Modern password security for users

User-focused recommendations for creating and storing passwords

By Ian Maddox and Kyle Moschetto, Google Cloud Solutions Architects

This guide describes password guidance and recommendations for users of online applications that require authentication. It establishes a set of user-focused recommendations for creating and storing passwords, including balancing password strength and usability. A related guide, Modern password security for system designers, offers guidance for the engineers who build online applications that require authentication.

The technology world has been trying to improve on the password since the early days of computing. Shared-knowledge authentication is problematic, because information can fall into the wrong hands or be forgotten. The problem is magnified by systems that don't support real-world secure use cases and by the all-too-common decision of end users to take shortcuts.

Best practices quick reference

DO	DON'T
Use a password manager.	Write down your passwords or store them unencrypted.
Use unique passwords for every site and application.	Reuse passwords on multiple sites.
Make long, random passwords.	Use short passwords or obvious character substitutions (such as @ for a).
Use multi-word passphrases.	Use single dictionary words (such as “password”).
Use salted passphrases or algorithms for security questions.	Answer security questions honestly.
Trust and review authoritative sources for best practices.	Think personal web security practices you learned 10 years ago still apply.

Overview

This document outlines the following:

- Trusted sources of thoughtful and researched information about password security
- Recommendations for safely handling end-user passwords
- Common anti-patterns and myths around password security
- Additional technologies to explore

Terminology

entropy
: Related to passwords, a measurement of how unpredictable a password is. Commonly represented as a number of bits.

hash
: In cryptography, a hash is the result of one-way, irreversible, deterministic encryption algorithm. It is mathematically extremely hard to guess the value that was used to create a hash. Similarly it is extremely unlikely to find two values that produce the same hash. Because a hashing algorithm will always produce the same hash for a given input, hashes can be used as secure stand-ins for sensitive data such as passwords. Login attempts can be authenticated by hashing the provided password and comparing it to the hash on file.

MFA and 2FA
: *Multi-factor authentication and two-factor authentication*. Methods for additional verification, traditionally in addition to a password.

password manager
: Software that assists in creating, storing, and retrieving passwords.

rainbow table
: A systematically generated table of precomputed hashes and their pre-hashed values. Commonly used to match a hash to a password or passphrase.

salt
: Random data added to an input, usually used to produce a non-predictable variant hash of the original input and to break rainbow tables

When to use this guide

This document covers many of the options for using passwords to control access to resources in the cloud-native world. This document does not attempt to provide all of the answers, nor to provide specific solutions. The goal is merely to raise awareness among end users, security professionals, system designers, and cloud architects.

An overview of passwords in 2019

The topic of password security seemingly contains as many strong opinions as there are possible passwords. Organizations draw different lines between what behaviors are secure and those that they consider insecure. This document does not intend to provide definitive guidance on password use in every application, but instead to outline a collection of topics that all users and engineers should consider

whenever they use or design password-based systems. Each application will have a unique security posture consistent with the sensitivity of the data protected by the application.

Passwords are in use everywhere. They remain the most common authentication mechanism, and are used across every vertical, industry, and application type. Unlike other common security features, passwords stand alone in that the industry is still debating how they should be implemented, managed, and stored.

The main reason for this lack of agreement is that the world of security evolves quickly. What we find secure today might be considered deprecated, compromised, broken, or simply out of date tomorrow. The things we feel are cutting edge now won't be for long as new technology, tools, and procedures supersede them. Users and systems designers must stay knowledgeable about the trends and changes in the world of password security.

Authoritative sources

There are a number of trusted sources for the latest guidance on the use of passwords. The following list is far from exhaustive, but represents highly regarded opinions on the matter:

- The National Institute of Standards and Technology (NIST) is part of the US Department of Commerce and provides guidance in many areas. Their Digital Identity Guidelines (SP 800-63) cover a range of password-related topics.
- The National Cyber Security Centre (NCSC) is the UK's independent authority on cybersecurity. They provide guidance in many areas, including password security.
- The Open Web Application Security Project (OWASP) is a worldwide not-for-profit organization focused on improving the security of software. Their collection of documentation available on GitHub includes cheatsheets on authentication, password storage, and more.
- Google provides beginner documentation on creating strong passwords, making your Google account more secure, managing your passwords, and the 12 best practices for user account, authorization and password management.

Password considerations for users

Because every member of an online community has to use a password at some point, understanding password care from a user perspective is a great place to start.

Creating passwords

When you create passwords, you have two primary concerns: the length and strength of your passwords, and the operational security of your passwords. The following sections discuss these concerns.

Length and strength

At its most basic level, the use of a password is designed to prevent an unauthorized person or device from accessing a resource. Prevention means making it mathematically difficult to guess your password. How this difficulty is measured is called *password entropy*. Put simply, entropy is a measure of complexity and randomness. This number is measured in bits.

Entropy calculation

Although there are a few competing ideas of how to calculate true password entropy, one simple example is the following formula:

$$\textit{password length} \times \log_2(\textit{possible characters}) = \textit{password entropy}$$

By taking the $\log_2$ of the number of characters available and multiplying it by the character length of the password, you can calculate the number of bits of password entropy. The more bits of entropy that your password has, the more difficult it is for a computer to guess, predict, or successfully attack it by brute force. Each bit of entropy mathematically doubles the difficulty of guessing the password correctly. For example, 28 bits of entropy represents 2^{28} or 268,435,456 possible passwords. A password that consists of lowercase English letters (26 characters in the set), and is 6 characters in length, has ~28 bits of entropy.

Table 1 illustrates the rapid growth of password complexity. The dictionary column on the right represents whole words chosen from the *Oxford English Dictionary* instead of individual characters.

Charset	a–z	a–z, A–Z	a–z, A–Z, 0–9	a–z, A–Z, 0–9, symbols	Full UTF-8	Dictionary
Charset size	**26**	**52**	**62**	**95**	**137,000**	**171,476 words**
Chars/ words	**Bits of entropy**					
4	19	23	24	26	68	70
6	28	34	36	39	102	104
8	38	46	48	53	137	139
12	56	68	71	79	205	209
16	75	91	95	105	273	278
32	150	182	191	210	546	556
60	282	342	357	394	1,024	1,043

Table 1. Growth of entropy in password complexity

Dictionary words and passphrases

If you have a dictionary like the *Oxford English Dictionary* with 171,476 headword entries (for words in current use), then that's the size of your "character" set. If you choose one word, that's the equivalent of choosing a single character password from a 171,476-character alphabet. Using the preceding formula, a password based on a single dictionary word has:

$$1 \times \log_2(171{,}476) = 17 \text{ bits of entropy}$$

By increasing the length of the password to 4 words (that is, by creating a 4-word passphrase), you get 70 bits of entropy. This level of entropy is mathematically very good as long as you aren't up against a quantum computer. Table 2 illustrates that this passphrase would take up to 2.7 million years to guess.

Compromised passwords and rainbow tables

Because passwords are normally stored as hashes in password databases, having a large list of known password-to-hash values can be extremely valuable to a password hacker. If a hacker obtains your hashed password and that hash matches a known password-to-hash value, then the hacker knows your password. Collections of these password-to-hash values are known as *rainbow tables*. These tables are used like a reverse directory of hashes to passwords.

The values hashed in rainbow tables are generated through several methods. Two basic approaches are 1) to generate every possible combination of characters for a given charset, or 2) to gather a list of previously stolen passwords. In the second case, when the list of passwords is obtained, each entry is hashed and the

hash result is stored next to the plaintext password in the rainbow table. Passphrase rainbow tables can also be generated using entire words.

Skilled rainbow table makers apply statistics, historical data, and human psychology to generate vast stores of pre-computed hashes for passwords and passphrases. These people often tailor the tables to the password complexity rules provided by a website or service to generate only valid guesses. They understand commonly used shortcuts for password generation and can quickly generate a breathtaking number of hashes that meet the criteria for personal password strategies like "a dictionary word with l337 characters plus one special character and a single digit at the end," which covers passwords like p455#w0rd9.

An advanced threat actor might go further and generate a rainbow table based on information they know about their target. They can generate a custom dictionary based on website, emails, social media content, public facts, and stolen information. If you choose the words in a passphrase manually, you might be inadvertently biased toward words you use or like and ultimately generate a key that is easier to guess based on your activity. Adding salted values to each word, as outlined in the password salting section, not only mitigates the effectiveness of rainbow tables, but can decrease predictability (increase entropy) many times over even for a reduced-size character set.

Brute force and entropy

When you understand password entropy, you can compare the speed of modern processors to find out how long it would take a computer to guess a specific password. This is commonly referred to as the *work factor* to break a password. For example, if a computer can perform 1 million calculations per second and there are 10 million possible combinations for a specific password, then it would take that computer 10 seconds to guess all possible combinations of a password.

The password guess rate or hashing rate depends greatly on the power of the computer and the algorithm being used. The preferred algorithms for generating passwords, such as Argon2, PBKDF2, Scrypt, and Bcrypt, employ several strategies to make it difficult to guess passwords efficiently by brute force. These techniques include math that resists computational acceleration by specialized hardware, functions that require large amounts of RAM, and calculations designed deliberately to be much slower than deprecated hashing strategies such as MD5 or SHA1.

Bits of entropy is an interesting and valuable way to look at complexity, but it’s often more practical to think in terms of how long it would take a real computer to guess a password. Table 2 examines password complexity in terms of time to crack.

Charset	a–z	a–z, A–Z	a–z, A–Z, 0–9	a–z, A–Z, 0–9, symbols	Full UTF-8	Dictionary
Charset size	**26**	**52**	**62**	**95**	**137,000**	**171,476 words**
Chars/ words	**Max time to crack @ 10,000,000 hash/sec**					
4	< 1 sec	< 1 sec	1.5 sec	8.1 sec	1.1 million years	2.7 million years
6	30.9 sec	33 min	1.6 hours	20.4 hours	21 trillion years	> universe lifespan
8	5.8 hours	2 months	8.3 months	21 years	> universe lifespan	> universe lifespan
12	302.6 years	1.2 million years	10 million years	1.7 billion years	> universe lifespan	> universe lifespan
16	138 million years	9 trillion years	151 trillion years	> universe lifespan	> universe lifespan	> universe lifespan
32	> universe lifespan	> universe lifespan	> universe lifespan	> universe lifespan	> universe lifespan	> universe lifespan
60	> universe lifespan	> universe lifespan	> universe lifespan	> universe lifespan	> universe lifespan	> universe lifespan

Table 2. Password complexity and time to crack

This table demonstrates the amount of time required to generate every possible password combination for a given complexity if the computer can guess 10 million times per second. The dictionary column on the far right represents whole words chosen from the *Oxford English Dictionary* instead of individual characters. The amount of effort quickly exceeds the expected lifespan of our universe. Using a 6-word randomly generated passphrase is about as strong as a 32-character lowercase password.

Based on these explanations, only the following two factors will enhance entropy, and therefore enhance the overall strength of your passwords:

- Increasing the available characters in a set.
- Increasing the length of the password.

By using many possible character sets in your passwords, including uppercase characters, lowercase characters, digits, multilingual plane characters, symbols, emojis, and so on, you make it more difficult for a computer to guess. By increasing the overall length of a password, you increase the amount of time it will take for a computer to guess all possible combinations of that previous character set.

Character substitution

Many people believe they can throw off password cracking attempts by replacing individual characters with similar-looking characters—a process called *character substitution*. This is most commonly done by switching the word “password” to something like “p@$$w0rd”. Such character substitutions are common and anticipated by password crackers.

The most commonly used substitutions are so well known that they add minimal complexity to a password. A malicious actor with an exfiltrated database of password hashes can, without great effort, crack a

substantial percentage of passwords that use only this technique. The software most commonly used to generate rainbow tables has the ability to perform common character substitutions like the following:

Character	Common replacements
A	4, /-\, /_\, @, /\, Д
B	8,|3, 13, |}, |:, |8, 18, 6, |B, |8, lo, |o, j3, ß
C	<, {, [, (, ©, ¢
D	), |}, |], |>
E	3, £, £, €
F	|=, ph, |#, |"
G	[, -, [+, 6, C-
H	4, |-|, [-], {-}, }-{, }{, |=|, [=], {=}, /-/, (-),)-(, :-:, I+I
I	1, |, !, 9
J	_|, _/, _7, _), _], _}
K	|<, 1<, I<, |{, I{
L	|_, |, 1,][
M	44, |V|, ^^, /\\, /X\, []V][, []V[],][\\//][, (V),//., .\\, N\
N	||, /\/, /V,][\\][, И
O	0, (), [], {}, <>, Ø, oh
P	|o, |O, |>, |*, |°, |D, /o, []D, |7
Q	O_, 9, (,), 0, kw
R	|2, 12, .-, |^, I2, Я, ®
S	5, $, §
T	7, +, 7`, '|' , `|` , ~|~ , -|-, ']['
U	|_|, _\, /_/, _/, (_), [_], {_}
V	\/
W	\/\/, (/\), \^/, |/|, \X/, \\', '//, VV, _|_/, \\/\\//, Ш, 2u, \\/
X	%, *, ><, }{,)(, Ж
Y	`/, ¥, |/, Ч
Z	2, 5, 7_, >_,(/)
0	O, D, space
1	I, L
2	Z, e
3	E, ❤

4	h, A
5	S
6	b, G
7	T, j
8	X
9	g, J

Table 3. Common character substitutions

These tricks aim to increase entropy by increasing the randomness of a password. A more mathematically sound approach is to salt the passwords with truly random characters, as described later in the password salting section.

Previous use

Reusing passwords on multiple sites and applications significantly degrades the security of the password. This happens because password hackers commonly use rainbow tables of previously compromised passwords. These tables contain previously leaked or discovered passwords on a massive scale. Hackers can use these known password lists to look up repeated uses in other systems. Any key that has been previously leaked or is available in a dictionary file or rainbow table is effectively as unsecure as the password "password". It's prudent to assume any previously leaked password can be broken by modern password cracking software in less than a second.

Passphrases

A password might have dozens or hundreds of options for each character, but a passphrase makes several selections out of a dictionary with potentially hundreds of thousands of words. If employed correctly, a passphrase can offer near impenetrable security.

It's difficult to talk about password security without referencing the now-legendary xkcd comic strip. In this example, the author uses four common words, displayed in an easy-for-humans-to-remember method, as an example of a strong password. These types of passphrases are good when you have to memorize a key and are unable to use a password manager. You might encounter this situation at a retail store kiosk or other standalone device or system.

Passphrases consist of randomly chosen dictionary words, so the strength of a passphrase depends greatly on the dictionary size. It is bad practice to use names unless they were chosen using the same random process. Picking randomly from the Oxford English Dictionary's 171,476 words is vastly more secure than choosing from the limited number of pet names, streets, and favorite associations you hold. The latter might be only a few dozen words in total, reducing complexity from trillions of combinations to a few million that can be cycled through in seconds.

The secret to a well-made passphrase is to use several truly randomly chosen words from a large dictionary.

Password salting

You can add further complexity to passwords and passphrases by adding random characters. This method to increase password strength is called *salting*. This technique increases the overall randomness of the password without making it much more difficult to remember. Salting, when discussing passwords, describes the act of adding true randomness to mitigate a dictionary attack and make the password harder to guess.

Consider the passphrase "DogCatFishRabbit". This is a reasonable passphrase that has sixteen mixed-case characters yielding 91 bits of entropy. However, this passphrase is a nonrandom set of common and related words. You can increase the overall strength of this passphrase by injecting a few random characters. Take the same password and salt it with the characters “4”, “$”, “{“and “z” and you would end up with something similar to “D4ogCa$tFis{hRabzbit”. This becomes a hybrid password or passphrase that is 20 characters long (131 bits) and is not as subject to a passphrase rainbow table attack.

Making a strong password

To create a strong password, you should avoid easy-to-guess patterns while embracing strategies that ensure randomness. A strong password should satisfy the following rules:

- Avoids well-known character substitutions.
- Disregards character replacement (that is, it doesn't swap “@” for “a”).
- Is used for one site only.
- Uses the largest possible character set.
- Uses a length sufficient to deter modern hacking techniques.
- Doesn’t use numbers, words, or phrases found in your daily life.

These rules make it difficult for humans to generate strong passwords on their own. A long and completely random string of diverse characters is the most difficult to guess. Passwords like these might be as difficult for people to recall as they are to generate.

One approach is to use a password manager, as explained later. If you must manually generate a strong password or passphrase, we recommend employing a combination of randomization strategies to increase password entropy. For example, take the short password "mango2". Here are several techniques that can make this password harder to guess:

- Shift-doubling involves typing the password twice, one time holding shift, one time without: "mango2MANGO@".
- Mirroring is handy where special characters or mixed case are not available: "mango22ognam".
- Stuttering is a simple and effective approach: "mmaannggoo22".
- Concatenation combines two different passwords: "mango2tater7".
- Interleaving two different words eliminates dictionary attacks: "mTaAnTgEoR2&".
- Combining these and other algorithms can make a low-entropy password into a high-entropy password: "maaangoo2ttater7&RETATT@OOGNAAAM".

If you apply all of these rules when you create a password, a malicious actor trying to crack your hashed password would have exponentially more work even if they knew some of the characteristics of the password or the words used to generate it.

Password managers

Password managers are an important aspect of modern security. They are a vault of credentials and other secrets like credit card numbers, and they often come with software to simplify generating new passwords and filling out login forms. Password managers are locked with a single master password.

A common criticism of password managers is that all of your secrets are in one high-value target. Losing control of that vault means losing control of all the accounts that it contains. This is why it is important to choose a password manager that has a high level of trustworthiness, transparency, and multiple layers of security controls.

Here are the minimum requirements for a good password manager:

- It encrypts data at rest (not just passwords).
- It encrypts data in transit.
- It has revocable recovery keys that you can print and store offline.
- It supports multi-factor authentication that does not use SMS (text messages).
- It has a built-in password generator.

In addition, you might want a password manager that has the following:

- Self-hosting
- No way for the password manager vendor to recover your data
- Open source code that you can compile yourself
- Cross-platform support
- Self-locking database access
- Client-managed encryption keys
- Trust no one (TNO) security
- Time-delayed account recovery for transfer upon death or other emergency

General concerns about operational security

Your first choice for new passwords should be the generator built in to your password manager. Free online sites that generate passwords and passphrases might produce good results, but the generated contents might be observed by a man in the middle, by third-party scripts on the page, or by the site itself. If this happens, someone else will have a list of possible passwords associated with your IP address and whatever other identity information they could glean from your browser. And you won't significantly reduce this risk by generating many passwords but only using one.

Your next step is to make sure you have a backup plan. You could experience a computer theft, malicious hack, house fire, or any number of other tragedies that could lead to the loss of your credentials. Use a password manager, and keep an offline copy of the recovery codes you can use if the password for the password manager is lost.

Think about those around you. Each person with a digital identity should consider what will happen to their data should they die prematurely. This applies to financial records, email, cryptocurrencies, and social media. Most services don't publish or even have a policy on how to handle the data of the deceased without a court order. Using a password manager with time-delayed recovery can enable one's survivors to access their data in weeks instead of months or years. Such a feature lets preapproved individuals request account access, and lets you specify the number of days or weeks to wait before the information is released.

Make sure the devices that have access to your password manager are secure and kept up to date with the latest patches. Your overall security is only as strong as the weakest link.

Security questions … and why they are terrible

Security questions are one of the weakest links in account security. Service operators often don't encrypt the answers, and the questions can often be answered by searching public records, deduction, or social engineering. The questions are effectively back-door passwords.

An algorithmic approach to security questions sidesteps this problem by providing an answer that is not intuitively tied to the question asked. The following are some examples of useful algorithms for the alternate use of security questions:

- **Basic: nth letter**
  A basic example of a security question algorithm is to type the first character (or some other specific character position) of each word with no punctuation, whitespace, or capitalization. For example, the first letters of the phrase "What is your favorite kind of bird?" would yield the answer "wiyfkob".

- **Intermediate: indexed letters**
  Add a secret to the algorithm to improve on the nth letter algorithm. For example, you might take a phone number you will always remember and use it as an index to pick the letters. Take the phone number 867-5309 and the question "What is your favorite color?" If you remove all punctuation and whitespace from the question, and count from the first character, the 8th letter is the 'o' in “your”. The 6th letter is the 's' in “is”. Completing the process, we get the letters "osyiaru".

  You don't have to stop at a simple numeric lookup, either. As long as your approach is deterministic and something you can perform in your head, it is far better than simply answering with your actual favorite color.

- **Passphrase**
  Create a passphrase based on the question itself. For example, you could take the last letter of every word in the security question and create a memorable phrase:

  > "What was your high school mascot?"

  The security question has the last letters "T S R L T". You could then use a random word generator and repeatedly run it until you have words that start with each of your letters:

  > "tree surplus rebuilds loud title".

As with manual password generation, the techniques used to generate answers to security questions should be your own. Keep in mind that security questions often have rules that greatly restrict your inputs. Many disallow punctuation, some block anything but letters, and most are case insensitive. Use a combination of strategies for the greatest level of security. Better still, disable security question authentication and use 2FA if that's an option. If it's not an option, consider using an unrelated passphrase or a unique randomly generated string of letters.

If you are prompted to enter your own security question, be certain not to divulge any information that would benefit someone trying to hack that account or any other account of yours.

Alternatives to passwords

No guide to passwords would be complete without mentioning the alternatives to passwords. It seems every few months there's an article with a headline claiming that passwords are dead. The article's content is often a nuanced view on web security and is promoting 2FA, or it's touting a password manager.

There are a handful of legitimate alternatives to passwords, and this section covers them.

Digital certificates

Authentication by digital certificate is one of the oldest non-password mechanisms in use today. It's rarely used for individuals on websites, but it's common among people who must remotely log in to servers by using SSH. A digital certificate is an authentication credential, typically stored in a text file, that can optionally be encrypted with a password. In simple terms, a digital certificate is a massive and practically unguessable password stored in a system file. As with any secret, you must protect the file.

SQRL

The Secure Quick Reliable Login Protocol is a recent addition to the security space. It's designed for end-user authentication to websites and applications. SQRL users run a small client application on their computer or in their browser. Instead of giving sites a password that the site must store securely, the client

provides a public key that is unique to the application or domain to which the user wants to authenticate. The site server provides a unique value to the client, and the client then uses their private key to sign and return that secret. The server verifies the signature using their public key and authenticates the user. Most importantly, a compromised site or service cannot expose its users' credentials in a way that impacts any other site or service.

This system appears to be thoroughly thought through and implemented with an eye toward the evolving threats online and the way real humans interact with security controls. Users can expect to see the option for SQRL login to appear in more places in the coming years.

Biometrics

In an ideal world, biometrics are one of the best authentication mechanisms. Verifying identity using the unique and inherent attributes of an individual is appealing. However, most consumer-grade biometric security systems such as fingerprint, iris, or facial recognition are vulnerable to spoofing attacks, which makes them a poor choice for highly sensitive systems. Other problems arise when your biometric data is leaked, for example, when individuals leave a fingerprint on a glass in a restaurant or when someone obtains a high-resolution photo of their hand or face. Use cases such as sharing passwords or turning over access of a deceased user are difficult to manage.

This field is improving each year, but biometric security is most effective when used as a 2FA, instead of as a primary credential.

Device-based authentication

Device-based authentication is an increasingly popular authentication mechanism. It relies on the user having a trusted and sufficiently secure device such as a personal smartphone or private computer. One common scenario is a user logging into a website on their computer. The user presents their identity to the website, which then uses a secure channel to the user's device to present a prompt verifying the authentication request.

This approach is similar to device-based 2FA, but it relies on an existing authenticated session between the device and service as the primary authentication and sends the prompt as a verification.

What's next

- Read the 12 best practices for user account, authorization, and password management.
- Review Google research on good account hygiene's effectiveness against hijacking.
- Read more about Modern password security for system designers.
- Try out other Google Cloud Platform features for yourself. Have a look at our tutorials.