Earth has a mass of around 5.9×10^{24} kg [63]. If we assume a password can be tested with 64 logical operations (a single round of MD5, for example), and a user wishes their cryptword to be unbreakable until an average of 100 years of computation on such a device, the cryptword would need to be generated randomly with equal probability from $\log_2\left(\frac{5.4\times10^{50}\times5.9\times10^{24}\times2\times100\times365.2425\times24\times3600}{64}\right) \approx 277.4$ bits.

Using the diceware system, such a random passphrase would consist of 22 words, such as "embargo flint canon requisite pointed bargraph unicycle likely unsaved jasmine selective chloride unfair unsorted graduate pedicure buddhist squishier nail skirmish gangrene deflector". An alphanumeric random password would consist of 47 characters, such as:

dqEDJkJjXYFoXYRQRR9dDjvXHnrUqRjmiqXgipK2yh974RU

2.2.4 Cryptography Key Lengths. We can apply lessons from minimal encryption key strength estimation to the cryptwords used to produce those keys. Cryptwords, like the cryptography keys they generate, are independently and identically distributed.

Giry maintains a site [30] which lists related work, and calculates encryption key strength based on the related work.

Blaze et al. show [15] in 1996 that every year we should add $\frac{2}{3}$ of a bit to key strength. In 2001 Lenstra and Verheul gave a better bound [40], adding $\frac{23}{30}$ bits every year. This is the most recent paper which estimates key lengths as years pass: subsequent related works justify their recommendations by citing this paper.

These estimates are partially derived from Moore's Law (Lenstra and Verheul also add budget inflation), and it appears to be slowing down [39]. Our paper sidesteps Moore's law entirely by offloading estimating the future to the user of our algorithm; ideally needing only to estimate at most a few years into the future at a time.

3 AUTHENTICATION PASSWORDS

Passwords used to authenticate are subject to both online and offline attacks, as covered in Section 1. Passwords intending to defend against offline attacks must be much stronger, and hence less usable [29].

3.1 Online Attack Resistance

Password authentication involves two parties — the authenticating party that presents their token, and the relying party that compares the token against expected values. Because there are two parties, the relying party can *rate limit* authentication attempts or *lockout* accounts. Both reduce opportunities for an attacker to guess authentication credentials, and are completely independent of technology advances, as they are determined by policy. In such cases it is trivial to determine minimal random password entropy for a given acceptable risk level, such as in the case of bank card PINs.

Many of the top websites do not protect accounts via rate limiting or lockouts [44]. It is reasonable to model these parties as less concerned or aware about security more generally, and characterize them as *negligent systems.* Attackers may conduct guessing attacks against negligent systems bounded only by computational resources and the speed of the communications channel between the attacker and the negligent system. However, as long as the negligent system uses a method of comparing the credential which has a fixed cost despite hardware increases, there is an effective rate limit.

If we model the daily odds of detecting an attack of from 10–1000 guesses per second are constant and not less than $\frac{1}{100}$ per day, an attack will be detected with $1 - \left(1 - \frac{1}{100}\right)^{365.2425} \approx 97.5\%$ probability after 1 year. Some ways an administrator may detect an attack include user reports of sluggishness, occasional sporadic checking of log files, and noting unusual traffic or CPU activity levels in management dashboards. Given the odds of detection are constant, even if an attacker slows down 100×, they will be detected with equal probability, and would take 100 years to guess the same number of passwords. At 1000 guesses per second, a login server would need to spend a substantial amount of computation time running the pwKDF for attacker guesses; we assume that above 1000 guesses the login server will either be so noticeably hobbled that the odds of detection increase, or it is simply unable to process more requests due to the effective rate limits of the pwKDF.

We can estimate adequate bit strength for random authentication passwords with just two parameters. We'll let a represent the total number of guesses we will allow the attacker, and r denote the acceptable risk of a successful guess. b represents the minimum size of the pool in bits from which the random password must be generated to provide at least the desired security level.

$$b = \log_2\left(\frac{a}{r}\right) \tag{1}$$

At 1000 guesses per second, an attacker can guess up to $a = 365.2425 \times 24 \times 3600 \times 1000 \approx 31{,}556{,}952{,}000$ passwords in a year. To have a similar risk factor as common 4-digit bank card PINs of $r = \frac{3}{10^4} = 0.0003$, online guess resistant passwords even at negligent systems should never need be longer than $\log_2\left(\frac{31{,}556{,}952{,}000}{0.0003}\right) \approx 46.6$ bits of entropy. This is higher than other numbers in the literature [29, 65, 66], which use $10^6 \approx 19.9$ bits [29].

Table 1 provides some varied examples of random authentication passwords at different strengths. It is important to note that security properties of random passwords are related to the underlying entropy, and not to the character classes represented or the length of the random password.

Endholm's law [23] predicts bandwidth and data rate doubling every 18 months, as with Moore's law. Despite this, if defenders use a pwKDF tuned to their modern hardware, the system remains a constant bottleneck for attacks, and the load on the system from attacker guessing should eventually provoke investigation. Ideally, administrators will use at least rate limiting to protect user accounts, allowing their users to use shorter random passwords to achieve the same level of security.

3.2 Offline Attacks

Offline attacks against authentication passwords require an attacker to have privileged access to the authentication verifier database. As this database should be well protected, it is possible that an attacker with this access may have access to read passwords in plaintext as they enter the system (solvable via aPAKEs such as OPAQUE [38]), or to inject JavaScript into an authentication webpage with the same trusted privileges as the site itself to intercept passwords as they are entered (not solvable by aPAKEs at the same privilege level). Additionally, not all passwords are stored securely by all systems, and the authentication verifier database may store passwords in clear text or with reversible encryption. We discuss threat models for offline attacks against authentication passwords further in Section 6.1.

Because passwords are subject to various attacks which can reveal them other than offline guessing, offline guessing resistance is a secondary consideration. If a defender believes they can detect a password database breach within k days from a passive attacker, it would be nice if the passwords are stored securely and are strong enough to resist guessing attacks for at least $k + w$ days, where w is the time allocated for notified users to change their passwords. Passwords at rest should be protected by a pwKDF, tuned to introduce a reasonable load on the authentication server(s). We provide equations in Section 4.3 for estimating minimal strengths to defend against offline attacks. These can also be used to estimate time and resources to crack random authentication passwords at rest.

Additionally, for an attacker to benefit from an attack, there will be some evidence of the attack [9]. This evidence can protect a body of users in aggregate, even if the defender optimistically overestimates k, as easier targets in the population serve to detect attacks. We discuss the likelihood of these attacks more in Section 6.1, finding password-related losses unlikely, as only uncommon threat models need the password in addition to whatever they could access when acquiring the password verifier.

3.3 Authentication Passwords Conclusions

Online attacks are subject to policy, so authentication passwords do not need to get longer each year. Authentication passwords are also subject to offline attacks, but the pwKDF parameters can and should be tuned to maintain constant strength despite hardware advances [53].

The information in this section is not new, but we discuss it in the context of cryptwords to frame our contribution, and to demonstrate that like cryptwords, authentication passwords need not get longer.

4 CRYPTWORDS

Cryptwords are passwords used to generate encryption keys; to encrypt. They should be randomly generated (see Section 1), and we analyze only randomly generated cryptwords in this section.

Encryption is fundamentally a different environment than authentication. Once an attacker has access to the encrypted information, they cannot be rate limited or locked out. Encryption can be used to protect secrets in a public environment, or the encrypted information can be kept private. Recovery requires proactively storing copies of the encryption key in other locations; these should also be kept safe. As proxies for encryption keys, cryptwords are subject to the same properties.

We focus on cases where the encrypted information is kept private. This includes disk encryption and file encryption where the

Table 1: Various example random authentication passwords, bit strengths, and generation rules

Random Password	Bits	Notes
0154	13.3	4 digits, for a bank card with 3 guess lockout
}x*	19.7	3 chars from all 95 printable ASCII
vg9u	19.8	4 lowercase alphanumeric, without any of "l1IO0", very close to [29]
454537	19.9	6 digit PIN, the exact entropy from [29]
glad bamboo	22.0	2 words from BIP-39 [49]
kfqaa	23.5	5 lowercase letters, easy to enter on mobile devices [31]
ceremony geranium	25.9	2 diceware words [55]
>v9K	26.3	4 chars from all 95 printable ASCII
kiwi credit library	33.0	3 words from BIP-39 [49]
839hxfm	34.7	7 lowercase alphanumeric without "l1IO0"
kJs59i	35.7	6 upper and lowercase alphanumeric
dyaaufct	37.6	8 lowercase letters, easy to enter on mobile devices [31]
sculpture blurb unchanged	38.8	3 diceware words [55]
g8MeJ8bD	47.6	8 upper and lowercase alphanumeric

files are kept on the computer or shared only with trusted computers, which keep the encrypted files in sync. Software examples include offline password managers, and SSH keys stored on disk. We characterize these as data-at-rest encryption.

Encryption is also useful to protect and authenticate in protocols. In these cases, a verifier is inherently available as part of normal operations. Cryptwords can also be used in these cases to produce encryption keys, such as with Wi-Fi passwords (WPA-PSK and WPA2-PSK), and cryptocurrency Brain Wallets.

Online password managers may have the database password double as an account password, and only send the database to users with a valid password. This provides effectively the same level of security as an offline password manager if the provider's servers are all no less trusted than user's computers.

4.1 Constant Security Despite Evolving Threats

In cases where the encrypted data E containing secrets S is kept private, such that an attacker cannot read it and all copies can be updated, we can update our security level by updating the pwKDF parameters P_{new} and generating a new encryption key K_{new} when the user enters their cryptword C. We use the old pwKDF parameters P_{old} to generate the old encryption key K_{old}, decrypt the secrets S with K_{old}, and then encrypt them freshly with the new key K_{new}. Note that $K_{new} = pwKDF(K_{new}, C)$ and $K_{old} = pwKDF(K_{old}, C)$.

One simple optimization is to avoid re-encrypting all of S, which can be slow in the case of full disk encryption or other large secrets, by generating a data encryption key DEK strong enough to itself be unguessable, use DEK to encrypt S, and then encrypt DEK with K. When pwKDF parameters are updated to P_{new}, only the DEK needs to be decrypted with K_{old} and re-encrypted with K_{new} instead of re-encrypting all of S. This makes updating pwKDF parameters P a constant speed operation, regardless of the size of S.

Cryptwords can be held at a constant length despite hardware improvements by regularly updating the pwKDF parameters P_{new} and re-encrypting using K_{new} until such time as the encrypted data E leaks to an attacker. The pwKDF algorithm can be replaced in like manner.

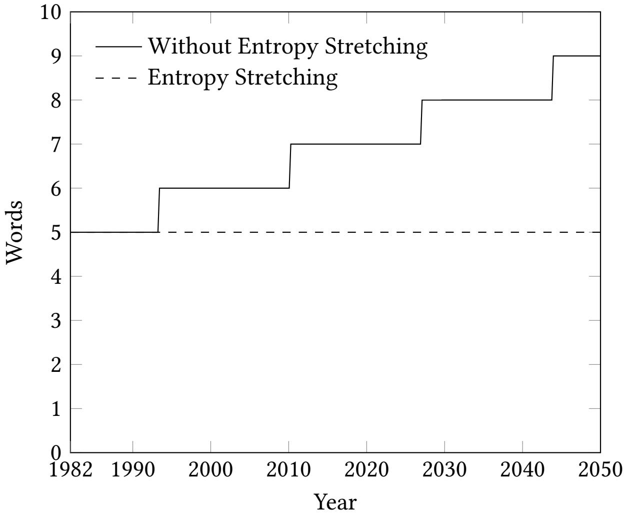


Figure 3: Diceware words with and without entropy stretching

If it is not possible to keep the encrypted data E from attackers, the pwKDF parameters P cannot be updated to derive a new encryption key K_{new} to encrypt the secrets S. In cases where E cannot be kept private, the user should generate a cryptword C strong enough to protect the secrets S for as long as they need to be protected. In cases where S itself can be replaced, such as an SSH key or a cryptocurrency wallet address, C can be sized to the replacement window. However, when generating a new secret S_2, the user will also need to generate a new cryptword C_2, as the encrypted data E_1 can typically be used as a verifier to guess C_1.

Figure 3 shows the impact of entropy stretching on diceware passphrases, and Figure 4 shows how cryptwords can be held at constant length despite replacing an encryption key of increasing strength. Notably, Figure 4 shows that the key strength minus the extra work is the cryptword length.

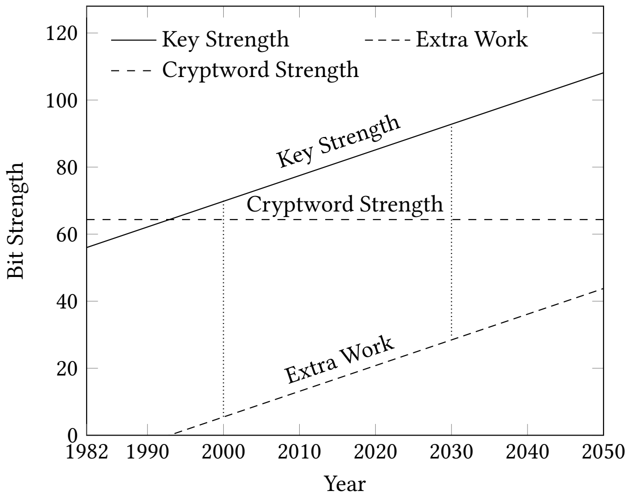


Figure 4: Using extra work, cryptwords can have constant length despite computational advances

4.2 Parameters

Kerckhoffs's principle states that the attacker knows the system; generating cryptwords from any distribution other than random gives the attacker an advantage. With a random distribution, no attacker guessing strategy can exceed parallel enumeration to encompass the whole set, or random guessing to reach some p-value for success if they cannot afford to enumerate the whole set. See Section 1.2 for a discussion of Zipf's law attackers.

4.2.1 Defensive. We model our defensive terms as follows. t is the upper bound users are willing to tolerate for pwKDF processing time, in seconds. t is used directly to derive the pwKDF parameters on the oldest supported hardware. r is the acceptable risk that an attacker will correctly guess a random password. For randomly generated 4 digit bank card PINs with a lockout after 3 failed attempts, this is $\frac{3}{10^4} = 0.0003$. b is the minimal number of bits ($\log_2$ of the number of possible cryptwords) required to achieve the desired performance. While this is useful as an output parameter when generating random passwords, it can also be used as an input when estimating costs or time for an attacker once a verifier for a generated cryptword has leaked.

4.2.2 Attacker Advantage. We assume the attacker has some form of computational advantage over the defender. This is natural if the defender needs to run the pwKDF on a slower processor, such as an older smartphone or an embedded system, or if the attacker manages to acquire a verifier produced on then-modern hardware at some point in the past. We call this parameter g to represent the gap in performance the attacker has over the defender, per equivalent computing unit. This parameter allows us to ignore the current year in all our equations, as we offload the task of forecasting environment changes to the user of our equations. It is based on the pwKDF parameter update frequency, which itself is based on how often older hardware is to be phased out of support (t values exceeding acceptable values). Due to the wide variability of environments in which this system may be deployed, and the relative advantage of predicting the future only 2–3 years in advance, we offload this to the defender. We discuss more in depth how to estimate this parameter in Section 5.3.

4.2.3 Attacker Capabilities. We provide two equations to estimate the attacker's capabilities. The first is a more traditional "attacker capabilities" equation. It assumes the attacker has a supercomputer or computing cluster with given capabilities and is willing to devote it to cracking cryptwords for some given duration. d is the upper limit of the number of days the attacker is expected to be willing to dedicate their hardware to cryptword guessing. p is the parallelism advantage of the attacker's hardware over the defender's. If the defender is using 4 core parallelism in their pwKDF and expect the attacker to dedicate 8000 cores to attacking, the attacker has a parallelism advantage of $\frac{8000}{4} = 2000$. Note that the relative speed of each core is not considered here; the parameter g represents how much better the attacker's computers are. In short, p is an estimate of the extra resources the attacker will bring to bear.

4.2.4 Attack Cost. Our second equation is somewhat novel. In 2001, Lenstra and Verheul [40] use a barebones 450 MHz PC to estimate attacker costs, recognizing that this may not fully capture the realities of economies of scale or dedicated hardware. AWS launched cloud computing services in 2006, Microsoft launched Azure in 2010, and Google launched Compute Engine in 2012. Infrastructure-as-a-service (IaaS) providers are in a competitive market, and so prices may reasonably be assumed to somewhat closely reflect the true costs of running a similar amount of computational capacity, including some profit margin. In some cases cloud computing costs may be lower than in-house costs after factoring in both capital and operating expenses, due to the provider's economies of scale. Even in the case of a large entity fully funding a dedicated computing farm for cryptword guessing, their true costs are likely to be within an order of magnitude of the cloud computing costs.

So, we estimate an attacker's cost to break a cryptword with our second equation. m represents the upper limit of money an attacker will be willing to spend. This should be expressed in the preferred currency of the largest cloud computing platforms, as services are likely to be cheaper without an implicit exchange fee. c is the cheapest hourly cost for equivalent cloud computing hardware, after factoring in bulk discounts, long reservations, more capable computers, etc. We use the hourly cost as this is commonly provided by current large providers; as all times are converted to seconds to cancel out the t term, it should be simple to adapt the equations to other durations.

Parameter Summary.

- b is the bitspace from which the cryptword must be generated
- r denotes risk — the final chance an attacker will guess correctly
- t is the upper tolerable compute time in seconds to encode the password on the slowest supported user's machine, in seconds
- g is the performance improvement of attacker computation over the slowest supported user device
- p is the number of cores the attacker will dedicate

- d is how many days the attacker dedicates their hardware to the cryptword. We convert this to seconds via the constant 86400, a simplified approximation of one day.
- c is the hourly cost for equivalent hardware. We convert this to seconds via the constant 3600.
- m is the amount of money an attacker may spend attacking

4.3 Equations

Both equations rely on r, t, and g, the latter two chosen by the developer. g is based on the maximum age of supported hardware, Moore's law, and the chosen pwKDF algorithm. t is based on user patience for security, keeping in mind that users may feel safer with a processing time on the order of 1 second [57]. r determines how much of the cryptword space an attacker can enumerate within the provided constraints.

This first equation computes the required guessing space in bits based on how many parallel guesses an attacker will make over a given duration, and represents a traditional risk equation. It should be used to estimate guessing space against an attacker with some given capability and persistence.

$$b = \log_2 \left(\frac{g}{rt} \times p \times d \times 86400 \right) \tag{2}$$

This second equation computes the required guessing space based on the cost an attacker might be willing to pay.

$$b = \log_2 \left(\frac{g}{rt} \times \frac{m}{c} \times 3600 \right) \tag{3}$$

Both equations can be solved for other variables: d for the safe duration for a given parallel computation capability and m for the cost to an attacker are particularly interesting.

$$d = \frac{2^b rt}{g} \times \frac{1}{86400p} \tag{4}$$

$$m = \frac{2^b rt}{g} \times \frac{c}{3600} \tag{5}$$

4.4 Limitations

The key cannot be regenerated until the user re-enters their password; thus, the encrypted information must be occasionally accessed. Infrequent use files will not have keys generated with updated pwKDF parameters, and could be weaker than intended as a result. Additionally, if not all copies can be updated — for example, a zip file shared with coworkers who have copies on their individual workstations — older copies will similarly not have updated pwKDF parameters applied.

We leave to the system administrator or developer to choose a correct value for the update window, and provide a correct scaling factor g. If the scaling factor is underestimated, security will not meet expectations, while if overestimated, the minimum cryptword length may be less memorable. To help mitigate this we provide a detailed example of estimating g in Section 5.3.

If the cryptword is regularly entered and the pwKDF parameters regularly updated, everything works as expected, and a single constant length cryptword can be used until a verifier leaks. If the encrypted information or a verifier is leaked or used publicly, the same equations can show how long until the information leaks. If the encrypted secret itself can be rotated, such SSH keys, Brain Wallets (by transferring funds to a new wallet), or Wi-Fi PSKs, password manager passwords, etc., there is a known safe window during which they can be used, based on estimates of g. If the secret cannot be rotated or must be posted publicly, the secret must be protected with a cryptword or encryption key of sufficient strength [40] to last the duration of the life of the secret.

Our system could incentivize attackers to gather and store verifiers for later attack when attacker hardware improves, giving the attacker an advantage beyond g. However, this is likely not an effective use of attacker resources. First, for cases where a verifier is public or easy to gather, we recommend choosing g to last the expected lifetime of the secret, as above. In cases where a verifier is not easy to obtain, such as a password manager stored only on user-controlled devices or in well-protected cloud services, we expect the attacker will almost always have easier ways to benefit from the attack. As mentioned in Section 3.2, we expect an attacker who acquires a verifier usually has more effective ways to gain value from the requisite access — for example, by installing a keylogger on the computer to directly acquire the cryptword. This allows additional opportunities to detect the attack and change cryptwords and protected secrets before any harm comes as a result of an attacker guessing the cryptword.

5 USING THE EQUATIONS

It is useful to show how these equations would apply in practice, and how to estimate some key parameters. Figure 5 shows the developer workflow, Figure 6 how a user would generate a cryptword for a specific application, and Figure 7 how a user would use a cryptword and a password manager for safe remote password authentication.

5.1 Estimating m

It makes sense to set the value of m, the minimum cost to an attacker, higher than the upper financial value of the thing being protected. Even if the value of something is purely financial, humans are loss averse. The cost of losing future options is higher than the cost of not gaining future options.

Most things have value beyond the mere financial. Loss of time, reputation, and memory are all important to factor in. Further, if the secrets protected could impersonate the defender, the cost of possible scams to others should be considered, even if not represented directly as monetary losses to the defender or the defender's reputation.

A useful indicator that a defender has chosen a sufficient value for m is if the defender says, "if an attacker is willing to pay m…more power to them". Essentially declaring that the cost to the attacker is far enough beyond the cost to the defender of failing to protect their secrets that the defender is resigned to a loss given the unequal value.

5.2 Estimating p

In Equation (2), p represents the parallelism advantage the attacker holds over the defender. As it is common for the defender to use a single device, an attacker dedicating 100 devices, or a device with 100 processors, would have a parallelism advantage of $p = 100$. For p we ignore differences in the effectiveness of the hardware such

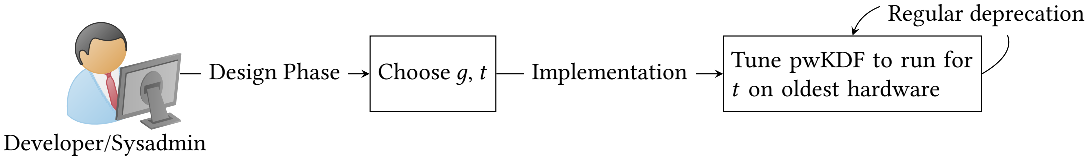


Figure 5: Developers choose g and t once, then regularly update pwKDF parameters

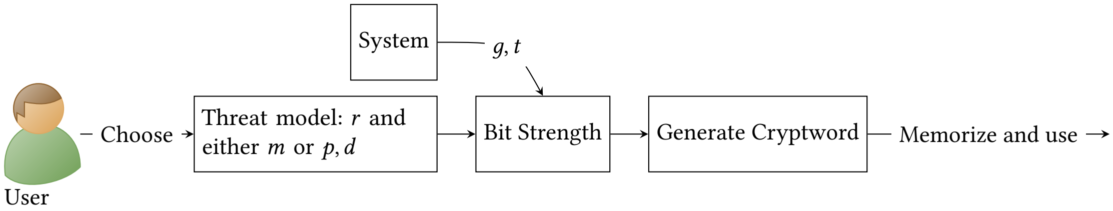


Figure 6: Users generate their cryptword based on their threat model

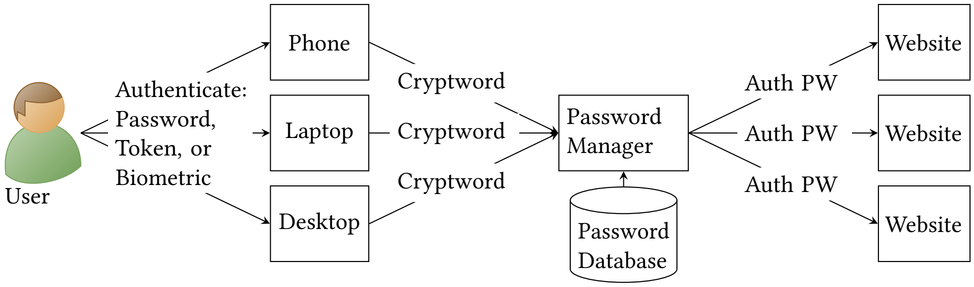


Figure 7: Typical user authentication flow using cryptwords and authentication passwords

as the relative number of cores: this is factored into g, discussed in Section 5.3. One way to determine a generous estimate of p is to consider the available power at some of the world's largest data centers as a potential upper-bound on what an attacker could reasonably harness.

The Switch Tahoe datacenter is one of the largest in the world, supporting a power draw of 850 Megawatts, at up to 55 kW per server cabinet [61]. Guessing passwords using modern pwKDFs is computationally bounded — thus, we can use the power draw as an estimate of parallelism. 803,608 is a reasonable upper bound estimate of the number of high power draw 1U servers the Switch Tahoe datacenter can support [1].

The US NSA is a large intelligence agency, and may serve as an example of a nation-state level attacker. The NSA recently built a datacenter in Utah with a reported power draw of 65 MW [8], more than 13 times smaller than the Switch Tahoe datacenter. Due to economies of scale, it seems unlikely an attacker could build a distributed server farm with more capability than the Switch Tahoe datacenter. Due to high costs and the number of involved workers, it seems unlikely that a nation-state actor could build a comparable datacenter without public awareness [32].

5.3 Estimating g

A correct understanding of g, the gap between the slowest supported hardware, and the attacker's hardware, is essential to the accurate usage of these equations. Predicting the future is hard, and grows harder the farther out one needs to predict. Additionally, which devices to support and for how long is an administrative decision which varies between different organizations. As a result, we leave determining g to the reader, based on their individual circumstances.

Some key things to consider when deciding g:

(1) **How long will hardware be supported for?** The pwKDF parameters are derived from the value of t and the slowest supported hardware, and are fixed until hardware is deprecated or the accepted value of t changes. To add support at the same t for slower hardware outside the current supported set, cryptwords would need to be updated to hold more entropy within themselves, to maintain the same levels of security to account for the loss in protection from the key stretching. This would run strictly contrary to the usability goal of not changing the cryptwords, so the pwKDF parameters must only be allowed to increase.
For organizations with largely homogeneous systems and frequent hardware update, such as in server farms, deprecation might be updated annually. For deployment scenarios with mostly heterogeneous systems and individualized ownership, deprecation of supported devices might happen much less often. In those cases, deprecated devices may still continue to receive support, but would continue to suffer

decreasing performance as the entropy stretching protects them from stronger attacker hardware.

There is no harm in determining minimum pwKDF parameters and allowing stronger ones on faster end user devices (perhaps locally generated from t), so long as those devices continue to check that they meet or exceed the minimum pwKDF parameters, and update their parameters when the minimums exceed their current settings.

(2) **How often will hardware deprecation be reviewed?** We recommend at a minimum an annual review of hardware support decisions and security levels, including tracking predictions of attacker capabilities.

(3) **How often will updated pwKDF parameters be shipped to end user devices?** Once pwKDF parameters have been updated, they must reach the user. Not all software is updated regularly. Developers need to have a sense of how often their users receive updates to estimate an update window. Automatic updates and update notifications help.

(4) **How often will users enter their cryptwords to decrypt, allowing updates to those parameters to take effect?** Even after software updates ship with new pwKDF minimum parameters, the software cannot rotate keys until the user re-enters their cryptword. For some systems such as daily use password managers this may be several times per week. In general, the longer between cryptword entries, the more difficult it will be to maintain in memory, so this should typically be somewhat frequent. If necessary, the user may also be prompted to re-enter their cryptword as part of the update process.

The above factors determine a window of time, which the cryptword must endure. If any of these are not possible to determine (for example, hardware will never be deprecated ever, even after 20 years), cryptwords should instead be generated based on the maximum expected valuable lifetime of the secret to protect.

As an example, when designing a new mobile password manager one might decide to support hardware which is currently 5 years old and to keep support for all hardware for up to 9 years, giving a 14-year technology advantage to the attacker. Decisions would be reviewed annually, at which point any technology older than 9 years may be deprecated, upgrading the attacker's advantage to 15 years. Updates would be deployed via an app store and users frequently enter primary passwords (the cryptword), so perhaps 2–3 months may be added as a safety buffer.

Once we have a window, we need to predict what the attacker will be capable of in the future. For the example above, we currently only need to predict 10 years into the future, and as we deprecate hardware, will only need to predict future hardware up until the next deprecation window (a much easier task). Lenstra and Verheul suggest [40] adding $\frac{23}{30}$ bits of strength each year to an attacker's capabilities, though this may differ by changing base assumptions in their equations (for example, Moore's law may be slowing, or budgets may be accelerating).

A simple algorithm follows:

(1) To estimate the gap, assume the attacker has access to the current fastest, nicest, most parallel hardware most specialized to cracking the chosen pwKDF, and will use all known attacks.
(2) Estimate the difference between the attacker hardware and the slowest supported hardware.
(3) Multiply for any known cryptanalytic attacker advantages.
(4) Multiply by $2^{\frac{23}{30}}$ or a personalized conversion factor for each year into the future the guess must reach.

For the example mobile password manager, using the search term "2017 smartphone popular", we find an article listing the top 20 by sales [50] which seems adequately reliable. After manually comparing each, we find the slowest is a quad-core 1.4 GHz Cortex-A53 on the Nokia 3 [33] with 2 GB of RAM. We will use the most recent version of Argon2 (not worrying at present about library support), which is designed to eliminate the advantage of custom hardware. The current fastest server CPU (in terms of threads × clock rate) is the EPYC 7773X, operating 128 threads at up to 3.675 GHz. This processor may be able to compute up to 84 times faster than our Nokia 3. The current best attack against Argon2i-B gives a 2× speedup at 256 MB of RAM with 3 passes over memory [5]. As we're predicting 10 years into the future, we set $g = 84\times2\times2^{\frac{23}{30}\times10} \approx 34{,}135.5$.

If not using a memory-hard pwKDF such as Argon2 or a successor, the attacker can use application-specific integrated circuits (ASIC) to perform computations possibly many times faster or more parallel than a commodity processor. In such cases the defender should factor this into g, and possibly p or c as appropriate.

The future is difficult to predict. We might make a breakthrough in quantum computing in the next 5 years which updates g to a much larger number. Cryptanalytic advances may render Argon2 much weaker than previously supposed. Regardless, g can be updated, the equations recalculated, if necessary cryptwords or secrets regenerated, and security maintained.

5.4 Updating pwKDF Parameters

As shown in Figure 5 and discussed in Sections 4.1 and 5.3, developers must regularly update the pwKDF parameters to provide constant security from fixed-length cryptwords. The pwKDF parameters are chosen so the pwKDF runs in t seconds on the slowest supported hardware. As hardware is deprecated, it is essential that the pwKDF parameters are increased so that g remains constant. Hardware deprecation may not be delayed but deprecated hardware may still be used, with the caveat that it will no longer run the pwKDF in only t seconds. This will eventually result in an unacceptable user experience on deprecated hardware.

The recommended method [14] to choose Argon2 parameters requires access to reference hardware. Developers would therefore tune their algorithm for the new slowest supported hardware during each deprecation cycle, choosing parameters which mitigate known attacks.

5.5 Worked Examples

We use $r = \frac{3}{10^4}$ to match accepted payment card PIN guessing risk.

We set $t = 0.8$ to provide a noticeable delay to users, as it both increases security and gives users a visceral sense that encryption is taking place [57]. Although t is not used for logging in to a system, it should be kept similarly short, as waiting for decryption is not a user's primary task. Additionally, each time it is doubled only saves a single bit from a random password. t has some natural minimum value > 0, though the duration of a single round of md5 decreases as hardware performance improves. In practice, values of $t < 0.1$ are likely to all be perceived equivalently to users, though large companies with frequent logins may be motivated to decrease it further to reduce the load on their login servers for password hashing.

We set $g = 34{,}136$, following the conservative estimates in Section 5.3.

The mean net worth of US citizens in 2019 peaks in the 65–74 age group, at roughly USD $1,215,920 [3]. To factor in replacement costs and losses (see Section 5.1), we round m up to USD $10,000,000. We estimate that this should be above the replacement cost of the average person's current net worth.

We set $c = 0.2179$, based on the current price for an AWS c6g.16xlarge compute instance after applying a 90% discount for Spot Instance use [4].

We set $d = 1826$ as an outlier; it seems unreasonable that any agency would be willing to dedicate a single piece of expensive cracking hardware or large datacenter to guessing a single cryptword for a longer duration. This may exceed a practical upper limit, but it should be at least equal to an upper limit.

We set $p = 803{,}608$, assuming the entire capacity of a large server farm is used to attack our passwords; see Section 5.2 for more details. The differences between phone parallelism and server parallelism are accounted for in g.

Summary of chosen values:

- $r = \frac{3}{10^4}$
- $t = 0.8$
- $g = 34{,}136$
- $m = 10^7 = \$10{,}000{,}000$
- $c = 0.2179$
- $d = 5 \times 365.24 = 1826.2$
- $p = 803{,}608$

$\frac{g}{rt} = 142{,}233{,}333.3$; equation (2) shows $b = \log_2(142{,}233{,}333.3 \times 803{,}608 \times 1826.2 \times 86400) \approx 73.9$ and equation (3) shows $b = \log_2(142{,}233{,}333.3 \times \frac{10{,}000{,}000}{0.2179} \times 3600) \approx 64.3$. Under the above generous assumptions, cryptwords of these lengths should be strong indefinitely if pwKDF parameters are updated regularly and used to regenerate encryption keys.

56-bit cryptwords are popular in the literature [18, 27, 34]. If we relax g to 105, keeping other parameters unchanged, we find that 56-bit cryptwords can also be secure indefinitely against the same financial threat, using equation (3). This would roughly correspond to using argon2 with a 3-year attack window (including hardware support, updates, and deprecation reviews) supporting current desktop processors. One could instead reduce the risk or threat model as in Section 6.3.

Some example cryptwords at these values:

- "gully hunger wistful reminder/" (56 bits, diceware [55] + random symbol)
- "TN2nssteT53" (64 bits, alphanumeric minus "1lI0O")
- "specks revenge smuggler dramatic quadrant" (65 bits, diceware [55])
- "tmaw qvak whsu tgrm" (75 bits, 26 lowercase, chunked in blocks of 4, easy to enter on a mobile device [31])
- "&DU7yg)}dq29" (78 bits, 12 printable ASCII characters)

5.6 Money to Days

As Equations (2) and (3) both estimate the same parameter, it is possible to set them equal to each other and solve for other parameters of interest, such as a lower bound on how long a money-bounded attacker would take to exhaust their budget.

$$d = \frac{m}{c \times 24 \times p} \tag{6}$$

Using the conservative estimates from Sections 5.1, 5.2, and 5.3, an attacker with a budget of $m = \$10{,}000{,}000$ and a parallelism advantage from renting an entire massive datacenter of $p = 803{,}608$ at a cost of $c = \$0.2179$ per guessing-hour would require $d \approx 2.38$ days to exhaust their budget.

6 DISCUSSION

6.1 Offline Attacks Against Authentication Passwords

Using a cryptword to authenticate does not protect from credential stuffing attacks, as credentials may be compromised via phishing, server compromise, or leaking insecurely stored verifiers from any relying party. Password managers, including stateless password managers (see Section 6.7) are a simple and effective solution to credential stuffing, and some significantly help with phishing.

If a user uses a unique password for each site, the offline attack resistance of their password only protects against an attacker which (a) can only access the password verifier instead of the password, and (b) requires the password to conduct their attack. Attackers may (1) only have access to stored password verifiers, (2) only have read access to account data and desire write access, or (3) desire to continue or conduct an attack after they have lost access to account data. Of note, all of these attackers would need to conduct their attack before the user changes their password, which will ideally follow shortly after administrators become aware of the breach. This limits the window during which an attacker can guess verifiers, as in Section 3.2.

Although these attacks are theoretically possible, they do not appear to be used in any substantial volume [32]. As the risk is therefore relatively small, the burden on users should be equally small [35]. Cryptwords for authentication do not appear to justify the usability costs even for infrequent scenarios where they must be entered manually, such as from a portable device. As a result, we concern ourselves principally with online attack resistance for authentication passwords.

6.2 On The Choice of Neologisms

In two-party authentication contexts, passwords can be protected using rate limiting and lockout schemes. As these passwords are inherently shared, their security also depends on the security of the party with which they are shared. Passwords used to generate encryption keys are subject to offline attacks, which are faster and cannot be slowed or prevented once an attacker has a verifier. These are not inherently shared, and must be stronger due to an attacker's relative advantage. Because they face different attacks and threat models, they should not be conflated. Using different words to refer to these will help system administrators and users be aware that each needs to protect against different threats.

Current language to refer specifically to passwords used to encrypt includes "primary password" (replacing the deprecated term "master password") and "strong password". "Primary password" is specific to password managers, and doesn't adapt as well to contexts like zip file encryption and Wi-Fi PSK. "Strong password" has the disadvantage of not inherently introducing a new concept: as it is a noun phrase, "strong" is weakly connected and could be interpreted as implying the same thing, just stronger. "Seed" is used to refer to a source of randomness which provides the initial state for a random number generator to produce the same output, and it is not wrong to apply the term to a password used to generate an encryption key.

An ideal neologism would (1) share some inherent parallelism with the term "password" to support analogical reasoning, (2) be pithy (terse and memorable), (3) and be both precise and accurate in meaning. "Cryptword" shares the second half of "password", and can be described as "a password used to encrypt", supporting such analogical reasoning. Though pithy, it inherits the same misuse of "word" from "password", where far more than words are represented (PINs, word sequences, non-word alphanumeric strings, etc.). "Cryptseed" as a neologism has the advantage of being pithy, precise, and accurate, but lacks inherent parallelism with "password". Both alternatives are short and can replace the word "password" in many UI designs with minimal changes, as their lengths are very similar. We have chosen the term "cryptword", as we value supporting analogical reasoning.

6.3 Bit Contribution

Because $\log_2(x \times y) = \log_2(x) + \log_2(y)$, it is easy to compare how many bits each parameter in the equation adds to the final bit strength. We analyze this using the parameter values from Section 5.5.

$r = \frac{3}{10^4}$ adds 11.7 bits to b, which could drop to as low as 1 bit at $r = 0.5$. For contrast, Diceware provides 12.9 bits of entropy per word added; this choice of r therefore requires one more word than that of someone willing to accept the same odds as a coin toss. $t = 0.8$ increases b by 0.3, and at human-relevant speeds has a fairly minimal impact on b, from -3.3 at 10 seconds to 3.3 at $\frac{1}{10}$th of a second. As large values of t can have a significant impact on the usability of the system without much security benefit, we recommend $t \leq 1$. $g = 34{,}136$ adds 15 bits. Like t, this is chosen by the developer and reflects the tension between two different usability factors in the security domain — shorter cryptwords vs longer device support windows. Finally, our threat models add 37.3 bits for money and 46.8 bits for the 5-year datacenter attack.

Notably, more than half of b comes from user-chosen parameters: the acceptable risk and the threat model.

6.4 End User Load

Random passwords are less memorable than unconstrained user chosen passwords, but are more secure [67]. Assigned cryptwords can be memorable [18, 27, 34], though perhaps not without a focus on memory training [22, 41]. Non-computerized password stores, such as paper in a wallet, can reduce human memory burdens by converting a cryptword into a "something you have" token until memorized [59].

Using a desktop, smartphone, or online password manager, it is easy to generate and store random passwords; these can be generated to be easy to enter [31] and remember for contexts precluding access to the password manager, such as logging into a computer or unlocking a mobile device. By using an encrypted password manager, users should only need to remember a single cryptword and a small handful of authentication passwords to access devices to access the password manager (possibly none if using biometric authentication for device access). If a user must have multiple password managers, we suggest storing easily-entered cryptwords for these password managers in a single portable primary password manager.

We model users as being willing and able to memorize between 5 and 7 distinct passwords with none or few strong against offline attacks, based on prior work [28]. A cryptword could be considered equivalent to 4–5 of these passwords, leaving sufficient remaining capacity to store an additional authentication password or two for accessing devices storing the password manager. Biometrics can further reduce the cognitive load, and secure non-computer password storage can reduce the cognitive load to only the effort to locate, read, and enter the few authentication passwords and the cryptword stored on the paper.

It is difficult to quantify the cost of using random passwords and cryptwords, especially when combined with password managers, and given the substantial variety of types of generated passwords (see Table 1 for some examples).

6.5 User Freedom

Users are not homogenous, and sometimes have goals and values which run counter to paternalistic security measures [26]. A common and simple example is how character class restrictions on websites can impede or make it more difficult to use user-generated random passwords, such as from a password manager.

With authentication passwords, there are two parties involved: the user, and the system they authenticate to. As an involved party, the system can reasonably require some controls on authentication methods to prevent misuse of resources by unauthorized parties. However, these controls should be reasonable and respect user autonomy.

Users should be allowed to select their own passwords. Although assigned random passwords have a known security strength, users may generate random passwords via other means which have equal or higher security. Preventing users from generating their own passwords can decrease a user's security. However, it would be helpful to generate a fresh random password at the site's security level as a suggested password nudge for users [68]. Authentication password maximum lengths should generally [62] be tuned to protect from denial of service attacks. 4096 bytes is a reasonable upper limit [11]. Similarly, systems should support password input of at least all 95 printable ASCII characters. Unicode is desirable, but should be tested for common problems [19]. Finally, password resets [24] should only be required in response to suspected server compromise.

Cryptwords have only one party involved: the user. Systems which use cryptwords should not have maximum or minimum lengths; the user should be permitted to use an empty string if it meets their security needs. Unmasking [37] (letting the user unhide their password during entry) is especially helpful for cryptwords as these are long and thus subject to increased typos [60]. Cryptword generators should allow users to control as many of the parameters in the equations as can be reasonably usable to the target population. In some cases, this may be as simple as a monetary value slider, with other values fixed. Systems should give the user enough information to decide what level of security they need. As an example, if a protocol publishes a verifier publicly, users will need to know to rotate their cryptwords and protected data after an expiration period of their choice (see Section 4.4). Additionally, users need to be aware that they are inputting a cryptword; using the label "cryptword" instead of "password" may help (see Section 6.2).

6.6 Lockouts and Rate Limiting

Lockouts and rate limiting can be used to conduct a denial of service (DoS) attack against a system or an individual account [42]. For usability, lockout limits should be higher than three failures [21, 48, 56].

Against a relatively unconstrained attacker, resetting passwords on account unlock provides little advantage to the defender [24] over simply unlocking the account. In brief, if an attacker is given a guesses against a random password of b bits, their odds of success before the account is locked are $\frac{a}{2^b}$. If the password is later reset to the same level of security, the attacker has another chance with odds $\frac{a}{2^b}$. If an account may be unlocked and the password reset to the same security level n times, the aggregate risk of an attacker guessing the random password is $1 - \left(1 - \frac{a}{2^b}\right)^n$. With a 4-digit PIN and a 3-strikes lockout policy, an attacker's odds are 0.03%. However, if we allow the attacker to continue guessing after PIN reset up to a maximum of 10 lockouts, their cumulative odds raise to $\approx 0.3\%$. It is simple and conservative to model this as allowing $a \times n = 30$ attempts against a single random PIN. So, if the user can unlock their account, the total attacks across all times the account may be unlocked should be factored into the attacker's capabilities such as in Equation 1.

Should an account be locked, it is reasonable to still provide access with additional security guarantees. These could include adding a CAPTCHA (increasing attacker costs though not preventing automation [6]), requiring a second authentication factor, increasing the minimum strength of a new random password out of reach for a reasonably unbounded attacker, or switching to an alternate authentication system which may be more secure but less usable. These may be used with a Risk-Based Authentication (RBA) system which adds enhanced protections for suspicious access.

6.7 Stateless Password Managers

Stateless password managers such as Spectre [12] allow a user to regenerate any password from just their primary password and a system, usually software. These have essentially the same advantages of passwords [17] except possible reliance on a computer: since passwords are typically entered on computing devices, this disadvantage would appear to have minimal impact. However, there are some security considerations which are not immediately obvious.

Any leaked password can serve as a verifier for the primary password, which is a cryptword used to generate all other passwords. While the primary password can be protected by a strong pwKDF, any updates to the parameters of the pwKDF require updating all generated passwords across all sites. As there is no list of sites, this may require supporting past pwKDF parameters in the UI, complicating usability. Additionally, if the primary password is not changed, a leaked password generated with previous parameters can still compromise the primary password.

While these problems can be solved with a password generation key itself encrypted via the primary password, the system would no longer be stateless, losing a unique advantage over other password managers. Without modification, these can be used securely by choosing for the primary password a cryptword which can resist attacks indefinitely, just as other cases where a protocol posts protected information publicly which cannot be rotated.

The "passwortkarte" [2, 46] ("password card") is another kind of stateless password manager. These require something to be carried but do not require a computer; the algorithm is performed by the user. If these are generated from a seeded random number generator [51], the same security considerations apply.

6.8 Quantum Computers

Bennett et al. proved [10] that a quantum computer can at best half the key space for brute force key searching. This means that if quantum computers are a legitimate threat model, b should be doubled. Additionally, any quantum attack speedup against a chosen pwKDF should be factored into g like any other attack, as discussed in Section 5.3.

7 CONCLUSIONS AND FUTURE WORK

We present two significant paradigm shifts. The first, that cryptword and random authentication password lengths can stop increasing — forever. Instead of requiring ever longer passwords, we can generate passwords and rehash them when users enter them to maintain a security value above some minimum threshold. The second, that passwords and cryptwords are subject to very different attack scenarios, and should be clearly distinguished in user interfaces. Ultimately, we feel "cryptword" should become a household term.

We have provided reasonable upper limits for authentication password lengths and proposed some reasonable cryptword lengths. We expect this to influence future authentication papers, informing the lengths they target and the kinds of usage scenarios they envision.

Future Work. There are many interesting areas of research to pursue in the future.

How well does the term “cryptword” resonate with end users? Will it change behavior after a brief explanation, or will users interpret it as just a stronger password?

How often are cryptwords entered? There is work related to authentication event frequency, but as cryptwords are not used to authenticate, there is no comparable work for cryptword use. This would inform the vulnerability window when choosing a suitable value of g.

We expect that with a fixed target to hit, researchers will identify memorability issues surrounding these longer “something you know” tokens, including innovating entry mechanisms beyond the keyboard, or memory beyond recall. We expect this to contribute greatly to the usability of both encryption and authentication in the future.

Future research might investigate how best to encode cryptwords for memorability. Shay et al. show [60] that for shorter random passwords where letters and words both fit within Miller's 7 ± 2 chunks for working memory [47], random passphrases and passwords are equally memorable. Do these results hold for cryptwords, which are longer, or does the reduced chunking from a larger dictionary aid memorability?

We need to better understand user's threat models to determine if a single secure generated cryptword for a password manager is an acceptable security tradeoff. Users may be more willing to accept cryptwords if they reduce the overall authentication burden, as the password manager can handle other authentication tasks.

Would users feel more comfortable if they can pick a cryptword from a selection of options, and the security level adjusted to subtract a corresponding number of bits (e.g., 3 bits if selecting from eight alternatives)?

There is also the issue of educating system administrators and security software developers. How can this radical notion reach them, so new systems are designed more usably, and current systems updated to become more usable?

Would redundancy in cryptwords aid recovery? Passwords can be recovered via alternate authentication; cryptwords may be recovered by writing them down and storing them somewhere secure. However, with forward error correcting codes in the bits of the cryptword, it may be possible to nudge users to re-train on the cryptword, or be more flexible in how it is entered without reducing security.

Cryptwords and authentication passwords should be randomly generated to provide guaranteed security, but because users have differing needs we recommend allowing users the freedom to generate these outside of system control (see Section 6.5). This means at least some will choose to forego alternate means of secure random generation, and use user-chosen passwords as cryptwords. These will naturally fall into a Zipf's law distribution. Can our equations and analysis be meaningfully adapted to analyze guessability of Zipf's law distributions? Can user-chosen cryptwords be strong enough to resist offline attacks under the same threat models?

Finally, future attacks will require new defenses. Quantum computers are one such advancement. A post quantum pwKDF will enable our results to remain relevant even as quantum computing becomes more available.

ACKNOWLEDGMENTS

The authors thank the NSPW attendees for their helpful feedback. We especially would like to recognize our shepherds, Elizabeth Stobert and David Balenson, who have significantly helped this paper to communicate clearly.

This work was partially supported by the National Science Foundation under Grant No. CNS-1816929 (https://nsf.gov/awardsearch/showAward?AWD_ID=1816929). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] Rack Solutions Inc. 2020. *How many servers does a data center have?* Rack Solutions Inc. Retrieved 2022-05-17 from https://www.racksolutions.com/news/blog/how-many-servers-does-a-data-center-have/

[2] 2020. *Passwortkarten: Konzepte und Generatoren.* Retrieved 2022-05-30 from https://www.e-passwordcard.com/de/passwortkarten-konzepte-und-generatoren/

[3] The Federal Reserve 2021. *Survey of Consumer Finances, 1989 - 2019.* The Federal Reserve. Retrieved 2022-05-17 from https://www.federalreserve.gov/econres/scf/dataviz/scf/table/#series:Net_Worth;demographic:agecl;units:mean;population:all;range:1989,2019

[4] Amazon Web Services, Inc. 2022. *Amazon EC2 Spot - Save up-to 90% on On-Demand Prices.* Amazon Web Services, Inc. Retrieved 2022-05-17 from https://aws.amazon.com/ec2/spot/

[5] Joël Alwen and Jeremiah Blocki. 2017. Towards practical attacks on argon2i and balloon hashing. In *2017 IEEE European Symposium on Security and Privacy (EuroS&P).* IEEE, 142–157.

[6] anti-captcha.com n.d. *Anti Captcha: Captcha Solving Service. Bypass reCAPTCHA, FunCaptcha Arkose Labs, image captcha, GeeTest, HCaptcha.* Retrieved 2022-07-26 from https://anti-captcha.com/

[7] Jean-Philippe Aumasson. 2019. *Password Hashing Competition.* Retrieved 2021-10-04 from https://www.password-hashing.net/

[8] James Bamford. 2012. The NSA is building the country's biggest spy center (watch what you say). *Wired, March* 15 (2012).

[9] Ken ZatykoDr John Bay. 2011. The Digital Forensics Cyber Exchange Principle. (2011).

[10] Charles H Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. 1997. Strengths and weaknesses of quantum computing. *SIAM journal on Computing* 26, 5 (1997), 1510–1523.

[11] James Bennett. 2013. *Issue: denial-of-service via large passwords | Security releases issued.* Retrieved 2022-08-15 from https://www.djangoproject.com/weblog/2013/sep/15/security/#s-issue-denial-of-service-via-large-passwords

[12] Maarten Bilemont. 2011. *Spectre: Passwords, Privacy-first.* Retrieved 2022-05-18 from https://spectre.app/

[13] A. Biryukov, D. Dinu, D. Khovratovich, , and S. Josefsson. 2021. *Argon2 Memory-Hard Function for Password Hashing and Proof-of-Work Applications.* RFC 9106. RFC Editor. https://www.rfc-editor.org/rfc/rfc9106

[14] Alex Biryukov, Daniel Dinu, and Dmitry Khovratovich. 2017. Argon2: the memory-hard function for password hashing and other applications. (24 Mar 2017). Retrieved 2022-11-21 from https://github.com/P-H-C/phc-winner-argon2/blob/master/argon2-specs.pdf.

[15] Matt Blaze, Whitfield Diffie, Ronald L Rivest, Bruce Schneier, and Tsutomu Shimomura. 1996. *Minimal key lengths for symmetric ciphers to provide adequate commercial security. A Report by an Ad Hoc Group of Cryptographers and Computer Scientists.* Technical Report. INFORMATION ASSURANCE TECHNOLOGY ANALYSIS CENTER FALLS CHURCH VA.

[16] Joseph Bonneau. 2013. *Moore's Law won't kill passwords.* Retrieved 2022-03-30 from https://www.lightbluetouchpaper.org/2013/01/17/moores-law-wont-kill-passwords/

[17] Joseph Bonneau, Cormac Herley, Paul C Van Oorschot, and Frank Stajano. 2012. The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *2012 IEEE Symposium on Security and Privacy.* IEEE, 553–567.

[18] Joseph Bonneau and Stuart Schechter. 2014. Towards reliable storage of 56-bit secrets in human memory. In *23rd {USENIX} Security Symposium ({USENIX} Security 14).* 607–623.

[19] Joseph Bonneau and Rubin Xu. 2012. Of contraseñas, סיסמאות, and 密码–Character encoding issues for web passwords. In *Web 2.0 Security & Privacy* (San Francisco, CA, USA). https://www.jbonneau.com/doc/BX12-W2SP-passwords_character_encoding.pdf

[20] S. Bradner. 1997. *Key words for use in RFCs to Indicate Requirement Levels.* RFC 2119. RFC Editor. https://www.rfc-editor.org/rfc/rfc2119

[21] Sacha Brostoff and M Angela Sasse. 2003. "Ten strikes and you're out": Increasing the number of login attempts can improve password usability. (2003).

[22] Boštjan Brumen. 2020. System-Assigned Passwords: The Disadvantages of the Strict Password Management Policies. *Informatica* 31, 3 (2020), 459–479.

[23] Steven Cherry. 2004. Edholm's law of bandwidth. *IEEE spectrum* 41, 7 (2004), 58–60.

[24] Sonia Chiasson and Paul C Van Oorschot. 2015. Quantifying the security advantage of password expiration policies. *Designs, Codes and Cryptography* 77, 2 (2015), 401–408.

[25] Michael Clark. 2019. *Fast hash threat model might be optimistic.* Retrieved 2022-05-04 from https://github.com/dropbox/zxcvbn/issues/272

[26] Steve Dodier-Lazaro, Ruba Abu-Salma, Ingolf Becker, and M Angela Sasse. 2017. From paternalistic to user-centred security: Putting users first with value-sensitive design. In *CHI 2017 Workshop on Values in Computing*. Values In Computing.

[27] Jayesh Doolani, Matthew Wright, Rajesh Setty, and SM Taiabul Haque. 2021. LociMotion: Towards Learning a Strong Authentication Secret in a Single Session. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems.* 1–13.

[28] Dinei Florêncio and Cormac Herley. 2007. A large-scale study of web password habits. In *Proceedings of the 16th international conference on World Wide Web.* 657–666.

[29] Dinei Florêncio, Cormac Herley, and Paul C Van Oorschot. 2014. An administrator's guide to internet password research. In *28th Large Installation System Administration Conference (LISA14).* 44–61.

[30] Damien Giry. 2020. *Cryptographic Key Length Recommendation.* Retrieved 2021-10-04 from https://www.keylength.com/

[31] Kristen K Greene, Melissa A Gallagher, Brian C Stanton, and Paul Y Lee. 2014. I can't type that! P@$$w0rd entry on mobile devices. In *International Conference on Human Aspects of Information Security, Privacy, and Trust.* Springer, 160–171.

[32] David Robert Grimes. 2016. On the viability of conspiratorial beliefs. *PloS one* 11, 1 (2016), e0147905.

[33] gsmarena.com n.d. *Nokia 3.* Retrieved 2022-05-11 from https://www.gsmarena.com/nokia_3-8572.php

[34] SM Taiabul Haque, Mahdi Nasrullah Al-Ameen, Matthew Wright, and Shannon Scielzo. 2017. Learning system-assigned passwords (up to 56 bits) in a single registration session with the methods of cognitive psychology. In *Proceedings of the Network and Distributed System Security Symposium (NDSS 2017), USEC*, Vol. 17.

[35] Cormac Herley. 2009. So long, and no thanks for the externalities: the rational rejection of security advice by users. In *Proceedings of the 2009 workshop on New security paradigms workshop.* 133–144.

[36] Cormac Herley and Paul Van Oorschot. 2011. A research agenda acknowledging the persistence of passwords. *IEEE Security & privacy* 10, 1 (2011), 28–36.

[37] Jack Holmes. 2014. *Password Masking.* Retrieved 2022-05-16 from http://passwordmasking.com/

[38] Stanislaw Jarecki, Hugo Krawczyk, and Jiayu Xu. 2018. OPAQUE: an asymmetric PAKE protocol secure against pre-computation attacks. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques.* Springer, 456–486.

[39] Charles E Leiserson, Neil C Thompson, Joel S Emer, Bradley C Kuszmaul, Butler W Lampson, Daniel Sanchez, and Tao B Schardl. 2020. There's plenty of room at the Top: What will drive computer performance after Moore's law? *Science* 368, 6495 (2020), eaam9744.

[40] Arjen K Lenstra and Eric R Verheul. 2001. Selecting cryptographic key sizes. *Journal of cryptology* 14, 4 (2001), 255–293.

[41] Michael D Leonhard and VN Venkatakrishnan. 2007. A comparative study of three random password generators. In *2007 IEEE International Conference on Electro/Information Technology.* IEEE, 227–232.

[42] Yu Liu, Matthew R Squires, Curtis R Taylor, Robert J Walls, and Craig A Shue. 2019. Account Lockouts: Characterizing and Preventing Account Denial-of-Service Attacks. In *International Conference on Security and Privacy in Communication Systems.* Springer, 26–46.

[43] Seth Lloyd. 2000. Ultimate physical limits to computation. *Nature* 406, 6799 (2000), 1047–1054.

[44] Bo Lu, Xiaokuan Zhang, Ziman Ling, Yinqian Zhang, and Zhiqiang Lin. 2018. A measurement study of authentication rate-limiting mechanisms of modern websites. In *Proceedings of the 34th Annual Computer Security Applications Conference.* 89–100.

[45] Philipp Markert, Daniel V Bailey, Maximilian Golla, Markus Dürmuth, and Adam J Aviv. 2020. This PIN can be easily guessed: Analyzing the security of smartphone unlock PINs. In *2020 IEEE Symposium on Security and Privacy (SP).* IEEE, 286–303.

[46] Peter Mayer, Alexandra Kunz, and Melanie Volkamer. 2017. Analysis of the Security and Memorability of the Password Card. (Dec 2017).

[47] George A Miller. 1956. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological review* 63, 2 (1956), 81.

[48] Gilbert Notoatmodjo and Clark Thomborson. 2009. Passwords and perceptions. In *Proceedings of the Seventh Australasian Conference on Information Security-Volume 98.* Citeseer, 71–78.

[49] Marek Palatinus, Pavol Rusnak, Aaron Voisine, and Sean Bowe. 2013. *Mnemonic code for generating deterministic keys.* Retrieved 2021-06-01 from https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki

[50] Paul. 2017. *The Top 20 most popular phones of 2017.* Retrieved 2022-05-11 from https://www.gsmarena.com/the_20_most_popular_phones_of_2017-news-28892.php

[51] pepsoft.org n.d. *Your PasswordCard - Algorithm.* Retrieved 2022-05-30 from https://www.passwordcard.org/algorithm.html

[52] Bc Vojtěch Polášek. 2019. Argon2 security margin for disk encryption passwords. (2019).

[53] Niels Provos and David Mazières. 1999. A Future-Adaptable Password Scheme.. In *USENIX Annual Technical Conference, FREENIX Track*, Vol. 1999. 81–91.

[54] Arnold Reinhold. 2014. *Time to add a word.* Retrieved 2022-03-30 from https://diceware.blogspot.com/2014/03/time-to-add-word.html

[55] A G Reinhold. 2022. *The Diceware Passphrase Home Page.* Retrieved 2022-03-11 from https://theworld.com/~reinhold/diceware.html

[56] Karen Renaud, Rosanne English, Thomas Wynne, and Florian Weber. 2014. You Have Three Tries Before Lockout. Why Three?.. In *HAISA.* 101–111.

[57] Scott Ruoti, Jeff Andersen, Travis Hendershot, Daniel Zappala, and Kent Seamons. 2016. Private Webmail 2.0: Simple and easy-to-use secure email. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology.* 461–472.

[58] Sarah Scheffler and Mayank Varia. 2021. Protecting Cryptography Against Compelled {Self-Incrimination}. In *30th USENIX Security Symposium (USENIX Security 21).* 591–608.

[59] Bruce Schneier. 1999. *Key Length and Security - Crypto-gram: October 15, 1999 - Schneier on Security.* Retrieved 2021-10-04 from https://www.schneier.com/crypto-gram/archives/1999/1015.html#KeyLengthandSecurity

[60] Richard Shay, Patrick Gage Kelley, Saranga Komanduri, Michelle L Mazurek, Blase Ur, Timothy Vidas, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. 2012. Correct horse battery staple: Exploring the usability of system-assigned passphrases. In *Proceedings of the eighth symposium on usable privacy and security.* 1–20.

[61] switch.com. n.d. *Switch TAHOE RENO Colocation Data Center Campus.* Retrieved 2022-07-18 from https://www.switch.com/tahoe-reno-campus/

[62] OWASP CheatSheets Series Team. 2021. *Implement Proper Password Strength Controls - Authentication - OWASP Cheat Sheet Series.* Retrieved 2022-08-15 from https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html#implement-proper-password-strength-controls

[63] The Astronomical Almanac Online! 2021. *2021 Selected Astronomical Constants.* Retrieved 2022-03-11 from http://asa.hmnao.com/static/files/2021/Astronomical_Constants_2021.pdf

[64] Ding Wang, Haibo Cheng, Ping Wang, Xinyi Huang, and Gaopeng Jian. 2017. Zipf's law in passwords. *IEEE Transactions on Information Forensics and Security* 12, 11 (2017), 2776–2791.

[65] Daniel Lowe Wheeler. 2016. *dropbox/zxcvbn: Low-Budget Password Strength Estimation.* https://github.com/dropbox/zxcvbn

[66] Daniel Lowe Wheeler. 2016. zxcvbn: Low-budget password strength estimation. In *25th {USENIX} Security Symposium ({USENIX} Security 16).* 157–173.

[67] Jeff Yan, Alan Blackwell, Ross Anderson, and Alasdair Grant. 2004. Password memorability and security: Empirical results. *IEEE Security & privacy* 2, 5 (2004), 25–31.

[68] Samira Zibaei, Dinah Rinoa Malapaya, Benjamin Mercier, Amirali Salehi-Abari, and Julie Thorpe. 2022. Do Password Managers Nudge Secure (Random) Passwords?. In *Eighteenth Symposium on Usable Privacy and Security (SOUPS 2022).* 581–597.