hashedpass

—

A standard and program for a decentralized, hashed based password manager.

Emil Arner

Version: 1.0.0

Introduction:

hashedpass is a program and a standard for managing passwords, utilizing hashes. It achieves this by combining a master password, website/service name, and a general ID in a hash, such that a hash digest will be used for the password. Due to the fact that password limitations and constraints often exist, *hashedpass* also describes a standard of managing and recording various limitations, to obtain consistent results. It is meant for easy memorization or writing down on a piece of paper. *hashedpass*, due to the reasons above, requires almost no storage and is almost completely decentralized—given the correct parameters, it will always give the requested password. Additionally, to prevent potential cracking of the master password, certain security measures have been put in place, which other algorithms like this have failed to address.

Hashing:

hashedpass will hash the master password, website/service name, and general ID in the following way, with SHA512:

master password=service=id

The master password should preferably be a sentence—or even a paragraph—worth of text. When the master password is used as a sentence worth of text, it should preferably be written in the correct grammatical form, according to the language that it is written in. It is the responsibility of the user to remember this.

ID names are general, meaning different things. An ID will always be made lowercase by the standard and program. An ID could be in reference to an email address, a username, or something similar. Hopefully, the service name should make it clear what the ID name references. Here the standard will define helpful ways to maintain consistency between different types of IDs:

- If the ID is an email:
    - It should contain an @ and the domain afterwards.
- If it is a username, it should remain exactly as it is, without garbage.

Service names, too, will be forced to be lowercase by the standard and the program. Service names should strive to be general. In other words, they should not be (if they do not need to be):

- URLs
- Links
- Domains
- Abbreviations

For example, good parameters would look like:

- Master Password: *The quick brown fox jumped over the lazy dog.*
- Service Name: *discord*
- ID: *johndoe@gmail.com*

After this first hash, the raw digest of the SHA512 algorithm will be fed into an Argon2 hashing algorithm, with the following default parameters:

- 256MiB of memory.
- A salt of b'11111111' (8 bytes of the '1' character).
    - This is to satisfy certain Argon2 library requirements.
- Key length (digest length) of 32 characters.
- Parallelism/Threads of 2.
- 24 iterations/time cost.

Of course, the user can set their own parameters as they see fit; however, they will have to write this down, so as to not lose the ability to regenerate their passwords. It should take around 1-5 seconds to make one hash—albeit, this is obviously relative to computer hardware. To express differing parameters for the Argon2 hashing algorithm, we have devised the following format:

memory:iterations:timecost:size:salt

After that, the Base64 digest of the resulting Argon2 will be sent to post processing to ensure that it will meet the criteria of services, whether it has constraints or not.

Constraints & Post Processing:

Note: indices of arrays in here assume that they start at 0.
Note: % refers to the modulo mathematical operation as it is seen in Python and C.

It is unfortunately quite common for services to reject *seemingly* random 32 character passwords. For this reason, there exists a need to standardize and simplify a method of consistently generating the same constrained password. It does, unfortunately, require one to record the constraint values for that password.

To ensure that most Base64 digests meet the criteria of most services, some automatic rules will always be applied. Let us define $seed$, which is defined by the numeric representation of the first character in the Base64 digest (Python: ord(digest[0])). At the index $seed \% (digestlen - 1)$, the character in the Base64 digest will be replaced by a number 0-9 as determined by $seed \% 10$. The last character in the Base64 digest will be replaced by a value within the following character set, the index of which determined by $seed \% 26$:

QWERTYUIOPASDFGHJKLZXCVBNM

A constraint string will contain various constraints that a password may have. The constraint string will be in the form of:

key1=value1;key2=value2; …

- where *;* and = are delimiters.

If a value is an array, then it will be in the form of:

[value1,value2,value3]

The following keys will exist:

Key Name:	Purpose & Values:
l	Contains the maximum length of the password. This will cause the program to truncate the Base64 digest of the password hash to meet the maximum password criteria. It will take in an integer.
oc	Stands for OR characters. This is used when a website requires at least one

	character from a list of special characters. This will be an array of characters. The first character in the Base64 digest will be replaced by a character in the list of OR characters, the index of which determined by $seed \% orlen$, where $orlen$ is the length of the OR characters array.
ac	Stands for AND character. This is used if a password must have all of the special characters within the provided list. This will be an array of characters. Starting after the first character, each character in the Base64 digest will be replaced by each of the AND characters. To not override the number put in earlier, the program must skip over its index by one, by checking if it is about to replace it.

If this is confusing, then it is important to look at the source code of the Python implementation of the *hashedpass* standard.

Conclusion:

After post processing and the constraints have been applied, the password is now ready for use with that specific service and ID. It is vitally important that one remembers their master password and the format of how they described the service, lest they will be unable to generate the same password again.