Assistance in Daily Password Generation Tasks

Karola Marky
Telecooperation / TU Darmstadt
Hochschulstr. 10, 64289
Darmstadt, Germany
marky@tk.tu-darmstadt.de

Verena Zimmermann
Institute of Psychology / TU
Darmstadt
Alexanderstr. 10, 64283
Darmstadt, Germany
zimmermann@psychologie.tu-darmstadt.de

Peter Mayer
SECUSO / Karlsruhe Institute of
Technology
Kaiserstrasse 89, 76133
Karlsruhe, Germany
peter.mayer@kit.edu

Nina Gerber
SECUSO / Karlsruhe Institute of
Technology
Kaiserstrasse 89, 76133
Karlsruhe, Germany
nina.gerber@kit.edu

Abstract

Passwords and PINs are used to protect all kinds of services in our everyday lives. To serve their purpose of protecting services against adversaries, passwords should have a certain degree of complexity which is often enforced through password policies that encompass various. This results in complicated passwords, which might not only be hard for users to create, but also hard to remember. Furthermore, users might reuse passwords which they feel are secure. We present a scheme for deterministic password generation that solves these problems by assisting the user in generating and remembering passwords. The passwords are generated based on previously stored meta data (e.g., policies) and a master password. Since the password generation is deterministic and only the master password is required to recreate the passwords. As proof of concept we implemented a mobile app and pre-evaluated it. The pre-evaluation indicates that our scheme offers a good usability.

Author Keywords

Password Generation; Password Assistance; Password Management.

Introduction

Passwords and PINs for different (web) services have become part of everyday life and represent the predominant authentication mechanism of the Internet. They are the barrier defending our accounts, e.g., for social networks and online shopping portals, from data thieves. To ensure a strong barrier, users have to choose and remember unique passwords for each service. Services use password policies enforcing certain properties for a password. Password policies might lead to three problems:

(1) users have problems in generating passwords [6, 11, 8],
(2) users struggle remembering their passwords [5, 8], and
(3) users often reuse passwords [5, 4].

A naive solution might be storing passwords in a password protected file. Such a file, however, constitutes a security problem, since the password protection might be circumvented trivially, if no encryption is used and files are often not tailored to address the security needs, that the protection of passwords demands. Password generators can assist users in the password generation process, but most password generators can only be used once, because the generated password is a random one. Therefore, they cannot assist the users in remembering the password. Password managers store passwords, such that users do not have to remember them. Despite this benefit, a widespread adoption has not occurred. This might be related to a missing trustworthiness towards the developer of a password manager or a poor user experience [1].

In this paper we present a *deterministic password generation scheme* that located between a random password generator and a password manager. It contains an algorithm that can deterministically generate random-looking passwords based on stored meta data describing the account (e.g. the username of the account), password properties required by the targeted service (e.g. the password length) and a secret master password. Thus, it assists the user in generating different passwords for each service, while the user only has to remember one master password. The passwords are not stored, but generated on demand whenever the user requires them. The generation algorithm is based on a combination of the hash algorithm BCrypt [15] and the key derivation function PBKDF2 [10]. The salt for these algorithms must not be random, thus it is derived from static meta data: the account name and the username. Other meta data used in the algorithm is: a character set (set of all possible symbols) and the length of the password. This meta data can be exchanged between different user devices, such that the users can generate their passwords on all of their devices.

The user only has to remember the master password, instead of the passwords for all different accounts. So, if an adversary takes possession of the meta data stored for generation, the adversary still has to perform a brute force attack. The only additional knowledge the adversary gains, is the length of the password. To evaluate our scheme, we have implemented an Android app and the results show that our scheme has a good usability.

Password Generation Scheme

In this section we explain the deterministic password generation scheme. We first describe the *meta data* from which the password is derived. Then we explain the *deterministic password generation* and show how the scheme can be used *device independently*.

Meta Data

For the deterministic generation the algorithm requires meta data. This meta data can either be user input or the meta data can be pulled automatically from the targeted web service, e.g., by crawling the password policy. In both cases the meta data is stored in a protected database. We denote the a set of meta data as *account* which consists of:

Account name: This can be the URL of a website, but it could also be the name of a (web) service.

Username: In case the user has more than one account on the same website. The username is optional.

Character set: Represents the character types a password consists of. Users can choose at least one of uppercase characters, lowercase characters, special characters, numbers.

Length: Length of the password.

Date: Date of the last meta data change to indicate the usage duration for a specific meta data set.

Version: Is used to create different passwords, if a password update without changes of the above-mentioned meta data and no change of the master password is intended (e.g., for changing the password after a breach at a web service). If no value is entered, the default value is "1" and is increased automatically during each password update.

Deterministic Password Generation

The password generation algorithm (see Figure 1) is based on the combination of two algorithms: The Public-Key Cryptography Standard *PBKDF2* [10] for key deviation and the hash algorithm *BCrypt* [15]. PBKDF2 can be executed with three different hash algorithms (SHA256, SHA384 and SHA512) and different numbers of iterations.

The master password from the user serves as the secret for the PBKDF2 algorithm. The password version, the account

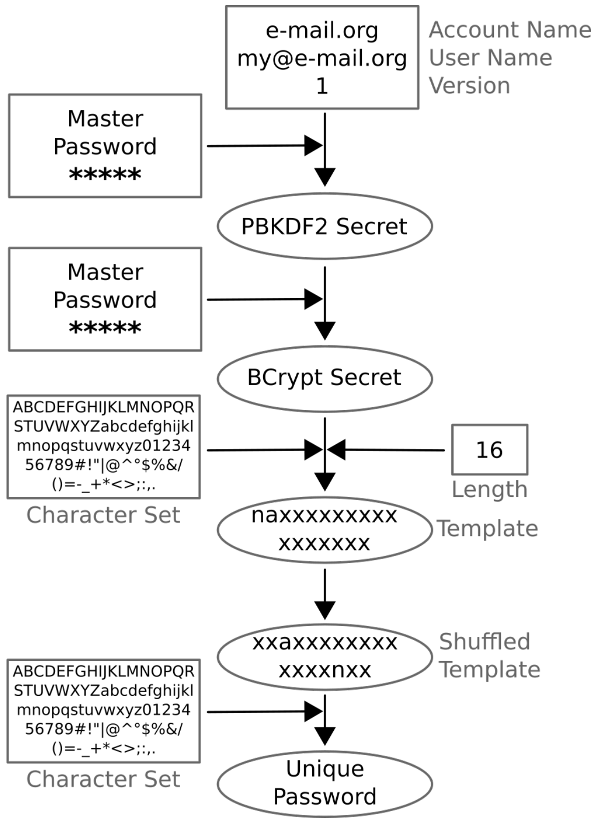
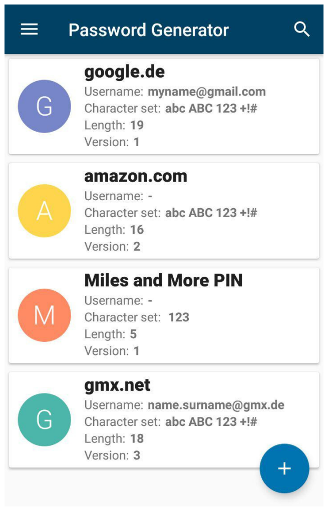


Figure 1: Simplified schematic overview of the generation algorithm. User input is depicted as squares, generated results are depicted as ovals.

name, the username and an optional device-binding string [1] are concatenated to a string and form the salt for PBKDF2. The result of PBKDF2 hashing is encoded into a special version of Base64 which is compatible with BCrypt and not

[1] Users might want to bind the passwords to their individual devices.



Password Generator

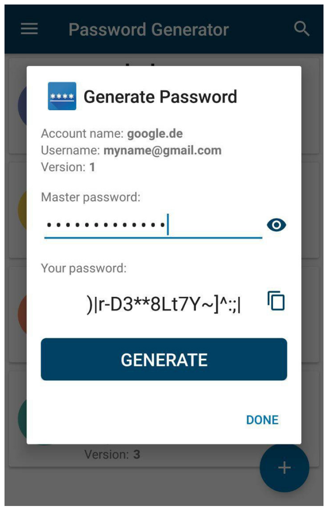
google.de
Username: myname@gmail.com
Character set: abc ABC 123 +!#
Length: 19
Version: 1

amazon.com
Username: -
Character set: abc ABC 123 +!#
Length: 16
Version: 2

Miles and More PIN
Username: -
Character set: 123
Length: 5
Version: 1

gmx.net
Username: name.surname@gmx.de
Character set: abc ABC 123 +!#
Length: 18
Version: 3

Figure 2: Account list.



Generate Password

Account name: google.de
Username: myname@gmail.com
Version: 1

Master password:

Your password:

)|r-D3**8Lt7Y~]^:;|

Figure 3: Generation dialog.

longer than 22 characters, because BCrypt demands this properties. The master password serves as secret for the BCrypt algorithm. The result of the PBKDF2 hashing combined with the string $2a$10$[2] at the beginning forms the salt for BCrypt. As the prefix and the salt are also part of BCrypt's result those are removed from the resulting byte-array[3]. The byte-array is used to deterministically choose characters out of the defined character set.

To ensure that at least one of each chosen character groups is included in the password, password templates are used. The template is a string generated for every password based on the character set. For instance a password with at least one number, one lowercase character and a length of 10 would be "naxxxxxxxx". Whereas "n" denotes a number, "a" denotes a lowercase letter and x denotes any symbol from the character set. To not deteriorate the password room of the resulting password the template string is shuffled based on the reverse of the resulting byte array.

Device Independent Generation

There are several possibilities to collect the required meta data. First, the user could enter the data manually, which is not very practical, but a good back-up option in case an automatic collection is not possible. Second, the user could indicate the targeted (web) service by, for instance, opening a website. Then the meta data could be crawled from the website's password policy if available. The meta data is stored in a protected database that can be synced among different user devices, e.g., by a syncing server. Software on these devices that implements the generation algorithm can generate the passwords on demand.

[2]The string $2a$10$ indicates the salt type and the 10 is the round value for BCrypt.

[3]Byte-arrays are used to exacerbate stealing the generated password from a device's RAM.

Pre-Evaluation

We implemented a mobile app in order to evaluate the password generation algorithm. In this section we present our *implementation*, describe the *evaluation method* and *results*.

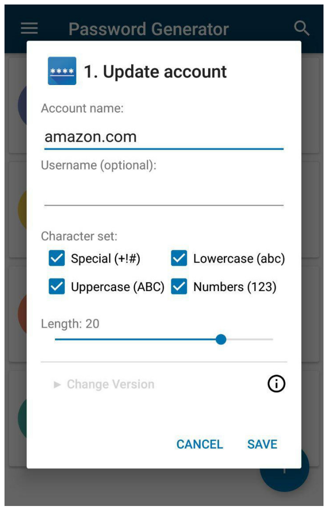
Implementation

We implemented our generation algorithm as an Android app in order to evaluate the password generation. The app is called "Privacy Friendly Password Generator" and its source code and the app are available on Github[4]. The app offers a password length between 4 and 25 characters. The number of PBKDF2 iterations can be set from 1000 up to 10000. We choose the range because it is recommended by the United States National Institute for Standards and Technology to use at least 1000 [17]. The number of iterations can be chosen in the expert section of the app's settings. To test the generation duration based on the given settings a benchmark can be executed in the settings.

To generate a password, a user has to add an account with the meta data explained above. This task has to be performed once, because the account is stored in a database. All accounts are displayed as a searchable list in the app (see Figure 2). To generate a password, the user chooses an account from this list by a click which opens a dialog (see Figure 3). There, the user enters his or her master password and presses a button with the label "generate". The app generates and displays the password which has to be copied by the user either manually or by a copy button next to the displayed password.

In case the user wishes to update the password, he or she chooses the account from the list and starts the update process by a long press (see Figure 4). After the data has

[4]https://github.com/SECUSO/privacy-friendly-passwordgenerator


Password Generator

1. Update account

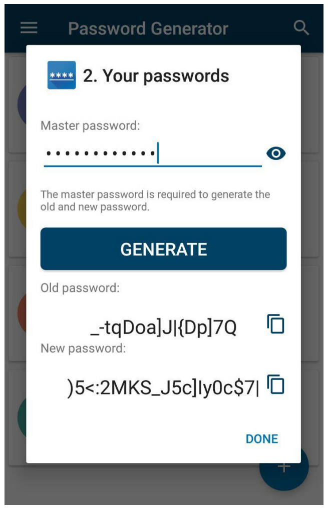
Account name: amazon.com

Username (optional):

Character set: ✓ Special (+!#) ✓ Lowercase (abc) ✓ Uppercase (ABC) ✓ Numbers (123)

Length: 20

Figure 4: Account update.


Password Generator

2. Your passwords

Master password: ••••••••••••

The master password is required to generate the old and new password.

Old password: _-tqDoa]J|{Dp]7Q

New password:)5<:2MKS_J5c]Iy0c$7|

Figure 5: Account update: Old and new password.

been stored in the database, the user has the possibility to generate and display both: old and new password (see Figure 5). This feature has been integrated to simplify the update process as the user sees both passwords at the same time and can copy one after another.

"Privacy Friendly Password Generator" is part of the *Privacy Friendly Apps* group [13] and uses the following mechanisms to enhance the privacy and the security of the password:

- *Blocking of screenshots.* By blocking screen shots users cannot save a screenshot of the password on the device. Furthermore, a malicious app cannot steal the password by taking a screenshot.
- *Usage of a stateless algorithm.* The generated password and the entered master password only "exist" during the execution of the algorithm.
- *Handling of the password in byte or char arrays as often as possible.* Avoidance of handling the password as a string should aggravate reading the password from a device's RAM.
- *No storage of passwords.* Passwords are not stored on the device to prevent leakage if the device gets lost or stolen.
- *No storage of the master password.* The master password is the only secret, it is not stored, such that passwords can not be generated if the device gets lost or stolen.
- *No usage of Android permissions.* "Privacy Friendly Password Generator" does not require permissions for its functionality. The usage of permissions would result in a over-privileged app.

Since our goal was to pre-evaluate the password generation, the app does not offer a possibility for syncing meta data.

Evaluation Method

In order to pre-evaluate the usability of the password generation, a user study with ten participants was conducted. The study was limited to the users of mobile devices. Nine participants were students of various subjects. One participant was a lecturer. Half of the participants were male and half of them were female. The mean age of participants was 22.3 years. The participants were recruited via word-of-mouth and e-mail invitation. Before starting the study, we explained the study to each participant who had to sign a consent form. Throughout the study, participants did not use their own smartphones, but were provided one with "Privacy Friendly Password Generator" pre-installed instead. The installation of "Privacy Friendly Password Generator" was reset after each participant.

The participants were asked to perform the following tasks[5]: (1) generating a password and therefore adding a new account, (2) updating a password and therefore updating an account and (3) deleting an account. Besides these tasks the participants had the opportunity to freely explore the app. The participants' interaction with the app was recorded by a screen capturing app. After exploring the app the participants were asked to fill out the System Usability Scale questionnaire (SUS) [3] and answer two further open-ended questions. Those questions were: (1) Do you have any improvement suggestions for "Privacy Friendly Password Generator"?, and (2) Do you have any additional feedback?

[5]If the app is started for the first time, it starts with a tutorial screen informing the user about these tasks. The tutorial can be re-accessed through the app's menu.

Results
"Privacy Friendly Password Generator" received an average SUS score of 83 ($Min = 77.5$, $Max = 92.5$) which corresponds to an adjective scale of "excellent" according to Bangor *et al.* [2] and indicates a "good" usability.

The first question (Do you have any improvement suggestions for "Privacy Friendly Password Generator"?) has mostly been answered with the need for a better explanation of the purpose of the master password. Therefore, an additional tutorial page explaining the master password in a tutorial which is displayed before the first generation of a password, was added to the app. Furthermore, participants struggled with copying the passwords from the generator to the web service, therefore we added a button that copies the generated password to the clipboard. The participants, furthermore, mentioned the need to generate the passwords on different devices and consider this property very useful. The screen captures were also analyzed but no mentionable conclusions could be made. All participants were able to perform the tasks that we asked from them.

Related Work
There are various password generation schemes available on the market and in the literature. *Web-based* schemes generate passwords on a website. *Mobile* schemes run on a smartphone or other mobile device in form of an app. Finally, *browser-based* schemes are included into a web browser.

Random Password Generator[6] is a web-based generator that creates random passwords. Users can choose between different password types. The length of the password can be chosen indirectly by picking one out of six security levels with *poor* being the lowest and *overkill* being the highest. If the password is lost it cannot be re-covered or re-generated.

xkpasswd[7] is web-based implementation of the *fastwords* concept [9]. The user chooses a quantity of English words in a defined length range and word transformations, e.g., alternating the case. Because the words are separated the user has the option to pick a separator, otherwise it is randomized. The user can choose padding digits and symbols for the beginning and end of the password. After password generation, the generator displays a strength estimation from "poor" to "strong". Although the configuration can be stored, the algorithm cannot re-generate passwords because the words are chosen randomly.

PwdHash [16] is a deterministic browser-based generator, which is also available as web-based generator[8]. PwdHash uses a HMAC-MD5 hash with the website's top-level domain[9] as salt and a master password as secret. The generated password is two characters longer than the master password and fulfills policies like the inclusion of specific character types. The implementation protects against some browser attacks [16], but it has several weaknesses and is susceptible to a brute force attack [12].

Master Password[10] is a mobile generator with implementations for different mobile operating systems. It generates passwords based on parameters and a master password. The user name serves as a seed for a Scrypt hash [14]. This hash, the website's name and a password counter are combined to a template seed by performing an HMAC-SHA256 with the key as secret and the website's name and password counter as salt. Based on the template seed the

[6]http://passwordcreator.org/ [accessed Jan. 30th 2018]

[7]https://xkpasswd.net/s/ [accessed Jan. 30th 2018]

[8]https://pwdhash.com/ [accessed Jan. 30th 2018]

[9]But this results in problems with umbrella domains like *.co.uk*.

[10]http://masterpasswordapp.com/ [accessed Jan. 30th 2018]

user can choose one of 30 templates with different lengths, character sets and characters sequences. But the limited number of templates reduces the number of possible passwords and eases a brute force attack.

The Password Assistance System (PAS) by Horsch [7] assists the user by minimizing the user's interaction with the password. PAS automatically generates passwords matching the password policies of a given website and stores it for the user in way that users can access the password from all devices. Thus, it is a password manager that can generate passwords. PAS has not been evaluated in a user study and therefore, metrics regarding its usability and trust are not available.

Discussion and Limitations

The pre-evaluation of the app implementation of the presented algorithm shows that the processes of adding, updating accounts, deleting accounts and generating passwords offer a good usability. However, only ten people in a specific age group participated in the study making the results not representative. But they serve as an input for improving the scheme before a deeper usability and trust evaluation. The mentioned need for password generation assistance on different user devices in the study indicates that users would welcome the assistance by our proposed scheme. As up to this point in time the generation is limited to mobile devices, we plan to extend the scheme from a standalone app to a device-independent password generation and management scheme and to implement a syncing service. This enables a deep evaluation of the entire scheme with a representative number of participants. We furthermore plan to investigate the automatic collection of meta data. Furthermore, password managers are only scarcely adopted due to trust issues [1], therefore determining the user trust in deterministic password generation based on a master password forms an important task of future work.

Conclusion

The presented scheme offers a possibility to assist the user in generating passwords based on a master password and meta data. Instead of remembering all the passwords for all services, the user only has to remember only one master password. The passwords generated by our scheme are random looking and thus hard to remember. But compared to random generators the password can be re-generated deterministically on demand. The pre-evaluation of the mobile app shows, that the password generation offers a good usability and that users would welcome the generation on different user devices. Therefore, we plan the extend the standalone app to a device-independent password generation and management scheme to further evaluate usability, user experience and trust.

REFERENCES

1. Nora Alkaldi and Karen Renaud. 2016. Why Do People Adopt, or Reject, Smartphone Password Managers?. In *Proceedings of the 1st European Workshop on Usable Security (EuroUSEC)*. Internet Society, Reston, VA, USA, 1–14.

2. Aaron Bangor, Philip Kortum, and James Miller. 2009. Determining What Individual SUS Scores Mean: Adding an Adjective Rating Scale. *Journal of Usability Studies* 4, 3 (2009), 114–123.

3. John Brooke. 1996. SUS - A Quick and Dirty Usability Scale. *Usability Evaluation in Industry* 189, 194 (1996), 4–7.

4. Anupam Das, Joseph Bonneau, Matthew Caesar, Nikita Borisov, and Xiao Feng Wang. 2014. The Tangled Web of Password Reuse.. In *Proceedings of*

Network and Distributed System Security Symposium (NDSS). Internet Society, Reston, VA, USA, 23–26.

5. Dinei Florencio and Cormac Herley. 2007. A Large-Scale Study of Web Password Habits. In *Proceedings of the 16th international Conference on World Wide Web (WWW)*. ACM, New York, NY, USA, 657–666.
6. Cormac Herley. 2009. So Long, and No Thanks for the Externalities: the Rational Rejection of Security Advice by Users. In *Proceedings of the New Security Paradigms Workshop (NSPW)*. ACM, New York, NY, USA, 133–144.
7. Moritz Horsch. 2018. *Generating and Managing Secure Passwords for Online Accounts*. Ph.D. Dissertation. Technische Universität Darmstadt.
8. Philip G. Inglesant and M. Angela Sasse. 2010. The True Cost of Unusable Password Policies: Password Use in the Wild. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)*. ACM, New York, NY, USA, 383–392.
9. Markus Jakobsson and Ruj Akavipat. 2012. Rethinking Passwords to Adapt to Constrained Keyboards. In *Proceedings of the Workshop on Mobile Security Technologies (MoST)*. IEEE, Piscataway, NJ, USA, 1–11.
10. Burt Kaliski. 2000. RFC 2898: PKCS# 5: Password-Based Cryptography Specification Version 2.0. (2000).
11. Saranga Komanduri, Richard Shay, Patrick Gage Kelley, Michelle L Mazurek, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, and Serge Egelman. 2011. Of Passwords and People: Measuring the Effect of Password-Composition Policies. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)*. ACM, New York, NY, USA, 2595–2604.
12. David Llewellyn-Jones and Graham Rymer. 2017. Cracking Pwdhash: A Bruteforce Attack on Client-Side Password Hashing. In *Proceedings of the 11th International Conference on Passwords (Passwords)*. Springer-Verlag, Cham, Switzerland.
13. Karola Marky, Andreas Gutmann, Philipp Rack, and Melanie Volkamer. 2016. Privacy Friendly Apps - Making Developers Aware of Privacy Violations. In *Proceedings of the 1st International Workshop on Innovations in Mobile Privacy and Security (IMPS)*. CEUR Workshop Proceedings, 46–48.
14. Colin Percival. 2009. Stronger Key Derivation via Sequential Memory-Hard Functions. `https://www.bsdcan.org/2009/schedule/attachments/87_scrypt.pdf`. (2009). Self-published, Online; accessed: 12-June-2018].
15. Niels Provos and David Mazieres. 1999. A Future-Adaptable Password Scheme. In *Proceedings of the USENIX Annual Technical Conference (ATC)*. Usenix Association, Berkeley, CA, USA, 81–91.
16. Blake Ross, Collin Jackson, Nick Miyake, Dan Boneh, and John C Mitchell. 2005. Stronger Password Authentication Using Browser Extensions. In *Proceedings of the USENIX Security Symposium*. USENIX Association, Berkeley, CA, USA, 17–32.
17. Meltem Sönmez Turan, Elaine Barker, William Burr, and Lily Chen. 2010. Recommendation for Password-Based Key Derivation. *NIST special publication* 800 (2010), 132.