Generating and Managing Secure Passwords for Online Accounts

Generierung und Verwaltung sicherer Passwörter für Onlinekonten
Zur Erlangung des akademischen Grades Doktor-Ingenieur (Dr.-Ing.)
genehmigte Dissertation von Moritz Horsch, M.Sc. aus Dernbach
Tag der Einreichung: 18.09.2017, Tag der mündlichen Prüfung: 28.11.2017
Darmstadt 2018 — D 17

1. Gutachten: Prof. Dr. Johannes Buchmann
2. Gutachten: Prof. Chris J. Mitchell, Ph.D.

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Fachbereich Informatik
Theoretische Informatik
Kryptographie und Computeralgebra

Generating and Managing Secure Passwords for Online Accounts

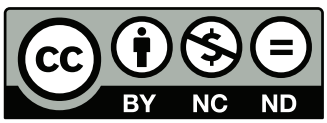
Generierung und Verwaltung sicherer Passwörter für Onlinekonten

Genehmigte Dissertation von Moritz Horsch, M.Sc. aus Dernbach

1. Gutachten: Prof. Dr. Johannes Buchmann
2. Gutachten: Prof. Chris J. Mitchell, Ph.D.

Tag der Einreichung: 18.09.2017
Tag der mündlichen Prüfung: 28.11.2017

Darmstadt 2018 — D 17



Erklärung zur Dissertation

Hiermit versichere ich, die vorliegende Dissertation ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 18.09.2017

_____

(Moritz Horsch)

Wissenschaftlicher Werdegang

Oktober 2012 – heute

Wissenschaftlicher Mitarbeiter in der Arbeitsgruppe von Prof. Dr. Johannes Buchmann, Fachbereich Informatik, Fachgebiet Theoretische Informatik – Kryptographie und Computeralgebra an der Technischen Universität Darmstadt.

März 2010 – Juli 2012

Studium Master of Science Informatik an der Technischen Universität Darmstadt.

Oktober 2004 – März 2010

Studium Bachelor of Science Informatik an der Technischen Universität Darmstadt.

List of Publications

[H1] **Moritz Horsch**, Johannes Braun, and Johannes Buchmann. The Wide Diversity of Password Requirements – And how to cope with it. In *(in submission)*, 2017. Cited on pages 15 and 63.

[H2] **Moritz Horsch**, Johannes Braun, and Johannes Buchmann. Password Assistance. In *Open Identity Summit (OID)*, Lecture Notes in Informatics, pages 35–48. German Informatics Society, 2017. Cited on page 15.

[H3] **Moritz Horsch**, Johannes Braun, Dominique Metz, and Johannes Buchmann. Update-tolerant and Revocable Password Backup. In *Australasian Conference on Information Security and Privacy (ACISP)*, Lecture Notes in Computer Science, pages 390–397. Springer, 2017. Cited on page 123.

[H4] **Moritz Horsch**, Mario Schlipf, Johannes Braun, and Johannes Buchmann. Password Requirements Markup Language. In *Australasian Conference on Information Security and Privacy (ACISP)*, Lecture Notes in Computer Science, pages 426–439. Springer, 2016. Cited on pages 63, 74, and 91.

[H5] **Moritz Horsch**, Mario Schlipf, Stefan Haas, Johannes Braun, and Johannes Buchmann. Password Policy Markup Language. In *Open Identity Summit (OID)*, Lecture Notes in Informatics, pages 135–147. German Informatics Society, 2016. Cited on pages 15 and 137.

[H6] **Moritz Horsch**, Andreas Hülsing, and Johannes Buchmann. PALPAS – PAsswordLess PAssword Synchronization. In *International Conference on Availability, Reliability and Security (ARES)*, pages 30–39. IEEE Computer Society, 2015. Cited on page 97.

[H7] **Moritz Horsch**, David Derler, Christof Rath, Hans-Martin Haase, and Tobias Wich. Open Source für europäische Signaturen. *Datenschutz und Datensicherheit*, 38(4):237–241, 2014. Cited on page 109.

[H8] **Moritz Horsch**, Max Tuengerthal, and Tobias Wich. SAML Privacy-Enhancing Profile. In *Open Identity Summit (OID)*, Lecture Notes in Informatics, pages 11–22. German Informatics Society, 2014. Cited on page 109.

[H9] David Derler, Christof Rath, **Moritz Horsch**, and Tobias Wich. Design und Implementierung eines Localhost Signaturgateways. In *D-A-CH Security*, pages 36–45. syssec Verlag, 2014. Cited on page 109.

[H10] **Moritz Horsch**, Detlef Hühnlein, Anja Lehmann, Johannes Schmölz, and Tobias Wich. Authentisierung mit der Open eCard App. *Datenschutz und Datensicherheit*, 37(8):507–511, 2013. Cited on page 109.

[H11] **Moritz Horsch**, Detlef Hühnlein, Christian Breitenstrom, Thomas Wieland, Alexander Wiesmaier, Benedikt Biallowons, Dirk Petrautzki, Simon Potzernheim, Johannes Schmölz, Alexander Wesner, and Tobias Wich. Die Open eCard App für mehr Transparenz, Vertrauen und Benutzerfreundlichkeit beim elektronischen Identitätsnachweis. In *13. Deutscher IT-Sicherheitskongress*, pages 391–403. SecuMedia Verlag, 2013. Cited on page 109.

[H12] Johannes Braun, **Moritz Horsch**, and Andreas Hülsing. Effiziente Umsetzung des Kettenmodells unter Verwendung vorwärtssicherer Signaturverfahren. In *13. Deutscher IT-Sicherheitskongress*, pages 347–359. SecuMedia Verlag, 2013.

[H13] Tobias Wich, **Moritz Horsch**, Dirk Petrautzki, Johannes Schmölz, Detlef Hühnlein, Thomas Wieland, and Simon Potzernheim. An Extensible Client Platform for eID, Signatures and More. In *Open Identity Summit (OID)*, Lecture Notes in Informatics, pages 55–68. German Informatics Society, 2013. Cited on page 109.

[H14] Detlef Hühnlein, Jörg Schwenk, Tobias Wich, Vladislav Mladenov, Florian Feldmann, Andreas Mayer, Johannes Schmölz, Bud P. Bruegger, and **Moritz Horsch**. Options for integrating eID and SAML. In *Digital Identity Management (DIM)*, pages 85–96. ACM, 2013. Cited on page 109.

[H15] **Moritz Horsch**, Johannes Braun, Alexander Wiesmaier, Joachim Schaaf, and Claas Baumöller. Verteilte Dienstnutzung mit dem neuen Personalausweis. In *D-A-CH Security*, pages 186–197. syssec Verlag, 2012.

[H16] Johannes Braun, **Moritz Horsch**, and Alexander Wiesmaier. iPIN and mTAN for Secure eID Applications. In *International Conference on Information Security Practice and Experience (ISPEC)*, Lecture Notes in Computer Science, pages 259–276. Springer, 2012. Cited on page 113.

[H17] Gerrit Hornung, **Moritz Horsch**, and Detlef Hühnlein. Mobile Authentisierung und Signatur mit dem neuen Personalausweis – Innovative technische und rechtliche Lösungsansätze. *Datenschutz und Datensicherheit*, 36(3):189–194, 2012. Cited on page 136.

[H18] Detlef Hühnlein, Dirk Petrautzki, Johannes Schmölz, Tobias Wich, **Moritz Horsch**, Thomas Wieland, Jan Eichholz, Alexander Wiesmaier, Johannes Braun, Florian Feldmann, Simon Potzernheim, Jörg Schwenk, Christian Kahlo, Andreas Kühne, and Heiko Veit. On the design and implementation of the Open eCard App. In *Sicherheit*, Lecture Notes in Informatics, pages 95–110. German Informatics Society, 2012. Cited on page 109.

[H19] Detlef Hühnlein, Johannes Schmölz, Tobias Wich, Benedikt Biallowons, **Moritz Horsch**, and Tina Hühnlein. Eine Referenzarchitektur für die Authentisierung und elektronische Signatur im Gesundheitswesen. In *GI-Jahrestagung*, pages 1651–1664. German Informatics Society, 2012.

[H20] Detlef Hühnlein, Johannes Schmölz, Tobias Wich, Benedikt Biallowons, **Moritz Horsch**, and Tina Hühnlein. Standards und Schnittstellen für das Identitätsmanagement in der Cloud. In *D-A-CH Security*, pages 208–218. syssec Verlag, 2012. Cited on page 109.

[H21] Detlef Hühnlein, Johannes Schmölz, Tobias Wich, and **Moritz Horsch**. *Daten- und Identitätsschutz in Cloud Computing, E-Government und E-Commerce*, chapter Sicherheitsaspekte beim chipkartenbasierten Identitätsnachweis, pages 153–168. Springer, 2012. Cited on page 134.

[H22] Johannes Braun, **Moritz Horsch**, Alexander Wiesmaier, and Detlef Hühnlein. Mobile Authentisierung und Signatur. In *D-A-CH Security*, pages 32–43. syssec Verlag, 2011. Cited on page 136.

[H23] Alexander Wiesmaier, **Moritz Horsch**, Johannes Braun, Franziskus Kiefer, Detlef Hühnlein, Falko Strenzke, and Johannes A. Buchmann. An efficient mobile PACE implementation. In *Asia Conference on Computer and Communications Security (ASIACCS)*, pages 176–185. ACM, 2011.

Patents:

[H24] Alexander Wiesmaier, Johannes Braun, and **Moritz Horsch**. EP 2639997 – Method and system for secure access of a first computer to a second computer. European Patent Office, 2014. EP Patent 2 639 997.

[H25] Claas Baumöller, Joachim Schaaf, **Moritz Horsch**, Alexander Wiesmaier, and Johannes Braun. EP 2600270 – Identification element-based authentication and identification with decentralised service use. European Patent Office, 2013. Pending Application EP Patent 2 600 270.

Implementation:

[H26] **Moritz Horsch**. Password Assistance Project. `https://passwordassistance.info`. Cited on pages 40, 66, 68, 75, 79, 87, 88, 89, 92, 110, 112, 132, 139, 140, 148, and 159.

Abstract

User accounts at Internet services contain a multitude of personal data such as messages, documents, pictures, and payment information. Passwords are used to protect these data from unauthorized access. User authentication based on passwords has many advantages for both users and service providers. Users can use passwords across many platforms, devices, and applications and do not need to carry an additional device. Service providers can implement password-based user authentication with little effort and operate it with low cost per user.

However, passwords have a key problem: the conflict between security and ease of use. For security reasons, passwords must be attack-resistant, individual for each account, and changed on a regular basis. But, these security requirements make passwords very difficult to use. They require users to create and manage a large portfolio of passwords. This poses three problems: First, the generation of attack-resistant passwords is very difficult. Second, the memorization of many passwords is practically impossible. Third, the regular change of passwords is very time-consuming. These problems are aggravated by the different password requirements, interfaces, and procedures of services. The preservation of passwords for users such as storing passwords on user devices mitigates the memorization problem, but it raises new problems: the confidentiality, availability, recoverability, and accessibility of the preserved passwords. Despite decades of research, the problems of passwords are not solved yet. Consequently, secure passwords are not usable in practice. As a result, users select weak passwords, use them across accounts, and barely change them.

In this thesis, we introduce the Password Assistance System (PAS). It makes secure passwords usable for users. This is achieved by automation and comprehensive support. PAS covers all aspects of passwords. It generates, preserves, and changes passwords for users as well as ensures the confidentiality, availability, recoverability, and accessibility of the preserved passwords. This reduces the efforts and activities of users to deal with passwords to a minimum and thus enables users to practically realize secure passwords for their online accounts for the first time.

PAS is the first solution that is capable of handling the different password implementations of services. This is achieved by a standardized description of password requirements, interfaces, and procedures. Moreover, PAS is solely realized on the user-side and requires no changes on the service-side. Both features ensure the practicability of PAS and make it ready to be used.

PAS solves the password generation problem by creating attack-resistant, individual, and valid passwords for users automatically. Users just need to provide the URL of a service to generate an optimal password for an account. Our uniform description of password requirements provides the information to generate passwords in accordance with the individual password requirements of services. PAS is able to generate the requirements descriptions automatically by extracting the password requirements of services from their websites. So far, this was done for 185,696 services. Moreover, PAS is equipped with an optimal password-composition rule set for the event that services do not explicitly state their password requirements, which is the usual case. By means of the optimal rule set, PAS also generates attack-resistant passwords with the best possible acceptance rate in case of unknown password requirements.

PAS solves the password memorization problem by preserving passwords for users. This releases users from memorizing their passwords and facilitates to use individual passwords for accounts. PAS makes users' password portfolios available on all their devices as well as automatically synchronizes changes. PAS achieves this without storing passwords at servers so that an attacker cannot steal them from servers. Moreover, PAS provides a backup solution to recover the preserved passwords in case of loss. Users need to create backups only once and do not have to update them even when their password portfolios change. Consequently, users can keep backups completely offline at secure, different, and physically isolated locations. This minimizes the risk of compromise and loss as well as enables an emergency access to the passwords for trusted persons. Moreover, PAS has a built-in revocation mechanism. It allows users to completely invalidate devices and backups in case they lose control over them. This guarantees that no passwords can be stolen from lost user devices and backups once revoked. Users always have full control of their passwords.

PAS solves the password change problem by changing passwords automatically for users. Users neither need to create new passwords nor manually log in to their accounts. Our uniform description of password interfaces and procedures provides the information to change passwords at arbitrary services. Moreover, PAS is the first solution that provides autonomous password changes. It changes passwords on a regular basis with respect to the security level of passwords as well as immediately after PAS detects a compromise of users' passwords.

The practicability of PAS is demonstrated by an implementation. The individual components of PAS can be used independently, integrated into other applications, and combined to a single user application, called a password assistant.

In summary, this thesis presents a solution that makes secure passwords usable. This is done by automation and comprehensive support in the generation and management of passwords.

Zusammenfassung

Nutzerkonten bei Internetdiensten enthalten eine Vielzahl von personenbezogenen Daten wie Nachrichten, Dokumente, Bilder und Bankdaten. Um diese sensiblen Daten vor unberechtigtem Zugriff zu schützen werden Passwörter verwendet. Eine Passwort-basierte Authentisierung hat für Nutzer und Dienstanbieter viele Vorteile. Nutzer können Passwörter bei verschiedenen Plattformen, Geräten und Anwendungen auf die gleiche Art und Weise nutzen. Dienstanbieter können eine Passwort-basierte Authentisierung leicht implementieren sowie mit wenig Aufwand und geringen Kosten betreiben.

Passwörter haben jedoch ein zentrales Problem: Der Konflikt zwischen Sicherheit und Nutzbarkeit. Aus Sicherheitsgründen müssen Passwörter diversen Angriffen wie Wörterbücher, Brute-Force, usw. widerstehen, sich zwischen Konten unterscheiden und regelmäßig geändert werden. Jedoch führen diese drei Sicherheitsanforderungen dazu, dass Passwörter sehr schwer nutzbar sind. Um die Sicherheitsanforderungen zu erfüllen, müssen Nutzer ein großes Portfolio an Passwörtern erstellen und verwalten. Dies führt zu drei Problem: Erstens ist es sehr schwierig sichere Passwörter zu erstellen. Zweitens ist es praktisch unmöglich sich viele Passwörter zu merken. Drittens ist es sehr zeitaufwendig Passwörter regelmäßig zu ändern. Die unterschiedlichen Passwort-Anforderungen sowie Schnittstellen und Prozeduren zur Nutzung von Passwörtern der Dienstanbieter tragen zu einer Verschärfung dieser Probleme bei. Durch das Speichern von Passwörtern auf Nutzergeräten müssen sich Nutzer zumindest ihre Passwörter nicht mehr merken. Jedoch schafft dieser Lösungsansatz neue Probleme: Die Vertraulichkeit, Verfügbarkeit, Wiederherstellbarkeit und Zugreifbarkeit im Notfall der gespeicherten Passwörter muss gewährleistet werden. Trotz jahrelanger Forschung und unzähligen Lösungsvorschlägen sind die Probleme von Passwörtern bis heute nicht gelöst. Daher können Nutzer in der Praxis sichere Passwörter nicht verwenden. Nutzer wählen daher im Alltag einfache Passwörter, verwenden die gleichen Passwörter für mehrere Konten und ändern Passwörter nur sehr selten.

In dieser Arbeit stellen wir das Password Assistance System (PAS) vor. Es ermöglicht Nutzern sichere Passwörter für ihre Konten bei Dienstanbietern im Internet zu verwenden. Dies geschieht durch eine Automatisierung und einer umfassenden Unterstützung bei der Generierung und Verwaltung von Passwörtern. PAS berücksichtigt und umfasst alle Aspekte und Bereiche von Passwörtern. Es generiert, speichert und ändert regelmäßig die Passwörter von Nutzern. Es stellt außerdem die Vertraulichkeit, Verfügbarkeit, Wiederherstellbarkeit und Zugreifbarkeit im

Notfall der gespeicherten Passwörter sicher. PAS reduziert damit den Aufwand und die Aufgaben von Nutzern hinsichtlich ihrer Passwörter auf ein Minimum. Nutzer sind mit PAS das erste Mal in der Lage sichere Passwörter in der Praxis einzusetzen.

PAS ist die erste Lösung, die mit den verschiedenen Implementierungen von Passwörtern bei Dienstanbietern umgehen kann. Dies wird durch eine einheitliche Beschreibung der Passwort-Anforderungen, -Schnittstellen und -Prozeduren erreicht. Außerdem ist PAS eine reine Nutzerseitige Lösung und erfordert keine Änderungen auf Seiten der Dienstanbieter. Mit diesen zwei Eigenschaften ist der Einsatz von PAS in der Praxis gewährleistet.

PAS löst das Problem der Erstellung sicherer Passwörter indem es automatisch Passwörter für Nutzer generiert. Diese Passwörter sind sicher gegen Angriffe, unterschiedlich für jedes Nutzerkonto und entsprechen den jeweiligen Passwort-Anforderungen der Dienstanbieter. Dazu müssen Nutzer nur die URL eines Dienstanbieters angeben. PAS generiert dann ein optimales Passwort automatisch. Eine einheitliche Beschreibung der verschiedenen Passwort-Anforderungen bildet die Grundlage um Passwörter im Einklang mit den jeweiligen Passwort-Anforderungen der Dienstanbieter zu generieren. PAS ist in der Lage diese Beschreibungen automatisch zu erstellen, indem es die Passwort-Anforderungen von den Webseiten der Dienstanbieter extrahiert. Die einheitlichen Beschreibungen der Passwort-Anforderungen wurden in dieser Arbeit bereits für 185.696 Dienstanbieter erstellt. Des Weiteren nutzt PAS optimierte Regeln für die Erstellung von Passwörtern für den Fall, dass die Passwort-Anforderungen eines Dienstanbieters nicht zur Verfügung stehen. Dies gilt für die Mehrheit der Dienstanbieter im Internet, wie wir in einer umfassenden Studie in dieser Arbeit zeigen. Mit diesen Regeln generiert PAS Passwörter, die sicher gegen Angriffe sind und von der Mehrzahl der Dienstanbieter akzeptiert werden.

PAS löst das Problem das sich Nutzer nur wenige Passwörter merken können indem es die Passwörter speichert. Damit ist es möglich unterschiedliche Passwörter für Nutzerkonten zu verwenden. PAS sorgt darüber hinaus dafür, dass die Passwörter auf allen Geräten der Nutzer zur Verfügung stehen einschließlich neuer und geänderter Passwörter. Dies geschieht jedoch ohne die Passwörter auf Servern im Internet zu speichern. Angreifer sind daher nicht in der Lage, die Passwörter von Servern zu stehlen. PAS stellt auch eine Backup Lösung bereit, so dass Nutzer ihre Passwörter nicht verlieren können. Ein Backup muss dabei nur einmal erstellt werden. Eine Aktualisierung ist nicht erforderlich, auch wenn sich Passwörter ändern. Diese Eigenschaft erlaubt es, Backups an sicheren und verschiedenen Orten aufzubewahren. Dies reduziert das Risiko, dass Backups selbst verloren oder zerstört werden, sowie Angreifer Zugriff auf diese erhalten. Darüber hinaus können damit Backups bei vertrauenswürdigen Personen hinterlegt werden, so dass diese im Notfall Zugriff auf die Passwörter haben. PAS bietet darüber

hinaus ein Revokationsmechanismus für Nutzergeräte und Backups an. Mit diesem sind Nutzer in der Lage darauf enthaltenen PAS-spezifischen Daten nutzlos zu machen. Angreifer können dann keine Passwörter von gestohlenen Geräten oder Backups entwenden. Nutzer haben damit zu jeder Zeit die volle Kontrolle über ihre Passwörter.

PAS löst ebenfalls das Problem der regelmäßigen Änderungen von Passwörtern. Dies erledigt PAS vollautomatisch. Nutzer müssen weder neue Passwörter erstellen, noch sich bei ihren Konten anmelden. Die einheitliche Beschreibung der Password-Schnittstellen und -Prozeduren der Dienstanbieter ermöglicht eine Automatisierung von Passwort-Änderungen bei beliebigen Diensten. Außerdem ist PAS die erste Lösung die Passwörter autonom ändert. Dies geschieht in regelmäßigen Abständen mit Hinblick auf das Sicherheitsniveau der Passwörter und sobald PAS ein Missbrauch von Passwörtern feststellt.

Die Praxistauglichkeit von PAS ist durch eine Implementierung demonstriert. Die Komponenten von PAS können dabei einzeln genutzt sowie in andere Anwendungen integriert werden. Verbunden zu einer Nutzerapplikation bilden sie den Password Assistant.

Zusammenfassend stellt diese Arbeit ein System bereit, das die Nutzung von sicheren Passwörtern bei Nutzerkonten ermöglicht. Dies wird durch eine Automatisierung und einer umfangreichen Unterstützung bei der Generierung und Verwaltung von Passwörtern erreicht.

Contents

List of Figures

List of Tables

1 Introduction

In the last decade, services on the Internet have changed from only providing static content to services which enable users to communicate, interact, and collaborate. User accounts for email providers, shopping portals, social networking sites, blogs, wikis, and video sharing platforms have become an integral part of our life. These accounts contain a multitude of personal data such as messages, documents, pictures, and payment information. To protect these sensitive data from unauthorized access, online accounts are usually protected by passwords.

Service providers can implement password-based user authentication with little effort and operate it with low cost per user [114]. Users are familiar with the concept of passwords and can use them across many platforms, devices, and applications [36]. They do not need to buy special hardware or need to carry additional devices. Moreover, passwords kept in users' minds appear to users to be always available, not threatened by loss like keys, and to be stored in the safest place on earth to which no one else has access and could steal them. Moreover, in urgent or emergency situations, users can give passwords to someone else to access accounts on their behalf just by telling them. These perceived benefits of passwords are very important for users [128, 131, 208, 244]. Regarding these advantages for users and services, it is very likely that passwords remain the dominant authentication scheme on the Internet in the future [115].

However, passwords have a key problem: the conflict between security and ease of use. The various attacks against passwords, such as brute-force [135, 235], dictionary [34, 234], and social engineering [53], make *secure* passwords indispensable. This means users must use attack-resistant and individual passwords for their accounts and change them on a regular basis. But, these three security requirements make passwords unusable. They require users to generate and manage a large portfolio of passwords. This poses the following *fundamental problems*:

- *Password generation*: Generating passwords that resist brute-force attacks is difficult. User-chosen passwords often include personal-related and service-related information and have patterns [141, 202, 224, 247]. This enables very efficient guessing attacks [162, 169, 235].
- *Password memorization*: Memorizing a multitude of passwords is practically impossible. Therefore, users create passwords that are easy to remember and reuse them across accounts [43, 66, 95, 129, 233, 241]. This bears the risk that an attacker can easily guess the passwords and get access to multiple accounts just by obtaining a single password.

- *Password change*: Regularly changing the passwords of all online accounts is very time-consuming. Consequently, users barely change their passwords [110, 212], even after security breaches at services or exceptional events like the Heartbleed bug [62, 121, 179].

The generation and change problem are aggravated by the different password implementations of services [37]. They raise further *implementation-related problems*:

- *Password requirements*: Generating passwords in accordance with the different, incomprehensible, incomplete, and often unknown services' password requirements is very difficult.
- *Password interfaces and procedures*: Changing passwords through the different password interfaces and procedures implemented by services is very challenging and cumbersome.

The memorization problem of passwords can be mitigated by preserving passwords for users such as storing them on user devices using a password manager. However, preserving passwords leads to additional *preservation-related problems*:

- *Password confidentiality*: Protecting passwords from unauthorized access is difficult. It is often done by another password, which raises the aforementioned fundamental problems.
- *Password availability*: Making passwords available on all devices is challenging. Manually copying passwords is inconvenient and an online synchronization raises security issues. But, users cannot access their accounts everywhere and anytime without their passwords.
- *Password recoverability*: Creating password backups, properly protecting them, keeping them up-to-date, and storing them at secure locations is practical impossible. But, users lose their entire digital life when they cannot restore lost passwords.
- *Password accessibility*: Making passwords available to trusted persons in urgent or even emergency situations is very challenging. Placing a copy of the passwords at trusted persons raises the same problems such as password backups.

The generation and memorization problem of passwords were already mentioned in 1979 [167]. Even after more than 35 years of research and numerous proposals to replace passwords [16, 25, 36], we are still using passwords and facing the same problems [113, 114, 211]. Even IT professionals cannot manage passwords properly [128, 151, 209].

Preserving passwords is the most suitable approach to realize at least a large portfolio of attack-resistant and individual passwords, but the confidentiality, availability, recoverability, and accessibility of the preserved passwords must be ensured. Moreover, the generation, change, and implementation-related problems remain.

As long as the fundamental, implementation-related, and preservation-related problems of passwords are not solved as a whole, user remain unable to use secure passwords in practice. Therefore, the research goal of this thesis is:

> *Make secure passwords usable for users.*

In this thesis, we introduce the *Password Assistance System* (PAS). It enables users to use secure passwords for their online accounts. This is achieved by automation and comprehensive support. PAS solves the password problems by performing three tasks for users:

1. Generating attack-resistant and individual passwords in accordance with the respective password requirements of services automatically.
2. Preserving passwords as well as ensuring their confidentiality, availability, recoverability, and accessibility.
3. Changing passwords in accordance with the different password interfaces and procedures of services automatically.

PAS reduces the efforts and activities of users to deal with passwords to a minimum. Moreover, it covers all aspects of passwords and supports users from the generation of passwords to their recovery in case of loss. By providing a ubiquitous solution that solves all password problems and addresses all conditions, concerns, and needs of users, PAS enables users to practically use secure passwords for their online accounts for the first time.

PAS is the first solution that is capable of handling the different password implementations of services. The diversity is solved by standardization. A uniform description of password requirements, interfaces, and procedures enables PAS to cope with the implementation-related problems of passwords. Moreover, PAS is solely realized on the user-side and completely independent from services. It does not require any changes on the service-side like new hardware, interfaces, or communication protocols. Handling the different implementations and requiring no changes by services ensure the applicability of PAS and make it ready to be used.

The individual components of the PAS can be used independently as well as integrated into other applications such as password generators and managers. In this way, users that prefer to continue to use their current applications also benefit from the features of PAS. Combined to a full-fledged user application, PAS facilitates the next generation of password management application, the *password assistant*.

Contribution and outline

The main contribution of this thesis is the Password Assistance System that makes secure passwords usable for users. PAS achieves this by generating and managing passwords for users automatically and providing comprehensive support. In the following, we summarize the structure of this thesis and the four parts of PAS.

Background (Chapter 2)

We start by presenting the relevant background for this thesis. This includes an explanation of the cryptographic primitives that we use in PAS and a description of the foundations of password-based authentication. We describe the usage of passwords for online accounts, the storage of passwords at services, and the various attacks against passwords.

Passwords require ubiquitous assistance (Chapter 3)

In this chapter, we detail the problem of users to use secure passwords for their accounts. We identify security requirements for passwords, conditions stated by services that affect the usage of passwords in practice, and usage requirements to ensure a practical and usable solution as well as address conditions, concerns, and needs of users. Based on these requirements and conditions, we define three tasks for users that enable them to realize secure passwords for their accounts. In brief, these tasks are generating, preserving, and changing passwords.

To realize these three password tasks in practice, we analyze the service conditions in detail. In the largest ever conducted survey on password requirements, we demonstrate that the requirements of services are quite different, incomprehensible, incompletely stated, and often not mentioned at all. In a second survey, we show that also the password interfaces and procedures of services are quite different.

Moreover, we examine existing user studies and proposals with respect to the realization of the three password tasks. It turns out that users cannot perform the tasks manually. None of the proposals covers all tasks and in many cases the service conditions prevent a practical use. Furthermore, the proposals do not address all conditions, concerns, and needs of users as well as often raise new problems.

With regard to this result, we outline PAS. It enables users to realize the three password tasks and thus secure passwords by automation and comprehensive support.

Automatic generation of attack-resistant and valid passwords (Chapter 4)

In this chapter, we describe the first part of PAS which realizes the first password task of generating passwords. We present the first solution that automatically generates attack-resistant and valid passwords for users. In contrast to existing approaches, users neither have to find out the password requirements of services nor to adjust generated passwords until they get accepted by services. Users just need to provide the URL of a service in order to generate an optimal password for an account.

We introduce a standardized description of password requirements, the first practical solution to handle the different requirements of services and the basis for the automatic generation of valid passwords. We present a tool that automatically creates the descriptions by extracting the password requirements from the services' websites. Moreover, we describe solutions to distribute the descriptions. We develop an optimal password-composition rules for the event that services do not explicitly state their password requirements, which is the usual case.

To evaluate the feasibility of our solution, we created standardized requirements descriptions of 185,696 services, implemented a password assistant that makes use of them, and also integrated them into an existing password generator.

Passwordless and seamless password synchronization (Chapter 5)

In this chapter, we present the second part PAS. It provides the first component for the second password task of preserving passwords. It preserves passwords for users and ensures their confidentiality and availability. We present the first usable and secure password synchronization scheme. It makes the password portfolios of users available on all their devices as well as seamlessly synchronizes changes. In contrast to common approaches, our password synchronization scheme does not store any passwords, neither at devices nor at servers on the Internet. Therefore, they cannot be stolen by compromising servers. Our solution also has a build-in revocation mechanism to invalidate the data stored on devices in an information-theoretical secure way. This prevents the theft of passwords from stolen devices.

Update-tolerant and revocable password backup with emergency access (Chapter 6)

We describe the third part of PAS in this chapter. It provides the second component for the second password task. We complement our password synchronization scheme from Chapter 5 with a secure and usable backup solution. It provides the recoverability and accessibility of the preserved passwords and ensures that users never lose their passwords. In addition, it allows users to make their passwords available to trusted persons in emergency situations.

Our backup system is the first solution that creates backups that do not need to be updated even when users' password portfolios change. Users need to create backups only once and can keep them completely offline at secure, different, and physically isolated locations. This minimizes the risk of compromise and loss. Further, it allows users to place backups at trusted persons to achieve an emergency access. Like our password synchronization scheme, our backup solution has a built-in revocation mechanism. It allows users to invalidate backups if they lose control over them. The revocation mechanism works without having access to backups and guarantees that no passwords can be leaked from them once revoked. Altogether, PAS provides the first secure and usable solution for the preservation-related problems of passwords. It ensures their confidentiality, availability, recoverability, and accessibility.

Automatic and autonomous password change (Chapter 7)

In this chapter, we describe the fourth and last part of PAS. It realizes the third password task of changing passwords. We present the first solution that completely automate this task so that users do not need to take care of this anymore. Users neither need to create passwords nor to log in to their accounts.

We introduce a standardized description for password interfaces and procedures, the first practical solution to handle the different password implementations of services and the foundation for automatic password changes at arbitrary services. We develop a tool for users to easily create the descriptions for their services. Moreover, we provide a service to distribute descriptions so that they must be created only once and then can be shared with all Internet users. In this way, our solution can support a large number of services with minor efforts and in a short time.

We realize an autonomous password change by two strategies. Password changes are performed on a regular basis with respect to the security level of passwords as well as immediately after PAS detects a compromise of users' passwords. In this way, passwords are changed in an intelligent manner and not simply after a fixed time interval like 90 days.

To evaluate the feasibility of our solution, we created standardized password interfaces and procedures descriptions of popular services and implemented a password assistant that makes use of them and automatically changes passwords.

Conclusion (Chapter 8)

Finally, in this chapter we conclude this thesis and discuss future work.

2 Background

In this chapter, we provide the necessary background for this thesis. First, we explain the cryptographic primitives that we use in our solution in Section 2.1. Second, we describe the foundations of password-based authentication in Section 2.2. We explain authentication in general as well as its different factors and types. Then, we describe the usage of passwords for user authentication at Internet services and the storage of passwords at services. Finally, we provide a detailed summary of the various guessing attacks against passwords.

2.1 Cryptographic primitives

In this section, we describe the cryptographic primitives that we use in PAS. We limit our explanations to the scope of this thesis.

2.1.1 Pseudo-random generator

A random generator produces a sequence of unpredictable bits. The randomness is collected from physical sources such as thermal noise, spinning hard drive, and input devices [76]. The number of randomness that can be gathered from such sources in a certain time interval is limited. Therefore, in practice often a pseudo-random generator (PRG) is used. It is seeded with a short sequence of truly random bits produced by a random generator. This short input seed is then expanded to a long sequence of pseudo-random bits [45]. A PRG is cryptographically secure if it is practically impossible for an attacker to distinguish between the output of a PRG and a sequence of truly random bits. This property is called pseudo-randomness. Such PRGs are sufficient for the usage in security-critical applications. A list of secure PRGs can be found in [19]. PAS uses a PRG to generate random passwords (cf. Chapter 4).

Rejection sampling

Rejection sampling is a technique to generate random values under certain constraints [27, Chapter 11.4]. PAS uses rejection sampling to generate random passwords under given requirements (cf. Chapter 4). A typical example is generating passwords that contain at least a number.

By means of rejection sampling, this works as follows: A random password is generated by using a PRG. Then, it is checked whether the password fulfills the requirements, e.g. containing at least one number. If not, the password is discarded and a new password is generated. This process is repeated until a password is found that fulfills all requirements.

While this process in theory might never terminate, in practice it ends after a few iterations. We consider the generation of a password with a length of 10 characters and consisting of letters and numbers and at least one number. The probability that after τ iterations such a password is not generated is $((1 - \frac{10}{62})^{10})^{\tau} = 0.17^{\tau} < 2^{-2\tau}$. This probability vanishes exponentially fast in the number of iterations. In practice this means for $\tau = 4$ the probability is less than 1%.

2.1.2 Hash function

A hash function maps data of arbitrary length to data of fixed length [45]. The input is usually called the message and the output the hash. For our solution we make use of cryptographic hash functions which have three properties: one-wayness, second-preimage resistance, and collision resistance [120, 187]. One-wayness means that a hash function is not invertible, i.e. the message for a given hash cannot be computed. Second-preimage describes the property that for a given message it is impossible to find another message so that both have the same hash. Collision resistance means that it is impossible to find two different messages with the same hash.

Hash functions are used to securely store passwords. Instead of storing passwords in plain, services just store the hashes of passwords. During a login, services hash the submitted password and compare the result to the stored hash. In Section 2.2.1 we describe the storage of passwords at services more detailed. The one-wayness ensures that an attacker obtaining a hash is not able to compute the corresponding password. Second-preimage resistance is irrelevant with respect to password hashing, because the attacker already knows the password. Finding another password with the same hash is pointless. The same applies for the collision resistance. Finding two passwords with the same hash does not provide any advantage for an attacker.

2.1.3 Encryption

Encryption is used to keep data secret. It can be used to protect stored data as well as to protect data which is transferred between entities over a public network such as the Internet. Encrypting and decrypting data is done with a corresponding encryption and decryption key. PAS uses symmetric encryption schemes such as AES [75] and a one-time pad (OTP) [22, 163] in which the same key is used for encryption and decryption. Encryption is used by PAS to protect the communication between entities as well as stored data.

2.1.4 Digital signature

A digital signature scheme is used to sign electronic data. A signature is created by using a secret key and can be verified by using the corresponding public key. Typically, signatures are used to sign emails or documents like contracts. But, they can also be used for authentication. PAS uses signatures to authenticate clients and servers. A client can authenticate itself to a server in the following way: The server generates a random nonce and sends it to the client. The client signs the nonce with its secret key and sends the signature back to the server. The server in turn verifies the signature using the public key of the client. If the signature is correct, the client is authenticated to the server. Note that this assumes that the server knowns the public key of the client. This can be achieved using certificates.

2.1.5 Certificate

A certificate binds a public key to a subject identifier [63]. In practice, subjects are URLs for websites or email addresses. Among other information, a certificate contains the identifier of the entity and its public key. A certificate is signed by a trusted Certificate Authority (CA) which attests that the public key belongs to the subject. The signature provides a provable binding which can be verified by others.

Besides the issuance of certificates, a CA operates a directory service for retrieving certificates and revocation information [41]. Servers in PAS are equipped with certificates issued by commercial CAs. For the issuance and management of client certificates PAS operates its own CA.

2.1.6 Transport Layer Security

Transport Layer Security (TLS) is a protocol that establishes an encrypted and authenticated communication channel between a client and a server [71]. The authentication is based on digital signatures and certificates for key distribution. TLS is the de facto standard for secure communication on the Internet. A prominent example is web browsing in which TLS secures the communication between a web browser and a web server. PAS uses TLS to protect the communication between clients and servers and for mutual authentication.

2.2 Password-based authentication

Authentication is the process of identifying and verifying the identity of an entity. We focus on the authentication of users. User authentication can be realized with various factors which can be divided into five categories [28, 36, 40]: something the user knows (e.g. a password), something the user has (e.g. a smart card), something the user is (e.g. biometrics like a fingerprint), somewhere the user is (e.g. at home), or somebody the user knows (e.g. a friend).

The different authentication means can be used on their own (single-factor authentication) and can be combined (two-factor or even multi-factor authentication) [102]. For user accounts at services on the Internet, today, mainly a single-factor authentication using a knowledge-based factor is used: passwords.

During the creation of an online account, the user and the service agree on a username and a password. The username identifies each user of the service and consequently must be unique. The password is used to verify that the user is who he claims to be and therefore must be kept secret. To log in to his account, the user needs to provide both, his username and his password, to the service. This means that the service must store these data which we describe in the next section. Figure 2.1 illustrates the authentication procedure in detail.

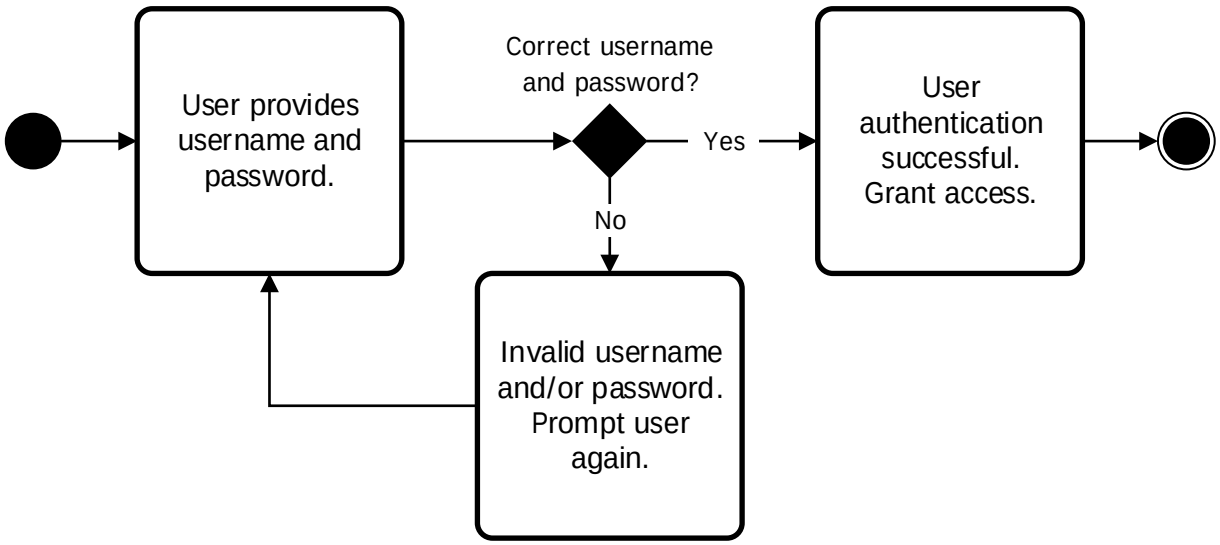


Figure 2.1: Activity diagram of password-based authentication.

2.2.1 Password storage

In this section, we explain the storage of passwords at services. Regarding the preservation of passwords on the user-side, we describe the various approaches in Section 3.6.2.

To properly protect passwords from leakage, best practice is to store them in a salted and hashed way [84, 107, 132]. A salt is a user-specific random value which is generated the service during account creation [167]. It is concatenated with the user password before computing the hash. The salt and the hashed password are then stored in a password database. The salt protects from attacks based on precomputed (reverse) lookup and rainbow tables [112, 117, 173, 239].

When a user attempts to log in to his account, the service looks up the salt using the submitted username. Next, it concatenates the salt with the submitted password and computes the hash. Finally, the service compares the computed hash with the stored hash value in its database. If both are equal the login is successful. Otherwise, the user provided a wrong password.

Storing passwords in a salted and hashed manner protects passwords and allows services to efficiently verify passwords during login. To prevent an attacker from computing the passwords from stolen hashes, a cryptographically secure hash function must be used (cf. [65]). We can expect that services currently use SHA-1 [122, 165].

2.2.2 Password guessing attacks

We describe the various kinds of guessing attacks against passwords in this section. A thorough understanding of these attacks is vital to develop effective countermeasures. It appears that any predicable patterns in passwords can be exploited to improve guessing attacks.

We start in Section 2.2.2.1 with the conceptually simplest guessing attack: brute-force. In Section 2.2.2.2 we describe a more sophisticated attack using dictionaries and in Section 2.2.2.3 we present guessing attacks based on probabilistic models. All these attacks can be performed online and offline. In an online attack an attacker tries to log in to an account using guessed passwords. In an offline attack an attacker has a hashed password and tries to find the same hash. We look at the online/offline attack scenario in detail in Section 3.1 and 3.2.1. In the following sections, we focus on the more general concepts of the guessing attacks.

Besides these guessing attacks, there are a number of further attacks against passwords from the domain of social engineering such as phishing [6], insider [223], social interactions [108], and malware [185]. Because these attacks cannot be prevented by technical means and are not specific to passwords, these are out of scope and we refer to [36, 185] for further details.

2.2.2.1 Brute-force attack

The concept of a brute-force attack is to simply try out all possible passwords. For instance, starting with the password candidate *aaaa*, an attacker tries *aaab*, *aaac*, *aaad*, and so on. This approach is well-suited for short passwords. However, with an increasing number of possible passwords, a brute-force attack quickly becomes very time-consuming and even impractical.

To optimize a brute-force attack, the password requirements of services can be taken into account. This is called a *mask attack*. Requirements like occurrences of characters or position restrictions can reduce the number of possible passwords and speeds up a brute-force attack.

In general, a brute-force attack is very inefficient regarding real-world password guessing, because it does not consider how users choose passwords in practice. As we describe in Section 3.6.1, user-chosen passwords usually have certain patterns and include user-related and service-related information. This can be exploited by more sophisticated strategies which we describe in the following sections.

2.2.2.2 Dictionary attack

In a dictionary attack an attacker selects password candidates from a dictionary [167]. Besides natural-language dictionaries [178], stolen password sets [135], common quotes [141], and wordlists created by *mangling rules* are used. Typical mangling rules are adding a number or special character at the end of passwords as well as transforming letters to uppercase or to numbers. For example, *password* can be transformed into *Password*, *password1*, and *pa33word*. Once invented to help users creating stronger passwords (cf. Section 3.6.1), nowadays many modern password cracking tools such as Hashcat [111], John the Ripper [177], and PasswordsPro [124] consider mangling rules to improve password guessing.

A dictionary attack is more efficient than a brute-force attack because it considers how users choose passwords in practice (cf. Section 3.6.1). The popular *Openwall wordlists collection* [178] contains approximately 40 million entries. This narrows the search space in comparison to a brute-force attack. A dictionary attack might not find all passwords, if they are not part of the dictionary or constructed by mangling rules. This particularly applies for random passwords. Another disadvantage is that password candidates are selected from a dictionary in a sequential order. This ignores the fact that some passwords and character sequences are more frequently used by users than others. Attacks based on probabilistic models, which we describe in the next section, take also advantage of this fact. They try more likely password candidates first.

2.2.2.3 Probabilistic models

A probabilistic password model assigns a probability value to a character sequence. It takes as input large password sets and outputs password candidates for a guessing attack ordered with descending probability. The idea behind this approach is that an attack is more efficient when passwords that are used more frequently are tested before passwords used less frequently.

There exist two types of probabilistic models: template-based and string-based [153]. A template-based model aims at patterns in passwords to determine the probability of password candidates. For instance, user-chosen passwords often start with letters and end with numbers and special characters (cf. Section 3.6.1). A string-based model focuses on the fact that certain character sequences are more likely than others, due to the layout of a typical keyboard or the language. For instance, *th* is the most frequent character sequence in English [246] and *1qazxcv* a keyboard pattern. In the following, we explain both models in detail.

Template-based model using probabilistic context-free grammar

A template-based password model can be realized by a *probabilistic context-free grammar* [235]. The grammar is used to define mangling rules to create additional password candidates for a dictionary attack. The production rules of the grammar are derived from a training set of plaintext passwords, which works as follows: First, the structure of the passwords is determined. For instance, *password12?* is represented by the structure $L_8N_2SP_1$, where L denotes letters, N numbers, and SP special characters. The indices denote the length of the respective character sequence. Moreover, the probabilities of occurrences are assigned to all observed structures. Second, the probabilities of occurrences of numbers and special characters of the training set is determined. This information is used to select suitable candidates for the variables N and SP. For instance, the two-digit number *12* and the special character *!* were the most frequent ones in the training data used in [235]. Based on this information, the aforementioned exemplary structure is refined to $L_8N_2SP_1 \rightarrow L_8$12!. Password candidates are finally generated by filling out the variable L by words form a dictionary with the same length. For instance, $L_8$12! is used to create password candidates like *computer12!*, *baseball12!*, and *superman12!*.

Probabilistic context-free grammars has found large application in the research area [53, 57, 70, 119, 135, 153, 158, 200, 220, 224, 234, 242]. They can be improved by taking more general patterns in user-chosen passwords into account [57]. This is also the advantage of string-based models which we describe next.

String-based models using markov chains and neural networks

A string-based probabilistic password model predicts the probability of the subsequent character in a password candidate based on the preceding characters, so-called *context characters*. For instance, given the context characters *passwor*, a string-based probabilistic model outputs the character *d* [162]. Similar to a template-based password model, it is trained using leaked passwords, natural-language dictionaries, and so forth.

String-based probabilistic password model can be realized using *markov models* [73, 169] and *neural networks* [162]. While markov models are very resource-intensive, neural networks provides a much more efficient solution and even allows to build an accurate client-side password meter [162].

Besides password cracking, markov chains are used in other applications such as password strength meters [54, 216] and password verification based on keystroke pattern analysis [56]. Moreover, various researchers have shown that markov chains and neural networks are more efficient than a probabilistic context-free grammar [70, 73, 153, 169].

3 Passwords require ubiquitous assistance

In this section, we detail the problem of users to use secure passwords for their online accounts. It turns out that the problem is not solved yet and we state that the Password Assistance System (PAS), presented in this thesis, is a promising solution.

We start in Section 3.1 with the specification of a system model and an attacker model. Then, we identify in Section 3.2 the requirements and conditions for the realization of secure passwords in practice. First, we define security requirements for passwords to resist the attacker we aim to protect against. Second, we identify conditions stated by services that affect the usage of passwords in practice. Third, we specify usage requirements to ensure a practical and usable solution as well as to address conditions, concerns, and needs of users.

Based on these requirements and conditions, we define three tasks for users that enable them to realize secure passwords for their accounts in Section 3.3. These tasks are generating, preserving, and changing passwords.

To realize these tasks in practice, we analyze the service conditions in Section 3.4 and 3.5 in detail. First, we present the largest survey on password requirements. We analyzed 185,696 services and show that their requirements are quite different, incompletely stated, and often entirely missing. Second, we present a survey on password usage at services. It appears that services have also different interfaces and procedures for login and password change.

Moreover, we examine existing literature and proposals with respect to the realization of the password tasks in Section 3.6. It follows from this examination that users have many problems with these tasks and cannot perform them manually. Further, existing proposals are unsatisfactory, impractical, or even insecure and do not enable users to use secure passwords.

We briefly describe our solution in Section 3.7. PAS enables users to realize the three tasks and thus secure passwords by automation and comprehensive support. We finally conclude this chapter in Section 3.8.

Parts of the contributions of this chapter were published in [H1, H2, H5]. This chapter extends the published contributions by (1) a detailed analysis of password-based authentication and its issues regarding security and ease of use and (2) an analysis of the password interfaces and procedures implemented by services.

3.1 System and attacker model

In this section, we present a system model that describes password-based authentication as it is used for online accounts at services on the Internet. Moreover, we describe the attacker model used in this thesis and characterize the attacker we aim to protect against.

System model

Entities

The entities involved in our system model are a user $\mathcal{U}$ and a service provider $\mathcal{SP}$. $\mathcal{SP}$ provides a service which is only available to legitimate users. The usage of the service requires $\mathcal{U}$ to authenticate himself by providing a username and a password. The username c_{un} identifies the user's account c at the service. The password c_{pw} acts as a credential conforming the legitimacy of $\mathcal{U}$. The objective of the password-based user authentication is that only $\mathcal{U}$ can access his account and use the service provided by $\mathcal{SP}$.

Password storage and preservation

In our system model, we expect that $\mathcal{U}$ preserves c_{pw} and c_{un} so that he does not need to memorize them. With respect to $\mathcal{SP}$, we assume that it also stores the password c_{pw} and the username c_{un}. The password is stored in a salted and hashed way. For a detailed explanation of the storage of passwords at services, we refer to Section 2.2.1.

Communication channel

The communication between $\mathcal{U}$ and $\mathcal{SP}$ is done over a public network, i.e. the Internet. Both entities establish a secure channel in which $\mathcal{SP}$ first authenticates itself to $\mathcal{U}$. The authentication of $\mathcal{U}$ using username and password is done through this secure channel. Both password c_{pw} and username c_{un} are transmitted in plain-text. In practice, the channel is established using TLS (cf. Section 2.1.6) and the authentication of $\mathcal{SP}$ is achieved by an X.509 certificate [63] issued by a trusted CA [42, 69].

Attacker model

Attacker goal

We consider an attacker $\mathcal{A}$ who aims at obtaining access to the account c of $\mathcal{U}$. To achieve this, $\mathcal{A}$ needs to have the username c_{un} and the password c_{pw} of $\mathcal{U}$. $\mathcal{A}$ succeeds when he impersonates $\mathcal{U}$ at the service, gets access to the account, and remains undetected.

Attacker capabilities

We assume $\mathcal{A}$ knows the username c_{un}. Obtaining c_{un} in practice is very easy as for most services the username corresponds to the user's email address or profile name in case of social network sites. With respect to obtain c_{pw}, we consider that $\mathcal{A}$ has the following three capabilities:

AC1 *Online attack*: $\mathcal{A}$ only knows c_{un}. He tries to obtain c_{pw} by repeatedly attempting to log in to the account c with a different password. For this attack, $\mathcal{A}$ uses the common login form of the service. The attack is successful when $\mathcal{A}$ finds c_{pw}.

AC2 *Offline attack*: $\mathcal{A}$ knows c_{un}, the salted and hashed password $H(c_{pw})$, and the used salt and hash function. $\mathcal{A}$ systematically try out all possible passwords offline. He uses the salt and applies the same hash function as $\mathcal{SP}$, until he finds a match for $H(c_{pw})$.

AC3 *Targeted attack*: $\mathcal{A}$ knows c_{un} and a password c'_{pw} of another account c' of $\mathcal{U}$. Within a *targeted online attack*, $\mathcal{A}$ attempts to log in to the account c with the password c'_{pw}. Moreover, if $\mathcal{A}$ has $H(c_{pw})$, he can verify in a *targeted offline attack* if c'_{pw} is a valid password for c by computing $H(c'_{pw})$ and comparing it to $H(c_{pw})$.

Attacker limitations

We exclude trivial attacks, attacks that cannot be prevented by technical means, and attacks that are not specific to passwords. This includes phishing [6], insider [223], social engineering [108], and malware [185] attacks. We also exclude attacks exploiting *account recovery mechanisms* [35, 96, 191] and breaking the secure channel between $\mathcal{U}$ and $\mathcal{SP}$ [62, 72, 166].

3.2 Requirements and conditions for secure passwords

In this section, we define security requirements, service conditions, and usage requirements for secure passwords. The security requirements are derived from the attacker model described in the previous section. For the realization of secure passwords, it is indispensable to take the service conditions into account. Passwords can only be used in practice under these conditions. The usage requirements are essential for a practical and usable solution as well as to address the conditions, concerns, and needs of users. They are deduced from the fact that users' preserve their passwords. In the following, we summaries the requirements and conditions and detail them in Section 3.2.1 to 3.2.3.

Security requirements

SR1 *Brute-force-resistant password*: A password must have a security level of at least 128 bits. If not feasible, the highest possible security level should be used. A password should neither have any patterns nor contain any user-related and service-related information.

SR2 *Individual password*: A password must be different for each account. It should not be reused (even not partially) for other accounts or password changes.

SR3 *Changing password*: A password must be changed on a regular basis and immediately after a compromise of the password is detected.

Service conditions

SC1 *Password requirements*: A password must comply with the individual password requirements of a service.

SC2 *Password interfaces and procedures*: A password can only be used through the password interfaces and according to the password procedures provided by a service.

Usage requirements

UR1 *Password confidentiality*: A password must be protected from unauthorized access.

UR2 *Password availability*: A password must be available on all user devices.

UR3 *Password recoverability*: A password must be recoverable in case of loss.

UR4 *Password accessibility*: A password must be available in emergency situations.

3.2.1 Security requirements

We provide a detailed justification for the security requirements in this section. These requirements are deduced from the attacker model described in Section 3.1. Table 3.1 illustrates how the security requirements SR1 to SR3 correspond to the attacker capabilities AC1 to AC3.

	AC1	AC2	AC3
SR1	■	■	
SR2			■
SR3	■	■	■

Table 3.1: Mapping of security requirements to attacker capabilities.

The security requirement SR1 is deduced from the general attacker's capability of guessing passwords. This might be just a handful of guesses within an online attack (AC1), or millions of guesses in case of an offline attack (AC2). Security requirement SR2 is derived from the attacker's capability of performing a targeted attack (AC3). And, requirement SR3 is deduced from all three attacker's capabilities and aims at invalidating passwords before the attacker can exploit them [212, 247].

SR1 – Brute-force-resistant password

Guessing a multitude of passwords through an online attack is time-consuming. It is limited by the bandwidth of the attacker and the services and can be thwarted by countermeasures such as blocking the attacker after multiple incorrect login attempts [66, 84, 90, 180]. But, in practice online attacks are feasible. Besides examples from the real world [14, 143], studies [37, 229] show that popular services are vulnerable against online guessing attacks.

While an online attack is always possible, an offline attack requires that an attacker first compromise a service and obtain its password database. Countless examples [122, 159, 165], even at popular services such as Adobe, eBay, and Twitter, show that such security incidents are likely and threaten millions of Internet users. The power of an offline attack has been demonstrated in case of the password breach at LinkedIn: 90% of the 117 million password hashes (SHA-1) could be cracked in 72 hours [89].

With respect to these results, users should not rely on services following best practices for thwarting online and offline attacks. Consequently, users should use passwords for their online accounts that withstand even millions of guesses. To measure the resistance of passwords

against brute-force attacks, the security level of passwords need to be determined. We provide a metric for this in Section 3.4.2. To withstand a brute-force attack, today, it is general consensus that a security level of 128 bits should be used [30, 44]. To this end, we require that passwords for online accounts have such a security level of 128 bits. However, as we show in Section 3.4, some services do not allow users to use attack-resistant passwords. To this end, we require that at least passwords with the best possible security level are used.

With respect to the more sophisticated attacks described in Section 2.2.2, it is important that passwords are not chosen from dictionaries, contain user-related and service-related information, and do not have any patterns which can be exploited. Such patterns occur when users use mangling rules or just try to randomly press keys on their keyboards. Without patterns, probabilistic models do not provide any benefit and an attacker needs to perform a brute-force attack which is very time-consuming and in case of a security level of 128 bits impossible.

Password-hashing function

A password-hashing function such as bcrypt [182], scrypt [176], and PBKDF2 [130] slows down a guessing attack. This might justify lower security levels to achieve security against offline brute-force attacks. However, in practice these functions are barely used [165]. Furthermore, such functions can only be considered secure when used with a proper number of iterations raising questions on their appropriate usage and actuality. For example the PBKDF2 standard recommending 1000 iterations goes back to the year 2000 [130].

In general, the research community does not consider current password-hashing functions secure anymore [15] and proposed new schemes such as Catena [88, 152] and Argon2 [26]. But, their implementations are still under development. And, most important, from the user perspective it is not visible how services are protecting their passwords. We make the pessimistic but robust assumption that services only salt and hash passwords (cf. Section 3.1) and use an ordinary hash function. This makes a security level of 128 bits necessary to protect from offline brute-force attacks.

SR2 – Individual password

In a targeted attack (cf. AC3) an attacker uses a stolen password, e.g. obtained by a successful online or offline attack, to get access to another account of the user. This attack is only possible when passwords are reused across accounts. Note that also a partially reuse of passwords allows to determine passwords with high accuracy [66, 230]. Consequently, the simple but effective countermeasure is to use an individual password of each account.

Besides a simultaneous reuse of passwords, it is also essential not to reuse old passwords. There exists no period of time until a compromised password becomes invalid. Reusing a password that was stolen even years ago is a security threat, as it can be misused by an attacker at any time. A prominent example is the hack of Mark Zuckerberg's Twitter and Pinterest account in 2016 [196]. Besides the fact that he used the weak password *dadada*, the attacker obtains his password from a password breach at LinkedIn four years before [89].

In general, we need to keep in mind that the compromise of passwords might remain undetected and it is impossible to determine which passwords can be reused without posing a security threat and which ones cannot. To this end, we require not to reuse passwords in general.

SR3 – Changing password

An online attack (cf. AC1) is always possible, even when services implement countermeasures like limiting the rate of login attempts. An attacker can easily adapt the rate of his attack to stay below the threshold triggering such a countermeasure. Although very time-consuming, it allows an attacker to test a multitude of passwords over months. As we cannot expect that services inform users about failed login attempts or users just do not notice such reports, it is necessary to change passwords regularly. This destroys all the attacker's knowledge about the user's password (in form of an exclusion list) and he must start from scratch.

In case of an offline attack (cf. AC2), an attacker can make millions of guesses per day. Besides the attacker's performance and time, the only limiting factors are the security level of the password and the way how the service had stored the password. With regard to security requirement SR1 and our assumption that services store passwords in a salted and hashed way, stolen passwords resist even offline brute-force attacks. Nevertheless, it is reasonable also to regularly change attack-resistant passwords due to the following reasons:

- *Service-side password storage*: Hash functions might become insecure (e.g. [31, 232]). In this way, an attacker is able to obtain the passwords without guessing them.
- *Password transmission*: Passwords are usually transmitted to services in plain-text [37]. Even when this is done through a TLS channel, an attacker might find security vulnerabilities to break the encryption (e.g. [62, 72, 166]).
- *Password compromise*: There are attacks against passwords in which the security level of passwords is irrelevant such as a phishing attack [185]. Such attacks can hardly be solved by technical means. The only practical mitigation is changing passwords regularly and thus invalidate potentially compromised passwords.

3.2.2 Service conditions

In this section, we briefly summarize the conditions for passwords made by services. For a detailed description we refer to Section 3.4 and 3.5. Taking these service conditions into account is essential because passwords can only be used in practice under these conditions.

SC1 – Password requirements

Password requirements are fixed rules regarding the password length and the allowed and/or required characters. Users can only select passwords for their accounts that comply with the individual requirements of services. We present in Section 3.4 a survey on password requirements and show that they are quite different, incompletely stated, and often missing at all.

SC2 – Password interfaces and procedures

Besides the selection of passwords, the usage of passwords is bound by services. Passwords at services can only be used through their provided password interfaces. In practice, these are the HTML forms for login, password change, and password reset at the services' websites. Moreover, the interaction with these interfaces is predefined by certain procedures. For instance, to change an account password, a user needs to log in to his account, browse to the password change form, and submit his new password. We describe the password interfaces and procedures of services in Section 3.5 in detail and demonstrate that they are also quite different.

3.2.3 Usage requirements

We next explain the usage requirements. They are essential for a practical and usable solution for the realization of secure passwords. Moreover, they address user conditions such as having numerous accounts [82], user concerns such as being afraid of password loss [160, 203, 208], and user needs such as having passwords available on all their devices in order to access account everywhere and anytime [105].

Due to security requirements SR1 and SR2, users need to cope with large password portfolios. Additionally, passwords must be regularly changed because of security requirement SR3. Memorizing such a huge number of passwords is impossible [2, 59, 82, 85, 164, 208, 237].

Therefore, a technical solution is necessary that preserves passwords for users. Typical realizations are storing passwords or regenerating them on demand (cf. Section 3.6.2). Such a solution needs to fulfill the following four usage requirements.

UR1 – Password confidentiality

Preserved passwords are vulnerable to unauthorized access. Typical situations are the loss or theft of persistent passwords. Therefore, it is necessary to protect them.

UR2 – Password availability

Today, users have multiple devices like desktop computers, tablets, and smartphones. Passwords must be available on all devices, so that users are able to access their accounts at Internet services everywhere and anytime. Otherwise, users will continue to memorize passwords and consequently use weak passwords. Users will not take the risk of being unable to log in to their accounts, because they do not have access to their passwords.

UR3 – Password recoverability

Preserved passwords are also threatened by loss. To this end, it is indispensable that users can recover them. Users take the possibility of losing their passwords and thereby their entire digital life very serious [160]. They would not use a solution that is vulnerable to password loss.

UR4 – Password accessibility

Preservation of passwords releases users from memorizing them. However, not knowing the passwords makes it impossible to give someone else access to accounts in urgent or emergency situations. For instance, after a robbery or natural disaster, it is essential that users can send messages (via email or social media) or get a copy of their identity card stored on a cloud storage provider. Because of this, users must be able to give passwords to someone else even when they have no access to them. Otherwise, they will keep unprotected copies of there passwords which are vulnerable to unauthorized access [208].

3.3 Password tasks of users

In this section, we define tasks for users that enable them to realize secure passwords for their accounts. This is done in the following way: At the beginning, we introduce the *password life cycle*, which describes the various stages of a password of an online account. Based on this life cycle and in consideration of the requirements and conditions for passwords described in the previous section, we specify the password tasks of users. The tasks are generating, preserving, and changing passwords. We detail the tasks in Section 3.3.1 to 3.3.3.

Password life cycle

The password of a user account at an Internet service has different stages. The progression of these stages is represented by the password life cycle which is illustrated in Figure 3.1. Note that this cycle describes the stages from the user perspective and represents the stages of a password for a single account. Therefore, this life cycle exists for each password of a user. The rectangles of the cycle represent the password stages and the arrows the user actions.

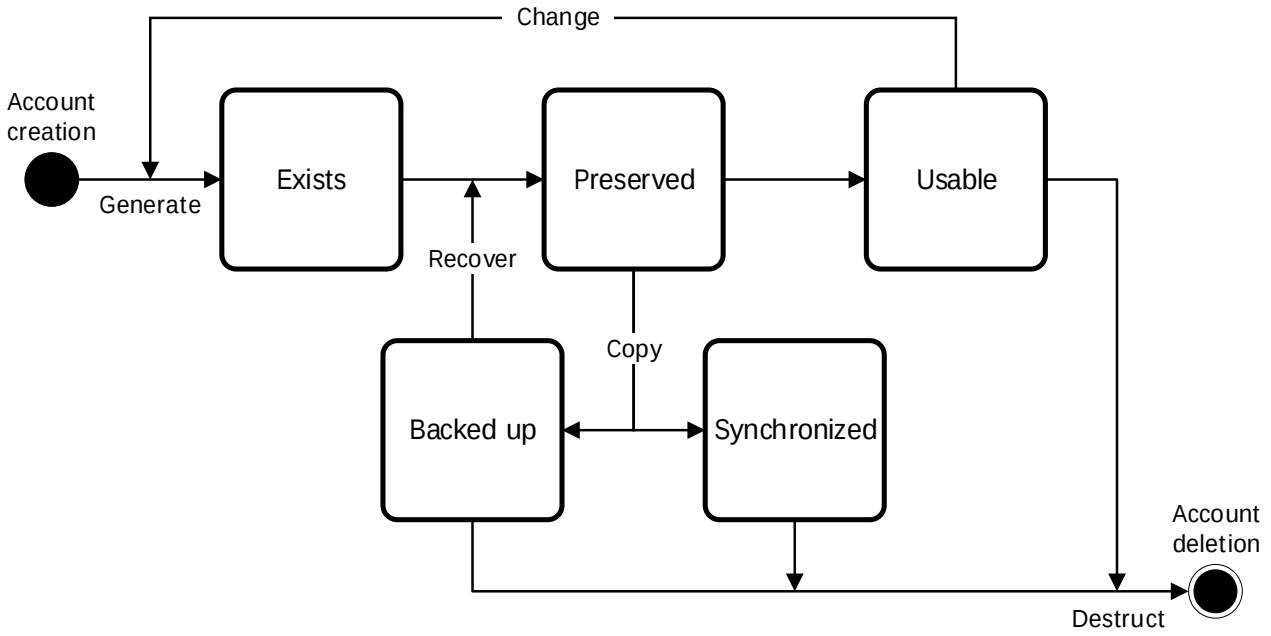


Figure 3.1: Password life cycle. It represents the progression of stages from the user perspective through which each password of an account passes.

We now describe the password life cycle in detail: During the account creation, the password is generated and its life cycle begins. After successfully submitting the password to the service and finishing the account creation, the password is preserved. To address the issue of loss, the preserved password is additionally copied to a backup location. In the event of loss, it can be recovered from this backup.

Against the background of multiple devices, the preserved password is synchronized between all user devices. When needed the password is retrieved from the preservation and becomes usable to users, for instance to log in to the account. In this situation, users can also change the password of the account which leads to the generation of a new password. This in turn leads to the preservation of this new password as well as the copy to the backup location and to the other user devices. Finally, the password is destructed when the account is deleted, including the backup and the copies located on other devices.

We remind the reader that we are focusing in this thesis on a pure user-side solution for the realization of secure passwords. To this end, we do not consider means of reseting passwords provided by services to solve the loss of preserved passwords. A typical mechanism is sending a new password to a recovery email address or mobile number. Note that such reset mechanisms have proven not to be reliable and secure [35, 96]. We address the issue of loss by creating a backup. This approach has the advantages that it does not rely on services properly implementing password reset mechanisms and it leaves users in control of their passwords.

Based on the password life cycle, security requirements, service conditions, and usage requirements for passwords, we define in the next sections tasks for users to realize secure passwords. Note that users need to do these tasks for every single password.

3.3.1 Password generation

The first password task of users is to generate a password. The password must fulfill the following three conditions:

1. The password must be attack-resistant (cf. SR1).
2. The password must differ from the passwords of other accounts (cf. SR2).
3. The password must comply with the password requirements of the service (cf. SC1).

Generating such a password is very difficult for users. We name this the *password generation problem*. In Section 3.6.1, we describe that users fail to generate such passwords in practice despite countless measures by services and proposals for users to generate passwords.

3.3.2 Password preservation

After generating the password, the second task for users is to preserve the password. In addition to this, users must perform the following four subtasks:

1. Protect the preserved password from unauthorized access (cf. UR1).
2. Make the preserved password available on all devices (cf. UR2).
3. Create a backup of the preserved password (cf. UR3).
4. Create a backup of the preserved password with emergency access (cf. UR4).

We refer to this as the *password preservation problem*. We summarize in Section 3.6.2 existing proposals for this problem. A prominent example is storing passwords in a password manager. In the following sections, we detail the subtasks of preserving the password.

3.3.2.1 Password protection

The first subtask of preserving the password is to protect it. We refer to this as the *password confidentiality problem*. We summarize in Section 3.6.2.1 existing proposals for this problem. The typical approach of encrypting preserved passwords with another password is insecure.

3.3.2.2 Password synchronization

The second subtask of users is to make the preserved password available on all their devices. We name this challenge the *password availability problem*. We summarize in Section 3.6.2.2 existing proposals for this problem. In brief, manually copying the preserved password to all devices is not a practical solution and the common approach of synchronizing preserved passwords between devices over the Internet leads to security issues.

3.3.2.3 Password backup

The third subtask is to create a backup of the preserved password. Users must protect the backup from unauthorized access and must place it at a secure location to protect it from loss as well. With respect to malware or physical damage like fire a different and distant location is important, e.g. at a friend's place.

We refer to this as the *password recoverability problem*. In Section 3.6.2.3 we examine proposals for creating backups of preserved passwords. It follows from this examination that placing backups at secure locations to protect them as well from loss and unauthorized access and keeping them up-to-date at the same time is an unsolved problem in practice.

3.3.2.4 Password backup with emergency access

The last subtask of users is to create a backup with emergency access in order to grant access to the preserved password to a trusted person in urgent or even emergency situations. Users must protect the backup from unauthorized access and must enable an emergency access. Finally, the user must place the backup at a trusted person.

We name this the *password accessibility problem*. In Section 3.6.2.4 we summarize proposals to realize an emergency access to preserved passwords. It turns out that they are unsatisfactory, impractical, and even insecure.

3.3.3 Password change

The third user task is to change the password regularly and immediately after a compromise of it is detected. This is deduced from security requirement SR3. Changing the password of an account consists of three subtasks:

1. Generate a new password. This implies the password generation problem which we described in Section 3.3.1.
2. Log in to the account and change the password. This must be done through the password interface and procedure of the service (cf. SC2).
3. Preserve the new password. This includes the password preservation problem that we describe in Section 3.3.2.

We denote this as the *password change problem*. In Section 3.6.3 we describe that users in practice barley change their passwords and that existing proposals for this problem are unsatisfactory or even insecure. A key challenge are the different password interfaces and procedures implemented by services (cf. Section 3.5).

To conclude, the realization of secure passwords for online accounts requires three complex and extensive tasks. Performing these tasks manually is practically impossible for users. In Section 3.6, we show that there is no solution for this problem. PAS solves this problem by automation and comprehensive support.

3.4 Service conditions for passwords: password requirements

In the following section, we detail the first type of service conditions for passwords: password requirements. In practice, users can only use passwords that comply with the password requirements of services. To perform the first password task of generating attack-resistant, individual, and particularly valid passwords (cf. Section 3.3) it is therefore essential to have a thorough understanding of the password requirements of services.

We present the first investigation of password requirements on a global scale by analyzing 185,696 services. We start in Section 3.4.1 with an overview of the application of password requirements and the usage of the following three main password requirements: password length, character sets, and required occurrences of certain characters. The characterization of the requirements reveals three results: First, only 30.98% of the services describe password requirements at all. Second, there is a large variety of requirements. We found 1401 distinct password requirements sets. Third, the requirements are often incompletely stated. Only 4017 (2.15%) services fully specify the essential information to create a valid password: lengths and allowed characters.

Then, we analyze in Section 3.4.2 the security implications of the password requirements. This is done by determining the security levels achievable with the given password requirements of the services. We show that none of the services enforces passwords that resist offline brute-force attacks and that every fifth service even does not allow such passwords at all.

For the collection of such a large sample of password requirements we developed a tool that extracts the requirements form the websites of services automatically. We analyzed more than 3 million websites to find services offering online accounts and therefore must have some sort of password requirements. We describe our tool later in Section 4.3 along with an evaluation and limitations. In brief, in an evaluation of 250 services our tool correctly extracted the requirements for 91.2% of the services. Consequently, we can expect the vast majority of our collected password requirements to be correct. Only a small fraction was incompletely or incorrectly extracted. Consequently, the concrete numbers for the application of password requirements and the security levels presented in this section might not exactly reflect reality. On the one hand, this can hardly be achieved because services may change their requirements at any time. On the other hand, our collected data provides such clear results that small deviations in the absolute numbers do not affect the key points of our contributions.

3.4.1 Application of password requirements

During our analysis of 3,236,319 websites, we detected 185,696 services with a sign-up form. Consequently, these services must have some sort of password requirements. However, only 57,536 (30.98%) services explicitly describe requirements on their websites. We noticed that popular and newer websites state requirements more often than smaller and older ones.

Table 3.2 shows the application of password requirements and lists the number of services specifying password lengths, character sets, and occurrences of certain characters. Altogether, only 4017 (2.15%) services fully specify the essential information to create a password: lengths and allowed characters. In total, we found 1401 distinct password requirement sets.

Password requirement	#Services	$\%\Sigma_{All}$
Password lengths	52,276	90.86%
Character sets	17,751	30.85%
Occurrences of characters	8,419	14.63%

Table 3.2: Number of services specifying password requirements and the percentage with respect to the 57,536 services (Σ_{All}) that specify password requirements at all. Services may be counted multiple times.

Our findings confirm Bonneau and Preibusch's [37] results from 2010 considering a set of popular 150 services. This shows, that the desolate state regarding well-documented password requirements is rather stable, despite the fact that users create better passwords when given advice as documented by various user studies [49, 77, 198, 220, 226].

For users, this lack of well-documented requirements is very problematic. Passwords have to be created using the *trial and error* approach: Users need to enter a password and wait for feedback by a password meter, or even worse, submit a password to find out whether it fulfills the requirements of a service. This makes password generation very onerous and challenging. Leading to resignation, this is certainly one of the reasons for weak passwords found by many user studies [123, 138, 208].

In the following sections we take a closer look at the three aforementioned types of requirements. In Section 3.4.1.1, we consider the password lengths. It appears that the minimal password lengths are quite similar between services but the maximum password lengths differ significantly. In Section 3.4.1.2, we present which character sets are accepted by the services. We discover that special characters and spaces find little application. Finally, Section 3.4.1.3 focuses on the restriction of occurrences of certain characters.

3.4.1.1 Password length

In the following, we focus on minimum and maximum password lengths. In Table 3.3 we list the number of services specifying a minimum and/or a maximum password length.

Password length	#Services	$\%\Sigma_{PWL}$	$\%\Sigma_{All}$
Minimum	22,948	43.90%	39.88%
Only minimum	11,310	21.64%	19.66%
Maximum	40,966	78.36%	71.20%
Only maximum	29,328	56.10%	50.97%
Minimum and maximum	11,638	22.26%	20.23%
Minimum or maximum	52,276	100.00%	90.86%

Table 3.3: Number of services specifying a password length as well as the corresponding percentage regarding the 52,276 services (Σ_{PWL}) specifying password lengths and the percentage regarding the 57,536 services (Σ_{All}) that specify requirements at all.

The minimum password length is the key factor for the lower bound of the achieved security level (cf. Section 3.4.2). It is highly questionable that 60.12% of the 57,536 services do not specify this requirement. The resulting lack of a minimum security level for the user passwords bears the risk that many passwords get compromised in case of a password breach at the services.

Two possible reasons why services specify a maximum password length are as follows: First, in order to prevent attacks (e.g. overflow attacks). Second, more likely, as a consequence of a limitation of their password implementations. Examples for such limitations are storage constraints of the password database or performance constraints when processing passwords.

The fact that services specify much more often a maximum than a minimum password length indicates that performance and service-side efforts caused by long passwords are more important than security considerations. This confirms the conclusion of Florêncio and Herley [83], based on an analysis of 75 services, that security requirements are not the driving factor of services' password requirements.

Minimum password length

Figure 3.2 illustrates the distribution of the minimum password lengths among 22,948 services. Astonishingly, we found more than 300 services that allow passwords with one or two characters. The most famous example is `wikipedia.org` with a minimum password length of a single character. The majority of services (35.40%) have a minimum length of 6 characters.

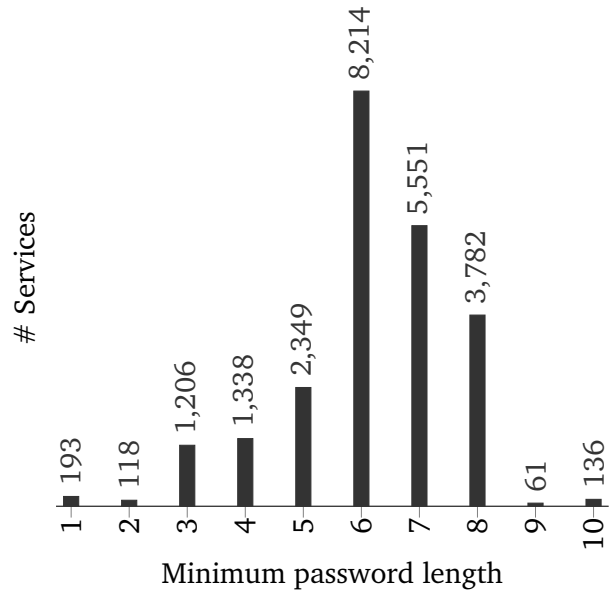


Figure 3.2: Distribution of the minimum password lengths of 22,948 services.

In 1999, Zviran and Haga [247] reported on the characteristics of 860 user-chosen passwords. The distribution of the length of these passwords (cf. [247, Figure 2]) is very similar to the distribution of the minimum password lengths that we found at the services 18 years later. Moreover, our findings confirm former and more recent studies [91, 92, 229] based on up to 50 services. Consequently, there is no progress towards longer and therefore stronger passwords. It seems that services still follow the NIST FIPS 112 publication from 1985 saying that "passwords should be, at a minimum, 6 characters in length" [47].

Maximum password length

Figure 3.3 depicts the maximum password lengths of 40,966 services. In comparison to the minimum password lengths, as illustrated in Figure 3.2, there is a much wider diversity. We exclude 36 password lengths which are specified by only one or two services (e.g. 37, 101, 64000) from the figure for readability reasons.

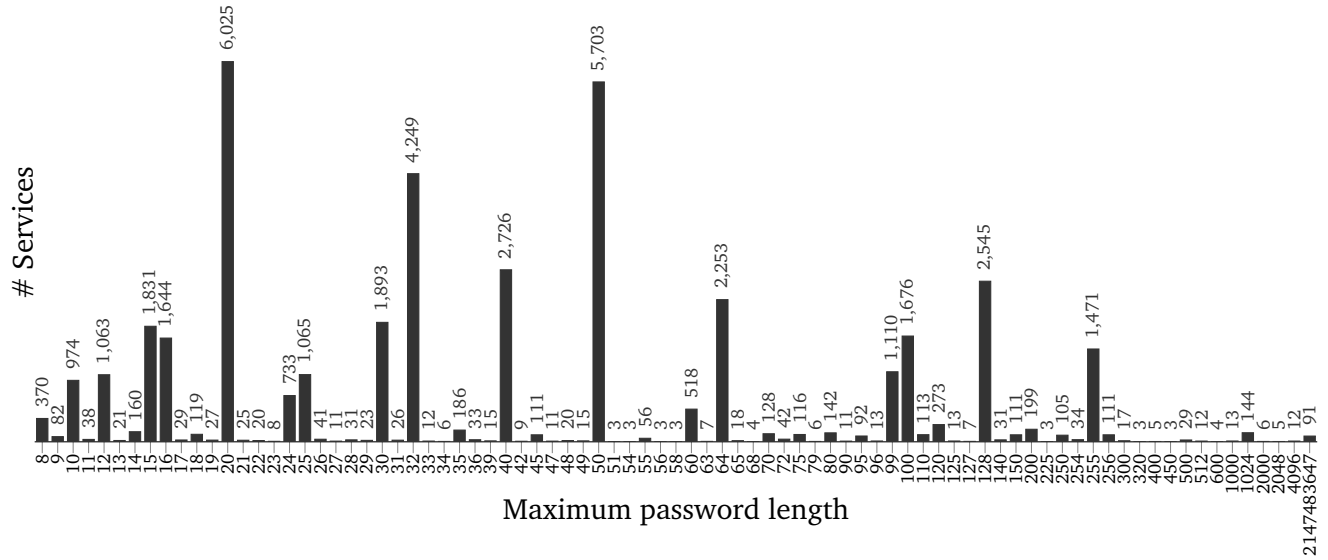


Figure 3.3: Distribution of the maximum password lengths of 40,966 services.

While many of the chosen maximum lengths appear arbitrary, the peaks reveal patterns which we discuss in the following:

- *Password lengths as a multiple of ten*: Services have a preference to select password lengths as a multiple of ten (e.g. 10, 20, 30). This is explicable with human-chosen lengths and reflects the general human preference to use smooth numbers. As illustrated in Figure 3.3 the most common lengths are 20 (14.70%) followed by 50 (13.92%).
- *Password lengths as a power of two*: Another characteristic of the distribution is that services tend to select the length as a power of two (e.g. 16, 32, 64). Services might believe that this is a benefit for the encryption or hashing algorithm they use when processing passwords for storage. Passwords that fit to the block size of a cipher or hash function need not to be padded. However, in practice such pursued benefits are questionable as it seems unlikely that users create passwords exactly matching the maximum length.
- *Password lengths suitable for numeric data types*: The high number of services with a maximum password length of 15, 255, and 2147483647 is an indication that the chosen maximum length has something to do with the service's password implementation. 15 is the maximum value representable by a nibble (half-byte), 255 by a byte, and 2147483647 by an integer. It seems that these services store the password length in a separate field in their database. That might indicate that these services store the passwords in plain-text instead of hash values. A hash value always has a fixed length independent of the input (e.g. in case of SHA-256 a 32 bytes value). Consequently, hashed passwords would not require to store the length. That services still store passwords in plain-text is shown by password breaches [122, 159, 165] and a survey by Bauman et al. [20].

A reasonable maximum password length must be a trade-off between the security level and performance. Random passwords with 1000 characters do not provide any security benefit in comparison to random passwords with 100 characters [207]. Also a maximum length of 2147483647 is nonsense and even dangerous because it might allow overflow and denial-of-service attacks. Encoding each character of a password with 1 byte, a password with 2147483647 characters would have a size of 2 gigabytes. It is very likely that creating an account with a 2 gigabyte password will crash a service. The services with a password length of 24 or 25 characters might have chosen such a reasonable length. These 1798 services provide an average security level of 145 bits, which fulfills our security requirement SR1 of attack-resistant passwords (cf. Section 3.2.1).

3.4.1.2 Character sets

In this section, we analyze the character sets that are accepted by services. For the 17,751 services specifying character sets we provide detailed numbers in Table 3.4.

Note that the character set *Letters* include services stating "uppercase characters", "lowercase characters", and "English characters". In case that a service states "alphanumeric characters" we count this as letters and numbers.

Character set	# Services	$\%\Sigma_{CS}$	$\%\Sigma_{All}$
Letters	12,416	69.95%	21.58%
Numbers	15,448	87.03%	26.85%
Specials	3,581	20.17%	6.22%
Spaces	1,201	6.77%	2.09%

Table 3.4: Number of services specifying character sets as well as the corresponding percentage regarding the 17,751 services (Σ_{CS}) specifying character sets and the percentage regarding the 57,536 services (Σ_{All}) that specify password requirements at all. Services may be counted multiple times.

Surprisingly, more services allow numbers than letters. Most probably, services allow letters by default and only state additional requirements. We noticed in a manual evaluation of 250 services that all accepted letters and numbers (cf. Section 4.3.3).

It is not that problematic that services focus on letters and numbers as long as they allow reasonably long passwords so that attack-resistant passwords can be used.

The application of special characters and spaces is very low. This seems to be another evidence that the design decisions for password requirements are driven by implementation constrains. Besides a limited password length, legacy systems only support passwords consisting of letters and numbers. Furthermore, accepting special characters and spaces demands a solid implementation which has a proper encoding and escaping of spaces and special characters. Otherwise, the implementation is vulnerable to various attacks such as SQL injections [106].

3.4.1.3 Occurrences of characters

Requirements regarding the occurrences of certain characters such as "use at least one number" are the least stated password requirements. Only 8,419 (14.63%) of the 57,536 services specify such a requirement. In Table 3.5 and 3.6 we provide detailed numbers for which character sets services specify minimum and maximum occurrences. We did not find a service specifying occurrences for spaces. The fact that minimum occurrences appear much more often than maximum occurrences indicates that this requirement serves the purpose to enforce that a character set is actually used by users when creating passwords.

Character set	# Services	$\%\Sigma_{Occ}$	$\%\Sigma_{All}$
Letters	3,661	43.48%	6.36%
Numbers	6,014	71.43%	10.45%
Specials	1,176	13.97%	2.04%

Table 3.5: Number of services specifying minimum occurrences as well as the corresponding percentage regarding the 8,419 services (Σ_{Occ}) specifying occurrences and the percentage regarding the 57,536 services (Σ_{All}) that specify password requirements at all. Services may be counted multiple times.

Character set	# Services	$\%\Sigma_{Occ}$	$\%\Sigma_{All}$
Letters	21	0.25%	0.04%
Numbers	103	1.22%	0.18%
Specials	25	0.30%	0.04%

Table 3.6: Number of services specifying maximum occurrences as well as the corresponding percentage regarding the 8,419 services (Σ_{Occ}) specifying occurrences and the percentage regarding the 57,536 services (Σ_{All}) that specify password requirements at all. Services may be counted multiple times.

3.4.2 Security levels resulting from password requirements

We analyze the security implications of the password requirements present in the previous section. This is done by determining the security levels achievable with the given password requirements of the services. We show that none of the 57,536 services enforces attack-resistant passwords and that every fifth service even does not allow such passwords at all.

We start by describing in Section 3.4.2.1 the assumptions that we make throughout the analysis and in Section 3.4.2.2 the metric we use to measure the security levels. Then, we present in Section 3.4.2.3 the results of our analysis.

3.4.2.1 Assumptions

As shown in Section 3.4.1, by far not all services explicitly state requirements. We base the analysis of the security implications presented in this section on the 57,536 services that provide requirements of any kind. If these requirements are incompletely stated, we make the following assumptions throughout the analysis:

- If no minimum password length is specified, we assume the service has no lower limit.
- If no maximum password length is specified, we assume the service has no upper limit.
- If no character sets are specified, we assume that the service allows letters and numbers.

The assumption that all services accept letters and numbers is based on the following facts:

- As shown in Table 3.4 (Page 33) approximately 70% of the services explicitly accept letters and nearly 90% allow numbers.
- The huge number of 52,276 (90.86%) services specifying a password length use phrases such as “at least 10 characters”. We interpret the term *characters* as letters and numbers.
- Research based on leaked password databases shows that passwords are usually alphanumeric (cf. [50, 224, 234]).
- In our evaluation set of 250 services that we used for our password requirements extraction tool (cf. Section 4.3.3) all services accepted letters and numbers.

To this end, we assume that all services that we analyzed accept letters and numbers. However, for languages based on non-Latin alphabets this assumption must be reviewed. For instance, Chinese services focus on numbers [229].

Note that assuming special characters as generally accepted by services cannot be justified given the relatively low number of services explicitly stating these as valid characters (cf. Table 3.4, Page 33). Furthermore, in practice there is no consensus observable which characters are actually to be subsumed as special characters. The ASCII character encoding standard defines 32 special characters plus space [11]. In our evaluation set of 250 services none of them defines these 32 characters. Sometimes services provide examples for special characters or use more puzzling phrases like "standard special characters" (e.g. `united.com`). We even found counterexamples such as `costco.com` and `target.com`. They claim to allow special character, but actually only a subset is allowed (e.g. < and > are not allowed). Therefore, we consider special characters and spaces only as accepted in our analysis if this is explicitly stated in the password requirements of a service.

3.4.2.2 Security metric

For the comparison and analysis of the security level of the password requirements, a uniform representation of security level is required. We determine the security level of password requirements by evaluating the security of passwords created under these requirements.

We adopt the metric by Florêncio and Herley [83] and calculate the security level of a randomly generated password by $S = L \cdot \log_2(C)$, where C is the cardinality of the character set and L is the password length. We consider a password consisting of letters and numbers and a length of 10 characters. Then, is $C = 26 + 26 + 10 = 63$, $L = 10$, and $S = 10 \cdot \log_2(62) = 59$ bits.

The security level provided by the password requirements of a service is defined as the range between a minimum security level S_{min} and a maximum security level S_{max}. These levels are calculated as $S_{min} = L_{min} \cdot \log_2(C)$ and $S_{max} = L_{max} \cdot \log_2(C)$, where L_{min} and L_{max} is the minimum and maximum password length, respectively.

For instance, Google has a minimum password length of $L_{min} = 8$ and maximum password length of $L_{max} = 100$. Furthermore, it allows uppercase letters, lowercase letters, numbers, special characters, and spaces resulting in $C = 26 + 26 + 10 + 20 + 1 = 83$. Note that Google states "common punctuation" characters, however does not list them in detail. We successfully tested the following 20 special characters: ! " § $ % & / ( ) = \ ? + - * # . : , ;. Under these password requirements, the security level is $S_{min} = 51$ bits and $S_{max} = 638$ bits.

Our metric ignores the password requirements of minimum and maximum occurrences of characters. We consider this as an acceptable simplification because the requirement of maximum occurrences is almost never used (cf. Section 3.4.1.3). And, the requirement of minimum occurrences lowers the security level by at most 1 bit in the worst case (given a password length

$L \geq 4$) but on average the reduction is much lower: Let C_i for $i = 1, ..., n$ be the cardinalities of the allowed character sets and $C = \sum_i C_i$. Given for each of the character sets, one occurrence is enforced, then $S = \log_2(C^L - \sum_i (C - C_i)^L) = \log_2(C^L \cdot (1 - \sum_i (\frac{C-C_i}{C})^L)$. As each of the terms $(\frac{C-C_i}{C})^L$ approaches zero for sufficiently large L, S approaches $\log_2(C^L) = L \cdot \log_2(C)$. Now consider the worst case found in all analyzed password requirements, which occurs for the combination of the character sets letters and numbers. Then is $C = 62$, $C_i = 52$ and $C_2 = 10$. For $L = 4$, $(1 - \sum_i (\frac{C-C_i}{C})^L = 1 - (\frac{52}{62})^4 - (\frac{10}{62})^4 > 0.5)$ which implies, that less than 1 bit of security level is lost.

Furthermore, the metric provides an upper bound for the achievable security level and is appropriate under the assumption of randomly generated passwords. Note that user-chosen passwords usually follow certain patterns that can be exploited as we explain in Section 2.2.2.3 and 3.6.1. Therefore, the actually realized security levels of user-chosen passwords are often much lower. However, we use our metric only to compute the security level of random passwords as well as to compare the security level of password requirements. To this end, it is adequate for our needs. For measuring the security level of user-chosen passwords, we recommend to use *guessability*, which provides the number of guesses needed by an attacker to guess a given password [33, 34, 135, 222, 234].

3.4.2.3 Findings

We calculated the minimum and maximum security levels for the 57,536 services using our aforementioned metric. Figure 3.4 illustrates which percentage of the services stays below a certain security level given the specified password requirements. The upper blue line represents the distribution regarding minimum security levels, the lower red line the distribution regarding maximum security levels.

The sharp increases of the security levels are caused by the strong dependence of the security level on the specified password lengths and mirror the peaks in Figure 3.2 and 3.3 (Page 31 and 32). For instance, there is a strong increase at 35 bits at the line depicting the minimum security levels. This is caused by an increase of the minimum password length from 5 to 6 characters (cf. Figure 3.2, Page 31). A same characteristic exists near 120 bits for the maximum password length (cf. Figure 3.3, Page 32) which is caused by a length of 20 characters.

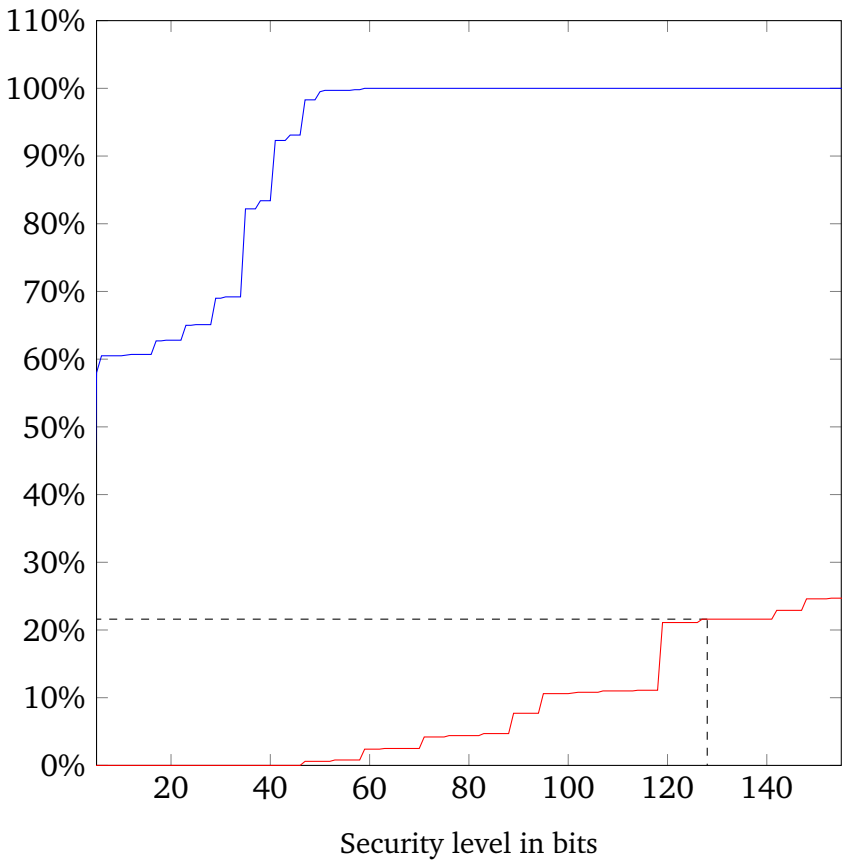


Figure 3.4: Percentage of services staying below a certain security level. The blue line represents the distribution regarding minimum security levels, the red line the maximum levels.

None of the services enforces attack-resistant passwords

Regarding the serious threat of offline brute-force attacks (cf. Section 3.2.1) it would be reasonable that services enforce attack-resistant passwords as a counteractive or precautionary measure (besides properly storing passwords).

Nevertheless, not a single service enforces attack-resistant passwords as the blue curve of Figure 3.4 shows. Around 60% of the services do not enforce any security level at all. Figure 3.4 also reveals that the vast majority of services (99.5%) already accept passwords with a security level of 50 bits. But, what does it mean in practice and is this a security problem? To answer this question, we need to take a look at the feasibility of brute-force attacks in practice and how users typically create passwords.

Current password cracking tools like Hashcat [101, 111] can perform more than 2^{50} SHA-1 hashes on a single day using standard hardware. So, from a technical perspective, a brute-force attack for a security level of 50 bits is absolutely feasible. But, does an attacker really need to brute-force a search space of 2^{50} bits?

Our calculated security level assumes randomly generated passwords which can be seen as an upper bound. In practice users do not create random passwords. Their passwords often include user-related and service-related information and have exploitable patterns (cf. Section 3.6.1). As we explained in Section 2.2.2, this makes attacks much more effective.

Moreover, user studies show that users choose passwords that narrowly comply with the minimum password requirements [200, 201, 221]. On average, they choose one character more than required. This also allows optimized attacks. Consequently, we expect that user-chosen passwords have a lower security level than calculated with our metric and an attacker must perform less guesses to obtain many passwords. This has clearly been shown in practice: 90% of the 117 million LinkedIn password hashes (SHA-1) could be cracked in 72 hours [89].

Every fifth service does not allow attack-resistant passwords

As we described in Section 3.2.1 both users and services can tackle the threat of offline brute-force attacks by employing a high security level for passwords. When considering security conscious users, the question evolves, whether they would be able to use attack-resistant passwords in practice.

The maximum achievable security levels are depicted in the lower red graph of Figure 3.4. It shows that, 21.6% of the services prevent passwords with a security level greater or equal to 128 bits. For these services, users cannot use attack-resistant passwords, even if they want to. From the viewpoint of automatic password generation based on the specified password requirements (cf. Chapter 4) this means that such an approach leads to attack-resistant passwords for 78.4% of services that specify their requirements.

To conclude, the security implications of the password requirements specified by the services on the Internet are unsatisfactory and leave much room for improvement. First, the wide acceptance of weak passwords (i.e. security level of 50 bits) is a huge security problem, as security unaware users are not guided towards secure passwords. That this is indeed a problem was demonstrated by countless examples (cf. [122, 159, 165]). Second, by preventing users to use attack-resistant passwords for their accounts it in many cases even affects password security in a negative way, contradicting its desirable goal of leading to more security.

3.5 Service conditions for passwords: password interfaces and procedures

In this section, we characterize the second type of service conditions for passwords: password interfaces and password procedures. In practice, users can only access their accounts through the services' login forms and change their passwords through the respective password change forms available at the services' websites. Moreover, the interaction with these interfaces, i.e. forms, is predefined by certain procedures. With respect to password changes such a procedure requires for instance that users need to additionally enter the current account password along with the new password. To perform the third password task of changing passwords (cf. Section 3.3.3) and thus realize secure passwords, it is therefore necessary to have a thorough understanding of the actual implemented password interfaces and procedures of services.

We conducted an analysis of password interfaces and procedures of 200 representative services[1] In the following, we present the results of a subset of 10 popular services with regard to password changes. The services a listed in Table 3.7. At the beginning, we provide in Section 3.5.1 an overview of the results and exemplary describe the password change procedures of Google and Facebook. It appears that the services' password interfaces and procedures differ in many ways. This includes the number of actions that are necessary to perform a password change, the way to firstly reach the password change form, and to actually change a password.

Then, we take a closer look at the interfaces and procedures implemented by services for the login in Section 3.5.2 and for the password change in Section 3.5.3. It turns out that the login is very similar at services. But, the challenge of generating passwords under the services' password requirements, that we detailed in Section 3.4 for the sign-up of accounts, occurs again when users need to generate a new password during the password change.

Our analysis reveals insights in the different implementations and technologies used by services and provides the basis to cope with this problem. In Chapter 7 we present a standardized description of password interfaces and procedures that provides the first practical solution to handle this problem.

[1] The Alexa Top 500 US list [8] reduced by websites with pornographic and illegal content, non-English websites, and websites that do not have or allow the creation of online accounts (e.g. banking websites). The list of services is available at [H26].

3.5.1 Overview

For a thorough analysis of the password interfaces and procedures for changing passwords at services, we divide a password change procedure into three sub-procedures:

1. *Login*: To change an account password, a user logs in to his account.
2. *Select password change*: After the login, a user navigates to the password change form where he can actually change his account password.
3. *Perform password change*: After reaching the password change form, a user enters his new password into the form and submits it to the service. He might need to provide the current password again and/or need to confirm the new password by entering it twice.

We examined the aforementioned sub-procedures of the 10 services. For each sub-procedure, we counted the actions that users need to perform. Actions include entering a string (e.g. username or password) and clicking on a button to submit the data or to navigate through the website.

In Table 3.7 we provide the number of actions that users need to perform to change the password of an account at the respective service. In the second to fourth column, we list the number of actions for the respective sub-procedures. According to these numbers, the login at the services is quite similar, but the number of actions to change passwords significantly differs between the services. While users just need 5 actions to change a password of a Wikipedia account, they need 10 actions in case of an eBay account.

Service	Login	Select password change	Perform password change	Σ
Google	4	4	5	13
Facebook	3	3	4	10
Amazon	3	4	4	12
Reddit	3	2	4	9
Yahoo	4	4	5	13
Wikipedia	3	2	3	8
Twitter	3	3	4	10
eBay	3	4	6	13
LinkedIn	3	3	4	10
Netflix	3	3	4	10

Table 3.7: Number of actions for password change.

Password change procedure of Google

We illustrate in Figure 3.5 the procedure to change a password of a Google account. Each rectangle represents a user action such as entering a string or clicking a button. Moreover, the actions in Figure 3.5 are divided into the three aforementioned sub-procedures: login, select the password change, and finally perform the password change.

Google has a two-step login procedure, where the username and the password is entered on a separate login screen (cf. Section 3.5.2). At the first login screen, a user only enters his username and then clicks the "Next" button. At the second login screen, the user enters his password and then clicks on "Sign in". To reach the password change form, the user needs to navigate through multiple extensive account settings pages: First, he needs to click on the *account icon* and then on the "My Account" button. In the account overview, the user needs to click the "Sign-in & security" button and then the "Password" button.

The actual password change consists of five actions: First, the user needs to enter his current password at the login form again (second login screen). Second, he needs to click on "Sign in". The third and fourth action is to enter and confirm the new password. Finally, the user needs to click on "Change password" to submit the new password to Google. In total, changing a password of a Google account takes 13 actions.

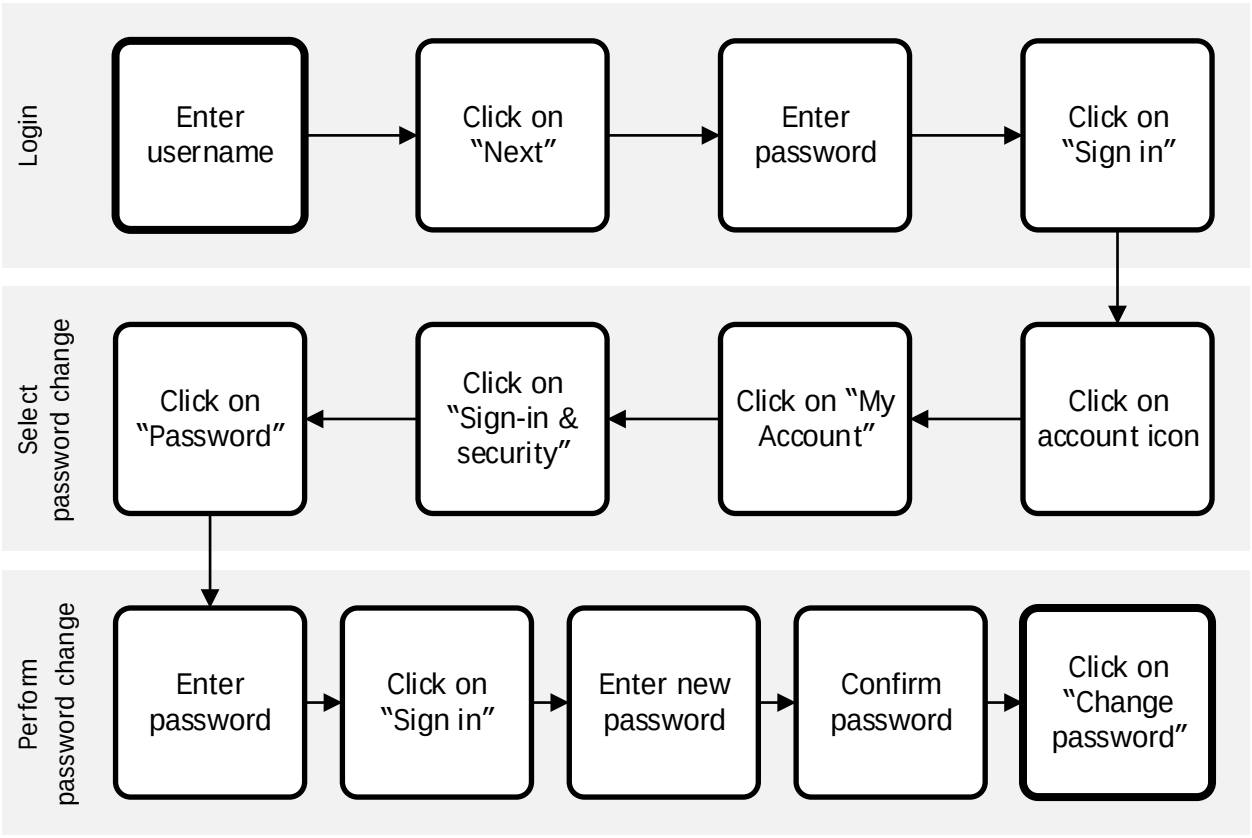


Figure 3.5: Password change procedure of Google.

Password change procedure of Facebook

In Figure 3.6 we illustrate the actions to change a password of a Facebook account. It is obvious that this requires less actions and can be done faster and is a less cumbersome task for users.

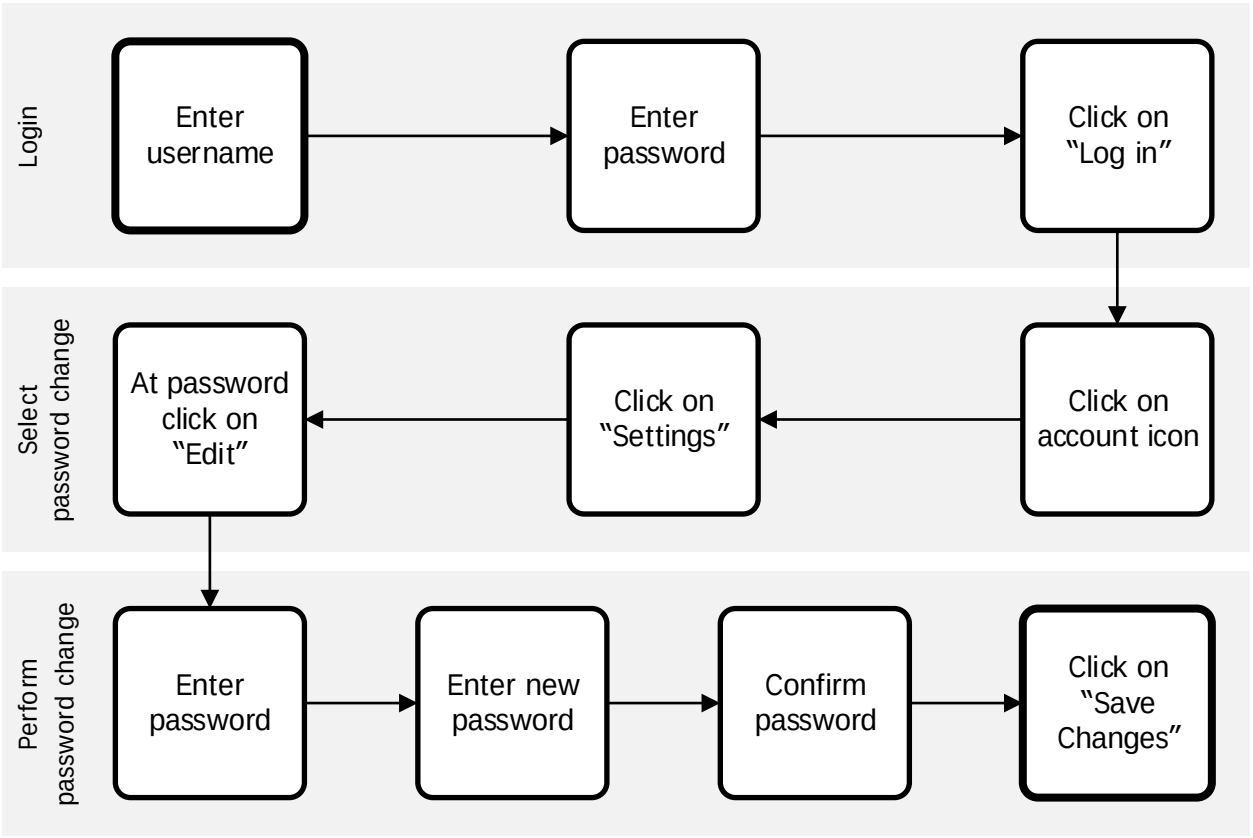


Figure 3.6: Password change procedure of Facebook.

3.5.2 Login interfaces and procedures

In this section, we focus on the interfaces and procedures of the login at the evaluated services. In general, we can distinguish between two types of realizations:

1. *One-step login procedure*: A user simply enters his username and password and clicks on a button, which sums up to three user actions. As shown in Table 3.7 such a one-step login procedure is implemented by the majority of services that we have evaluated.

2. *Two-step login procedure*: Entering the username and the password is divided into two steps and two respective login screens. First, a user enters his username and clicks on a button. Second, he enters his password and completes the login by clicking on a further button.

Regarding the analyzed services the two-step approach is only implemented by Google and Yahoo. Introduced in 2015, Google aimed at an easier integration of other authentication means

into the same login screen, e.g. one-time passwords [100]. This approach also allows to add a user-specific image or message into the second screen to tackle phishing attacks. A similar system was already introduced by Yahoo in 2007 [3]. As we see in the next section there is also a similarity between the realization of the login and the password change procedure.

3.5.3 Password change interfaces and procedures

We now focus on the interfaces and procedures to actually change the password of an account. As already mentioned, we divide this into two sub-procedures. First, we evaluate in Section 3.5.3.1 the actions to reach the password change form. Second, we examine in Section 3.5.3.2 the actual password change which consists of entering the new password and submitting it to the service through the respective password change form.

3.5.3.1 Select password change

To actually change the password of an account, a user needs to navigate to the password change form where he can enter his new password. In Table 3.8 we enumerate the buttons that a user needs to click in order to reach the password change form at the respective service.

Service	Buttons to reach a password change form
Google	Account icon → “My Account” → “Sign-in & security” → “Password”
Facebook	Account icon → “Settings” → “Password”
Amazon	Account icon → “Your Account” → “Login & Security Settings” → “Edit”
Reddit	“Preferences” → “password/mail”
Yahoo	Account icon → “Account Info” → “Account security” → “Change password”
Wikipedia	“Preferences” → “Change password”
Twitter	Account icon → “Settings and privacy” → “Password”
eBay	Account icon → “Account settings” → “Personal Information” → “Edit”
LinkedIn	Account icon → “Settings & Privacy” → “Change”
Netflix	Account icon → “Your Account” → “Change password”

Table 3.8: Actions to reach a password change form.

Besides the number of buttons users need to click, there is also a large variety in the naming of these buttons. While Amazon uses a precise description “Login & Security Settings”, eBay has a more puzzling naming “Personal Information”. Greene and Choong [103] pointed out that users

have problems with the ambiguous terminology used in password requirements stated by services. It is very likely that users also struggle with the different ways to reach a password change form at a service. An indication for this are the various tutorials how to change passwords at services available at `http://howdoichangemypassword.com` and `http://www.wikihow.com`.

3.5.3.2 Perform password change

Besides different a number of actions for reaching the password change form, there is also a large variety regarding the number of actions to actually change a password (cf. Table 3.7, Page 41). We noticed that services that implement a one-step login procedure also implement a simpler password change form where users just need to enter the current password and two times the new password. We made the same observation with the two-step login procedure, which is implemented by Google and Yahoo. When changing a password at both services, a user sees the second login screen again and needs to enter his current account password. Then, both services provide a form where a user just needs to enter his new password twice.

However, we found two exceptions: First, in case of Wikipedia users do not need to provide the current password and can just enter and confirm the new password. Because of this, a password change at Wikipedia requires the fewest user actions. Second, like Google and Yahoo, eBay users see the login screen again and need to enter the current password. But, then a user must enter his current password again at the subsequent password change form. In summary, changing a password of an eBay account requires to enter the current account password three times and the new password two times.

Password requirements at the password change forms

In Section 3.4, we showed that generating a valid password for an online account is very difficult for users because the password requirements of services are quite different, often incompletely stated, or entirely missing. But, in the analysis in Section 3.4 we evaluated the password requirements available at the sign-up forms of services. It is unclear whether services properly state their password requirements at the password change forms. Unfortunately, our analysis reveals that the lack of well-documented password requirements occurs again.

Our analysis was done by evaluating the password change forms of the services. We evaluated whether the services state their requirements at the password change form and whether they provide a password meter (we describe password meters in detail in Section 3.6.1).

Due to the lack of well-documented password requirements at the majority of the services, we additionally determined the requirements by trying out a multitude of passwords with different lengths and character sets. Note that we were not able to determine the maximum length of some services such as Facebook, as illustrated by the question mark (?) in Table 3.9. We skip our tests when a service accepted passwords with more than 1024 characters. The results of our analysis are presented in Table 3.9 and discussed in the following.

Service	Password requirements	Password lengths	Character sets	Password meter
Google	■	8 – 100	L, N, SP, S[a]	■
Facebook	□	6 – ?	L, N, SP, S	■
Amazon	□	6 – 1024	L, N, SP, S	□
Reddit	□	6 – ?	L, N, SP, S	□
Yahoo	□	7 – 128	L, N, SP, S	□
Wikipedia	□	1 – ?	L, N, SP, S[b]	□
Twitter	□	6 – ?	L, N, SP, S	■
eBay	■	6 – 64	L, N, SP[c]	■
LinkedIn	■	6 – ?	L, N, SP, S	■
Netflix	■	4 – 60	L, N, SP, S	□

[a] A password cannot start or end with a space.

[b] A single space as a password is not allowed, but a single number or letter.

[c] The special characters <, >, [,], / are not allowed.

Table 3.9: Password requirements and password meters available at password change forms. The character sets are abbreviated by L = letters, N = numbers, SP = special characters, and S = spaces. The symbol ■ denotes that a service states password requirements or provides a password meter. In contrast, the symbol □ illustrates that a service do not provide password requirements or a password meter at its password change form.

Application of password requirements

The results of our analysis in Table 3.9 show that only 4 of the 10 services actually state their password requirements at their password change form. Consequently, we can see that there is the same lack of well-documented password requirements as at the sign-up forms (cf. Section 3.4.1). Furthermore, we found out that services that do not provide password requirements at their password change form also do not state their requirements at the sign-up form.

Password lengths

The distribution of the minimum password lengths of the services fits to the results of our large-scale survey of password requirements stated at sign-up forms. There is also the same large variety for the maximum password length (cf. Section 3.4.1.1).

Character sets

Regarding the allowed character sets, we found a larger acceptance of special characters and spaces in comparison to our results presented in Section 3.4.1.2. However, we also encountered the missing consensus which characters are actually to be subsumed as special characters. For example, eBay states to accept special characters, but does not allow <, >, [,], /. Moreover, we found limitations regarding spaces. Passwords at Google cannot start or end with a space. Anecdotally, Wikipedia does not accept a single space as a password while a single number or letter is fine. Regarding the occurrences of characters only eBay requires that besides letters a password must also include at least one number or one special character.

Password meters

Besides the missing password requirements there is also little application of password meters. A password meter provides a graphic feedback whether the entered password fulfills the password requirements or not. Furthermore, it rates the security level of the password using colored bars or labels such as *weak*, *normal*, or *strong* which indicate the strength of the password. Only half of the services provide a password meter. Although, there are many issues with password meters in practice, researchers have documented that they lead to stronger user-chosen passwords. We describe password meters including their advantages and disadvantages in Section 3.6.1.

3.6 Passwords in practice and the state of the art

In this section, we describe that the problem of users to use secure passwords for their online accounts is not solved yet. This is done in the following way: For each of the password problems detailed in Section 3.3, we provide an overview of the related literature. We summarize studies on how users cope with the problems in practice. And, we describe proposals that aim at the mitigation of the problems. We also evaluate whether these proposals are suitable for the realization of the three password tasks of generating, preserving, and changing passwords. It turns out that none of the proposals address all password problems and that the service conditions often prevent their practical use.

We begin with the password generation problem in Section 3.6.1. It turns out that an enforcement of proper passwords by services leads to circumvention strategies by users. Moreover, password-composition advices for users do not create attack-resistant passwords and tools for password generation are impractical because they do not consider the different password requirements of services. To this end, the password generation problem is unsolved.

Then, we focus on the password preservation problem and evaluate existing approaches for the realization of the second password task of preserving passwords in Section 3.6.2. Subsequently, we analyze proposals for the password confidentiality problem in Section 3.6.2.1. It becomes apparent that properly protecting preserved passwords is an unsolved problem and the common approach of using a master password is insecure. We examine proposals to solve the password availability problem in Section 3.6.2.2. It follows from this examination that the common approach of storing preserved passwords on Internet servers is not preferred by users and leads to many security issues. Afterwards, we focus on the password recoverability problem in Section 3.6.2.3. According to user studies, the risk of password loss is the main obstacle for users to preserve passwords. It follows from the examination of proposals that placing password backups at secure locations to protect them as well from loss and unauthorized access while keeping backups up-to-date at the same time is an unsolved problem in practice. We evaluate proposals for the password accessibility problem in Section 3.6.2.4. It becomes apparent that existing proposals do not meet user needs. Users are in the dilemma of either losing control of their passwords or having access to them in emergency situations. In summary, there exist no solution for the realization of the second password task of preserving passwords and ensuring their confidentiality, availability, recoverability, and accessibility.

Finally, we discuss the password change problem in Section 3.6.3. According to studies, users change their passwords very seldom. Proposals to automatically change passwords on behalf of users struggle with the different password implementations at services and still let users in charge of triggering password changes. Consequently, there exist no fully automated solution to realize the third password task of regularly and if necessary immediately changing passwords.

3.6.1 Password generation

Users have well-defined strategies to generate passwords [208, 221]. These strategies often contain user-related and service-related information [202, 247]. Examples include terms related to love, sexual terms, profanity, animals, food, money [224], dates [225], phrases from music lyrics, movies, literature, television shows [141], and locations [221] as well as the names or the URLs of services [184, 221]. Furthermore, the users' strategies lead to patterns in passwords such as starting with letters and ending with numbers or special characters [82, 138, 153, 184,

202, 203, 212, 247]. These highly predictable information and patterns make passwords easy to guess [34, 135, 138, 158, 200, 201, 221]. Besides user studies, countless password breaches [122, 159, 165] show the tremendous and persistent dimension of the password generation problem. It remains a problem for users for decades [167, 211].

There exists various proposals to solve this problem, which we discuss in the following. First, we examine in Section 3.6.1.1 the approach of services to enforce proper passwords. Second, we evaluate password-composition advices for users to generate proper passwords in Section 3.6.1.2. Finally, we analyze tools for users to generate passwords in Section 3.6.1.3. It turns out that none of the approaches solves the password generation problem and can be used to realize the first password task of generating attack-resistant, individual, and valid passwords.

3.6.1.1 Enforcement

In the following, we analyze the approach of services to enforce proper passwords. It appears that users often respond with circumvention strategies and enforcement does not solve the password generation problem.

Password requirements

Password requirements are rules for passwords with respect to the length and the allowed and/or required characters. Services implement password requirements to guide users to select proper passwords but also to prevent incompatibilities with their password implementations. Typical examples for incompatibilities are encoding problems with an umlaut or a backslash.

The general problem of password requirements is that they cannot prevent users from choosing and reusing weak passwords such as *password1234* [197]. Such a password on the one hand might fulfill the requirements of a service, while on the other hand is easy to guess. In case passwords get rejected by services, users have well-defined coping strategies. They append special characters or numbers and always use the same character for this purpose [208, 221].

As we have shown in Section 3.4.2 as well as done by other researchers [37, 229], the password requirements of services fail to serve their purpose with respect to security. They do not enforce attack-resistant passwords and, even worse, some services do not allow them.

The design decisions on which services choose their password requirements are often unclear [229]. Security needs seem not to be the driving factor as some of the largest and most attacked services allow weak passwords [83]. There is also no noticeable improvement towards better password requirements [91, 92], despite various well-engineered proposals (cf. [135, 201]).

Password requirements often lead to user frustration [123, 158]. Users often need multiple attempts to generate passwords in accordance with password requirements [138, 200, 201, 226]. One reason is the ambiguous terminology used in password requirements. Users are confused by terms such as *non-alphanumeric*, *symbols*, *special characters*, and *punctuation marks* [103]. As we mentioned in Section 3.4.2.1, in practice there is also no consensus observable on the service-side which characters are actually to be subsumed as special characters. It is likely that this also applies for symbols and punctuation marks.

Password meters

Password meters provide a graphic feedback whether the entered password fulfills the password requirements or not. Furthermore, it rates the security level of the password using colored bars or labels such as *weak*, *normal*, or *strong* which indicate the strength of the password.

Various studies have shown that password meters lead users towards the generation of stronger passwords [77, 198, 218, 220]. However, this applies only for important accounts. In case of low-value accounts users continue using weak passwords even with the presence of a password meter [77]. Moreover, only password meters that rate passwords very strictly actually lead to stronger passwords in the sense of guessing-resistance [220]. But, strict password meters make the password generation more time-consuming so that users find them more annoying [220].

In practice, password meters are highly inconsistent in assessing the security level of passwords. Even weak passwords such as *password1* are rated as strong which makes users believe that they have chosen a proper password, but actually they have not [50, 51]. Many implementations of password meters also have security issues, including the leakage of passwords to third parties and the transmission of passwords over the Internet in plain-text [1]. The design of password meters at Internet services is also often not well-engineered [50], despite various proposals for accurate password meters (cf. [54, 216, 218, 228]).

Blacklists

Services also try to enforce proper passwords by rejecting common passwords such as *123456*, *password*, and *letmein*. Blacklists often contain commonly and frequently used passwords [192]. However, large blacklists demand an proper implementation to ensure an efficient lookup [23, 205]. Although recommended by the NIST [46], blacklists are not widely adopted [229] and there is a huge difference between their implementations [50]. Moreover, they only lead to a small improvement regarding guessing attacks [135, 198, 201]. Users circumvent blacklists by simply changing the capitalization or inserting a number to their blacklisted passwords [104].

System-assigned passwords

Another approach is to assign users passwords generated by services. Such system-assigned passwords can be generated with respect to services' countermeasures against online and offline attacks. Moreover, they solve the problem of password reuse (as long as users cannot change passwords). Despite these security benefits, system-assigned passwords suffer from poor memorability [199, 237].

To mitigate this issue, there is the idea of *persuasive passwords* [87, 118]. User-chosen passwords are strengthened by the service by adding random characters at random positions. However, this approach fails in practice because users compensate the service-side improvement by choosing simpler initial passwords [87]. In general, an attacker knowing the strengthening algorithm can easily guess the strengthen passwords so that the security improvement is negligible [195].

3.6.1.2 Password-composition advices

Researchers identified the users' lack of security knowledge as a major reason for weak passwords [128]. This indicates that the problem might be solved by providing instructions on how to proper generate passwords. In this section, we examine advices for users to generate passwords. It turns out that they cannot be used to generate attack-resistant passwords.

Complex passwords

First and foremost, there is the ancient advice of using complex passwords: 8 characters with a mixture of uppercase and lowercase letters, numbers, and special characters [226]. Having more different types of characters makes passwords harder to guess. The security benefit of complex passwords is also known by users [132]. However, today passwords with 8 characters provide negligible security [135, 200, 201]. They can be guessed by a brute-force attack in a few hours and therefore longer passwords are necessary [101, 111].

Mangling rules

Memorizing complex passwords is difficult for users. To solve this, one proposal is to use a word or a phrase and transform it to a complex password by means of mangling rules. A typical rule is to transform letters to uppercase or numbers. For example, *password* can be transformed into *Password*, *password1*, and *pa33word*. But, nowadays password cracking tools (cf. [111, 124, 177]) consider mangling rules so that such passwords are guessed easily.

Passphrases

Instead of short, complex passwords, another proposal is to use longer passwords consisting of multiple words, so-called passphrases. Being longer can provide more security because an attacker needs longer to guess such passwords [135]. Moreover, using words allows users to generate passphrases with a meaning to them, thus being easier to memorize [135, 138].

However, passphrases and in general longer passwords negatively influence the ease of use of passwords. Users need longer to create such passwords and make more mistakes while entering them which yield to a higher number of login failures [133, 134, 201]. Login failures caused by typographical errors can be addressed by storing multiple variants of a passphrase [18, 161].

In practice, users select passphrases like *passwordpassword* [200], so that "passphrases are vulnerable to dictionary attacks like all schemes involving human choice" [39]. Using large dictionaries collected from the Internet (e.g. Wikipedia, news sites, and blogs) it is possible to successfully guess passphrases [184], even with up to 20 characters [206]. Only when including numbers and special characters also passphrases withstand a powerful attacker [200]. But, also this becomes obsolete with an increasing computational power of the attacker [200, 201]. Therefore, users must use very long and complex passphrases which are practically impossible to memorize. This makes it very likely that users again trade security for memorability and ease of use. Consequently, passphrases are also no solution for attack-resistant passwords.

Anecdotally, passphrases were already proposed by Kurzban in 1985 [142]. However, he insists that passphrases are generated by services due to security reasons.

Mnemonic phrase-based passwords

Similar to passphrases, there is the advice of using so-called mnemonic phrase-based passwords. A user choose a sentence and select the first character of each word. The resulting string is the password. For example, the sentence "Passwords are always stronger on the other side" is condensed into the password *Paasotos*. Mangling rules can also be applied to further strengthen the password for instance to *Paa50t05*.

The security of mnemonic phrase-based passwords highly depends on the fact that users choose a unique sentence [238]. But, in practice users select sentences from music lyrics, movies, and literature [141]. This makes such passwords easy to guess [137].

3.6.1.3 Tools

Despite the various approaches such as password meters and password-composition advices, user-chosen passwords remain vulnerable to guessing attacks. Security experts recommend to use password generators. They create random passwords that do not have patterns. Therefore, an attacker can only use time-consuming brute-force attacks to guess such passwords. Attacks using dictionaries and probabilistic models do not provide any advantage. Users are aware of the security benefit of random passwords [95]. But, similar to system-assigned passwords, passwords generated by password generators suffer from poor memorability [148].

Password generators exist as web (e.g. `random.org`) and stand-alone applications (e.g. [214]) as well as are an integral part of most password managers (e.g. [67, 144, 186]). They create passwords based on predefined password-composition rules, which specify the length and the included characters of the generated passwords.

The rules of common password generators are sub-optimal as we describe in Section 4.6.3. Generated passwords are not attack-resistant and often get rejected by services, because they do not fulfill the services' password requirements. The only solution for users is to manually look up the password requirements for each individual service and to configure the password generator accordingly. With respect to the wide diversity as well as the huge lack of password requirements that we showed in Section 3.4, this is a very cumbersome task. This inconvenience induces users not to employ password generators and rather stick to weak passwords.

One solution would be that all services use the same password requirements [9, 84, 207]. Because of the various password implementations and missing official standards, it is very unlikely that this will happen.

Moreover, there exist password generators creating passwords which are easy to remember for users (e.g. [67]). They do not include ambiguous characters (e.g. *Il*, an uppercase *i* and a lowercase *L*) or make the password easily pronounceable (e.g. *nenesotifexe*). Such schemes often based on the algorithm of Morrie Gasser [74], but such passwords are insecure [93].

To conclude, password generators are the only option to generate attack-resistant passwords because the generated passwords have no patterns. To this end, they are the best candidate to realize the first password task. But, existing generators are impractical because they do not take the password requirements of services into account.

PAS solves this problem by generating attack-resistant and valid passwords for users automatically (cf. Chapter 4).

3.6.2 Password preservation

In this section, we evaluate approaches to preserve passwords. We also describe two alternatives to address the memorization problem of passwords without preserving passwords. In Section 3.6.2.1 to 3.6.2.4, we evaluate the preservation-related problems: confidentiality, availability, recoverability, and accessibility.

Storing passwords

The most common solution to preserve passwords it to store them on user devices. A prominent example is a password manager. It saves passwords in a local database. While this seems to be a suitable approach, we describe in Section 3.6.2.1 that providing the confidentiality of stored passwords is very difficult.

Storing password generation data

Instead of storing passwords, they can be regenerated when needed. This can be done for instance based on service-related information, some information that users need to memorize, or some data stored on devices. A survey of existing approaches is available at [155].

Earlier schemes generate passwords by hashing a user-chosen master password and the URLs of services [105, 189]. As long as users know their master password they can regenerate their passwords at any time. But, an attacker who steal a password can perform an offline guessing attack to obtain the master password and then generate all passwords [150]. One solution is to use an attack-resistant seed for the password generation instead of a predictable user-chosen password. But, the seed must be stored because it is practically impossible to memorize it.

Furthermore, the approach of simply hashing a master password or a seed together with the URLs cannot be used to generate new passwords for accounts. This problem can be solved by adding an additional salt in the hashing for each account [105]. However, memorizing the salts is not possible for users so that also the salts must be stored on devices. Storing the seed and the salts leads to the confidentiality, availability, recoverability, and accessibility problem of this data. For instance, when the seed and the salts are not available on a device, a user cannot generate his passwords.

At first glance storing password generation data instead of passwords does not provide any benefits. But, it follows from the examination of the confidentiality, availability, recoverability, and accessibility problem in the next sections that this approach allows to solve these problems.

Remembering passwords

Various researchers have shown that users can memorize random passwords using repetition [38], cues [5, 29], and training videos [109]. However, in these studies users only needed to learn a single password. As such approaches are very time-consuming and sometimes cost weeks to learn a password [38], they are not applicable for a multitude of passwords.

Reusing passwords

Another approach is the reuse of passwords across accounts. Thus, users only need to memorize a small number of passwords. But, this approach raises a major security threat. In case an attacker obtains a password of a user, he gets access to all the other accounts where this password is used. This is called the *domino effect* of password reuse [129].

Despite this obvious security threat, there are researchers that recommend password reuse as a suitable coping strategy for the memorization problem of passwords instead of preserving them. Users should categorize their online accounts based on the importance and use proper passwords for high-value accounts and reuse passwords for low-value accounts [85, 243]. However, in practice users do the opposite. They often reuse the passwords of their valuable accounts and in general reuse passwords that they use often [17, 233]. The reuse of passwords also increases over time, because users have more and more accounts but do not create more passwords [95]. Even when users do not exactly reuse passwords, they often make only minor adjustments [66, 138, 202, 221, 242]. Although users know that reusing passwords is totally insecure, they justify their behavior as necessary because otherwise they need to memorize too many passwords [170, 219, 241].

3.6.2.1 Password confidentiality

We analyze proposals to ensure the confidentiality of preserved passwords in the section. It becomes apparent that the common approach of encrypting preserved passwords by a user-chosen master password does not properly protect them against an attacker.

The confidentiality of preserved passwords is realized by encryption. In case the encryption key is derived form a user-chosen master password there is the risk of efficient attacks (cf. Section 2.2.2). One mitigation approach is to use a password-hashing function such as PBKDF2 [130] to derive a proper encryption key. This makes an attack more time-consuming, however, it is still feasible. Another approach is to randomly generate the encryption key, store it along with the encrypted preserved passwords, and encrypt the key with a master password. However, this just

shifts the problem. In practice, the derivation of encryption keys is often poorly implemented so that efficient attacks are possible even when users use proper master passwords [21, 52, 245]. Besides guessing the master password, there is also the risk of other vulnerabilities such as in the storage format which allow an attacker to obtain the passwords without guessing the master password [94, 149, 204, 210].

Another mitigation approach to prevent the guessing of the master password are *decoy passwords* [32, 55]. When an attacker uses a wrong master password for decryption, he gets a set of plausible-looking decoy passwords. This forces him to verify the passwords online to distinguish between the decoy passwords and the real passwords of users. However, generating decoy passwords that are indistinguishable from user-chosen passwords is very difficult [97], if not impossible at all [175].

Moreover, there exist proposals that protect preserved passwords by making use of additional devices [160, 231]. However, such approaches require users to always carry an extra device. This is burdensome and bears the risk of loss.

In essence, protecting preserved passwords by a master password does not ensure the confidentiality of passwords. Assuming a master password with an appropriate securely level also just provide a first line of defense. In Chapter 5, we present the first secure solution for the confidentiality problem. We introduce a revocation mechanism to invalidate preserved passwords which prevents that the passwords can be stolen.

3.6.2.2 Password availability

In this section, we examine proposals to realize the availability of passwords on all user devices. It follows from this examination that manually copying preserved passwords between devices is not a practical solution. Storing preserved passwords online for synchronization leads to the confidentiality issues described in previous section.

Offline synchronization

Manually copying preserved passwords between user devices is not a practical solution. It is burdensome and time-consuming, because users must do this after any changes of their password portfolio. This happens frequently due to the continually increasing number of accounts and regular password changes.

Online synchronization

Another approach is to store preserved passwords at servers on the Internet to make them available on all devices. This is done by various password managers (cf. [67, 144]) but users can also use a simple cloud storage provider to realize this. However, storing preserved passwords on the Internet generally raises various security issues.

First and foremost, there is the risk of server compromise [145, 146, 174] in which an attacker steals the preserved passwords. This leads to the confidentiality issues discussed in Section 3.6.2.1. The attacker can perform a powerful offline attack or exploit security flaws in the storage format. Moreover, there is the risk of governmental access [61]. This includes that server operators are forced to hand over the preserved passwords to governmental agencies or even forced to implement backdoors that enable an access to the passwords. Note that if such backdoors become public they can also be misused by an attacker. Users take these risks very serious and worry about giving control of their passwords to an online service [131]. The security issues are very serious if the preserved passwords contains the passwords. This applies for the majority of the existing proposals such as password managers. The security of the preserved passwords then solely relies on the security level of the user-chosen master password. It is unlikely that such passwords withstand a large-scale offline brute-force attack.

In case the preservation of passwords is done by storing password generation data, this data can also be stored on servers to realize a synchronization of the preserved passwords. Like PAS (cf. Chapter 5), this is done by AutoPass [156]. Passwords are made available on all devices by generating them using some secret data that is stored on devices and some non-sensitive data that is stored on servers. The data on servers enable all devices to generate the same passwords as well as synchronize new passwords and password changes. But, the data on the servers does not allow an attacker to obtain the passwords. Nevertheless, even this approach has a security issue. There is the risk that a malicious server in cooperation with a malicious service determine the secret data stored on the user devices and thus obtain all passwords [156]. PAS is the first solution that is not vulnerable to this attack.

3.6.2.3 Password recoverability

We next evaluate approaches to create and maintain backups of preserved passwords. It turns out that placing backups at secure locations to protect them as well from unauthorized access and loss while keeping them up-to-date at the same time is an unsolved problem in practice.

Offline backup

Backups of preserved passwords can be stored on external hard drives or local network storage devices. This also allows an automatic update. But, they are threaten by malware [24, 136]. Therefore, it is recommended to store backups on read-only memories such as CDs and put them at secure, different, and physically isolated locations [78, 217]. For instance, at friends which also has the benefit that backups are protected from physical damage like fire.

However, this runs contrary to the requirement of keeping backups up-to-date. After any changes to the users' password portfolio backups must be updated because outdated backups are useless. However, regularly creating new backups and putting them at a friend's place is practically impossible. Having the preserved password on multiple devices also does not serve as a proper backup solution because of the serious threat of malware [79].

Online backup

Backups can also be stored on servers. This allows users to protect their backups from fire or burglars. However, when the backups contain the passwords this approach poses the same security issues as already discussed in Section 3.6.2.2. An attacker can get access to the servers, steal a backup, and finally obtain the passwords. As we explained in Section 3.6.2.1 the confidentiality of stolen preserved passwords can hardly be archived by a master password. In case the preservation of passwords is realized by generation data and the data is independent of passwords, online backups are suitable. This applies for PAS (cf. Chapter 5).

3.6.2.4 Password accessibility

In this section, we examine proposals to solve the password accessibility problem. It follows from this examination that existing proposals to make preserved passwords available to trusted persons in urgent or even emergency situations are impractical and raise new security issues.

Copy of preserved passwords

One approach is to place copies of the preserved passwords at trusted persons. However, this is not a practical solution because users need to keep the copies up-to-date. Moreover, the copies must be properly protected, for instance by a master password. This is necessary because the copies can be stolen from the trusted persons. Moreover, users might want to approve the access to their passwords. But, telling a very long and complex master password to trusted persons in urgent or emergency situations, for instance via phone, is error-prone.

Sharing account credentials

If the preserved passwords are stored online, users can tell trusted persons their credentials (i.e. username and password for the storage account). But, this approach has many drawbacks. First, telling the password via phone is error-prone. Second, trusted persons get access to all passwords. Third, this approach is not possible if the storage account is protected by a second factor (e.g. one-time-key, SMS on a mobile device, device-specific secret [4]). It also does not work if the preservation of passwords is realized by generation data. This requires additional secret data stored on user devices, but this data is not available to trusted persons.

Instead of sharing the credentials in case of emergency, users can provide them to trusted persons in advance, e.g. written on a piece of paper [4]. However, in this case users have no possibility to approve the emergency access to their preserved passwords. Moreover, there is the risk that the credentials get stolen from trusted persons and an attacker gets access to the preserved passwords, e.g. in case of a burglary.

Predefined emergency access

Another approach is to set up a predefined emergency access including a mechanism to approve the access. The approval can be realized by a *waiting period*, which can be chosen by users [67, 144]. If a trusted person requires emergency access, the access to the preserved passwords is granted after this period. During the waiting period, users can refuse the access. However, selecting an appropriate waiting period is impossible. For emergency purposes, a very short period is important, but then users might not be able to refuse an unauthorized emergency access. A typical situation would be an attacker misusing the emergency access of a trusted person. Against this background, a longer waiting period might be more appropriate but then an immediate access in urgent situations is impossible.

PAS allows to defined an emergency access to passwords and provides an approval by means of a PIN (cf. Chapter 6). This solves both issues. Trusted persons cannot access the passwords without the PIN and users can approve the access to their passwords immediately by handing over the PIN.

3.6.3 Password change

Users barely change their passwords [110, 212, 247], even after security breaches at services or exceptional events like the Heartbleed bug [62, 121, 179]. Against this background, some services enforce regular password changes. While this is common practice at companies, universities, and government services, banks and general services do not implement it [83, 84].

Despite the security benefits of reducing the time an attacker has to guess passwords and invalidating possibly compromised passwords (cf. Section 3.2.1), enforcing regular password changes fails to serve its purpose in practice if users need to memorize passwords. Users develop strategies for the generation of the new passwords which leads to patterns [242]. This makes it very easy to determine the new passwords from old ones.

Preserving passwords allows users also to use attack-resistant passwords when regularly changing passwords. But, changing passwords through the different password interfaces and procedures implemented by services is very challenging and cumbersome for users (cf. Section 3.5). One approach to solve the password change problem is to change passwords on behalf of users. This can be realized by an application running as a web browser extension [144, 157]. However, this is inconvenient and error-prone, because users cannot use the browser while changing passwords and might even interrupt the process by closing the browser. Moreover, it is not a suitable solution to change passwords of all accounts. It also does not work on mobile devices. Another realization is performing the password changes by a server [67, 68]. This enables a simultaneous change of all passwords but the server finally knows the passwords of the users which raises serious security issues and user concerns [131].

Another issue of existing approaches (e.g. [67, 144]) is the limited number of supported services. Because these approaches are proprietary, users have no option to use it with other services. Like our solution, the proposal by Mayer et al. [157] can be used with arbitrary services. But, it is also realized in the browser leading to the aforementioned issues.

Another drawback of all existing proposals is that users are left in charge of triggering password changes. This means they need to keep track of doing it regularly. Moreover, it does not ensure that passwords are immediately changed in case of a password compromise. It can take weeks until users get informed about a password breach at a service and change their passwords.

PAS is the first solution that solves the aforementioned issues (cf. Chapter 7). It works with arbitrary services, is independent of applications and platforms, and provides fully-autonomous password changes. PAS changes passwords on a regular basis with respect to the security level of passwords as well as immediately after it detects a compromise of users' passwords.

3.7 Realization of secure passwords by ubiquitous password assistance

The three password tasks of generating, preserving, and changing passwords enable users to use secure passwords for their accounts. But, in practice these tasks are so complex and extensive that it is practically impossible for users to perform them manually. Existing proposals often do not consider the conditions of services and therefore are quite useless for practical use. Moreover, they only cover some parts of the tasks, but to make secure passwords usable for users it is necessary to perform all of them. It follows from countless studies that users will only adapt a solution when it encompasses all aspects of passwords. If users are only able to partially use secure passwords they will continue using weak passwords.

In this thesis, we present the Password Assistance System (PAS). It enables users to use secure passwords for their online accounts. This is achieved by automating the three passwords tasks and providing comprehensive support. PAS consists of four parts:

1. *Password generation*: The password generation problem is solved by generating attack-resistant and valid passwords for users automatically. Users just need to provide the URL of a service in order to generate an optimal password for an account.

2. *Password synchronization*: The preservation, confidentiality, and availability problem is tackled by password synchronization scheme. It protects preserved passwords and makes them available on all user devices.

3. *Password backup*: The password recoverability and accessibility problem is solved by backups. Backups do not need to be updated even when passwords change, they can be revoked without physical access, and they provide a fully controllable emergency access.

4. *Password change*: The password change problem is solved by changing passwords automatically. Users neither need to create new passwords nor need to log in to their accounts. Passwords are changed on a regular basis with respect to the security level of passwords as well as immediately after compromise of users' passwords is detected.

PAS is the first solution that is capable of handling the different password implementations of services. A uniform description of password requirements, interfaces, and procedures enables it to cope with the various implementations. We describe the four parts of PAS in Chapter 4 to 7.

3.8 Conclusion

In this chapter, we detailed the problem of users to use secure passwords for their accounts. Secure passwords are indispensable to protect the multitude of personal data contained in accounts. Yet, users are not able to use secure passwords in practice.

We defined requirements and conditions for secure passwords. First, we defined security requirements for passwords. In essences, passwords must be (1) attack-resistant, (2) different for each account, and (3) changed on a regular basis and immediately after a compromise is detected. This prevents an attacker from guessing and misusing stolen passwords. Second, we detailed the conditions of services for passwords. We presented the largest ever conducted survey of password requirements and detailed their characteristics and security implications. Password requirements at Internet services are quite different, incompletely stated, and often missing at all. None of the 57,536 analyzed services enforce attack-resistant passwords and every fifth service even does not allow such passwords at all. Furthermore, we analyzed the password interfaces and procedures of services for the first time. It follows from this survey that here again users need to cope with very different password implementations. Third, we defined usage requirements which ensure a practical and usable solution as well as address conditions, concerns, needs of users.

Based on these requirements and conditions, we defined three tasks that enable users to realize secure passwords for their accounts: generating, preserving, and changing passwords. But, it is practically impossible for users to perform these tasks manually. It is therefore obvious that users are not able to use secure passwords for their accounts in practice. To verify this claim, we examined how users cope with the password tasks in practice. Based on an extensive survey of existing literature regarding users' behaviors and perceptions of passwords as well as proposed solutions it became apparent that the problem of using secure passwords is still not solved.

Finally, we outlined our solution: Password Assistance System. It enables users to realize secure passwords. This is achieved by automating the three passwords tasks and providing comprehensive support. We present the individual parts of PAS in the next chapters.

4 Automatic generation of attack-resistant and valid passwords

In Chapter 3, we have shown that the generation of passwords that are attack-resistant and compliant with the individual password requirements of services is very difficult for users.

In this chapter, we present the first part of PAS. It solves the password generation problem by automation. Brute-force-resistant and valid passwords are generated automatically by password assistants. Users just need to provide the URL of the service in order to generate an optimal password. PAS realizes this by making the services' requirements available to password assistants and providing optimal fallback password-composition rules for the case that no explicit requirements for a service are available. We provide a conceptual description of our solution in Section 4.1.

Then, we describe the four building blocks of our solution: First, Password Requirements Descriptions (PRD), a standardized description of password requirements that can be processed by password assistants (cf. Section 4.2). Second, the automatic generation of PRDs for the numerous services that already exist on the Internet (cf. Section 4.3). Third, the distribution of PRDs to make them available to password assistants (cf. Section 4.4). Fourth, optimal fallback password-composition rules for password assistants that can be used in the event that the password requirements for a service are partially or entirely not available (cf. Section 4.5).

We present a practical evaluation and implementation of our solution in Section 4.6. We present a large-scale generation of PRDs for 185,696 services. Furthermore, we complement our solution by password assistants that make use of PRDs. Moreover, we show that our optimal password-composition rule set is indeed a good fallback solution for services with no explicit password requirements. We conclude this chapter in Section 4.7.

The contributions of this chapter were published as parts of [H1, H4]. This chapter extends the published contributions by the development of a stand-alone PRD-based password assistant.

4.1 Conceptual description

This section provides a conceptual description of our solution which solves the password generation problem. We introduce the next generation of password management applications, a password assistant. It supports users with the first password task of generating passwords. The password assistant performs the task for users and generates attack-resistant and valid passwords automatically.

The entities involved in our solution are a user, a password assistant, and a repository for Password Requirements Descriptions (PRD). A PRD described the password requirements for services in a standardized way. We consider that the user needs to generate a password for a new account. The password assistant supports the user in this challenge and generates a password for him. The repository provides the PRD of the service for which the password is used. The general application flow of the password generation is illustrated in Figure 4.1 and briefly described in the following:

1. The user enters the URL of the service (e.g. `example.org`) into the password assistant for which he needs to generate a password.
2. The password assistant uses the URL to request the PRD of the service from the repository. In case the requirements of the service are partially or entirely not available, the assistant makes use of a set of fallback password-composition rules.
3. The password assistant generates an attack-resistant password compliant with the received password requirements and hands it over to the user.

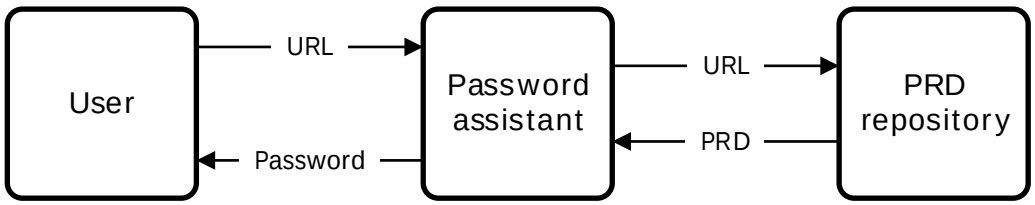


Figure 4.1: Data flow of the automatic password generation procedure.

We now introduce a solution to automatically generate attack-resistant and valid passwords. As illustrated in Figure 4.2 it consists of two components. First, a cryptographically secure Pseudo-random Generator (PRG) that generates a random value. Second, a Password Generator (PG) that derives a password from the random value in accordance with given password requirements. In Section 4.6.2 we describe the implementation of both components in detail.

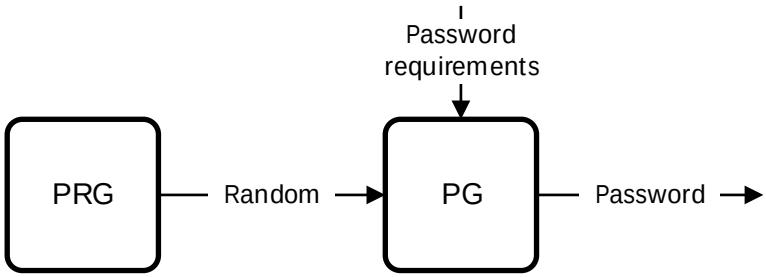


Figure 4.2: Password generation procedure.

Despite existing approaches, our concept is the first secure and usable solution for the password generation problem for the following reasons:

- Brute-force-resistant passwords are generated. If not supported by services, at least password with the best security level are generated.
- Valid passwords are generated by considering the individual requirements of services.
- Fallback password-composition rules are used if the password requirements of a service are incomplete or missing.

Altogether, generating optimal passwords is as secure, easy, and comfortable as possible. Users only need to provide the URL of the service. The realization of our solution consists of four building blocks which are presented in the following Section 4.2 to 4.5.

4.2 Uniform description of password requirements

In Section 3.4, it was shown that password requirements of services are very different and described in various ways. For instance, a maximum password length of 10 is expressed by “not more than 10”, “do not use 11 or more”, or “at most 10 characters”. In total, we found 1401 distinct password requirements sets. This makes it very problematic for password assistants to handle and consider the services’ password requirements during password generation.

In this section, we solve this problem by presenting a uniform description of password requirements. It allows to specify the many different password requirements of services in a standardized format. Such descriptions can be processed by password assistants in order to consider the individual password requirements of services during password generation.

In the following, we introduce our uniform description language and provide examples for standardized password requirements descriptions.

The Password Requirements Markup Language (PRML) specifies a framework for Password Requirements Descriptions (PRDs). A PRD is a standardized description of the password requirements of a service. It provides all information to automatically generate a valid password.

We conducted an analysis of password requirements for 200 representative services[1] in order to provide a comprehensive specification for PRDs. Based on the results, we identified common password requirements and specified PRML. As illustrated in Figure 4.3, a PRD consists of two parts: First, some metadata used to identify and manage PRDs properly. Second, the actual password requirements of services. In the following, we describe both parts and in particular how the individual password requirements of a service can be described by a PRD.

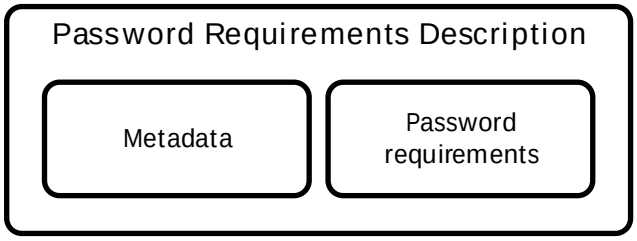


Figure 4.3: Structure of a PRD.

We encode PRDs in XML and provide a full XML Schema at [H26]. XML is well-specified and supported by many programming languages. This enables an easy integration of PRDs into password assistants. In practice, an XML-encoded PRD has a file size of 1 – 2 kilobytes.

A PRD is represented by a <prd> XML element. It has three attributes which form the metadata: `url`, `version`, and `prmlVersion`. The `url` specifies the URL of the service associated with the PRD. The `version` number allows password assistants to differentiate between multiple versions of a PRD and to update it if necessary. The `prmlVersion` specifies the version of PRML that is used to describe the PRD.

The password requirements and their structure, which can be specified by a PRD, are illustrated in Figure 4.4 and described in the following:

- *Character sets*: The <`characterSets`> element defines a list of allowed character sets. Each character set is described by a <`characterSet`> element which in turn defines a name (e.g. *numbers*) and the list of actual characters (e.g. *0123456789*).

[1] The Alexa Top 500 US list [8] reduced by websites with pornographic and illegal content, non-English websites, and websites that do not have or allow the creation of online accounts (e.g. banking websites). The list of services is available at [H26].

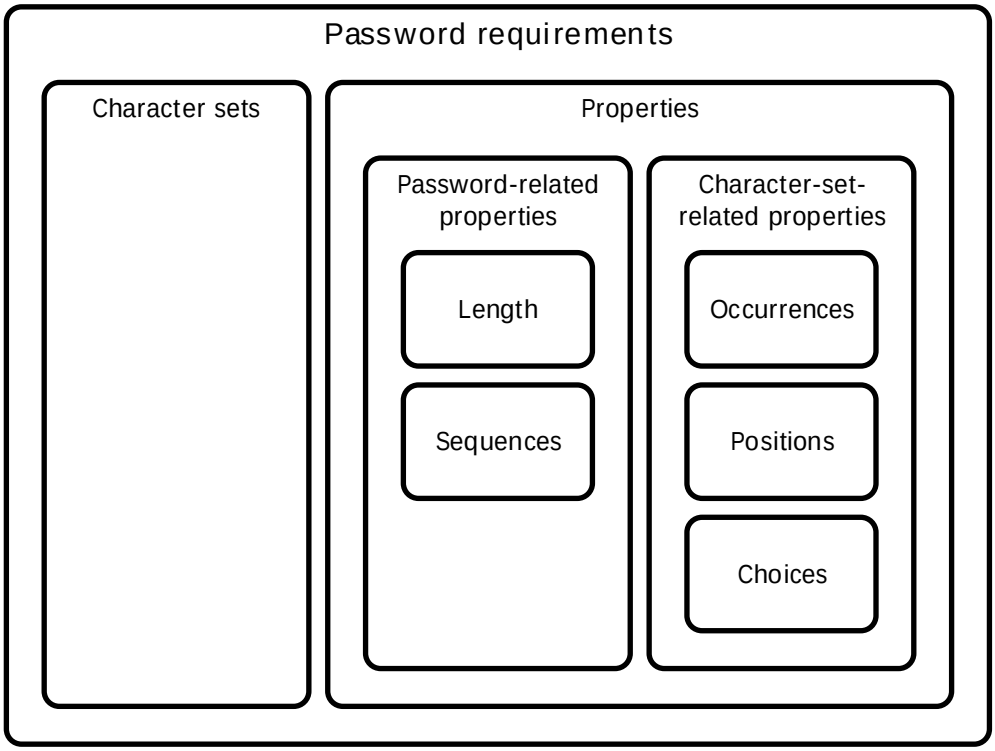


Figure 4.4: Structure of the password requirements definition.

- *Properties*: The <properties> element contains password-related and character-set-related properties. The former defines further restrictions for passwords as a whole, the latter further restrictions for the previously specified character sets. The restrictions for the character sets are defined within a <characterSettings> element and a reference to a character set specified in the <characterSets> element.

  Password-related properties:

  – *Minimum and maximum password length*: The minimum and maximum length of passwords are specified by a <minLength> and <maxLength> element.

  – *Sequences*: The <maxConsecutive> element specifies the number of allowed consecutive characters. It allows to prevent the usage of sequences like *1111* or *1234*.

  Character-set-related properties:

  – *Minimum and maximum occurrences*: The occurrences of characters from a certain character set is defined by a <characterSet> element. The element defines an attribute name which refers to a character set (as defined in the overall list of character sets, cf. <characterSets> element) as well as a minimum (<minOccurs>) and a maximum (<maxOccurs>) occurrence. This allows defining requirements such as “the password should contain at least one number and one special character”.

– *Positions*: A <positionRestriction> element is used to restrict the occurrences (minimum (<minOccurs>) and maximum (<maxOccurs>) occurrence) of a character set to a single or a range of positions (<positions>). Multiple <positionRestriction> elements must be combined by a logical *AND*. They can be used to express requirements such as “the password must start with a letter”.

– *Choices*: Choices of character sets are defined by a <requirementGroup> element which contains a list of <requirementRule> elements. Each rule refers to a character set and defines a minimum and maximum occurrence. Moreover, each <requirementRule> can be restricted to certain positions by a <positions> element. The number of rules that must be fulfilled is specified by the <minRules> element. The <requirementRule> elements are combined with a logical *OR*. A <requirementGroup> allows defining conditions such as “the password must contain at least two from the three character sets letters, numbers, and special characters”.

Note that a PRD does not contain a blacklist of passwords that are not accepted by a service or to be considered as insecure in general (e.g. *password*, *123456*, *qwerty*, and *letmein*). PRDs are intended for password assistants to generate random and attack-resistant passwords. They are not intended to assess or validate user-chosen passwords like password meters.

In the following, we provide two exemplary PRDs. One for a fictitious service and one for the existing service PayPal. Further 185,696 PRDs are available at [H26].

PRD for a fictitious service

We consider a fictitious service with the URL `example.com` for which the PRD is presented in Listing 4.1. The service accepts passwords consisting of letters, numbers, special characters, and spaces. This is represented by the corresponding <characterSet> elements within the <characterSets> list. As defined in the <characterSettings> element, passwords must contain letters, numbers, and special characters. Spaces may be used, which is represented by the <minOccurs>0</minOccurs> part. However, at least two numbers must be used, but at most one special character (cf. the <minOccurs> and <maxOccurs> elements, respectively).

Furthermore, the service restricts the first character of passwords. As specified by the <positionRestiction>, the first character of a password must be a letter. Moreover, the password should not have more than three consecutive identical characters. Finally, the minimum password length accepted by the service is 10 and the maximum is 20 characters, which is represented by the <minLength> and <maxLength> element, respectively.

```
<prd url="https://www.example.com" version="1.0" prmlVersion="1.0">
  <characterSets>
    <characterSet name="Letters">
      <characters>abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ</characters>
    </characterSet>
    <characterSet name="Numbers">
      <characters>0123456789</characters>
    </characterSet>
    <characterSet name="Specials">
      <characters>,;.:-_</characters>
    </characterSet>
    <characterSet name="Spaces">
      <characters> </characters>
    </characterSet>
  </characterSets>
  <properties>
    <characterSettings>
      <characterSet name="Letters"/>
      <characterSet name="Numbers">
        <minOccurs>2</minOccurs>
      </characterSet>
      <characterSet name="Specials">
        <maxOccurs>1</maxOccurs>
      </characterSet>
      <characterSet name="Spaces">
        <minOccurs>0</minOccurs>
      </characterSet>
      <positionRestriction characterSet="Letters">
        <positions>1</positions>
        <minOccurs>1</minOccurs>
      </positionRestriction>
    </characterSettings>
    <maxConsecutive>3</maxConsecutive>
    <minLength>10</minLength>
    <maxLength>20</maxLength>
  </properties>
</prd>
```

Listing 4.1: PRD of a fictitious service. The PRD defines the character sets letters, numbers, special characters, and spaces. Passwords must contain letters, numbers, and special characters. Spaces may be used. However, at least two numbers must be used, but at most one special character. Furthermore, the first character must be a letter. Passwords should not have more than three consecutive identical characters. The minimum password length is 10 and the maximum is 20 characters.

In addition to the fictitious example, we now present a PRD for an existing service showing how PRDs look in practice. In the following, we list the password requirements of PayPal and point out the corresponding elements in the PRD (cf. Listing 4.2). The requirements have been extracted from PayPal's sign-up form available at `https://www.paypal.com/signup`.

1. "8 characters or longer."
2. "Do not use more than 20 characters."
3. "Include at least 1 number or symbol (such as !@#$%^)."
4. "Do not use your email address."
5. "Please avoid 4 or more consecutive repeated characters (like 1111)."
6. "Please avoid consecutive numbers (like 1234 or 4321)."
7. "Please avoid key sequences (like qwer or rewq)."
8. "Please avoid key and number sequences (like qwer, rewq, 1234 and 4321)."
9. "Do not use any spaces."

The Requirement 1. and 2. are represented by the <minLength> and <maxLength> element, respectively. The Requirement 3. appears as the <requirementGroup> element in the <characterSettings> element, as part of the <properties> element. For each character set numbers and special characters (PayPal calls it symbols) there exists a <requirementRule> element. Both define a minimum occurrence of one character. The <minRule> element finally specifies that at least one of these two rules must be fulfilled. The Requirement 4. is not part of the PRD, because PRDs are intended for generating random passwords and therefore do not contain a blacklist. The Requirement 5. to 8. are represented by the <maxConsecutive> element. Finally, Requirement 9. is implemented such that spaces are not defined in the set of allowed character sets in the <characterSets> element.

```
<prd url="https://paypal.com" version="1.0" prmlVersion="1.0">
  <characterSets>
    <characterSet name="Letters">
      <characters>abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVQXYZ</characters>
    </characterSet>
    <characterSet name="Numbers">
      <characters>0123456789</characters>
    </characterSet>
    <characterSet name="Specials">
      <characters>.,:;-+*?%&amp;amp;=</characters>
    </characterSet>
  </characterSets>
  <properties>
    <characterSettings>
      <characterSet name="Letters"/>
      <requirementGroup>
        <minRules>1</minRules>
        <requirementRule characterSet="Numbers">
          <minOccurs>1</minOccurs>
        </requirementRule>
        <requirementRule characterSet="Specials">
          <minOccurs>1</minOccurs>
        </requirementRule>
      </requirementGroup>
    </characterSettings>
    <maxConsecutive>3</maxConsecutive>
    <minLength>8</minLength>
    <maxLength>20</maxLength>
  </properties>
</prd>
```

Listing 4.2: PRD of PayPal. Passwords can consist of letters, numbers, and special characters. Besides letters, at least one number or one special character must be used. A password should not have more than three consecutive identical characters. The minimum password length is 8 and the maximum is 20 characters.

4.3 Automatic generation of password requirements descriptions

In the previous section, we introduced Password Requirements Descriptions (PRDs) to solve the problem of the different descriptions and presentations of password requirements. To enable password assistants to use PRDs in practice, the PRDs for services are required. However, creating these PRDs manually for the huge number of existing services is not feasible.

In this section, we solve this problem by generating PRDs automatically. We present the Password Requirements Crawler (PRC), an application that automatically extracts the password requirements of a service from its website and translates them into a PRD. The password requirements are extracted from the sign-up forms of services. This is the common location where users are confronted with them while generating passwords for their online accounts.

Figure 4.5 illustrates the components of the PRC and the data flow of the automatic PRD generation. The PRC consists of two components and takes as input the service's sign-up form as an HTML document and outputs the PRD of the service. The Password Requirements Extractor (PRE) extracts password requirements and the Password Requirements Description Generator (PRDG) generates the PRD based on the PRE's output.

We describe later in Section 4.6.1 how we find the sign-up form on a service's website and download it as an HTML document in order to use it as input for the PRE. The subsequent Section 4.3.1 and 4.3.2 describe the PRE and the PRDG, respectively. Section 4.3.3 provides an evaluation of the PRC. Finally, in Section 4.3.4 we discuss limitations.

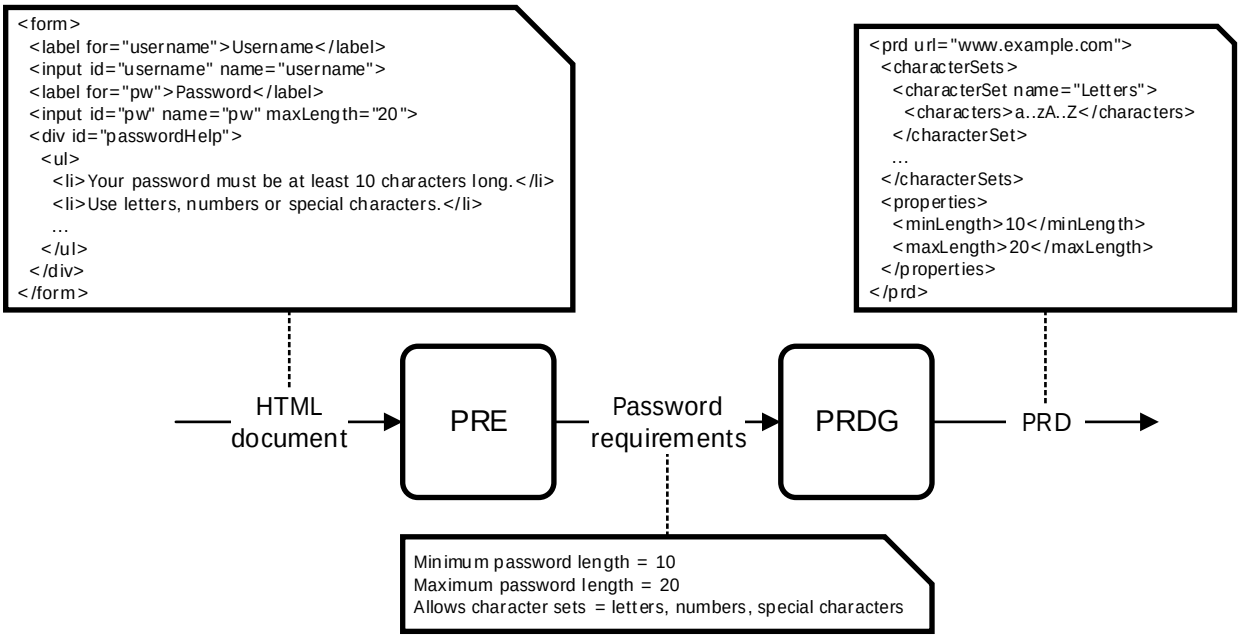


Figure 4.5: Data flow of the automatic PRD generation.

4.3.1 Extraction and interpretation of password requirements

The Password Requirements Extractor (PRE) extracts password requirements from an HTML document. It outputs the extracted requirements in a uniform description, which is then used by the PRDG to create the PRD of the service.

In practice, the description and presentation of password requirements are extremely diverse. Services describe their requirements in natural language instead of a structured format like PRDs. For instance, a maximum password length of 10 is expressed by "not more than 10", "do not use 11 or more", or "at most 10 characters". This diversity makes it extremely challenging to identify and extract password requirements from websites. For instance, regular expressions would be too narrow to cover all the different expressions. We apply *Natural Language Processing* technologies [48] to precisely identify and extract password requirements from websites.

Figure 4.6 provides an overview of the processing steps performed by the PRE. Password requirements are extracted in two ways. First, natural language processing using the Apache UIMA framework [13, 80] is used to analyze the textual content of the HTML document (dashed path). Second, information provided by the HTML document itself is analyzed by the HTML Metadata Extractor (solid path). Both results are finally combined. We describe the requirements extraction in the following.

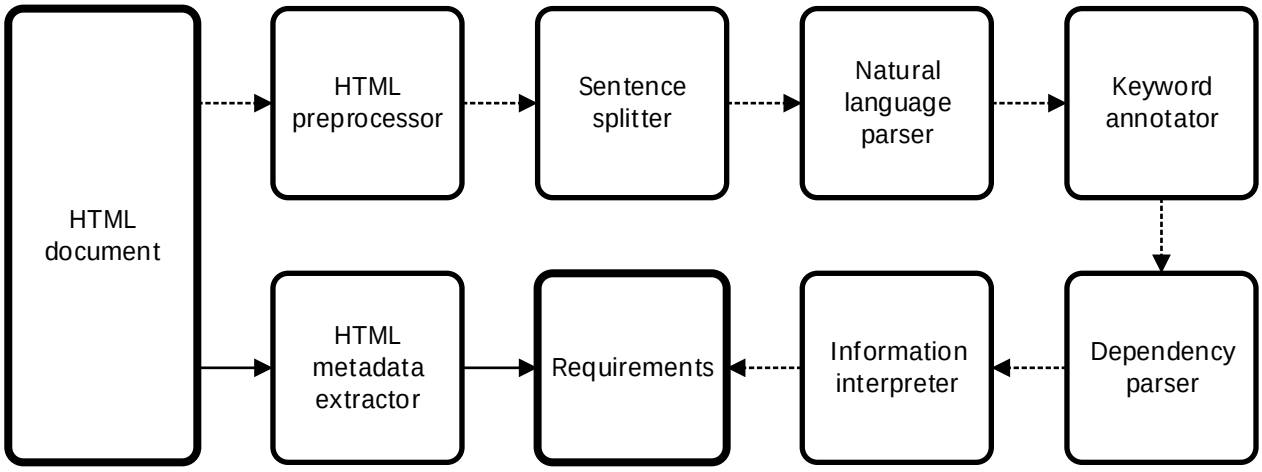


Figure 4.6: Execution steps of the password requirements extraction.

- *HTML preprocessor (HP)*: It removes content from the HTML document that in general does not contain password requirements, such as the header, images, and drop-down lists. Furthermore, it adds sentence delimiters to list items, text blocks, and paragraphs in order to improve the subsequent detection of sentences in the content. The HP outputs plain-text without any HTML markup.

- *Sentence splitter (SS)*: It takes as input the loose collection of words and punctuation marks from the HP and splits it into sentences. It detects beginnings and ends of sentences and annotates them with a delimiter.
- *Natural language parser (NLP)*: It determines the grammatical structure of each sentence. It detects groups of words that belong together and identifies the category of each word (verb, subject, or object). It outputs a dependency tree for each sentence, which describes the sentence's grammatical structure and the textual relations of its words.
- *Keyword annotator (KA)*: It searches for keywords to identify sentences that potentially contain password requirements. Relevant keywords were identified based on 20 representative websites and extended by linguistically related words (cf. [194, H4] for details). In essence, we use keywords related to character sets like *uppercase* and *numbers* and keywords focusing on password lengths such as *at least* and *more than*. The KA outputs a list of sentences containing these keywords.
- *Dependency parser (DP)*: It combines the results from the NLP and KA and puts both into a relation. For the example "use 6 to 12 characters" the dependency tree provides the information that *6* and *12* are numbers and the KA identified the word *characters* as a keyword. The DP now determines the relation between the numbers 6 and 12 and labels it as a number range.
- *Information interpreter (II)*: It finally takes all information into account and extracts the password requirements from the sentences. The example sentence "use not more than 16 characters" is interpreted as follows. The number *16* refers to the object *characters* and the keyword *more than* indicates that at least 16 characters must be used. Then, the negation *not* is considered and the sentence is interpreted as such that the maximum password length is 16. The II outputs the extracted requirements in a structured format. All the different expressions of password requirements used by services are now boiled down to a uniform format.
- *HTML metadata extractor (HME)*: Besides the textual analysis of the document, the HME analyzes the HTML source code to extract additional information about the password requirements. The HTML standard specifies an optional `minlength` and `maxlength` attribute for input fields. If present, the HME stores these values to the final list of extracted requirements as the minimum and maximum password length. If password lengths are found by both II and HME, the higher minimum length and lower maximum length are used.

4.3.2 Generation of password requirements descriptions

The Password Requirements Description Generator (PRDG) takes as input the requirements found by the PRE and generates a PRD. The requirements stated by services are often incomplete as we have shown in Section 3.4. A prominent example are special characters. Some services allow them or even require them, but they usually do not list them. To mitigate this problem the PRDG uses the following default rules for generating PRDs:

- *Missing character sets*: If a service does not name allowed character sets, the allowed character sets are set to letters and numbers.
- *Missing password lengths*: If a service does not specify a minimum and/or maximum password length, these values are also undefined in the PRD.
- *Missing special characters*: If a service allows or require special characters but does not name them, we define the following special characters in the PRD: . : , ; - + * ! ? % & =.

In Section 3.4.2.1 we provided a detailed justification for these design decisions. To close the gap of the undefined minimum and/or maximum password length, we provide optimal lengths with respect to acceptance rate and security level in Section 4.5.

4.3.3 Evaluation

In the following, we present an evaluation of the PRC showing that it achieves a ratio of 91.2% for a correct generation of PRDs. Moreover, we discuss implications for (1) the results of our large-scale survey of password requirements (cf. Section 3.4) and (2) password assistants using the PRDs created by our PRC (cf. Section 4.6.2).

We evaluated the PRC based on 250 services to verify that it extracts password requirements correctly. We used 200 websites selected from the Alexa Top 500 US list [8] and further 50 randomly selected websites with explicitly stated password requirements from the Alexa Top 1 Million list [7]. For this set, we manually looked up the sign-up forms and the password requirements stated there. The PRC was independently applied to the same set for automatic requirements extraction. Finally, we compared our manually extracted requirements with the automatically extracted ones. In case of 228 (91.2%) services the PRC exactly extracted the password requirements stated on the services' websites. For 8 (3.2%) services the PRC did not extract all requirements and for 14 (5.6%) services it extracted (parts of) them incorrectly. Table 4.1 provides details about the affected requirements. The data set used for the evaluation is available at [H26].

Password requirement	Not extracted	Incorrectly extracted
Minimum password length	5	2
Maximum password length	1	4
Character sets	4	6
Occurrences of characters	1	6

Table 4.1: Number of services for which password requirements were incompletely or incorrectly extracted by the PRC. Services may be counted multiple times.

4.3.3.1 Implications for our password requirements survey

The password requirements for our large-scale survey of password requirements presented in Section 3.4 were collected using the PRC. With respect to the outcome of the previously presented evaluation, we now discuss implications for the results of this survey.

The evaluation of the PRC shows that we can expect the vast majority of collected password requirements to be correct. Only a small fraction was incompletely or incorrectly extracted. Therefore, the concrete numbers for the application of password requirements (cf. Section 3.4.1), the security levels (cf. Section 3.4.2), and the acceptance rates of passwords (cf. Section 4.5) might not exactly reflect reality.

On the one hand, this can hardly be achieved because services may change their requirements at any time. On the other hand, our collected data provides such clear results that small deviations in the absolute numbers do not affect the key points of our contributions presented in Section 3.4 and 4.5.

The evaluation shows that the PRC did not extract the minimum password length for 5 of the 250 services. So, in practice there might be more services specifying a minimum password length than we presented in Section 3.4.1.1. In the two cases where the PRC selected an incorrect minimum length it extracted a higher length than specified by the services. To this end, we can expect that the distribution of the minimum password length as well as the minimum security levels are even a little bit lower in practice.

Regarding the maximum length the PRC failed only for one service. In case of the 4 services with an incorrect length, we found two cases with a lower and two with a higher length than specified by the services. Consequently, we expect only a very little deviation of our results regarding the maximum password length.

The cases of missing and incorrect character sets were dominated by special characters and spaces. This might be caused by a lack of proper training data for the PRC, because both are rarely used in practice. We found nearly the same number of cases where the PRC wrongly extracted an unstated character set as well as where it did not find an allowed one. To this end, we expect a very little deviation of our results regarding character sets. The major issue regarding the occurrences was that the PRC interpreted a minimum password length as a minimum required occurrence. For instance, the requirement "passwords must be 8 letters or more and contain at least 1 number" was interpreted as a minimum occurrence of 8 letters and a minimum occurrence of 1 number. Correct would be a minimum password length of 8. Thus, we can assume that in practice services specify minimum password lengths more often than we presented, but less minimum occurrences. This issue could be mitigated by an upper bound for minimum occurrences. However, because of the rare adoption of this requirement and thus the lack of suitable training data, we could not identify an appropriate upper bound.

4.3.3.2 Implications for password generation

We now discuss implications for password assistants that use PRDs created by our PRC. When password assistants use the PRDs of the 250 services as well as our optimal fallback password-composition rule set they generate valid passwords for 97.6% of the services.

As already mentioned, in the two cases where the PRC selected an incorrect minimum length it extracted a higher length than specified by the services (cf. Table 4.1). To this end, password assistants would still generate valid passwords. Regarding the maximum password length, we found two cases with a lower and two cases with a higher length than specified by the services. Consequently, at least in two cases password assistants would generate valid passwords.

In 5 of the 6 cases with incorrect character sets, password assistants would still generate valid passwords. The character sets that were wrongly extracted are actually accepted by the services, even though they were no explicitly stated as allowed. Regarding the character sets that were not found by the PRC only one service actually requires to use such a character set. As already mentioned, the major issue regarding the occurrences was that the PRC interpreted a minimum password length as a minimum required occurrence. Thus, also in this situation, password assistants would generate passwords compliant with the services' requirements.

Altogether, in case of the 14 services were the PRC extracted (parts of) the password requirements incorrectly, only for 4 services invalid passwords would be generated by password assistants. This sums up to a rate of 95.2% for valid password generation based on the information provided by the PRDs.

Besides the 14 cases with incorrectly extracted password requirements, our evaluation showed that in case of 8 services not all requirements were extracted. In Section 4.5, we present an optimal fallback password-composition rule set for password assistants that can be used in case the password requirements of a service are partially or entirely unavailable. This rule set specifies the character sets letters and numbers and a maximum password length of 22 characters. Using these rules, in case of the aforementioned 8 services with incomplete password requirements, password assistants would generate valid passwords for 6 of the 8 services. This sums up to an acceptance rate of 97.6%.

4.3.4 Limitations

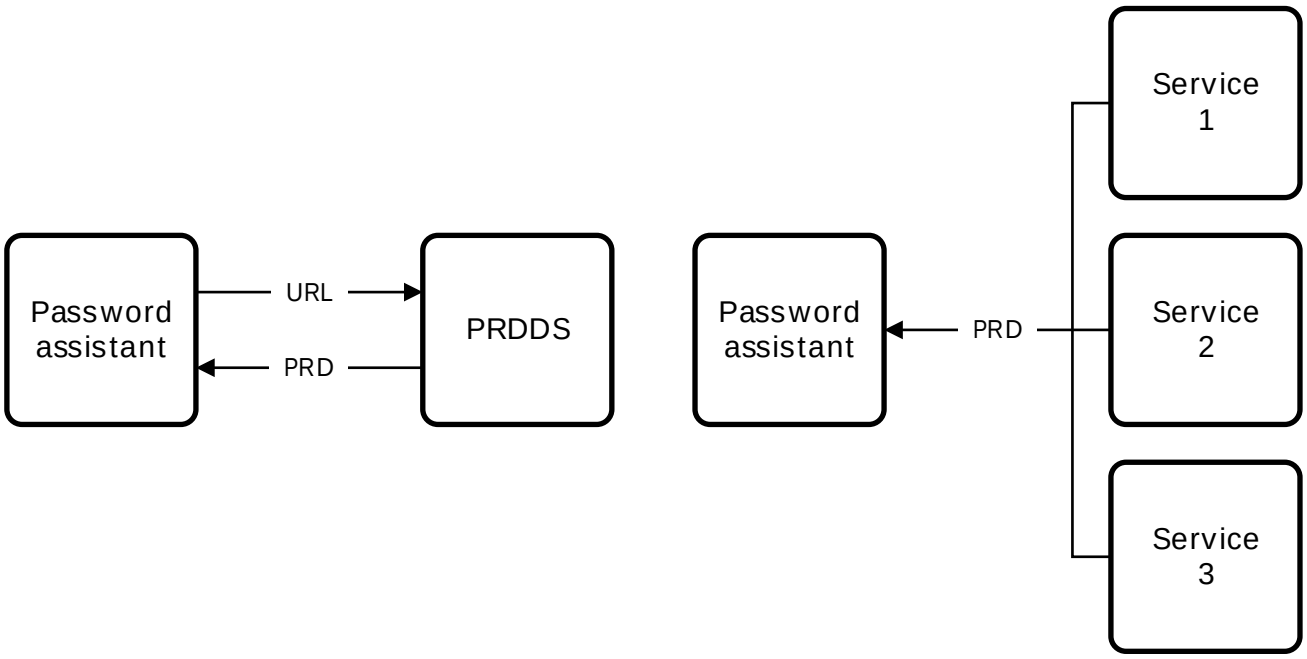
In this section, we discuss limitations of our PRC and their effects for our large-scale survey of password requirements presented in Section 3.4.

The PRC interprets password requirements stated in English. Although, this covers 52.7% of all websites on the Internet [227], it might have excluded the requirements of some services. As we explained in Section 4.3.1, the PRC extracts requirements from the textual content and the source code. While the textual extraction fails for foreign languages, the PRC is still capable of extracting the minimum and maximum password length from the attributes of the password input field. To this end, the limitation to English mainly affects our results regarding the character sets and occurrences of characters, that we have presented in Section 3.4.1.2 and 3.4.1.3. The PRC could be adapted for other languages, but this is out of scope of this thesis.

The PRC is capable of extracting the major password requirements: password length, character sets, and occurrences of characters. Further requirements such as position restrictions (e.g. “the password must start with a letter”) are not extracted. While ignoring such requirements might affect our calculations regrading the acceptance rate of passwords (cf. Section 4.5), the effect is very limited because such requirements are barely used in practice. For instance, the manually analyzed set of 250 services only contains four services with position restrictions. This very low application in practice was also found by Florêncio and Herley [83].

4.4 Distribution of password requirements descriptions

To use PRDs during password generation, they must be available to password assistants. In this section, we present two solutions to realize the distribution of PRDs. First, we present in Section 4.4.1 a central service that distributes PRDs (cf. Figure 4.7a). Second, we present in Section 4.4.2 an alternative decentralized approach in which services themselves provide their PRDs to password assistants (cf. Figure 4.7b).



(a) Service-independent centralized solution. **(b)** Privacy-preserving decentralized solution.

Figure 4.7: Distribution of PRDs.

4.4.1 Service-independent centralized solution

In order to distribute PRDs, we realize the Password Requirements Description Distribution Service (PRDDS), which is available at [H26]. It is developed as a Java Web Service and provides an interface through which password assistants can search and receive the PRDs of services (cf. Figure 4.7a). Searching and identifying PRDs are based on the URL specified by the `url` attribute in a PRD.

By making the PRDs available through our PRDDS, they must be created only once and can used by all Internet users and their password assistants. This allows an immediate deployment of PRDs for a large variety of services without demanding any efforts by services.

The communication between password assistants and the PRDDS is secured using TLS. This prevents manipulations of the PRDs as well as eavesdropping on the requests and thus user profiling by a third party attacker.

The PRDDS allows users to submit PRDs of unknown services. The submission system checks the plausibility of the specified requirements and rejects PRDs with obvious errors. The checks for instance verify that the minimum length must be smaller than the maximum password length, the sum of the maximum occurrences of characters must be smaller than the maximum password length, and any position restriction must be in range of the minimum and maximum password length.

To prevent an attacker from adding manipulated PRDs, a submitted PRD is also manually evaluated by the PRDDS provider. To address incorrect or outdated PRDs, the PRDDS also provides a feedback system that allows users to report issues with PRDs. Moreover, the PRDDS keeps a history and change log of each PRD.

The disadvantage of the centralized solution is that the PRDDS itself in principle learns the services used by users and could create user profiles. Well-known mitigations to this problem are connections over web proxies or VPN networks to anonymize the communication (cf. [12, 215]). An even stronger guarantee for anonymity can be achieved by using the TOR network [213] or our decentralized solution that we present in the next section.

4.4.2 Privacy-preserving decentralized solution

An alternative to the PRDDS and the aforementioned mitigation approaches is the provision of PRDs by the services themselves (cf. Figure 4.7b). However, this requires a uniform mechanism to obtain PRDs from services.

We propose to use the *well-known location* scheme as specified in the RFC 5785 [171]. The PRD of a service is then available at the URL `https://www.example.org/.well-known/prd.xml`. This enables password assistants to retrieve the PRD directly from each service in a unified way. Moreover, it removes the privacy issue and the requirement to trust in the correctness of the PRDs provided by the PRDDS.

On the negative side, such a decentralized distribution relies on the cooperation of services and would entail a presumably long transition time. This gap is bridged by our PRDDS. In practice, password assistants should first try to receive a PRD directly from a service. If unavailable, password assistants should try to fetch the PRD from the PRDDS. If also this fails, password assistants should use fallback password-composition rules for password generation, which we present in the next section.

4.5 Optimal fallback password-composition rules for password assistants

Our large-scale survey of password requirements presented in Section 3.4 shows three challenges while generating passwords: First, services have often different password requirements. Second, password requirements are often incompletely stated. Third, the majority of services do not state any requirements at all. In this section, we solve the incompleteness of password requirements by an optimal fallback password-composition rule set, which password assistants can use if the requirements for a service are partially or entirely unavailable.

The wide diversity of password requirements can be handled by using PRDs. This guarantees the acceptance of the passwords. Moreover, the results of our security analysis of the password requirements in Section 3.4.2 show that in this way attack-resistant passwords are theoretically possible for 78.4% of analyzed services. For the other 21.6% of the services with explicit requirements still the best possible passwords can be generated by using PRDs.

However, this solution cannot cope with the incompleteness of password requirements. As we have shown in Section 3.4.1 only 31% of the services explicitly state password requirements. For the majority of the services it is unknown how the passwords must look like. Furthermore, only 4017 (2.15%) services fully specify the essential information to generate valid passwords: lengths and allowed characters. Thus, for a practical solution fallback password-composition rules must be in place that password assistants can use if the password requirements for a service are partially or entirely unavailable.

Because for those services no data is available, we determine an optimal password-composition rule set based on the password requirements of the 57,536 services (cf. Section 3.4). Given that the implicit requirements have the same characteristics as the ones of services that provide their requirements, this optimal password-composition rule set is also optimal for services without explicit or partial requirements. We validate this assumption as well as the quality of our optimal rule set in the evaluation in Section 4.6.3.

In Section 4.5.1, we present a password-composition rule set that achieves the absolutely minimal rejection rate. We show that the definition of a single password-composition rule set that fits for all services, i.e. a single rule set that achieves an acceptance rate of 100%, is impossible.

In Section 4.5.2, we refine the former password-composition rule set into an optimal rule set that creates both attack-resistant and valid passwords with respect to the password requirements of services. We show that the security and acceptance rate of passwords are conflicting objectives and present a possible trade-off based on the variation of the password length.

4.5.1 Optimization of the acceptance rate

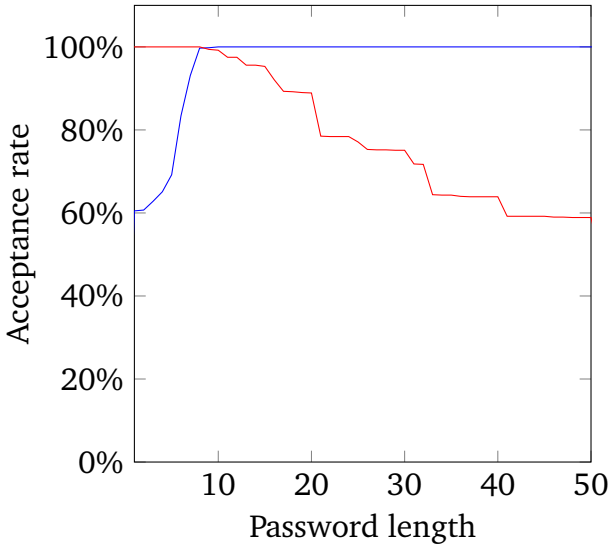
In order to develop a password-composition rule set which generates passwords in accordance with the services' password requirements, an optimal password length and character set must be found. We solve this optimization problem in two steps: First, we determine an optimal password length in Section 4.5.1.1. Second, we define an optimal character set for our password-composition rule set in Section 4.5.1.2.

The compliance to the password requirements of the services is measured by the acceptance rate of passwords created under a given password-composition rule set. Note that the evaluation is done under the assumptions presented in Section 3.4.2.1. In brief, if no minimum/maximum password length is specified we assume that services have no lower/upper limit and if no character sets are specified we assume that services allow letters and numbers.

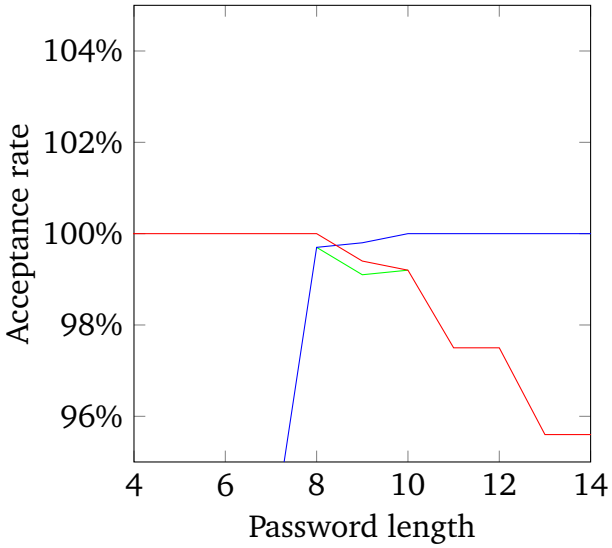
4.5.1.1 Optimal password length

We examined password lengths from 1 to 50 characters and counted the number of services that accept such a length. The results are illustrated in Figure 4.8a. The blue line represents the acceptance rate with respect to the minimum password length and the red line the maximum password length, respectively. The acceptance rate regarding the minimum password length is increasing until 10 characters. Whereas, the acceptance rate with respect to the maximum password length decreases constantly after 8 characters. The sharp increases and decreases of the acceptance rates correspond to the peaks in Figure 3.2 and 3.3 (Page 31 and 32).

Figure 4.8b highlights the acceptance rate around the intersection of both lines where the optimum is to be found. The green line represents the acceptance rate considering both, minimum and maximum password length, at the same time. This is decisive for the optimal password length. It shows the maximum at a password length of 8 characters with 99.7%, followed by 10 characters with an acceptance rate of 99.2%. Between 8 and 10 characters neither for the minimum nor for the maximum password length full acceptance is achieved simultaneously. This is caused by services where the minimum password lengths conflict with the maximum lengths of other services. As illustrated in Figure 3.2 (Page 31), there are 197 services with a minimum password length of 9 or 10 characters. At the same time, there are 370 services with a maximum length of 8 (cf. Figure 3.3, Page 32). Therefore, there exists no password length that is supported by all services.



(a) Acceptance rates for password lengths between 1 and 50 characters.



(b) Acceptance rates for password lengths between 4 and 14 characters.

Figure 4.8: Acceptance rates for different password lengths. The blue line represents the acceptance rate with respect to the minimum password length and the red line the maximum password length, respectively. The green line in (b) represents the acceptance rate considering both, minimum and maximum password length, at the same time. The highest acceptance rate of 99.7% is reached for a password length of 8 characters followed by 10 characters with 99.2%.

4.5.1.2 Optimal character set

We now determine an optimal character set for our password-composition rule set. This is done by counting the number of services accepting different combinations of character sets. We combine the character sets and determine the acceptance rate based on the numbers presented in Table 3.4, 3.5, and 3.6 (Page 33 and 34). The acceptance rates are listed in Table 4.2.

Following from this, the absolute maximum of the acceptance rate is reached for the password-composition rule set that specifies a password length of 8 characters and the character sets letters and numbers. This password-composition rule set has an overall acceptance rate of 97.6% and a security level of $S = L \cdot \log_2(C) = 8 \cdot \log_2(62) = 47$ bits (cf. Section 3.4.2.2).

The lower acceptance rates for only letters or only numbers compared to their combination, results from the number of services demanding both letters and numbers by the minimum occurrences requirement (cf. Table 3.5, Page 34). The combination of letters and numbers does also not achieve full acceptance because of the demand for special characters by 1,176 services. However, it marks the optimum under the given assumptions.

Combination of character sets	Acceptance rate
Letters	88.11%
Numbers	92.00%
Letters, numbers	97.96%
Letters, numbers, specials	6.22%
Letters, numbers, specials, spaces	0.64%

Table 4.2: Acceptance rate of combined character sets.

To this end, there exists no single password-composition rule set that creates valid passwords for all services. There is neither a password length nor a (set of) character set(s) that is accepted by all services that we analyzed in our survey presented in Section 3.4. This shows that our approach of service-specific PRDs, which we presented in Section 4.2, is necessary.

4.5.2 Optimization of the acceptance rate under the condition of 128-bit security

We showed that there exists no single password-composition rule set to generate valid passwords for all services. The aforementioned rule set with an optimal acceptance rate creates insecure passwords. More precise, passwords with a security level of 47 bits which can be brute-forced on a single day using standard hardware. We now refine this insecure password-composition rule set under the restriction that the generated passwords resist offline brute-force attacks to achieve both attack-resistant and valid passwords under the conditions of the given password requirements of the services.

In Section 4.5.1.2 the conjunction of letters and numbers was identified as the optimal character set. Using only letters or numbers decreases the security level compared to the combined set for a fixed password length. Increasing the character set by special characters leads to a drop of the acceptance rate which cannot be compensated by a potential decrease of the password length. Therefore, the conjunction of letters and numbers is also optimal when additionally taking into account the resulting security level. As a consequence, the only remaining parameter that can be varied to steer the resulting security level is the password length.

Under these conditions, we can calculate the necessary password length to realize the desired security level of 128 bit providing resistance against offline brute-force attacks. With the metric presented in Section 3.4.2.2 the required password length is given by $L = \left\lceil \frac{128}{\log_2(62)} \right\rceil = 22$ characters which results in a security level of 130 bits.

In order to determine the related acceptance rate of the refined password-composition rule set we counted the services accepting letters and numbers as well as a password length of 22 characters. We assumed that services accept such passwords when they do not specify password lengths, character sets, or occurrences of certain characters (cf. Section 3.4.2.1). As a result 44,099 of the 57,536 services accept passwords created under our password-composition rule set which implies an acceptance rate of 76.6%.

In Figure 4.9, we illustrate the security level for the character set comprising letters and numbers depending on the password length. Additionally, the figure depicts the acceptance rate for this character set also depending on the password length. The figure shows, that the two objectives (1) increasing the acceptance rate and (2) increasing the security level are contradicting for password lengths greater than 8 characters. Moreover, Figure 4.9 shows a possible trade-off between these objectives which can be realized by varying the password length.

To conclude, our optimal password-composition rule set specifies the character sets letters and numbers and a password length of 22 characters. It provides a security level of 130 bits and an acceptance rate of 76.6%.

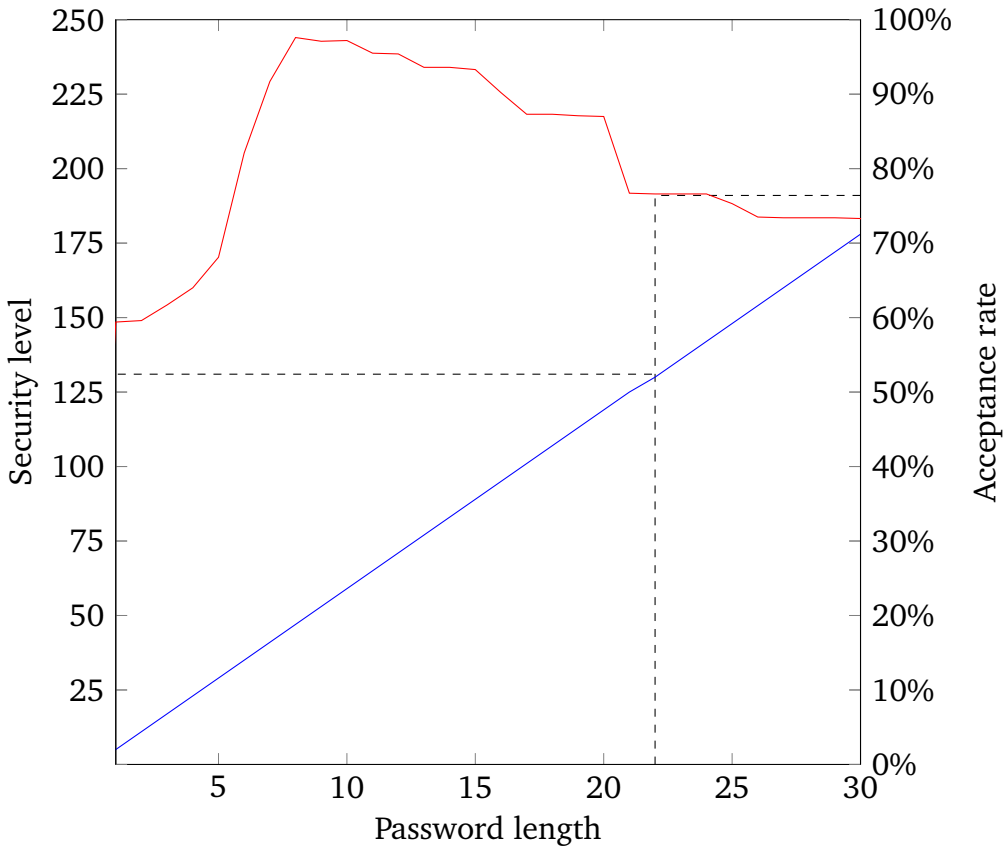


Figure 4.9: Security level (blue line) and acceptance rate (red line) depending on the password length for the combined character set letters and numbers. The aimed security level of 128 bits is achieved for 22 characters with an acceptance rate of 76.6%.

4.6 Implementation and practical evaluation

In this section, we present an evaluation of our solution. We show that our concept of automatically generating attack-resistant and valid passwords is applicable in practice.

We start in Section 4.6.1 by showing that our PRC, as presented in Section 4.3, can be used to automatically generate PRDs on a large scale. We used the PRC to create PRDs for 185,696 services. This huge set served as the data basis for our survey on password requirements (cf. Section 3.4) and lays the foundation for a practical application of PRD-based password assistants. Such password assistants that make use of PRDs are presented in Section 4.6.2. We present stand-alone password assistants as well as the integration of PRDs into an already existing password generator. Finally, we evaluate in Section 4.6.3 the application and impact of our optimized password-composition rule set, which was presented in Section 4.5.

4.6.1 Large-scale creation of password requirements descriptions

In the previous sections, we built the foundation for password assistants that are capable of generating attack-resistant and valid passwords automatically. However, to actually solve the password generation problem in practice, the PRDs for existing services are required. In this section, we present a large-scale generation of PRDs for 185,696 services.

Because the PRC requires the sign-up form of a service as an HTML document to generate a PRD, we developed the Sign-up Form Crawler (SFC). It takes as input the root URL of a service (e.g. `example.com`) and, if it exists and is found, downloads and stores the service's sign-up form as an HTML document. This is then used as input for the PRC to generate the PRD.

In Section 4.6.1.1, we explain how the SFC finds the sign-up form of a service. In Section 4.6.1.2, we then describe how we selected a sample of services and finally we discuss in Section 4.6.1.3 limitations of our approach.

4.6.1.1 Finding password requirements

Naturally, our survey is limited to services where users can choose their passwords. Thus, we ignore services with system-assigned passwords such as online banking services. The SFC identifies relevant services, where users can choose passwords, by checking for the presence of a sign-up form on the service's website. It takes as input the URL of a service and, if found, downloads the sign-up form and stores it as an HTML document.

The procedure to find a sign-up form is the following: The SFC takes as input the URL of the service (e.g. `example.com`). It browses the service's website and searches the available links on the root page for the keywords *registration*, *sign up*, *create account*, and *join* and checks if they lead to a sign-up form. After following two links in sequence without detecting a sign-up from, the search is aborted and it is assumed that the service has none. To identify a sign-up form and filter out login and newsletter forms, the SFC checks the HTML `<input>` elements. A sign-up form usually contains at least an input field for the password and additional ones for user-related data, such as name, email, or address. The SFC requires a minimum of three input fields (one password and two additional ones) to accept a form as a sign-up form. The employed search strategy was developed based on a manual analysis of 200 websites[2].

4.6.1.2 Selecting a sample of services

We collected 3,999,883 URLs from four data sets provided by the ranking companies Alexa, Majestic, and Quantcast. These sets contain the top US and global websites and after removing duplicates consist of 3,236,319 unique URLs (cf. Table 4.3). It is important to note that the number of unique URLs is not equal to the number of services, which is indeed much smaller. For instance, the list of unique URLs contains 68 URLs with `google.com.*` (e.g., `google.com.co`, `google.com.fr`) which all point to Google and thus must be considered as the same service.

Dataset	Date	# URLs	# Unique URLs
Alexa [7]	2015	1,000,000	999,959
Majestic [154]	2016	1,000,000	704,523
Quantcast [183]	2015	999,882	854,131
Quantcast [183]	2011	1,000,001	677,706
		3,999,883	3,236,319

Table 4.3: Data sets of URLs.

After removing duplicates, we checked the availability of the 3,236,319 URLs. It turned out that about 20% of the URLs were not available and were therefore ignored after performing several connection attempts. For the remaining URLs we used our SFC to find the sign-up forms, as described in the previous Section 4.6.1.1.

[2] The Alexa Top 500 US list [8] reduced by websites with pornographic and illegal content, non-English websites, and websites that do not have or allow the creation of online accounts (e.g. banking websites). The list of services is available at [H26].

For about 75% of the URLs no sign-up form could be detected. In comparison to a manual analysis of the Alexa Top 500 websites [8], where 60% have no online accounts, and the fact that the set contains many old and static websites this is plausible. For 185,696 services the sign-up form was found and downloaded. The stored HTML documents were then used as input for our PRC to create the PRDs of the services, which was described in detail in Section 4.3. The processing of the 3,236,319 URLs took approximately 3 months running 4 computers in parallel and produced more than 3 TB of traffic.

4.6.1.3 Limitations

We only considered the sign-up forms and not services' help or FAQ pages to find password requirements. This might have prevented requirements of some services from being found. However, this limitation was necessary: First, it filters out password requirements that are mentioned in articles or news and do not represent the requirements of an actual service (e.g. an article about selecting appropriate password requirements). Second, the sign-up form is the most likely place to find requirements, because here users need to select passwords. Compared to searching the entire websites of services this provides a much better performance and facilitates the analysis of a huge sample. In our analysis of the 250 websites[3], we only found a single service which provided more detailed password requirements on its help page.

Furthermore, the SFC only captures password requirements that are available before submitting the sign-up form. The SFC selects the password input field before downloading the form in order to capture password requirements that are not written in the initial sign-up form and loaded by a separated AJAX request. However, the sign-up form is not submitted with random passwords to get error messages that might contain password requirements. This approach would require to submit the form many times with different passwords because we do not know which passwords are invalid. A proper sample would consist of more than 5000 passwords to cover a reasonable number of different password lengths (cf. Figure 3.2 and 3.3, Page 31 and 32) and character sets (cf. Table 4.2, Page 84). While such an extension of the SFC might reveal more password requirements, we did not implement it, because it causes heavy load on the services' infrastructure and leads to unforeseeable consequences.

[3] 200 websites selected from the Alexa Top 500 US list [8] and further 50 randomly selected websites with explicitly stated password requirements from the Alexa Top 1 Million list [7]. The list is available at [H26].

4.6.2 Usage of password requirements descriptions in password assistants

In this section, we present password assistants that make use of PRDs and the optimal fallback password-composition rule set. These password assistants are the place where the previously presented building blocks are put together and our solution comes alive. It is also the place where the password generation problem is actually solved by PAS in the users' daily life.

We start in Section 4.6.2.1 with presenting our PRD-based password assistant realized as a web and mobile application. In Section 4.6.2.2 we demonstrate that PRDs can also be integrated into existing password generators.

4.6.2.1 PRD-based password assistant

We present the development of a PRD-based password assistant in this section. We implemented our solution for password generation, consisting of a Pseudo-random Generator (PRG) and a Password Generator (PG), that we introduced in Section 4.1 in JavaScript. Along with a connector for the PRDDS and a parser for PRDs, this builds the code base for the deployment of our password assistant on multiple platforms.

We used the Angular framework [98] to build a web application and the NativeScript framework [181] to build mobile applications for Android and iOS. Our implementation is available at [H26]. In the following, we explain the realization of the PRG and the PG in detail.

Pseudo-random Generator

The PRG generates a random value with a desired number of bits. We use the PRG provided by the Web Cryptography API [236]. Other programming languages provide similar PRGs, e.g., the class `SecureRandom` in case of Java. In the situation that the platform-provided PRG is not able to provide the desired number of bits, the PRG can be called multiple times and the output can be concatenated to a single bitstring of the desired length.

Password Generator

The PG maps the random value generated by the PRG to a password which complies with requirements defined by a PRD. This is done in three steps:

1. The PG calculates the required number of bits N that the PRG needs to produce. This is done by $N = L \cdot \lceil \log_2(C) \rceil + B$, where L is the desired password length, C the cardinality of the allowed character set, and B the security buffer. We choose a security buffer of at least 128 bits and calculate it as $B = 128 + L - (128 \bmod L)$.

   Using at least 128 bits more than required guarantees a negligibly biased distribution over the character set. More specifically, for each value v in the character set, the probability that v is chosen deviates from $\frac{1}{C}$ at most by 2^{-128}. Please note that adding fewer additional bits (e.g. 20 bits) recognizably increases the probability of biased passwords.

   We consider a PRD that specifies a password length of 22 characters and a character set of letters and numbers, like our optimal password-composition rule set (cf. Section 4.5). The cardinality is $C = 62$, the password length is $L = 22$, and the security buffer $B = 128 + 22 - (128 \bmod 22) = 128 + 22 - 18 = 132$. The number of bits that the PRG must produce is $N = 22 \cdot \lceil \log_2(62) \rceil + 132 = 22 \cdot 6 + 132 = 264$.

2. The random value is mapped to a sequence of characters. This works as follows: The random value is considered as a bit string and divided into L blocks of N/L bits. The bits of each block are converted to a positive integer X and then reduced to a number between 0 and $C - 1$ using the modulo operator. The final number X then serves as an index in the character set. For instance, $0 = a$ and $51 = Z$ in our example character set. The resulting characters of each block are concatenated and forms the generated passwords.

3. The password requirements are verified. For instance, the given PRD specifies that a password must consist of at least one number. To generate valid passwords while ensuring that passwords are uniformly distributed, we use rejection sampling (cf. Section 2.1.1).

   In brief, the PG checks if the generated password fulfills all requirements. If not, the password is discarded and the PG start again with Step 1 and obtains a new pseudo-random value from the PRG. A password is generated according to Step 2 and verified. This process is repeated until the password fulfills all requirements stated by the PRD.

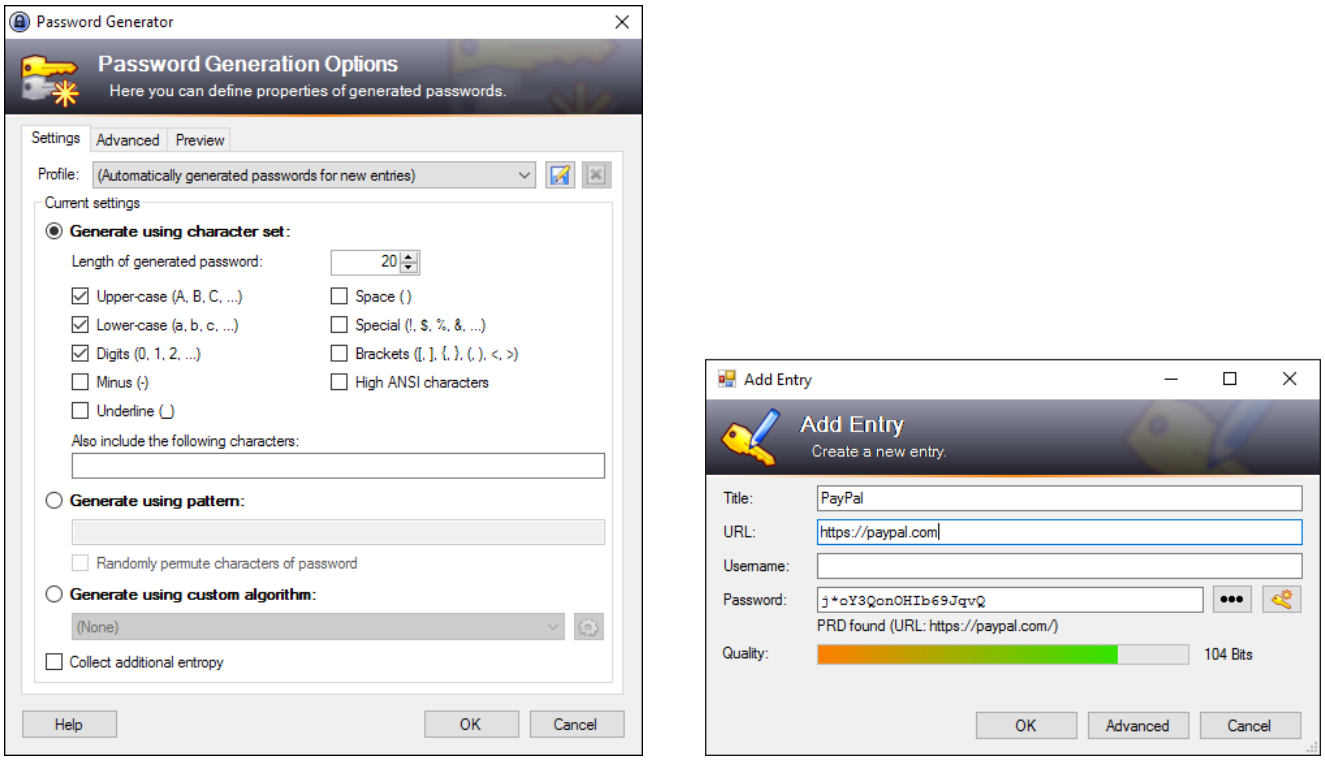
   Please note that a straight-forward approach of considering requirements during the password generation (e.g. generate $i - 1$ characters, check if they contain a number, if not select a digit for the i-th character) would produce biased passwords.

   For the event that the given PRD is incomplete, the PG makes use of our optimal fallback password-composition rule set. Moreover, for very large maximum password lengths like 100, the PG automatically reduces the maximum length to generate passwords with at least 128-bit security level but with the least needed characters. The reduced password length is determined by $L = \left\lceil \frac{128}{\log_2(C)} \right\rceil$.

4.6.2.2 KeePass extension

To demonstrate the usage of PRDs by existing password generators we implemented an extension for KeePass. KeePass [186] is an open-source password manager. It provides an extension framework that allows third-party developers to enhance it with additional functionality, such as our PRDs for automatic generation of attack-resistant and valid passwords.

Our extension allows users to generate attack-resistant and valid passwords by just entering the URL of the service. After entering the URL, our extension retrieves the corresponding PRD from our PRDDS, configures the internal KeePass' password generator accordingly, and generates a password. Users neither need to find out the password requirement nor configure the KeePass' password generator manually (cf. [193, H4]).



(a) Original password generator of KeePass.

(b) PRD-based password assistant for KeePass.

Figure 4.10: Password generation with KeePass.

Figure 4.10 contrasts the original way to generate passwords with KeePass and the novel one provided by our extension. As shown in Figure 4.10a users need to cope with a multitude of different input fields and checkboxes to generate a password of certain characteristics. However, to do so, users first need to find out the password requirements of the service. In case of our extension, as depicted in Figure 4.10b, users just need to enter the URL of the service.

4.6.3 Application and impact of optimized fallback password-composition rules

In Section 4.5, we presented an optimal password-composition rule set that generates passwords with a security level of 130 bits and an acceptance rate of 76.6%. We now evaluate the quality of this rule set. First, in Section 4.6.3.1 we show that our rule set is a good fallback solution for services not stating requirements. Second, in Section 4.6.3.2 we demonstrate that our rule set significantly improves the currently implemented rules of common password generators.

4.6.3.1 Application to services without explicit requirements

For the development of our optimal password-composition rule set in Section 4.5, we assumed that the available password requirements are a good estimator for services that do not state any requirements. To verify this assumption, we evaluated our optimal rule set on 100 randomly chosen services that do not state any password requirements.

We manually tested, whether the services accept a password generated under the optimized password-composition rule set. The acceptance rate turned out to be 80%, which is very close to the estimated acceptance rate of 76.6% (cf. Section 4.5.2). The only found reason for password rejection were length constraints, which as well is as expected. The maximum lengths—in some cases found in error messages—appeared to exclusively be a choice of 10, 12, 15, 16 or 20. So, our 22-character password that we used for the evaluation was just too long. These length constraints again exactly fit to the peaks we found in explicit password requirements as shown in Figure 3.3 (Page 32). The data set used for the evaluation is available at [H26].

Altogether this confirms our assumption and subsequently that our optimized password-composition rule set is a good fallback solution for password assistants when no explicit or only partial password requirements are available. Moreover, we tested our optimized password-composition rule set for the services which we analyzed regarding their password procedures and interfaces for changing a password (cf. Section 3.5). In Table 3.9 (Page 46) we showed that 6 of the 10 services do not state any password requirements at the password change form. Our optimized password-composition rule set generates valid passwords for all of them.

4.6.3.2 Impact on password generators

In this section, we show that our optimal password-composition rule set significantly improves the currently implemented rules of common password generators. We analyzed the password-composition rules of password generators and compared them to our optimized rule set.

Table 4.4 lists 15 password generators, their default password-composition rules as well as the resulting security level and acceptance rate. We calculated the security level based on the metric described in Section 3.4.2.2. The acceptance rate is determined based on the results described in Section 4.5.1. Astonishingly, we observed the same wide diversity of password-composition rule sets as with the password requirements of services on the Internet.

Password Generator	Password length	Character sets	Security level	Acceptance rate
http://passwordsgenerator.net	16	L, N, SP	101 bits	5.65%
http://www.freepasswordgenerator.com	10	L, N	59 bits	97.18%
https://identitysafe.norton.com/password-generator	12	L, N, SP, S	76 bits	0.59%
https://lastpass.com/generatepassword.php	12	L, N, SP	76 bits	6.10%
https://strongpasswordgenerator.com	15	L, N, SP, S	95 bits	0.50%
https://www.dashlane.com/password-generator	12	L, N, SP	76 bits	6.10%
https://www.passwort-generator.com	8	L, N, SP	50 bits	6.15%
https://www.random.org/passwords/	8	L, N	47 bits	97.64%
https://www.safepasswd.com	10	L, N, SP	63 bits	6.19%
1Password (https://1password.com/)	16	L, N	95 bits	90.15%
KeePass (http://keepass.info)	20	L, N	119 bits	87.01%
Password Gorilla (https://github.com/zdia/gorilla)	8	L, N, SP	50 bits	6.15%
Password Safe (https://pwsafe.org)	12	L, N	71 bits	95.43%
PWGen (http://pwgen-win.sourceforge.net)	12	L, N	71 bits	95.43%
RoboForm Password Generator (http://www.roboform.com)	8	L, N	47 bits	97.64%
Optimized password-composition rules	22	L, N	130 bits	76.65%

Table 4.4: Default password-composition rules of common passwords generators and their related security level and acceptance rate.

The security level of the generated passwords is by far too low to resist offline brute-force attacks in most cases. Only KeePass with 119 bits almost reaches the desired security level of 128 bits. With 50 bits or less, several password generators have security levels which, as already discussed, can easily be brute forced.

A further problem can be seen in the low acceptance rates resulting from the inclusion of special characters and spaces by default. This leads to many rejected passwords. Currently, the only solution for users is to look up the password requirements of each service manually and configure a password generator accordingly. This is very inconvenient for users or might even not be possible at all. As we showed in Section 3.4.1, only one third of the services actually state their password requirements. Moreover, this process is error-prone. Even when services allow special characters and users configure their password generator accordingly, it is likely that the generated password gets rejected because of the missing consent on special characters (cf. Section 3.4.2.1). Under these circumstances users can only reconfigure the password generator repeatedly in a trial-and-error approach until they finally succeed in generating a password that is accepted by the service.

So how can password generators benefit directly from our findings? First, half of the listed generators can significantly increase both, acceptance rate and security by simply changing their standard settings to our proposed password-composition rules. A second improvement that can directly be derived from our findings is the simplification of the user interface of password generators. Instead of presenting a multitude of configuration settings (cf. Figure 4.10a, Page 91), for which the effects are unclear to most of the users, a simple slider could be presented which allows to make use of the security-acceptance-rate trade-off discussed in Section 4.5.2. Such a slider easily covers the standard rules of the second half of the password generators. Adding one single check box to include special characters appears to be sufficient to cover nearly 100% of the services password requirements.

4.7 Conclusion

In this chapter, we solved the password generation problem. PAS enables password assistants to fully automate the generation of passwords by creating attack-resistant and valid passwords for users automatically. So far, users have struggled with the generation of passwords. They had to manually find out the password requirements of each service and configure a password generator accordingly or adjust generated passwords until they finally got accepted. Our solution only prompts for the URL of a service to generate an attack-resistant and valid password. PAS thereby makes the generation of optimal passwords for online accounts as secure, easy, and comfortable as possible for users.

We presented PRML which is intended to specify PRDs for Internet services. A PRD uniformly describes service's password requirements and makes them thereby available to password assistants. We have shown that this solution is necessary because there exist no single password-composition rule set that fits to all services. PRD-based password assistants generate the best-possible passwords and guarantee their acceptance. Moreover, we presented an optimal fallback password-composition rule set. It solves the problem that the password requirements of the majority of services on the Internet are partially or entirely unavailable. Our rule set generates attack-resistant passwords with an estimate acceptance rate of 76.6%. In an evaluation of 100 services even an acceptance rate of 80% was reached.

The practicality of our solution and its capability to solve the password generation problem have been extensively demonstrated. We presented the PRC which creates PRDs automatically by extracting the password requirements form the services' websites. Our evaluation shows, that the PRC achieves a rate of 91.2% for the correct generation of PRDs. In conjunction with our optimal fallback password-composition rule set, valid passwords for even 97.6% of 250 evaluated services are generated.

The PRC was used to generate PRDs for 185,696 services so far. The PRDs are available through our PRDDS. This huge set allows to use PRD-based password assistants already in practice. An application that already uses PRDs is AutoPass [156]. We presented the implementation of a PRD-based password assistant which is available as a web and mobile application and the integration of PRDs into KeePass.

Our contribution presented in this chapter is also highly valuable for existing password generators in general. We have shown that our optimal fallback password-composition rule set significantly improves the current default rules of common password generators. Moreover, our results have shown that the user interfaces of password generators can be drastically simplified while still covering nearly 100% of the services requirements.

In this chapter, we presented the first part of PAS. It enables the automatic generation of attack-resistant and valid passwords. In the subsequent Chapter 5, we present a password synchronization scheme that uses PRDs to automatically generate attack-resistant, individual, and valid passwords for users and makes them available on all their devices.

5 Passwordless and seamless password synchronization

In Chapter 3, it was shown that properly protecting preserved passwords is very challenging. Moreover, an offline synchronization of preserved passwords is very inconvenient for users and existing online synchronization approaches are insecure.

In this chapter, we present the second part of PAS which solves the password preservation, confidentiality, and availability problem. We introduce the PAsswordLess PAssword Synchronization (PALPAS) scheme. It automatically generates attack-resistant, individual, and valid passwords for online accounts and preserves them for users. Moreover, it protects the preserved passwords, makes them available on all user devices, and automatically synchronizes changes. But, PALPAS does not store passwords, neither at user devices nor at servers on the Internet. The preservation of passwords is done by storing a secret on all devices and some data on a central server. PALPAS generates the same passwords on all devices by combining the device-side secret and the service-side data. Updating the data on the server makes changes to password portfolios seamlessly available to all devices. But, the data on the server is statistically independent of the passwords. Thus, the data enables a passwordless synchronization of passwords between devices and it is impossible for an attacker to steal passwords from the server. In addition, PALPAS provides a revocation mechanism that allows users to invalidate the secret on their devices. This guarantees that the seed cannot be stolen from lost devices once revoked and an attacker cannot obtain the users' passwords. We detail PALPAS in Section 5.1.

We present a realization of PALPAS consisting of a client and a server application in Section 5.2. Further, we provide a detailed security evaluation in Section 5.3 and deduce in particular that PALPAS is the first secure online password synchronization scheme. Finally, we conclude this chapter in Section 5.4.

The main contributions of this chapter were published as part of [H6]. This chapter extends the published contributions by a revocation mechanism for user devices and an additional protection against a compromised synchronization server.

5.1 Solution for password preservation and synchronization

This section describes the PAsswordLess PAssword Synchronization (PALPAS) scheme. It automatically generates attack-resistant, individual, and valid passwords for online accounts and preserves them for users. Moreover, it protects the preserved passwords, makes them available on all user devices, and automatically synchronizes changes. But, PALPAS does not store passwords, neither at user devices nor at servers on the Internet. It retrieves passwords on devices on demand by regenerating them.

We start in Section 5.1.1 by describing the system architecture of PALPAS. Then, we detail the generation of passwords in Section 5.1.2 and explain how this preserves passwords and enables their retrieval. Next, we explain in Section 5.1.3 how a password portfolio is made seamlessly available on all user devices without storing passwords on devices and servers. Finally, we describe the management of user devices including the installation of PALPAS and the revocation of devices in case of loss in Section 5.1.4.

5.1.1 System architecture

This section provides details about the components of the PALPAS system, the data used by PALPAS, and the storage location of the individual data.

Components

The PALPAS system consists of two components:

- *PALPAS server (PLS)*: A central server that stores some user data. All devices of a user have access to the data and can add, update, and delete it.
- *PALPAS client (PLC)*: An application installed on all devices of a user. It generates the user's passwords on demand using some data stored locally on a device and some data received from the PLS.

We impose minimal trust assumptions on the PLS. We only consider the PLS to be trustworthy with regard to the availability of the data. For a detailed discussion about the trust relation between users and the PLS, we refer to Section 5.3.4. We present an implementation of both components in Section 5.2. In the following, we detail the data used by PALPAS and explain where it is stored.

Device-side data

All devices of a user store the following user-specific data:

- *PALPAS secret* (ps): The PALPAS secret consists of two parts:
  - *Seed*: A randomly generated secret that serves as input for the password generation.
  - *Data protection key* (k_{Data}): A randomly generated key that is used for privacy and integrity protection of the user data stored on the PLS. An encryption key $k_{\text{Data,Enc}}$ and an integrity key $k_{\text{Data,Mac}}$ (MAC, Message Authentication Code) is derived from k_{Data}.

  The PALPAS secret is created when a user uses PALPAS for the first time and it does not change over time. A device stores the secret $ps = (seed, k_{\text{Data}})$ encrypted by a one-time-pad (OTP) [22, 163]. Each user device (UD) randomly samples its individual one-time-pad key otp_{UD} when the PLC is installed for the first time. A device only stores the encrypted PALPAS secret $ps_{\text{UD}} = ps \oplus otp_{\text{UD}}$, the corresponding key otp_{UD} is stored on the PLS. This enables a secure revocation of the PALPAS secret in case of device loss (cf. Section 5.1.4.3).

Furthermore, each device of a user stores the following device-specific data:

- *Authentication key* (sk_{Auth}): A randomly generated secret key used to authenticate a device against the PLS. The key is created along with a corresponding public key pk_{Auth} during the installation of the PLC. $sk_{\text{Auth,UD}}$ is stored on the user device and $pk_{\text{Auth,UD}}$ on the PLS. A device stores $sk_{\text{Auth,UD}}$ encrypted with k_{mpw}. This encryption key is derived from a user-chosen master password (MPW) and not stored.

In Section 5.1.4, we describe the complete installation procedure of the PLC in which the aforementioned data is created. In the following, we explain which data is stored on the PLS.

Server-side data

Each PALPAS user has an account at the PLS which contains the following data:

- *Account data*: For each account c of the user at an Internet service, the PLS stores an account data object $data_{\text{c}} = (salt_{\text{c}}, prd_{\text{c}}, un_{\text{c}}, url_{\text{c}})$, with the following content:
  - $salt_{\text{c}}$: A random value that differs for each account. The $salt$ is used as input for the password generation and facilitates individual passwords for the different user accounts. Changing the $salt$ allows to create a new password for the account, as necessary for a password change. A salt is generated when the PLC initially creates the password, i.e. during the account registration.

– prd_c: The Password Requirements Description (PRD) specifies the password requirements of the service (cf. Section 4.2). The prd is for instance retrieved from the PRDDS (cf. Section 4.4.1) when the PLC initially creates a password. In the event of a password change the prd might be updated to comply with the recent password requirements of the service.

– un_c: The username for the account at the service, e.g. *me@example.org*.

– url_c: The service's URL where the user has the account, e.g. `https://example.org`.

Each $data_c$ is encrypted with the user-specific encryption key $k_{\text{Data,Enc}}$ by the PLC before it is transferred to the PLS. To retrieve only the account data of a specific account and not all of them, each $data_c$ is associated with an identifier id_c. It is generated by the PLC by $id_c = \text{HMAC}(k_{\text{Data,Mac}}, url_c)$ using a Keyed-Hash Message Authentication Code (HMAC) [168] and the user-specific key $k_{\text{Data,Mac}}$.

The account data is added, updated, or deleted during the synchronization of a password portfolio. We describe this in detail in Section 5.1.3. Besides the account data, the PLS stores the following device-specific data:

- *One-time-pad keys*: The individual one-time-pad keys (otp) of the user's devices on which the encrypted PALPAS secret *ps* is stored.
- *Authentication keys*: The individual public authentication keys (pk_{Auth}) of the user's devices that are allowed to access the data stored at the PLS.

We provide details of the account creation at the PLS and the registration of further devices in which the one-time-pad and authentication keys are added to the PLS in Section 5.1.4.

5.1.2 Password generation

We next detail how PALPAS in general generates attack-resistant, individual, and valid passwords for online accounts. Moreover, we describe the complete data flow of generating a particular password for a login at an account.

As depicted in Figure 5.1, PALPAS generates a password in two steps: First, a cryptographically secure Pseudo-random Generator (PRG) generates a random value using the $seed$ and a $salt$. Second, a Password Generator (PG) derives a password from the random value. The PG ensures that the password complies with the service's password requirements provided by the prd.

In case the requirements of a service are unavailable, PALPAS uses our optimal fallback password-composition rules and generates passwords consisting of letters and numbers with a length of 22 characters (cf. Section 4.5).

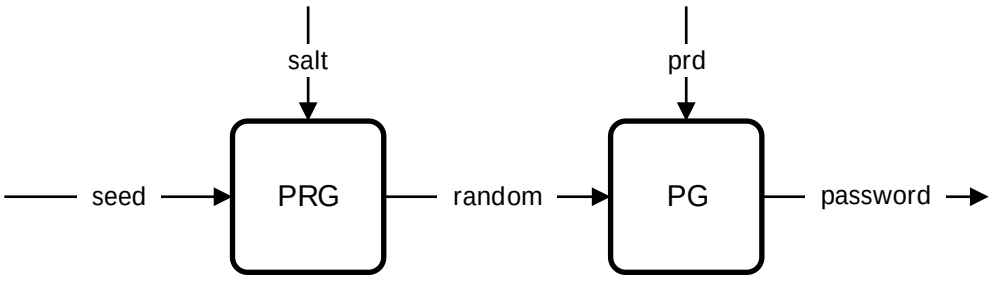


Figure 5.1: PALPAS password generation procedure.

The PRG and the PG are deterministic. Therefore, by using the same $seed$, $salt$, and prd, always the same password is generated. Because of this, PALPAS does not need to store passwords. It can regenerate them on demand. We present details about the implementation of the PRG and the PG in Section 5.2.1.

We describe the complete data flow of generating a password in the following. The user (U) wants to log in to his account c at the service (S) using his user device (UD). He has an account at the PLS and the UD is registered at the PLS. The PLC is installed on the UD and generates the user's password for the account. To simplify the description, we consider the PLC and the UD as a single entity and just use the term UD in our explanations in the next sections. The entire procedure of generating a password is described in the following and the corresponding data flow is illustrated in Figure 5.2.

1. U enters his master password (MPW) and the URL of the service on the UD (url_c).
2. The UD derives k_{mpw} from the MPW and decrypts $sk_{\text{Auth,UD}}$. Then, the UD and the PLS perform a mutual authentication in which the UD uses its $sk_{\text{Auth,UD}}$.
3. The UD requests otp_{UD} from the PLS. Then, it decrypts the locally stored PALPAS secret ps_{UD} with otp_{UD} by computing $ps = ps_{\text{UD}} \oplus otp_{\text{UD}}$. The UD now has $ps = (seed, k_{\text{Data}})$.
4. The UD derives $k_{\text{Data,Mac}}$ from k_{Data} and computes $id_c = \text{HMAC}(k_{\text{Data,Mac}}, url_c)$. Then, it sends id_c to the PLS and requests the corresponding account data.
5. The PLS responds with the related $data_c$.
6. The UD derives the encryption key $k_{\text{Data,Enc}}$ from k_{Data} and uses it to decrypt $data_c$. The UD has now $salt_c$, prd_c, and un_c. It next generates the password pw_c using the $seed$, $salt_c$, and prd_c. Finally, the UD logs in to the account c at S using pw_c and un_c.

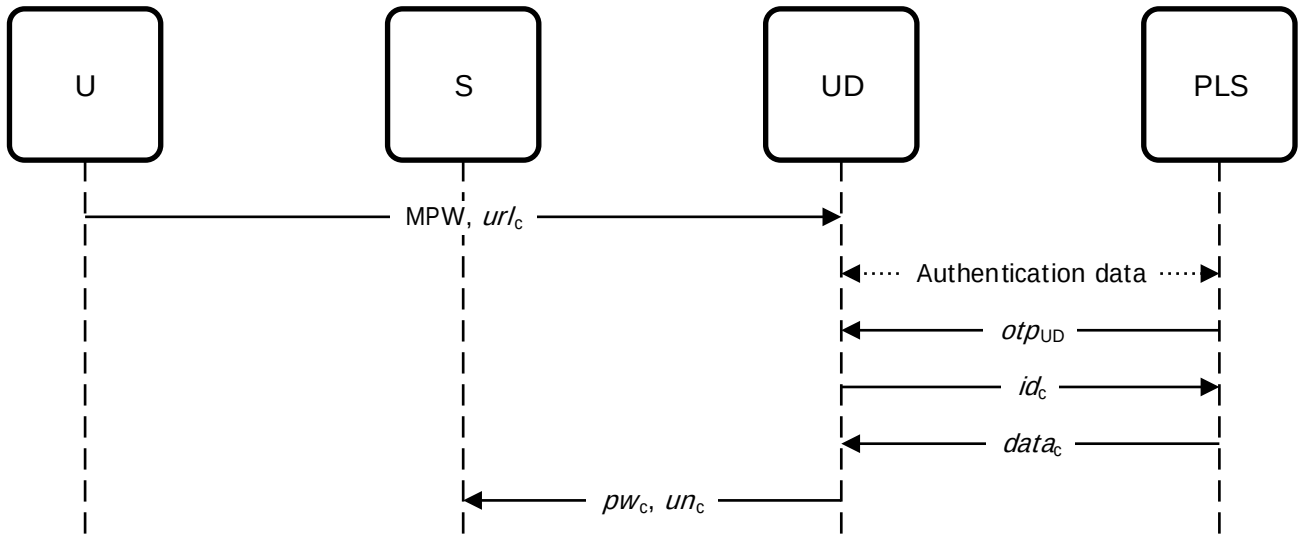


Figure 5.2: Data flow of the password generation procedure.

5.1.3 Password synchronization

In this section, we detail the synchronization of a password portfolio across user devices. It is realized by adding, updating, or deleting account data on the PLS. The account data is not cached on devices. Every time a password needs to be generated a device receives the corresponding account data from the PLS. In this way, it always gets the current account data and generates the latest passwords.

In the following, we describe the three situations affecting password portfolios and requiring a synchronization: First, adding a password to the user's portfolio after account creation at an Internet service in Section 5.1.3.1. Second, updating an account password in Section 5.1.3.2. Third, deleting a password from the portfolio in Section 5.1.3.3.

5.1.3.1 Synchronization of new passwords

After creating a password for a new online account it is necessary to make it available on all devices so that users can access their new account everywhere and anytime. PALPAS realizes this by adding the corresponding account data object on the PLS. Other devices are then able to receive the data, obtain the salt and the PRD, and thus to generate the password. The procedure of creating a new password for an online account and making it available on all devices is explained below. The corresponding data flow is depicted in Figure 5.3.

1. U enters his MPW and the URL of the service on the UD.
2. The UD derives k_{mpw} from the MPW and decrypts $sk_{\text{Auth,UD}}$. Then, the UD and the PLS perform a mutual authentication in which the UD uses its $sk_{\text{Auth,UD}}$.
3. The UD requests otp_{UD} from the PLS. Then, it decrypts the locally stored PALPAS secret ps_{UD} with otp_{UD} by computing $ps = ps_{\text{UD}} \oplus otp_{\text{UD}}$. The UD now has $ps = (seed, k_{\text{Data}})$.
4. The UD receives prd_{c} from the PRDDS. In case the PRD of the S is unavailable, U is prompted to use a default PRD (cf. Section 4.5) or to create one for the S manually. Then, the UD generates $salt_{\text{c}}$ as well as the account password pw_{c} using the $seed$, $salt_{\text{c}}$, and prd_{c}. U now selects a username un_{c} and uses un_{c} and pw_{c} to create an account at the S.
5. After the account creation, the UD derives $k_{\text{Data,Enc}}$ and $k_{\text{Data,Mac}}$ from k_{Data}. It creates $data_{\text{c}} = (salt_{\text{c}}, prd_{\text{c}}, un_{\text{c}}, url_{\text{c}})$ and encrypts it with $k_{\text{Data,Enc}}$. In addition, it computes the corresponding account data identifier $id_{\text{c}} = \text{HMAC}(k_{\text{Data,Mac}}, url_{\text{c}})$. Finally, it sends $data_{\text{c}}$ and id_{c} to the PLS.

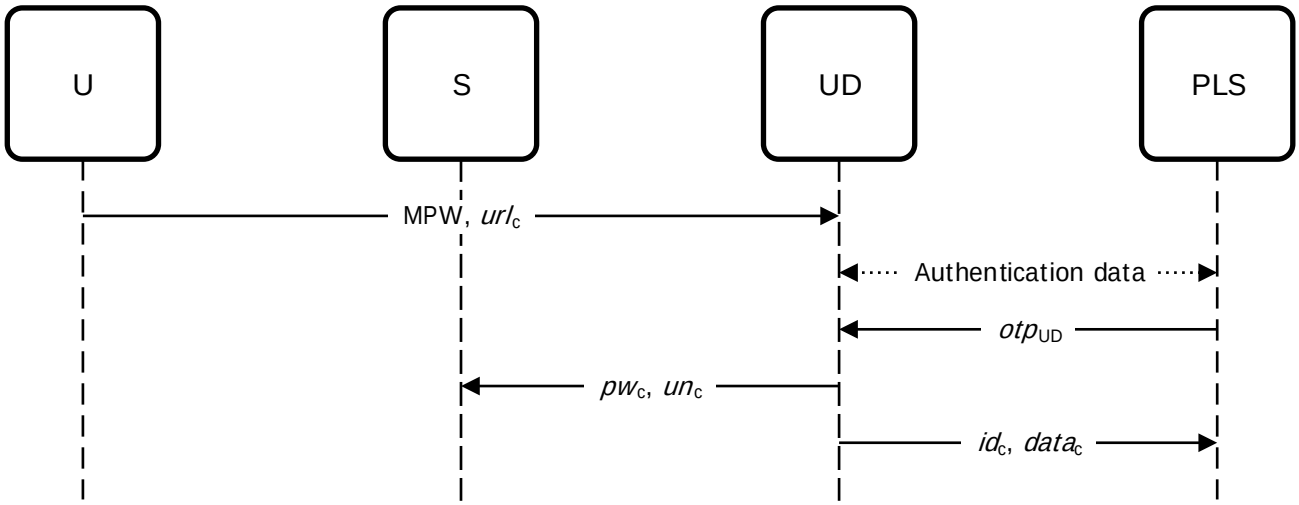


Figure 5.3: Data flow of the password synchronization procedure.

After storing the account data at the PLS, it can be immediately received by the other devices of $\mathcal{U}$. Thus, the new password is seamlessly available on all devices.

Other approaches, such as storing a password database on a server, have the drawback that devices have to regularly checking for updates to prevent outdated copies of passwords. This causes heavy network load. Furthermore, to new passwords seamlessly available on all devices users have to manually trigger a synchronization. PALPAS is doing this automatically.

5.1.3.2 Update of passwords

Changing the password of an online account requires to make the new password available on all user devices. PALPAS realizes this by generating a new salt for the account and updating the related account data at the PLS. To tackle the issue of device failures during this procedure, PALPAS temporary stores the old and the new salt at the PLS. This prevents situations like (1) the salt on the PLS was updated but the account passwords is not changed and (2) the password was changed but the new salt is not send to the PLS. Typical situations are application crashes or network interrupts. Moreover, this mechanism signals other devices that a password change is in progress. This is necessary for our solution of automatic password changes presented in Chapter 7. The complete procedure is described in the following. The corresponding data flow is shown in Figure 5.4:

1. U enters his MPW and the URL of the service on the UD.
2. The UD derives k_{mpw} from the MPW and decrypts $sk_{\text{Auth,UD}}$. Then, the UD and the PLS perform a mutual authentication in which the UD uses its $sk_{\text{Auth,UD}}$.
3. The UD requests otp_{UD} from the PLS. Then, it decrypts the locally stored PALPAS secret ps_{UD} with otp_{UD} by computing $ps = ps_{\text{UD}} \oplus otp_{\text{UD}}$. The UD now has $ps = (seed, k_{\text{Data}})$.
4. The UD derives $k_{\text{Data,Mac}}$ from k_{Data} and computes $id_{\text{c}} = \text{HMAC}(k_{\text{Data,Mac}}, url_{\text{c}})$. Then, it sends id_{c} to the PLS and requests the corresponding account data.
5. The PLS responds with the related $data_{\text{c}}$.
6. The UD derives the encryption key $k_{\text{Data,Enc}}$ from k_{Data} and uses it to decrypt $data_{\text{c}}$. The UD has now $salt_{\text{c}}$, prd_{c}, and un_{c}. It next generates the password pw_{c} using the $seed$, $salt_{\text{c}}$, and prd_{c}. Finally, the UD logs in to the account c at the S using pw_{c} and un_{c}.
7. The UD generates a new salt $salt'_{\text{c}}$ and the corresponding password pw'_{c}. Next, it created a new account data object $data'_{\text{c}} = (salt_{\text{c}}, salt'_{\text{c}}, prd_{\text{c}}, un_{\text{c}}, url_{\text{c}})$ which includes the old and the new salt. The UD encrypts $data'_{\text{c}}$ with $k_{\text{Data,Enc}}$ and sends it to the PLS.
8. The UD changes the account password at the S to pw'_{c}.
9. After a successful password change, the UD creates $data''_{\text{c}} = (salt'_{\text{c}}, prd_{\text{c}}, un_{\text{c}}, url_{\text{c}})$ only containing the new salt, encrypts it with $k_{\text{Data,Enc}}$, and sends it to the PLS.

Note that in Step 7, the UD also queries the PRDDS for an update of prd_{c} and, if available, receives the latest version. prd'_{c} is then also included into $data'_{\text{c}}$ and $data''_{\text{c}}$. After storing the updated account data at the PLS, the new password is immediately available on all devices.

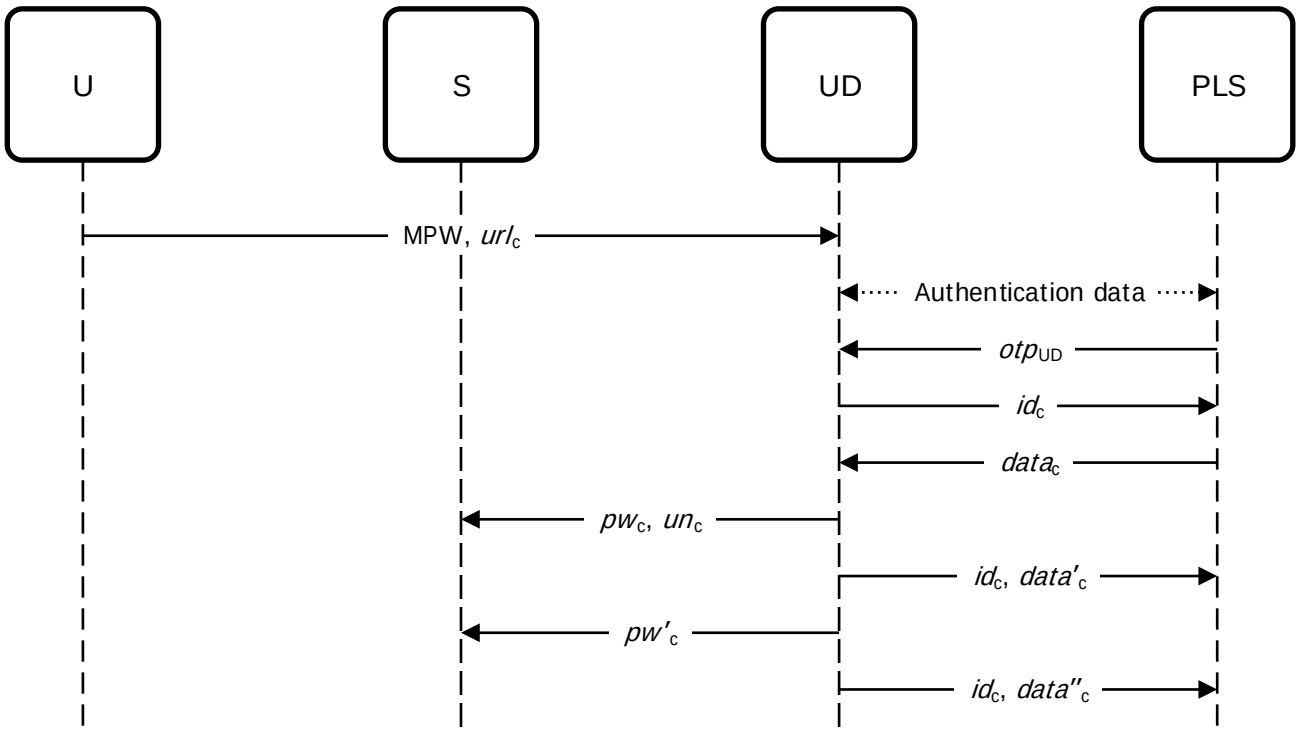


Figure 5.4: Data flow of the password update procedure.

The UD might not be able to successfully change the password (Step 8) or to finally update the account data (Step 9). This situation can be solved by any device of U, which works as follows: The UD receives $data'_c$ and generates the passwords pw_c and pw'_c. If it can log in with pw_c, it continues with Step 8. Otherwise, the UD performs Step 9.

5.1.3.3 Deletion of passwords

Finally, we explain the last operation of synchronizing a password portfolio across devices, the deletion of a password. The procedure consists of four steps which are enumerated below. The related data flow is depicted in Figure 5.5.

1. U enters his MPW and the URL of the service on the UD.
2. The UD derives k_{mpw} from the MPW and decrypts $sk_{Auth,UD}$. Then, the UD and the PLS perform a mutual authentication in which the UD uses its $sk_{Auth,UD}$.
3. The UD requests otp_{UD} from the PLS. Then, it decrypts the locally stored PALPAS secret ps_{UD} with otp_{UD} by computing $ps = ps_{UD} \oplus otp_{UD}$. The UD now has $ps = (seed, k_{Data})$.
4. The UD derives $k_{Data,Mac}$ from k_{Data} and computes the account data identifier $id_c = HMAC(k_{Data,Mac}, url_c)$. Then, it sends id_c to the PLS and requests the deletion of the related account data object $data_c$ at the PLS.

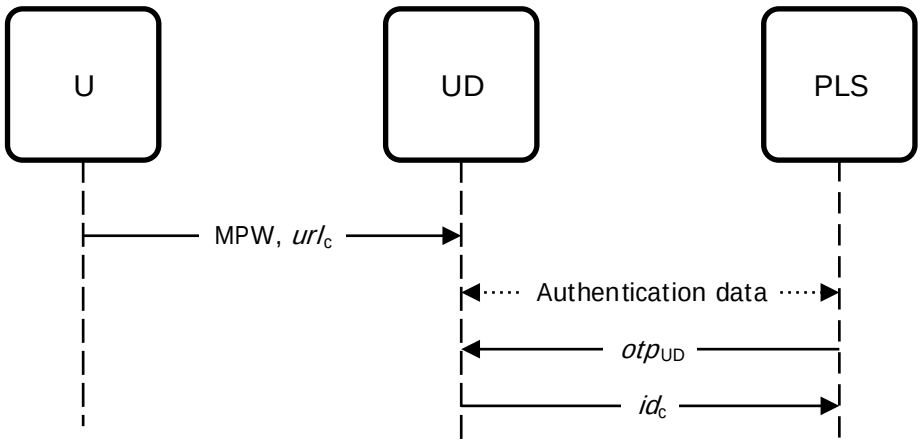


Figure 5.5: Data flow of the password deletion procedure.

In case U now tries to generate the password again and the UD requests the corresponding account data, the PLS responds with an error. Because the account data is not stored on user devices the password is immediately unavailable on all devices. In case of other approaches users need to worry that the password is still stored on other devices.

5.1.4 Device management

We detail the management of user devices in this section. First, we describe the case where users use PALPAS for the first time in Section 5.1.4.1. Second, we explain the case where users want to set up PALPAS on further devices in Section 5.1.4.2. Third, we provide details about the revocation of devices in the event of loss or theft in Section 5.1.4.3. To address the loss of the PALPAS data in such situations, we present a backup solution for PALPAS in Chapter 6.

5.1.4.1 Initial installation

The initial setup of PALPAS includes the installation of the PLC on a device and the creation of an account at the PLS. The installation procedure is described in the following and the corresponding data flow is illustrated in Figure 5.6.

1. After installing the PLC on the UD, U chooses a MPW. The UD derives k_{mpw} from the MPW. Then, it generates the PALPAS secret $ps = (seed, k_{\text{Data}})$. Afterwards, the UD randomly samples a one-time-pad key otp_{UD} and encrypts the PALPAS secret by computing $ps_{\text{UD}} = ps \oplus otp_{\text{UD}}$. The UD only keeps ps_{UD} and deletes ps. Then, it generates an authentication key pair $sk_{\text{Auth,UD}}$ and $pk_{\text{Auth,UD}}$ and stores $sk_{\text{Auth,UD}}$ encrypted by the k_{mpw}.

2. The UD connects to the PLS and the PLS authenticates itself to the UD.

3. The UD sends $pk_{\text{Auth,UD}}$ and otp_{UD} to the PLS. The PLS creates and account and stores $pk_{\text{Auth,UD}}$ and otp_{UD}. Finally, the UD deletes otp_{UD}.

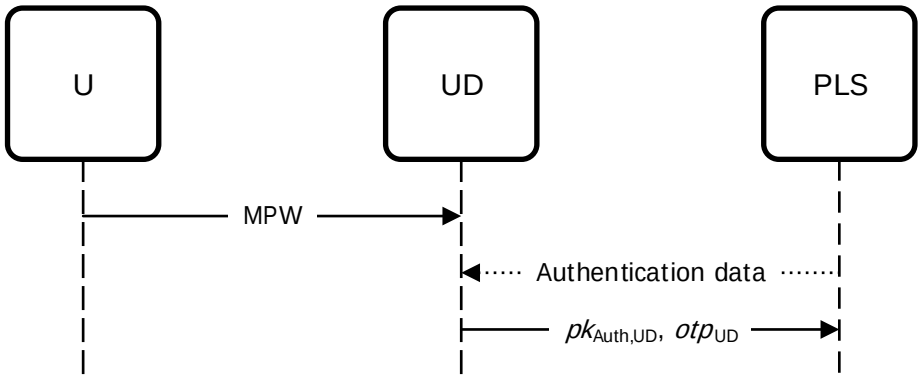


Figure 5.6: Data flow of the initial installation procedure.

The user account at the PLS is now created and U can synchronize his passwords (cf. Section 5.1.3) or set up PALPAS on his further devices, which we explain in the next section.

The registration at the PLS does not require any personal information of users, not even an email address. Moreover, the registration is as easy as possible. Users do not need to provide a username, a valid email address to obtain a conformation link, nor select a password which might be leaked during a later password breach.

5.1.4.2 Installation on further devices

We next describe the case where users want to set up PALPAS on further devices. This must be done only once for each device and includes the installation of the PLC on the further user device (UD′), the transfer of the PALPAS secret, and the registration of the device at the PLS.

1. U enters his MPW at the UD.

2. The UD derives k_{mpw} from the MPW and decrypts $sk_{\text{Auth,UD}}$. Then, the UD and the PLS perform a mutual authentication in which the UD uses its $sk_{\text{Auth,UD}}$.

3. The UD requests otp_{UD} and an authentication token t_{Auth} from the PLS. t_{Auth} is randomly created by the PLS and has a limited validity. It is used to temporary authenticate the UD′ at the PLS and to link the UD′ to the user's account.

4. The UD decrypts the locally stored PALPAS secret ps_{UD} with otp_{UD} by computing $ps = ps_{\text{UD}} \oplus otp_{\text{UD}}$. Then, the UD makes $ps = (seed, k_{\text{Data}})$ and t_{Auth} available to U. This can be done by storing the data in a file or encoding it as a QR code.

5. U installs the PLC on the UD′ and chooses a MPW. The UD′ derives k_{mpw} from the MPW.

6. U transfers $ps = (seed, k_{\text{Data}})$ and t_{Auth} to the UD′.

7. The UD′ connects to the PLS which authenticate itself to the UD′.

8. The UD′ randomly samples a one-time-pad key $otp_{\text{UD}'}$ and uses it to encrypt the PALPAS secret by $ps_{\text{UD}'} = ps \oplus otp_{\text{UD}'}$. It only keeps $ps_{\text{UD}'}$ and deletes ps. Then, it generates an authentication key pair $sk_{\text{Auth,UD}'}$ and $pk_{\text{Auth,UD}'}$ and stores $sk_{\text{Auth,UD}'}$ encrypted with k_{mpw}. Afterwards, the UD′ sends $pk_{\text{Auth,UD}'}$, t_{Auth}, and $otp_{\text{UD}'}$ to the PLS. The PLS verifies t_{Auth} and stores $pk_{\text{Auth,UD}'}$ and $otp_{\text{UD}'}$. Finally, the UD′ deletes $otp_{\text{UD}'}$.

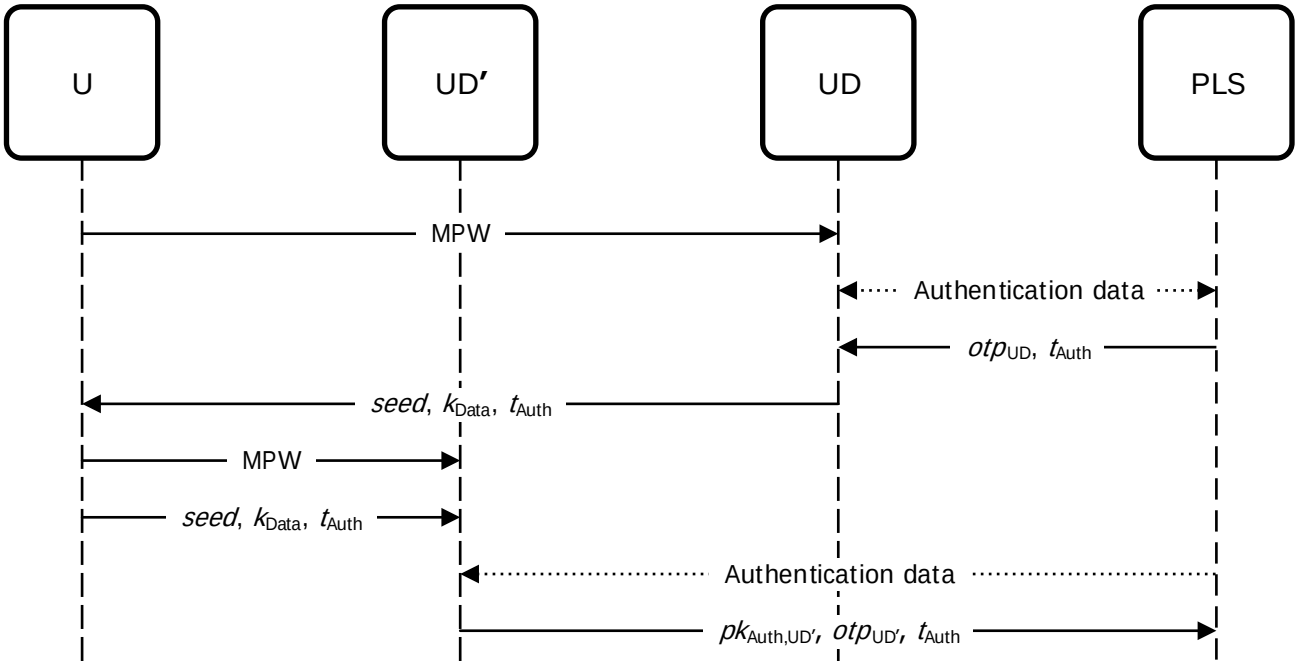


Figure 5.7: Data flow of the installation procedure on further devices.

The further user device UD′ has now access to the user's account at the PLS and can retrieve the account data for generating passwords or an authentication token to register another device.

U can choose the same MPW for the UD and the UD′. But, he can also use different passwords, because the derived encryption key k_{mpw} is only used to protect the locally stored secret authentication key. We discuss the security benefit of the feature in Section 5.3.3.4.

5.1.4.3 Revocation of devices

With regard to the risk of device loss and theft, a protection of the PALPAS data stored on devices is required. The encryption of the data using a user-chosen master password only provides a first line of defense, because such a password does not withstand a large-scale brute-force attack.

PALPAS has a revocation mechanism to solve this problem. A revocation invalidates the PALPAS secret on a device and prevents the leakage of passwords. In the following, we explain the revocation mechanism and refer to Section 5.3 for a detailed security evaluation.

All existing user devices are registered at the PLS with their individual pk_{Auth} and otp. To revoke a device a user deletes the related $pk_{\text{Auth,UD}}$ and otp_{UD} at the PLS. Now the device can no longer retrieve any data from the PLS. This includes the account data, authentication tokens, and particularly the otp_{UD}. Therefore, the device cannot generate passwords, register new devices, and decrypt the PALPAS secret ps_{UD} anymore. Moreover, the deletion of otp_{UD} invalidates ps_{UD} and it is impossible to recover the PALPAS secret ps from the revoked device. Thus, the user device is useless for an attacker and the passwords of the user cannot be stolen.

Alternative revocation methods

Users might not have access to another device, which is registered at the PLS, to do the revocation. Therefore, alternative methods to perform a revocation are necessary. A typical situation is the theft of a mobile device while being on holiday. It takes days until users come home and can revoke a stolen device from their desktop computer. We present two solutions to solve this problem and to ensure a secure revocation at any time.

- *PALPAS backup*: The first solution is a PALPAS backup which is placed at a friend. Users can call their friends and ask them to revoke their stolen or lost device on their behalf. This solution can be realized with our backup solution present in Chapter 6.
- *Alternative authentication means*: The second solution is to allow users to revoke their devices with other means of authentication such as electronic identity cards. We present a solution to support multiple authentication means on the PLS-side in [H20]. To enable users to use their different authentication means on devices, we present various solutions based on electronic identity cards [H11, H10, H18], signature cards [H7, H9, H13], and the Single Sign-on systems using SAML (Security Assertion Markup Language) [H8, H14].

These solutions enable users to securely revoke their devices everywhere and anytime.

5.2 Implementation

In this section, we present a realization of our solution. We describe the implementation of the PALPAS client application in Section 5.2.1 and the PALPAS server in Section 5.2.2.

5.2.1 Client application

We start in this section by describing the architecture of the PLC and then provide details of the implementation of the PALPAS password generation scheme and the other cryptographic primitives used in PALPAS. We implemented the PLC in Java. The source code is available at [H26]. For the realization of the cryptographic primitives we used Bouncy Castle [147].

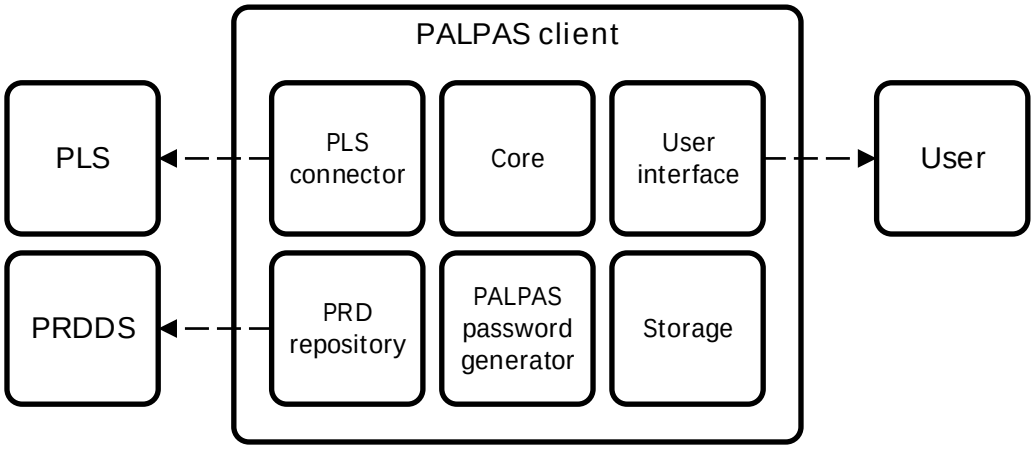


Figure 5.8: Architecture of the PALPAS client.

Architecture

The architecture of the PLC is depicted in Figure 5.8. It consists of six main components which we briefly describe below.

- *Core*: It manages the application and instantiates the other components.
- *User interface*: It provides a graphical user interface for the user. It allows the user to generate passwords for his new online accounts and to manage his password portfolio.
- *Storage*: It stores necessary data on the user device such as the encrypted PALPAS secret and the secret authentication key for the PLS.
- *PLS connector*: It establishes the communication to the PLS. The connector performs the authentication and retrieval of the account data stored on the PLS.

- *PRD repository*: It manages the PRDs of the services for which the user has accounts. In case the user adds an account to PALPAS, the repository requests the corresponding PRD of the service from the PRDDS (cf. Section 4.4.1). In case the PRD is not available, the user is prompted to use a default PRD (cf. Section 4.5) or to create one for the service. The repository also checks for updates during a password change.
- *PALPAS password generator*: It implements the PALPAS password generation scheme (cf. Section 5.1.2). For the login at services, it triggers the PLS connector to receive the corresponding account data from the PLS. In case of a password change, it generates a new salt and corresponding password and instructs the PLS connector to update the account data at the PLS. We provide more details regarding its realization below.

PALPAS password generator

As described in Section 5.1.2, the password generation is done in two steps: First, the PRG generates a random value using the seed and the account-specific salt. Second, the PG derives a password from the random value according to the password requirements provided by a PRD.

We implemented the PRG using the block cipher AES 128 in CBC mode [19]. The seed is used as the key and the salt as input for the cipher. The output is the random. We iteratively increment and then encrypt the salt value until enough random bits for the PG are produced.

We already described the implementation of the PG using rejection sampling in detail in Section 4.6.2.1. In brief, the PG maps the random value to a password. Next, it verifies whether the password fulfills the password requirements. If not, the password is discarded and the PG obtains a new random value from the PRG and starts from scratch.

Realization of the cryptographic primitives

For the generation of the PALPAS secret $ps = (seed, k_{\text{Data}})$ we use the secure random number generator of Java (class `SecureRandom`). All secrets are 128-bit values. For sk_{Auth} and pk_{Auth} we generate a 2048-bit RSA key pair. k_{mpw} is derived from the user's master password using the PBKDF2 function [130]. $k_{\text{Data,Enc}}$ and $k_{\text{Data,Mac}}$ are derived from k_{Data} using an HMAC-based Extract-and-Expand Key Derivation Function (HKDF) [140].

For the encryption of the account data we use AES 128 in CBC mode. Each account data object is encrypted separately using a different initialization vector (IV). Both, ciphertext and IV are additionally protected using a HMAC [139, 168]. Ciphertext, IV, and the HMAC value are stored on the PLS.

5.2.2 Server application

We now provide details about the implementation of the PALPAS server, including its components, the user authentication, and its deployment. The source code and the detailed specification of its interfaces are available at [86, H26].

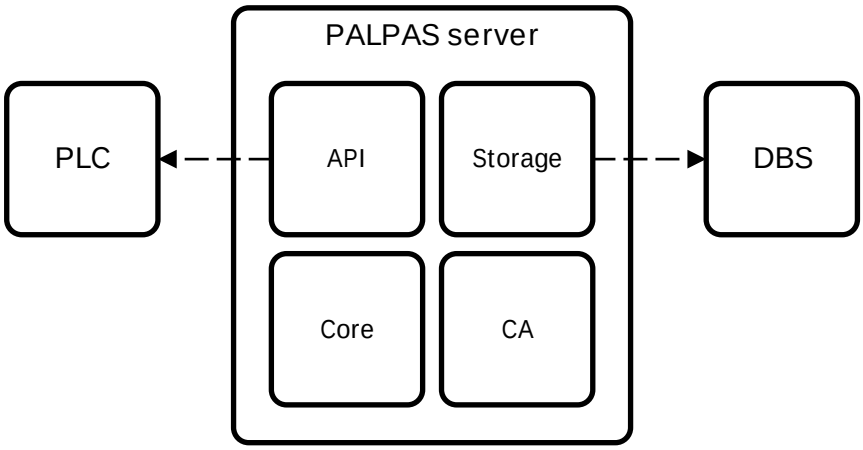


Figure 5.9: Architecture of the PALPAS server.

Architecture

The architecture of the PLS is depicted in Figure 5.9. It consists of four main components which we briefly describe in the following:

- *Core*: It instantiates the other components and manages the application. It also takes care of log files and error handling.
- *API*: The Application Protocol Interface provides means for creating new accounts as well as managing existing ones. This particularly includes adding, receiving, updating, and deleting account data. Moreover, the API provides means to receive authentication tokens and to revoke devices. The API is used by the PALPAS clients.
- *Storage*: It manages the stored user and authentication data. The data is saved on a separate database server (DBS).
- *CA*: The Certificate Authority component manages the certificates used for user/device authentication. We describe its realization below.

User authentication

As described in Section 5.1.1, the authentication at the PLS is based on device-specific authentication keys. To bind the individual public authentication keys to each device we use certificates (cf. Section 2.1.5). Besides its individual secret authentication key $sk_{\text{Auth,UD}}$, each user device has a certificate $ct_{\text{Auth,UD}}$ which is used for the authentication. For the issuance and management of the certificates, the PLS operates its own CA.

The issuance of a certificate works as follows: During the account creation at the PLS, the UD sends a Certificate Signing Request (CSR) [172] to the PLS. The CSR contains only the public key $pk_{\text{Auth,UD}}$ of the device and no further personal information. It is signed by $sk_{\text{Auth,UD}}$. The PLS creates an X.509 certificate [63] $ct_{\text{Auth,UD}}$ based on the CSR and transmits it to the UD. The subject of the certificate contains a user identifier (UID) and a device identifier (DID), which are both randomly generated by the PLS. The serial number of $ct_{\text{Auth,UD}}$ is stored in the user's account at the PLS.

We use TLS with client authentication to mutually authenticate the PLS and a UD. Within the TLS handshake the PLS presents its certificate $ct_{\text{Auth,PLS}}$ to the UD so that the UD can verify the PLS' identity. We equip the PLS with a certificate issued by a commercial CA. In addition, the PLS requests a certificate from the UD which responds with $ct_{\text{Auth,UD}}$. The PLS verifies the certificate and identifies the user based on the UID encoded in the subject field of the certificate. Then, the PLS checks if the serial number of $ct_{\text{Auth,UD}}$ is allowed to access the account. For a detailed description of TLS with client authentication, we refer to [63].

Authenticating each device by its own certificate in conjunction with individual OTPs has the advantage of a fine-granular revocation. In case of theft or loss the user can revoke the access for a device by removing the serial number from the list of allowed devices in his account. Please see Section 5.1.4.3 and 5.3.3.4 for further details.

Apart from that, users might want to additionally protect their accounts by multi-factor authentication. We present such a solution in [H16]. The integration such a multi-factor authentication in the PLS is future work.

Deployment

We developed the PLS as a Java web application providing a RESTful API. For the implementation of the CA we used Bouncy Castle [147]. For the storage of the user and authentication data we used the Java Persistence API and a MySQL server as a backend.

The PLS is deployed on a Tomcat server. The server is running behind an Apache server which acts a proxy and the TLS endpoint. Any TLS connections to the PALPAS clients are established and managed by the Apache server. For the account creation, it performs an unilateral authentication in which only the server authenticates itself to the UD using $ct_{\text{Auth,PLS}}$. In case a UD requests access to a user account, the Apache server requests the UD's certificate. After the connection establishment, the Apache server forwards the request to the PLS along with information about the UD's certificate $ct_{\text{Auth,UD}}$ such as the serial number. Based on this data, the PLS can perform the authorization, i.e. verifies that the UD is allowed to access the account.

Besides the TLS management, we equipped the Apache server with various countermeasures against denial-of-service attacks [240]. For instance, we limit the number of requests devices can make to the PLS. This setup also allows to balance the load to multiple servers which is necessary to support a huge number of PALPAS users.

5.3 Security evaluation

We evaluate the security of PALPAS in this section. At the beginning, we extend in Section 5.3.1 our current system and attacker model to take the PLS into account. Particularly, we describe the additional attacker's capabilities. Then, we describe the security properties of the PALPAS' password generation procedure in Section 5.3.2 and evaluate them with respect to the attacker's capabilities in Section 5.3.3. We show that the attacker is not able to obtain the passwords of users. Finally, we detail the trust relation between users and the PLS in Section 5.3.4. We describe that PALPAS imposes only a minimal trust assumption with respect to the PLS and users only need to trust in the availability of their data stored on the PLS.

5.3.1 Extended system and attacker model

We extend our system model described in Section 3.1 with a further entity: the PALPAS server $\mathcal{P}$. The user $\mathcal{U}$ has an account at $\mathcal{P}$ which contains the data described in Section 5.1.1. The access to the account requires that $\mathcal{U}$ authenticates himself against $\mathcal{P}$. This is done by the device-specific authentication keys. The communication between $\mathcal{U}$ and $\mathcal{P}$ is done over the Internet. Both entities establish a secure and mutually authenticated channel using TLS.

Attacker goal

So far, we considered an attacker $\mathcal{A}$ who aims at obtaining the password of $\mathcal{U}$ for an online account. We now assume that $\mathcal{A}$ aims at obtaining the PALPAS data of $\mathcal{U}$ in order to generate his entire password portfolio.

Table 5.1 provides an overview of the secrets used by the PALPAS system to generate passwords and to manage the account data. Note that the salts are non-security-sensitive data. In Section 5.3.3, we explain that $\mathcal{A}$ cannot obtain any information from them. Moreover, the public authentication keys of the user's devices are also useless for $\mathcal{A}$ because the are public information by nature and therefore uncritical. Their knowledge also raises no privacy issue because they are only used for the PLS and not for other services.

Secret	Objective	Properties
$seed$	Password generation	• Randomly generated by the PLC. • Created during the first installation of the PLC. • Same for all user devices. • Manually transfered by the user only once per device.
k_{Data}	Remote data protection	• Randomly generated by the PLC. • Created during the first installation of the PLC. • Same for all user devices. • Manually transfered by the user only once per device.
otp	Local data protection	• Randomly generated by the PLC. • Created during the installation of the PLC on a device. • Different for all user devices. • Stored on the PLS.
sk_{Auth}	Device authentication	• Randomly generated by the PLC. • Created during the installation of the PLC on a device. • Different for all user devices.
k_{mpw}	User authentication and local data protection	• Derived from user's master password. • Not stored. • Same/different for user devices. • Entered for every use of the PLC.

Table 5.1: PALPAS secrets.

Attacker capabilities

We assume that $\mathcal{A}$ knows which services $\mathcal{U}$ is using and the usernames of his accounts. This knowledge allows $\mathcal{A}$ to obtain the PRDs of the services. With respect to the new goal of obtaining the user's PALPAS data, we consider that $\mathcal{A}$ has the following four additional capabilities:

AC4 $\mathcal{A}$ knows the password c_{pw} of an account c of $\mathcal{U}$ at a service.

AC5 $\mathcal{A}$ knows $salt_c$ which was used to generate c_{pw} for the account c of $\mathcal{U}$.

AC6 $\mathcal{A}$ knows $salt_c$ and c_{pw} of the account c of $\mathcal{U}$.

AC7 $\mathcal{A}$ knows the encrypted PALPAS secret ps and an encrypted secret authentication key sk_{Auth} used to authenticate $\mathcal{U}$ at $\mathcal{P}$.

Attacker limitations

We consider the same attacker's limitations as described in Section 3.1. In particular, we assume that user devices are not compromised. Note that PALPAS has an inherent protection against phishing attacks because it uses public-key cryptography for user authentication at the PLS [10, 60]. Other password synchronization solutions (e.g. [144]) are vulnerable to an attacker luring users to a fraud website to obtain their credentials for a password synchronization server.

5.3.2 Security properties

In this section, we describe the security properties of the PALPAS password generation procedure. First, we explain that $\mathcal{A}$ cannot obtain the seed by knowing passwords and/or salt values. Second, we describe that PALPAS is the first solution that automatically generates attack-resistant, individual, and valid passwords for users.

In the first step of the password generation, PALPAS uses a PRG. It takes as input the seed and a salt and outputs a random value. We assume that the PRG is cryptographically secure. This means that $\mathcal{A}$ cannot distinguish between the output of the PRG and a truly random value. This property is named pseudo-randomness. As described in Section 5.2.1, we implement the PRG using the block cipher AES 128 in CBC mode which provides this property [19].

It follows from the pseudo-randomness of the PRG that $\mathcal{A}$ knowing the output of the PRG, cannot learn the seed or salt. $\mathcal{A}$ also cannot compute an input of the PRG by knowing another input. This means $\mathcal{A}$ cannot compute the seed by knowing a salt. $\mathcal{A}$ also cannot determine the PRD's output by knowing either a seed or a salt. This is only possible by having both inputs.

Computing the outputs of all possible seeds and salts is infeasible because the seed and the salt are 128-bit values. Such a brute-force attack is the only option of $\mathcal{A}$, because PALPAS generates seeds and salts completely random. Other approaches (e.g. [105, 189]) that use the URL of services or user-chosen master passwords as input allow efficient attacks.

In the second step of the password generation, the PG derives a password from the random value using the modulo operator. Further, the PG uses rejection sampling to ensure that the password complies with a given PRD. The input of the PG, i.e. the random value, is at least 128 bits longer than its output (cf. Section 4.6.2.1). For instance, in case of 2^{130} possible passwords the random value is 264 bits long. This 128-bit security buffer ensures a uniform distribution of the generated passwords while using the modulo operator. Moreover, there exist multiple random values for the same password. In case of the previous example and under the assumption that no further restrictions are applied, there exist 2^{136} random values that result in the same password. With the knowledge about the functioning of the PG and the publicly available PRDs, $\mathcal{A}$ can determine plausible random values for a password. But, he cannot determine if it is the correct one generated by the seed and salt of $\mathcal{U}$. With respect to the pseudo-randomness of the PRG, $\mathcal{A}$ can also not compute the seed and salt from a random value. To conclude, $\mathcal{A}$ cannot obtain the seed by knowing passwords and/or salt values.

PALPAS fulfills the security requirement SR1 of generating attack-resistant passwords (cf. Section 3.2). The seed and the salts are completely random values and not derived from any user-related or service-related information. Moreover, the PRG and the PG generates passwords in a uniform way. Consequently, the generated passwords do not have any patterns that can be exploited by $\mathcal{A}$. Further, PALPAS generates passwords with a security level of 130 bits which makes a brute-force attack infeasible. If attack-resistant passwords are not supported by services, PALPAS generates passwords with the best possible security level.

PALPAS also fulfills security requirement SR2 of generating individual passwords (cf. Section 3.2). This is ensured by using different salt values for accounts and always randomly generating them. Consequently, a password is never used twice.

Finally, PALPAS fulfills the service condition SC1 of generating valid passwords (cf. Section 3.2). This is done by the PG which takes the individual password requirements of services into account. In case the requirements of a service are partially or entirely unavailable, PALPAS makes use of our optimized password-composition rule set (cf. Section 4.5) to generate attack-resistant passwords with the best possible acceptance rate. To conclude, PALPAS is the first solution that automatically generates attack-resistant, individual, and valid passwords.

5.3.3 Attack scenarios

In this section, we evaluate the security properties of PALPAS described in the previous section with respect to the new attacker's capabilities defined in Section 5.3.1. We show that $\mathcal{A}$ is not able to obtain the seed of $\mathcal{U}$ and generate his password portfolio.

5.3.3.1 Scenario 1

In the first attack scenario, we evaluate the attacker's capability of knowing a password of $\mathcal{U}$ (cf. AC4). $\mathcal{A}$ obtained the password for instance by a password breach at the respective service. We show that $\mathcal{A}$ is not able to generate the other passwords of $\mathcal{U}$.

With respect to the assumptions about the PALPAS' password generation procedure described in Section 5.3.2, $\mathcal{A}$ is able to invert the PG and determine plausible random values for the stolen password. However, it follows from the pseudo-randomness of the PRG that knowing its output, i.e. the random value, it is infeasible to learn the seed and the salt value. To this end, $\mathcal{A}$ cannot generate the other passwords of $\mathcal{U}$.

The same applies for the cases in which $\mathcal{A}$ obtains multiple passwords of $\mathcal{U}$ from different services or passwords from a single service over a long period of time. $\mathcal{A}$ can obtain plausible random values for these passwords by inverting the PG, but he cannot learn the seed and the salt values from them. Due to the fact that PALPAS generates individual passwords for accounts, $\mathcal{A}$ can also not reuse stolen passwords to get access to other accounts of $\mathcal{U}$.

5.3.3.2 Scenario 2

In the second attack scenario, we consider that $\mathcal{A}$ knows a salt value of $\mathcal{U}$ (cf. AC5). $\mathcal{A}$ got it for instance by a security incident at $\mathcal{P}$. We show that $\mathcal{A}$ cannot generate the user's passwords.

The salts are randomly chosen values and statistically independent of the passwords and from the seed. Because of this, $\mathcal{A}$ cannot derive any information from them, e.g. for which service a salt was used. Furthermore, it follows from the pseudo-randomness of the PRG that only knowing a salt computing the output of the PRG without the seed is infeasible. To this end, $\mathcal{A}$ cannot generate the user's password portfolio.

5.3.3.3 Scenario 3

In the third attack scenario, we combine the previous scenarios and consider that $\mathcal{A}$ knows a password and the corresponding salt (cf. AC6). $\mathcal{A}$ obtained this information for example by a security breach at $\mathcal{P}$ and at a service. We show that even in this case $\mathcal{A}$ cannot generate the other passwords of $\mathcal{U}$.

With respect to the assumptions described in Section 5.3.2, $\mathcal{A}$ is able to invert the PG and determine plausible random values for the stolen password. However, it follows from the pseudo-randomness of the PRG that knowing its output and the salt it is impossible to learn the seed. To this end, $\mathcal{A}$ cannot generate the other passwords of $\mathcal{U}$.

5.3.3.4 Scenario 4

In the last attack scenario, we assume that $\mathcal{A}$ knows the encrypted PALPAS secret ps and an encrypted secret authentication key sk_{Auth} (cf. AC7). He got this data for instance by stealing a device of $\mathcal{U}$. We show that the PALPAS' revocation feature (cf. Section 5.1.4.3) prevents $\mathcal{A}$ from generating the user's password portfolio.

$\mathcal{A}$ cannot decrypt ps without the corresponding key otp. As part of PALPAS' revocation procedure, the key is deleted from $\mathcal{P}$. Therefore, it is impossible to recover ps because it is a one-time-pad encryption yielding information-theoretic security [22, 163]. To this end, $\mathcal{A}$ cannot obtain the seed of $\mathcal{U}$ and therefore cannot generate his password portfolio.

sk_{Auth} is encrypted with k_{mpw} which is derived from the user's master password. We expect that the MPW has an appropriate security level. This assumption is reasonable because this is the only password that $\mathcal{U}$ needs to memorize. Therefore, it provides a sufficient protection until pk_{Auth} is revoked by $\mathcal{U}$ and thus sk_{Auth} cannot be misused by $\mathcal{A}$ to access $\mathcal{P}$.

Alternatives to a master password

The encryption key k_{mpw} is only used for the protection of sk_{Auth} on a device. $\mathcal{U}$ can choose the same MPW for all devices, but can also use different ones. This allows him to choose a very strong password for his mobile devices because they are most threatened. Moreover, it is possible to facilitate a protection of sk_{Auth} with many different authentication means which especially are more secure than a user-chosen master password.

In case of mobile devices with a fingerprint reader it would be possible to randomly generate k_{mpw}, save it in a keystore stored on the device, and protect it with the user's fingerprint. The PLC then asks $\mathcal{U}$ for his fingerprint instead of requiring a long and complex master password. Another possibility is to store sk_{Auth} on a smart card and protect it with a PIN. The key is then located in the protected memory of the card and $\mathcal{A}$ has only a limited number of attempts to guess the PIN. This smart card approach is suitable for desktop computers with a card reader but also works with NFC-enabled mobile devices. Our backup solution for PALPAS presented in Chapter 6 makes use of this smart card approach.

Other approaches that use for instance a master password to encrypt passwords do not support different authentication means. Their security entirely relies on the questionable assumption that users select an attack-resistant master password or immediately change all their account passwords if there is a suspicion that the master password is compromised.

5.3.4 Trust relation

Finally, we evaluate the trust relation between $\mathcal{U}$ and $\mathcal{P}$. Certainly, $\mathcal{U}$ needs to trust $\mathcal{P}$ to some extent. However, PALPAS imposes only a minimal trust assumption with respect to $\mathcal{P}$. It only requires that $\mathcal{P}$ ensures the availability of the user data. PALPAS does not expect $\mathcal{P}$ to be trustworthy with respect to privacy and data integrity.

The user's privacy-sensitive information stored at $\mathcal{P}$ are the URLs of the services and his usernames. Both would allow $\mathcal{P}$ to create a detailed user profile. PALPAS protects the user's privacy in two ways: First, the account data which includes both, service URL and username, is encrypted with $k_{\text{Data,Enc}}$. The key is only available on the devices of $\mathcal{U}$. Second, account data identifiers are generated by computing an HMAC with $k_{\text{Data,Mac}}$ over the service's URL. The key is also only available on the devices of $\mathcal{U}$. Moreover, the encryption of the salts prevents a manipulation by a malicious $\mathcal{P}$ (cf. Section 3.6.2.4). More precise, $\mathcal{A}$ controlling $\mathcal{P}$ and a service cannot shuffle the account data of $\mathcal{U}$ so that he logs in to the service with the passwords of his other accounts.

For users that do not want to rely on the assumption that $\mathcal{P}$ ensures the availability of their data, we describe an alternative approach in Section 6.1.4. The basic approach is to redundantly store the account data on multiple PLSs to mitigate the risk of loss. Besides loss protection, this also mitigates unavailability during a potential temporary outage or maintenance of a PLS.

5.4 Conclusion

In this section, we presented PALPAS, the first secure and usable online password synchronization scheme. Users' passwords are generated on demand using the seed stored on devices and the salts synchronized with the PLS. Users only need to remember a single master password. PALPAS performs the first password task of generating attack-resistant, individual, and valid passwords for online accounts automatically for users.

The initial installation of PALPAS as well as the setup of further devices is very easy and convenient. The first installation of PALPAS only requires to choose a master password. In contrast to other approaches, users do not need to provide any personal information such as an email address. The setup of PALPAS on further devices must be done only once. It only requires to transfer some data which can be easily done by a QR code.

The practicality of PALPAS has been demonstrated by a realization of the PALPAS client and the PALPAS server. Moreover, we provided a detailed security evaluation of PALPAS. We explained that an attacker cannot obtain a seed and generate users' passwords, neither by compromising the PLS and/or services nor stealing user devices. Moreover, we described that PALPAS generates passwords according to the related security requirements and service conditions for secure passwords defined in Section 3.2. Finally, we stated that PALPAS only imposes minimal trust requirements on the PLS. This allows to operate a PLS with standard hardware and software. Users can even run their own PLS.

PALPAS is the second part of PAS. It solves the preservation, confidentiality, and availability problem of passwords. In the subsequent Chapter 6, we complement PALPAS with a backup solution called PASCO (PALPAS RECOVERY). It is the third part of PAS and ensures the recoverability and accessibility of preserved passwords. PASCO allows to cope with situations of data loss on both sides, user devices and the PLS. PASCO also provides an emergency access for backups which allows users to grant access to their passwords. Finally, in Chapter 7, we present the fourth part of PAS which provides automatic password changes and enables users to change their passwords regularly. PALPAS makes the new passwords automatically and seamlessly available on all devices. Moreover, PAS feature of automatic password changes also enables users to easily migrate from their weak to attack-resistant passwords generated by PALPAS.

6 Update-tolerant and revocable password backup with emergency access

In Chapter 3, we showed that creating and maintaining backups of preserved passwords for recovery and emergency access is very difficult for users. Backups must be placed at secure locations and kept up-to-date at the same time.

In this section, we present the third part of PAS which solves the password recoverability and accessibility problem. We introduce PASCO (PALPAS RECOVERY), a secure and usable backup solution for PALPAS (cf. Chapter 5). It ensures that users never lose their passwords by providing recoverability of the PALPAS data. Backups created by PASCO do not have to be updated even when users' passwords change. Users need to create backups only once and can keep them completely offline at secure, different, and physically isolated locations. This minimizes the risk of compromise and loss as well as enables an emergency access to the users' passwords by trusted persons. Moreover, PASCO backups have a built-in revocation mechanism. It allows users to completely invalidate backups if they lose control over them. The revocation mechanism works without having access to the backups themselves and guarantees that no passwords can be leaked from the backups once revoked. We detail PASCO in Section 6.1.

Afterwards, we present the fully controllable emergency access of PASCO backups in Section 6.2. Users can authorize trusted persons to access backups and obtain a set of pre-defined passwords in urgent or emergency situations.

Then, we present an implementation of PASCO in Section 6.3 as well as a security evaluation in Section 6.4. Finally, we conclude this chapter in Section 6.5.

The contributions of this chapter were published as part of [H3]. This chapter extends the published contributions by the description of the implementation and the security evaluation.

6.1 Solution for password backup

In this section, we describe PASCO (PALPAS RECOVERY), a secure and usable backup solution for PALPAS (cf. Chapter 5). It ensures that users never lose their passwords by providing recoverability of the PALPAS data.

The $seed$, the salts, and the PRDs are crucial for the PALPAS password generation procedure. Moreover, the secret authentication key sk_{Auth} and the encryption key k_{Data} are necessary to retrieve the account data from the PALPAS server (PLS). The availability of these three pieces of data must be guaranteed. Otherwise, the user's password portfolio is lost.

The account data is stored at the PLS. Their availability has to be guaranteed by the PLS, which is in line with our trust assumption (cf. Section 5.1.1). We assume that the provider of the PLS implements proper measures to restore the data at any time. As an alternative, we describe a user-side solution using multiple PLSs in Section 6.1.4. The PALPAS secret $ps = (seed, k_{\text{Data}})$ and the individual sk_{Auth} are exclusively stored on the user devices and thus are at high risk of loss. Typical situations are lost, stolen, or damaged devices as well as malware. The recoverability of the PALPAS data on the devices is ensured by PASCO.

PASCO backup device

PASCO uses a separate backup device (BD) to store a backup of the PALPAS data. We consider the BD to be a tamper resistant device that provides secure storage, user authentication, and basic cryptographic algorithms such as encryption and hashing. In Section 6.3 we present a practical realization of PASCO using an off-the-shelf smart card as a BD.

In the same way as user devices, the BD stores the PALPAS secret $ps = (seed, k_{\text{Data}})$ encrypted with a one-time-pad key otp_{BD}. Furthermore, it has its own secret authentication key $sk_{\text{Auth,BD}}$ for the PLS. The corresponding $pk_{\text{Auth,BD}}$ is stored on the PLS. To prevent the BD from unauthorized access, it is protected by a user-chosen PIN. A retry counter for the PIN prevents guessing attacks. After five wrong PIN entries the BD erases all stored data.

We explain the creation of update-tolerant PASCO backups in Section 6.1.1 and how to restore the PALPAS data from backups in Section 6.1.2. In Section 6.1.3, we describe the built-in revocation mechanism provided by PASCO backups.

6.1.1 Creation of backups

We now describe the creation of PASCO backups in order to protect the PALPAS data stored on user devices from loss. The user (U) already uses PALPAS on his user device (UD). He has an account at the PLS and the UD is registered at the PLS. The procedure to create a PASCO backup is described in the following and the corresponding data flow is illustrated in Figure 6.1.

1. U initializes the BD with a PIN.
2. The UD and the PLS perform a mutual authentication in which the UD uses its $sk_{\text{Auth,UD}}$.
3. The UD requests an authentication token t_{Auth} and otp_{UD} from the PLS.
4. The UD decrypts the locally stored PALPAS secret ps_{UD} with otp_{UD} by computing $ps = ps_{\text{UD}} \oplus otp_{\text{UD}}$. Then, it sends $ps = (seed, k_{\text{Data}})$ and t_{Auth} to the BD.
5. The BD randomly samples a one-time-pad key otp_{BD} and encrypts the PALPAS secret by computing $ps_{\text{BD}} = ps \oplus otp_{\text{BD}}$. It only keeps ps_{BD} and deletes ps. Then, it generates an authentication key pair and stores $sk_{\text{Auth,BD}}$. Afterwards, the BD sends $pk_{\text{Auth,BD}}$, t_{Auth}, and otp_{BD} to the PLS. The PLS verifies t_{Auth} and stores $pk_{\text{Auth,BD}}$ and otp_{BD}. Finally, the BD deletes otp_{BD}.

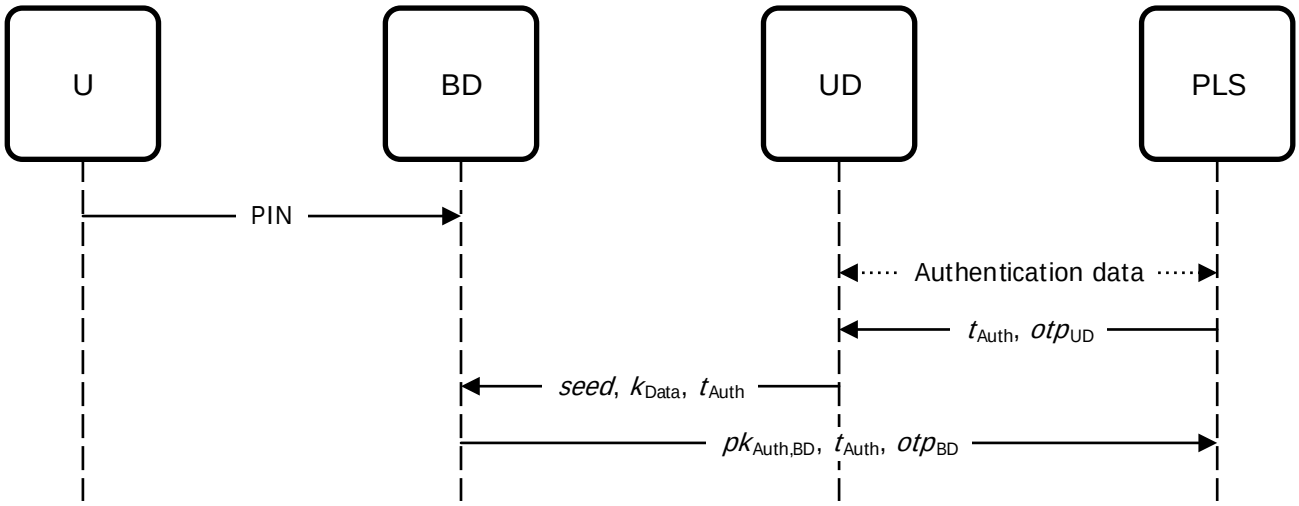


Figure 6.1: Data flow of the backup procedure.

An update of the BD is not necessary, even when the password portfolio of U is changed. Changing, adding, or deleting passwords only requires to update, store, or delete the related account data at the PLS. This is an integral part of PALPAS itself (cf. Section 5.1.3). As the PLS provides availability of the account data or the account data is stored on multiple PLSs (cf. Section 6.1.4), PASCO itself does not need to take care of it.

6.1.2 Data recovery from backups

Typical reasons for the loss of the PALPAS data are lost, stolen, or damaged devices as well as malware. To restore the PALPAS data on a user device, the user needs to have the BD and the corresponding PIN. The procedure is depicted in Figure 6.2 and works as follows:

1. U authenticates himself to the BD with his PIN.
2. The BD and the PLS perform a mutual authentication in which the BD uses its $sk_{\text{Auth,BD}}$.
3. The BD requests t_{Auth} and otp_{BD} from the PLS.
4. The BD decrypts the locally stored PALPAS secret ps_{BD} with otp_{BD} by computing $ps = ps_{\text{BD}} \oplus otp_{\text{BD}}$. Then, it transfers $ps = (seed, k_{\text{Data}})$ and t_{Auth} to the UD.
5. The UD randomly samples a one-time-pad key otp_{UD} and encrypts the PALPAS secret by computing $ps_{\text{UD}} = ps \oplus otp_{\text{UD}}$. It only keeps ps_{UD} and deletes ps. Then, it generates an authentication key pair and stores $sk_{\text{Auth,UD}}$. Afterwards, the UD sends $pk_{\text{Auth,UD}}$, t_{Auth}, and otp_{UD} to the PLS. The PLS verifies t_{Auth} and stores $pk_{\text{Auth,UD}}$ and otp_{UD}. Finally, the UD deletes otp_{UD}.

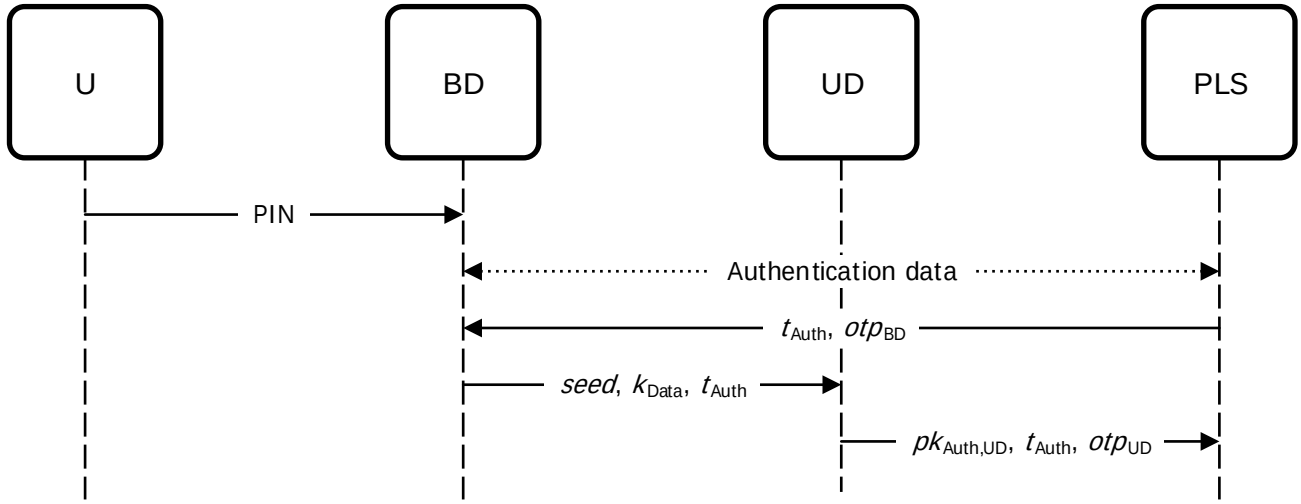


Figure 6.2: Data flow of the recovery procedure.

The UD has now access to the user's account at the PLS and can retrieve the account data for generating the passwords.

6.1.3 Revocation of backups

It is essential to place backups at various locations to protect them from loss and to enable an emergency access to passwords. However, this bears the risk that backups get lost, users cannot get them back, or just forget that they exist. An attacker might be able to compromise a stolen backup at some point in time and thus obtains the entire password portfolio of a user.

Like user devices, all existing backup devices are registered at the PLS with their individual pk_{Auth} and otp. This allows to revoke backup devices in the same way as user devices (cf. Section 5.1.4.3): The user deletes $pk_{\text{Auth,BD}}$ and otp_{BD} of a BD at the PLS. Now the BD can no longer access the PLS and requests data. Moreover, the deletion of otp_{BD} invalidates ps_{BD} on the BD. This means it is impossible to recover the PALPAS secret ps from the BD. Thus, the BD is useless and the passwords cannot get stolen. We refer to Section 6.4 for a detailed security evaluation for the revocation mechanism.

6.1.4 Recovery of server-side data

PASCO provides recoverability of the PALPAS data stored on user devices. So far, the recoverability of the account data stored on the PLS is to be guaranteed by the PLS provider. However, for users that do not want to rely on this assumption, we describe an alternative approach to protect the server-side PALPAS data from loss. The basic approach is to redundantly store the account data on multiple PLSs to mitigate the risk of loss. Besides loss protection, this also mitigates unavailability during a temporary outage or maintenance of a PLS.

Storing the data on multiple PLSs can be realized with minor adaption of the PALPAS client (PLC). Instead of storing the data on a single PLS, a PLC mirrors the account data to one or even more additional PLSs. The communication protocols and interfaces of the PLS providers do not need to be changed. The additional network traffic is negligible, because the account data only make up for a few kilobytes. To efficiently create individual authentication and encryption keys for different PLSs, we introduce a new scheme to generate these keys. As well, we show how a one-time-pad key can me masked in order to provide privacy protection.

In the original version of PALPAS k_{Data}, sk_{Auth} and the related pk_{Auth} are randomly generated. Instead of applying k_{Data} directly, it is used as input for a Key Derivation Function (KDF) to create PLS-specific encryption keys. Moreover, PLS-specific authentication keys are derived from a randomly generated secret k_{Auth}. As well, the OTP keys are masked before storing them on the PLSs.

This approach has two advantages: First, it protects the users' privacy because keys are not reused. Otherwise, collaborating PLS providers could identify users by comparing the authentication or OTP keys. Second, it does not require to store a multitude of keys on a user/backup device so it can be realized on a resource-constrained device like a smart card. Only a small random bit string needs to be additionally stored. In detail, the generation of PLS-specific encryption and authentication keys and the OTP masking work as follows.

PLS-specific encryption keys

When users use the PLC for the first time k_{Data} is randomly created. But, the PLC does not use the key directly, it creates a PLS-specific key $k_{\text{Data,PLS}} = \text{KDF}(k_{\text{Data}}, url_{\text{PLS}})$ on demand for each PLS, where url_{PLS} is the URL of the PLS. From $k_{\text{Data,PLS}}$ an encryption key $k_{\text{Data,PLS,Enc}}$ and a message authentication key $k_{\text{Data,PLS,Mac}}$ can be derived as in the original PALPAS. For setting up the PLC on other devices, users just transfer k_{Data} to a new device as done in the original version of PALPAS. The URLs of the PLSs can be included in this transfer or manually entered.

PLS-specific authentication keys

Instead of randomly generating an authentication key pair, the PLC randomly creates a secret k_{Auth} and stores it on the device like sk_{Auth} in the original PALPAS version. Then, it creates a PLS-specific secret $k_{\text{Auth,PLS}} = \text{KDF}(k_{\text{Auth}}, url_{\text{PLS}})$ for each PLS. $k_{\text{Auth,PLS}}$ serves as input for a PRG which is used for the key generation of PLS-specific authentication key pairs comprising $sk_{\text{Auth,PLS}}$ and $pk_{\text{Auth,PLS}}$. By storing k_{Auth} these keys can be regenerated at any time.

Note that in case of setting up the PLC on a further device an authentication token $t_{\text{Auth,PLS}}$ from each PLS must be requested. The tokens can be transferred together with the URLs of the PLSs, the $seed$, and k_{Data} by a file transfer or a QR code.

PLS-specific OTP key masking

When setting up a device, the PLC samples a random bitmask m in addition to the one-time-pad key otp that is used to encrypt the PALPAS secret on the device. Then, it creates a PLS-specific bitmask $m_{\text{PLS}} = \text{KDF}(m, url_{\text{PLS}})$. Instead of otp_{BD} (cf. Step 4, Section 6.1.1), the BD sends $otp_{\text{BD,PLS}} = otp_{\text{BD}} \oplus m_{\text{PLS}}$ to the PLS. m is stored on the device while m_{PLS} is recomputed on demand to recover otp_{BD} from $otp_{\text{BD,PLS}}$.

6.2 Emergency access to backups

This section describes our solution which solves the password accessibility problem. We extend PASCO to provide an emergency access for backups. User can authorize trusted persons to obtain a set of pre-defined passwords from a backup in urgent or emergency situations.

The extension of PASCO consists of two parts: First, in addition to storing the PALPAS data, the BD now also implements the PALPAS password generation procedure (cf. Section 5.1.2). Second, the PLS is equipped with a fine-grained access control system for the account data. For each pk_{Auth}, a user can specify different access rules. While, one pk_{Auth} may have access to all data, another pk'_{Auth} can only access the account data for the user's mail account.

We describe the creation and management of backup devices with emergency access in Section 6.2.1. Particularly, we explain how to control which accounts can be accessed with a specific BD. Then, we describe the procedure for an emergency access in Section 6.2.2.

6.2.1 Creation of backups with emergency access

The procedure for creating BDs with emergency access is nearly the same as described in Section 6.1.1. It only differs in the second step, where the UD requests t_{Auth}. The request is supplemented by an access control list (ACL). The ACL is basically a list of account data identifiers (cf. Section 5.1.3). The $pk_{\text{Auth,BD}}$ registered using t_{Auth} in Step 4 is later only granted access to the account data defined by the ACL. The ACL for each pk_{Auth} can be modified at any time by the user through an authorized device. For instance, the user can enable the access to his social media account during vacation and disable it afterwards. Both can be done without having physical access to the BD. Moreover, the user can also revoke the BD to invalidate it completely as described in Section 6.1.3.

The BD can simultaneously act as a backup and an emergency password generation device. Thus, depositing a single BD at a friend's place is sufficient. To provide both features, a BD is equipped with multiple authentication keys. One pk_{Auth} is allowed to request an authentication token t_{Auth} as needed for the restoring procedure (cf. Section 6.1.2). Another pk'_{Auth} can only retrieve certain account data and is used for the emergency access. To equip a BD with multiple authentication keys, the creation procedure described in Section 6.1.1 is performed multiple times with a different t_{Auth}, pk_{Auth}, ACL, and a different PIN. Depending on the PIN, a BD uses the corresponding sk_{Auth} for the authentication at the PLS. In accordance with the ACL associated to pk_{Auth}, the PLS allows to request an authentication token or only certain account data, e.g. for the mail account.

6.2.2 Access backups in case of emergency

We envision that the user (U) has deposited a PASCO backup (BD) at a friend's place. Allowing the friend, referred to as the backup holder (H), to create the user's password for an account works as follows. The data flow is depicted in Figure 6.3.

1. U tells H the emergency PIN of the BD and the URL of the service where H should access the user's account.
2. H uses the PIN to authenticate himself to the BD and also transfers url_c to the BD.
3. The BD and the PLS perform a mutual authentication in which the BD uses its $sk_{\text{Auth,BD}}$.
4. The BD requests it one-time-pad key otp_{BD} from the PLS. Then, it decrypts the locally stored PALPAS secret ps_{BD} with otp_{BD} by computing $ps = ps_{\text{BD}} \oplus otp_{\text{BD}}$. The BD now has $ps = (seed, k_{\text{Data}})$.
5. The BD derives $k_{\text{Data,Mac}}$ from k_{Data} and computes $id_c = \text{HMAC}(k_{\text{Data,Mac}}, url_c)$. Then, it sends id_c to the PLS and requests the corresponding account data.
6. The PLS responds with the related $data_c$.
7. The BD derives $k_{\text{Data,Enc}}$ from k_{Data} and uses it to decrypt $data_c$. The BD has now $salt_c$, prd_c, and the username un_c. Then, it generates the password pw_c using the $seed$, $salt_c$, and prd_c. Finally, the BD hands pw_c and un_c over to H. H can now browse the service and log in to the user's account.

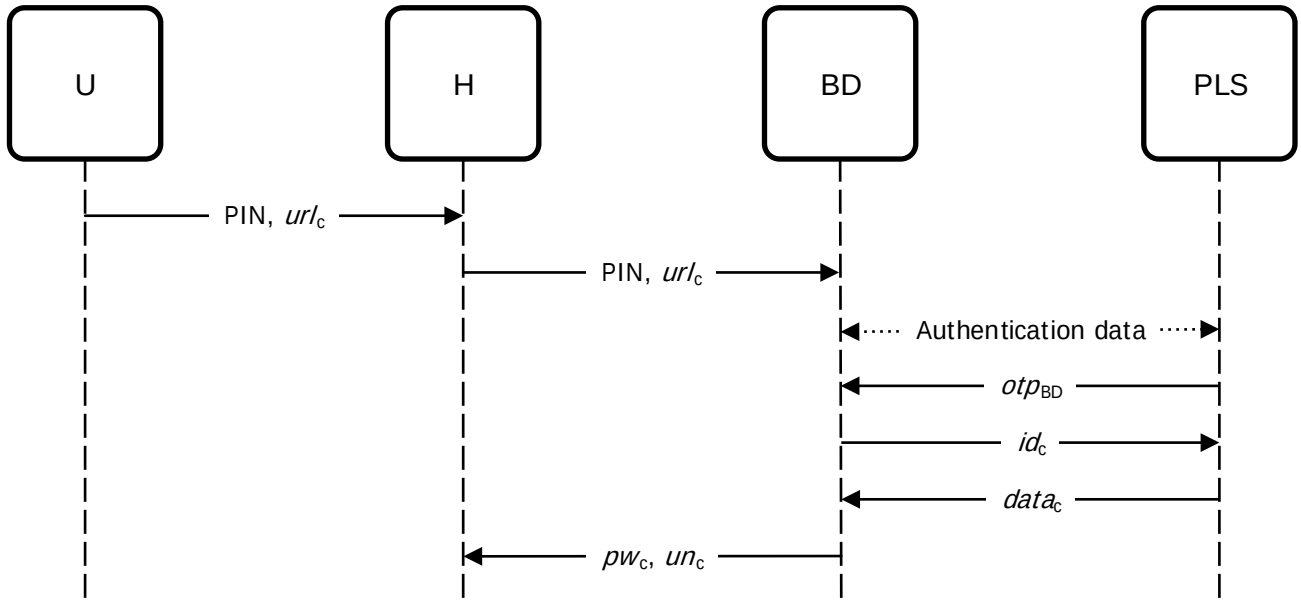


Figure 6.3: Data flow of the emergency access procedure.

6.3 Implementation

In this section, we present an implementation of our solution as well as demonstrate the realization of PASCO with an off-the-shelf smart card.

We start by explaining the implementation of a PASCO backup device using a smart card in Section 6.3.1. Then, we present a PASCO client in Section 6.3.2. Finally, we explain how the data flows of the conceptual description of PASCO are realized in practice in Section 6.3.3.

6.3.1 Backup device

In Section 6.1, we described the BD to be a tamper resistant device that provides secure storage, user authentication, and basic cryptographic algorithms. We realized the BD with a smart card. A smart card fulfills the pursued security features. Namely, it securely stores the PALPAS secret and protects it from unauthorized access and other threats like malware. Consequently, a smart card is a valid and practical realization of PASCO.

We used a Java card (NXP J3D081) and developed a Java Card Applet (Classic Platform) that implements the backup and the emergency access feature. The PALPAS secret is stored in the secure memory of the card and is protected by a PIN. For the data exchange we built on the standard Application Protocol Data Units (APDU) specified by the ISO 7816 [127].

6.3.2 Client application

In this section, we describe the PASCO client. It runs on user devices and implements the creation, recovery, and emergency access procedure. For the backup procedure it acts as an intermediate between the backup device and the PLS. With respect to the backup creation and data recovery, the PASCO client is installed in parallel to the PALPAS client (PLC). For the emergency access scenario it runs as a stand-alone application on the device of the backup holder.

The architecture of the PASCO client is depicted in Figure 6.4. It consists of six main components which we briefly describe in the following.

- *Core*: It manages the client and instantiates the other components.
- *User interface*: It provides a graphical user interface for a user. It allows to start the creation, recovery, and emergency access procedure as well as to configure the client.

- *Smart card connector*: It provides the communication to the smart card (BD) and implements the APDUs.
- *PLC connector*: It provides data exchange between the PASCO client and the PLC. It is used to access the PALPAS data such as the authentication key and certificate for the PLS.
- *PLS connector*: It establishes the communication to the PLS. The connector performs the authentication to the PLS and obtains the necessary certificates and signatures from the PLC connector (in case of the creation procedure) or smart card connector (in case of the recovery and emergency access procedure). It is also used to retrieve the account data, authentication tokens, and one-time-pad keys.
- *PALPAS password generator*: It implements the PALPAS password generation procedure (cf. Section 5.1.2) and is used in case of the emergency access procedure.

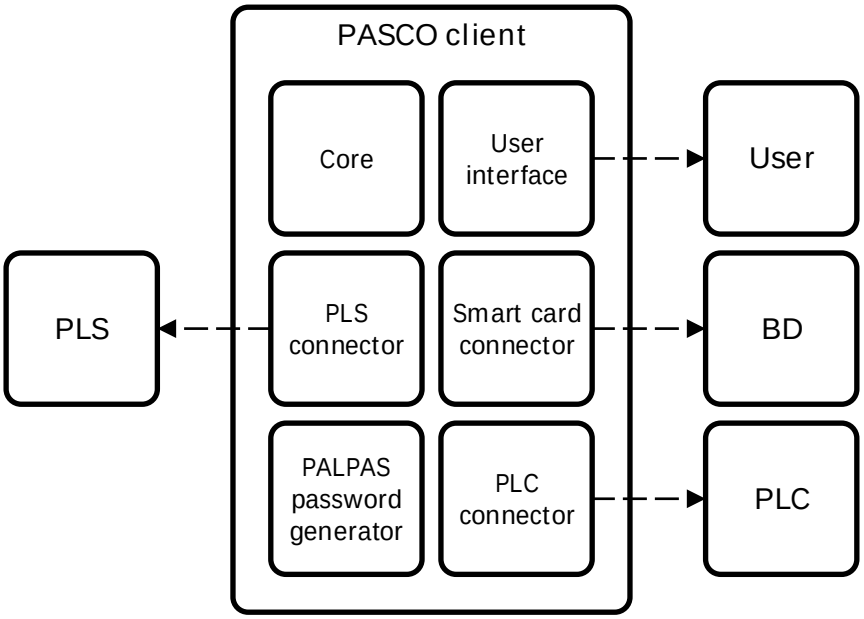


Figure 6.4: Architecture of the PASCO client.

We implemented the PASCO client in Java and used the Java Smart Card IO for the card communication. The implementation is available at [H26].

6.3.3 Details of operation

Smart cards and Internet services like the PLS use different communication protocols. Therefore, they cannot communicate directly with each other. A user device must act as an intermediate to translate between the different protocols and data formats. Moreover, smart cards have limited storage capacity and computation power so that some operations must be performed by user

devices. Because of this, the communication flows of our implementation slightly differ from the conceptual description of our solution presented in Section 6.1 and 6.2. We explain the differences in the following.

Authentication at the PLS

Because the BD and the PLS cannot communicate directly, the UD establishes the connection to the PLS in the restore and emergency access procedure. The UD also performs the authentication to the PLS on behalf of the BD using TLS with client authentication. This works as follows: The UD connects to the PLS which requests a certificate for authentication. The UD obtains the certificate from the BD and transfers $ct_{\text{Auth,BD}}$ to the PLS. Moreover, during the TLS handshake the UD sends a hash of all handshake messages to the BD. The hash is signed by the BD using $sk_{\text{Auth,BD}}$. This signature proves the possession of $sk_{\text{Auth,BD}}$ to the PLS.

For further details, we refer to RFC 5246 [71]. Note that this procedure does not raise security issues, because the security-sensitive secret authentication key $sk_{\text{Auth,BD}}$ never leaves the BD and the PIN is required to access the certificate on the BD and to generate the signature.

Shared password generation

Besides limited communication capabilities, smart cards have resource restrictions with respect to storage capacity and computation power. Therefore, the password generation in the emergency access procedure is jointly done by the BD and the PASCO client running on the device of the backup holder. The password generation (cf. Section 5.1.2) is done as follows:

1. *Random generation*: The first part of the password generation, the generation of the random, is done by the BD. The BD decrypts $ps = ps_{\text{BD}} \oplus otp_{\text{BD}}$ to obtain the $seed$ and k_{Data}. It then decrypts $data_{\text{c}}$ to get $salt_{\text{c}}$, prd_{c}, and un_{c}. Using the $seed$ and $salt_{\text{c}}$, it computes the random, and hands it along with prd_{c} and un_{c} over to the PASCO client.
2. *Password generation*: The second part is done by the PASCO client. It takes as input the random and the prd_{c} from the BD and computes the password.

The joint approach is necessary, because the processing of XML-encoded PRDs as used by PALPAS could not be realized on the smart card. Yet, this does not pose security issues, because the $seed$ and k_{Data} never leave the smart card.

6.4 Security evaluation

We evaluate the security of PASCO in this section. At the beginning, we extend in Section 6.4.1 our current system and attacker model to take PASCO backups into account. Particularly, we describe the additional attacker's capabilities and evaluate them in Section 6.4.2. We show that the attacker is not able to obtain the user's password portfolio from a PASCO backup.

6.4.1 Extended system and attacker model

We extend our system model described in Section 3.1 and 5.3.1. The user $\mathcal{U}$ has created a PASCO backup which contains the data described in Section 6.1. We assume that the BD was securely created and the BD is protected by a PIN. The BD is registered at the user's account at the PALPAS server $\mathcal{P}$. The access to this account requires that the BD authenticates itself to $\mathcal{P}$. This is done by the device-specific authentication key $sk_{\text{Auth,BD}}$.

Attacker goal

We again consider an attacker $\mathcal{A}$ that aims at obtaining the PALPAS data of $\mathcal{U}$ in order to generate his entire password portfolio (cf. Section 5.3.1). This time $\mathcal{A}$ focuses on obtaining this data from the BD of $\mathcal{U}$.

Attacker capabilities

With respect to the new goal of obtaining the user's PALPAS data from his BD, we consider that $\mathcal{A}$ has the following two additional capabilities:

AC8 $\mathcal{A}$ is able to obtain the BD of $\mathcal{U}$.

AC9 $\mathcal{A}$ is able to circumvent the PIN protection of the BD after same time and thus obtains the encrypted PALPAS secret ps_{BD} and the secret authentication key $sk_{\text{Auth,BD}}$.

Attacker limitations

We consider the same attacker's limitations as described in Section 3.1 and 5.3.1. In brief, we exclude trivial attacks, attacks that cannot be prevented by technical means, and attacks that are not specific to passwords. We provide a more general security evaluation of the usage of smart cards in [H21].

6.4.2 Attack scenarios

In this section, we evaluate the new attacker's capabilities defined in the previous section. We show that $\mathcal{A}$ is not able to obtain the PALPAS data of $\mathcal{U}$ from his BD and therefore cannot generate his password portfolio.

6.4.2.1 Scenario 1

In the first attack scenario, we evaluate the attacker's capability of obtaining the BD of $\mathcal{U}$ (cf. AC8). $\mathcal{A}$ for instance has stolen it from a trusted person of $\mathcal{U}$. We show that $\mathcal{A}$ is not able to generate any passwords with the stolen BD.

The BD is protected with a PIN. The PIN has a retry counter which only allows five PIN entries. Then, the entire data on the BD is deleted. Assuming $\mathcal{U}$ has chosen a PIN with four digits, the probability that $\mathcal{A}$ successfully guesses the PIN after five attempts is $5/10.000 = 0.05\%$. This is very unlikely, so that $\mathcal{A}$ does not obtain the PIN and therefore cannot generate any passwords from a stolen BD. After five wrong attempts the PALPAS data on the BD is deleted so that $\mathcal{A}$ cannot obtain it anymore.

6.4.2.2 Scenario 2

In the second attack scenario, we consider that $\mathcal{A}$ is able to circumvent the PIN protection after some time and to extract ps_{BD} and $sk_{\text{Auth,BD}}$ from the BD (cf. AC9). $\mathcal{A}$ might achieve this for instance by a side-channel attack [64, 116, 188]. We show that PASCO's revocation feature (cf. Section 6.1.3) prevents $\mathcal{A}$ from obtaining the user's PALPAS data and therefore cannot generate his password portfolio.

$\mathcal{A}$ cannot decrypt the PALPAS secret ps_{BD} without the corresponding one-time-pad key otp_{BD} which is stored at $\mathcal{P}$. Assuming that $\mathcal{U}$ revokes the BD before $\mathcal{A}$ is able to circumvent the PIN protection, the public authentication key $pk_{\text{Auth,BD}}$ and otp_{BD} are deleted from $\mathcal{P}$. Therefore, it is impossible for $\mathcal{A}$ to recover ps_{BD} because it is a one-time-pad encryption yielding information-theoretic security [22, 163]. To this end, $\mathcal{A}$ cannot obtain the PALPAS data of $\mathcal{U}$ and consequently cannot generate his password portfolio.

6.5 Conclusion

In this chapter, we presented PASCO which creates backups of the PALPAS data with the following three properties: First, backups do not have to be updated when users' password portfolios change. Second, backups can be revoked even without physical access in an information-theoretical secure way. Third, backups provide a fully controllable emergency access. These features address the key concerns of users regarding loss and inaccessibility of passwords [160, 208]. Besides providing access to passwords in urgent or emergency situation, PASCO backups can also be used to cover the event of death. By means of PASCO, users can make sure that their spouse get access to their passwords in case of death. So far, it was practically impossible for users to place backups of preserved passwords at secure locations and to keep them up-to-date simultaneously.

The practicality of PASCO has been demonstrated by a realization with an off-the-shelf smart card. A smart card can be easily deposited at secure locations such as a safe or a friend's place for recovery and emergency access purposes.

Apart from that, users can use a PASCO backup device as a secure mobile password generator. This is particularly a major advantage in the mobile environment. Storing the PALPAS data on a PIN-protected smart card within its protected memory provides much more security compared to security features available on mobile devices. The smart card that we used has a contactless interface and thus it is capable to communicate with NFC-enabled mobile devices (cf. [H17, H22]). Using the smart card as a password generator allows users to literally have their passwords in their wallet. With the two properties of PASCO, update-tolerance and revocability, users do not have to update the card when their password portfolio changes and they are able to revoke the smart card in case of loss at any time.

PASCO is the third part of PAS. It solves the password recoverability and accessibility problem. Based on PALPAS and PASCO, PAS provides the first full-fledged solution for the password preservation problem. PAS ensures the confidentiality, availability, recoverability, and accessibility of preserved passwords. In the subsequent Chapter 7 we present the fourth and last part of PAS. It provides automatic password changes and enables users to change their passwords regularly. The update-tolerance of PASCO backups releases users from updating their backups every time a password has been changed.

7 Automatic and autonomous password change

In Chapter 3, it was shown that changing passwords is a burdensome task. Besides the number of passwords, users are challenged with different password interfaces and procedures for password change implemented by services.

In this chapter, we present the fourth and last part of PAS. It solves the password change problem by enabling an automatic and autonomous password change. Users neither need to create new passwords nor to log in to their accounts. PAS realizes this by making the password requirements, interfaces, and procedures of services available to password assistants, which perform the password changes on behalf of users. We provide a conceptual description of our solution in Section 7.1.

Then, we describe the four building blocks of our solution: First, Password Policy Descriptions (PPD), a standardized description of password interfaces and procedures which can be processed by password assistants (cf. Section 7.2). Second, a tool to easily create PPDs for services that already exist on the Internet (cf. Section 7.3). Third, the distribution of PPDs to make them available to password assistants (cf. Section 7.4). Fourth, strategies for password assistants to autonomously change passwords (cf. Section 7.5).

We present an implementation of our solution in Section 7.6. We demonstrate the generation of PPDs for popular services. Furthermore, we complement our solution by a password assistant that makes use of PPDs and is capable of changing passwords automatically.

Moreover, we present easy-to-use passwords in Section 7.7, a further application of PAS. They solve the problem of using passwords on a device that does not belong to a user. In this situation entering passwords is inconvenient, error-prone, and dangerous because of malware. PAS sets a temporarily easy-to-use password for an account and replaces it by an attack-resistant one after the service usage on the other device automatically.

The contributions of this chapter were published as part of [H5]. This chapter extends the published contributions by the strategies for autonomous password changes and the concept of easy-to-use passwords.

7.1 Conceptual description

This section provides a conceptual description of our solution that solves the password change problem. We expand password assistants with the ability to automatically and autonomously change passwords. This enables assistants to perform the third password task of regularly changing passwords and immediately after a compromise of passwords is detected for users.

The entities involved in our solution are a user, a password assistant, a repository for Password Policy Descriptions (PPD), and a service. The user has an account at the service and his account password is preserved by his password assistant. The password assistant takes over the task of password changes and releases the user from any actions. The repository provides the PPD of the service. A PPD describes the service's password implementation including its password requirements, interfaces, and procedures. In particular, it specify how to change a password at the service. The general application flow of the automatic and autonomous password changes is illustrated in Figure 7.1 and briefly described in the following:

1. The password assistant uses the URL of the service to retrieve its PPD from the repository.
2. The password assistant generates a new password in accordance with the current password requirements of the service, which are described in the policy.
3. The password assistant logs in to the user's account at the service and changes the account password. This is done through the service's password interfaces and corresponding procedures described in the PPD. Finally, the password assistant preserves the new password.
4. The password assistant regularly changes the password with respect to the security level of the password as well as immediately after it detects a compromise of user's password.

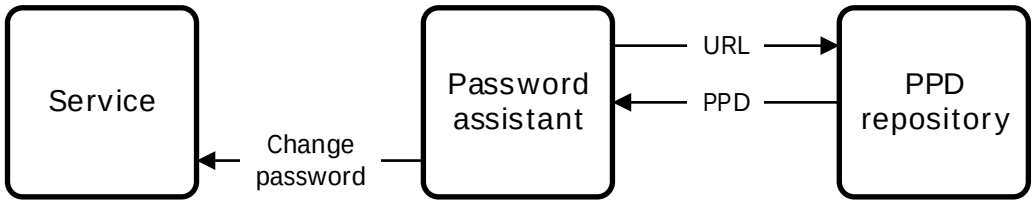


Figure 7.1: Data flow of the automatic password change procedure.

The fourth part of PAS that enables these automatic password changes consists of four building-blocks which are presented in the following Section 7.2 to 7.5.

7.2 Uniform description of password policies

In Section 3.5, we saw that services have different password interfaces and procedures for the usage of passwords at their websites. This makes it problematic for password assistants to change passwords automatically.

In this section, we solve this problem by presenting a uniform description of password interfaces and procedures. It allows to describe the different password implementations of services, e.g. for changing a password, in a standardized format. Such descriptions enable password assistants to automate password tasks. In the following, we introduce our uniform description language and provide an example of a description.

Password Policy Markup Language

We introduce the Password Policy Markup Language (PPML) which enables the definition of Password Policy Descriptions (PPD) for services. A PPD is a standardized description of the service's password implementation, including password requirements, interfaces, and procedures. The objective of a PPD is twofold. First, it provides all information to enable password assistants to automate the first and third password task (cf. Section 3.3.1 and 3.3.3). More precise, a PPD facilitates password assistants to (1) generate a valid password, (2) log in to an account, (3) change a password, and (4) reset a password on behalf of users. Second, if such an automation is not possible for a service (e.g. due to a CAPTCHA), a PPD provides all information to assist users with performing the password tasks manually. Namely, a PPD provides the URLs of the password interfaces so that password assistants can directly guide users to these interfaces. Therefore, users do not need to finding them on the service's website by hand. As seen in Section 3.5, finding the password interfaces of services is very inconvenient and challenging.

We conducted an analysis of password interfaces and procedures of 200 representative services[1] in order to develop a comprehensive and representative specification for PPDs. Based on these results, we identified common patterns and created a universal description for password interfaces and password procedures.

As illustrated in Figure 7.2, a PPD consists of three parts: First, the metadata which is used to identify and manage PPDs properly. Second, the password requirements of the service and third its password interfaces and procedures.

[1] The Alexa Top 500 US list [8] reduced by websites with pornographic and illegal content, non-English websites, and websites that do not have or allow the creation of online accounts (e.g. banking websites). The list of services is available at [H26].

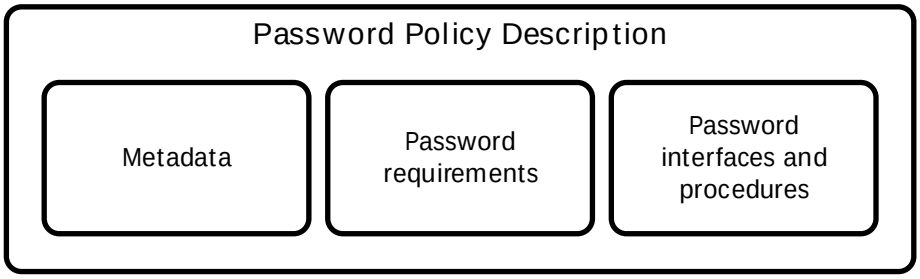


Figure 7.2: Structure of a PPD.

We encode PPDs in XML and provide a full XML Schema at [H26]. XML is well-specified and supported by many programming languages, which enables an easy integration of PPDs into password assistants. In practice, an XML-encoded PPD has a file size of a few kilobytes.

A PPD is represented by a XML element <ppd>. It has three attributes which form the metadata: `url`, `version`, and `ppmlVersion`. The `url` specifies the URL of the service associated with the PPD. The `version` number allows PAS to differentiate between versions of a PPD and to update it if required. The `ppmlVersion` specifies the PPML version that was used to describe the PPD.

In the following, we describe the second and the third part of a PPD in detail. An example of a PPD can be found in Listing 7.1 (Page 144).

Password requirements

A PPD allows to specify the same password requirements as a PRD (cf. Section 4.2). As PRDs focus on password generation and PPDs on password management (in particular password change), a PPD additionally supports to specify the expiration of passwords:

- *Expiration*: The <expires> element defines the number of days until a password expires and should be updated. We leave it to password assistants to keep track of the dates and to prompt users to change their passwords in time or do it autonomously.

Password interfaces and procedures

In addition to the password requirements, a <ppd> element contains a <service> element that describes the password interfaces and procedures of a service. As depicted in Figure 7.3, it provides information about the registration, login, password change, and password reset. These password interfaces are represented by a <register>, <login>, <passwordChange>, and <passwordReset> element, respectively.

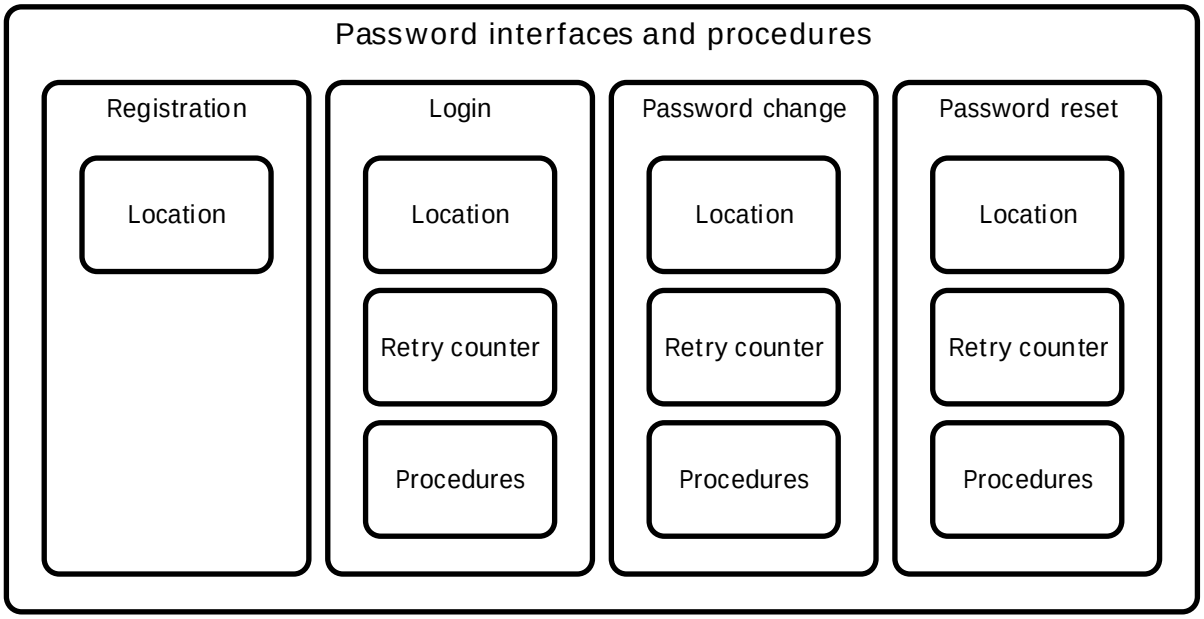


Figure 7.3: Structure of the password interfaces and procedures definition.

All elements contain a <location> element which is described in the following:

- *Password interface location*: The <location> element contains the URL of the HTML form for the registration, login, password change, and password reset at the service's website. This information is intended to directly guide users to these password interfaces, i.e. HTML forms, to access them manually. The information for accessing these interfaces automatically is part of the <procedures> element (see below).

In addition, the <login>, <passwordChange>, and <passwordReset> elements contain a <retryCounter> and <procedures> element which are described in the following:

- *Retry counter for passwords*: The <retryCounter> element specifies the number of attempts to enter passwords. In case of the login it defines the number of possible login attempts. For password changes it defines how often users can enter an incorrect old password and for password resets how often users can answer security questions (or enter recovery information) before an account gets disabled.

- *Password procedures*: The <procedures> element describes the password procedure for the login, password change, and password reset of a service. A procedure is a set of instructions telling password assistants what actions need to be performed and how the execution of these actions can be verified. For example, a login procedure describes how to log in to an account and how to verify that the login was successful. We provide more technical information about procedures in the following.

Procedures

Due to the different technologies that services use for their password interfaces, we annotate the procedures with an additional technology type. We define four types: HTTP, HTML, JavaScript, and Extended JavaScript. Password procedures are represented by the elements <*`LoginRoutine`>, <*`PasswordChangeRoutine`>, and <*`PasswordResetRoutine`> element where '*' indicates the type (`http`, `html`, `js`, or `extentedJS`, e.g. <`httpLoginRoutine`>).

The basis for all these procedures are HTTP POST and GET commands [81] which are used to interact with the service's password interfaces and to perform a login, password change, or password reset. A POST or GET command is defined by a <`post`> or <`get`> element, respectively. Both contain an <`url`> element which defines the target URL of the command. Furthermore, both include an <`assert`> element which is used to define a list of assertions that need to be verified in order to check whether the POST or GET command was performed successfully or not. We provide more information about assertions later. The <`post`> element additionally contains a list of <`data`> elements representing the data which is sent to a service within the POST command (e.g. the username and password within a login procedure). In the following, we describe the four technology types in detail:

- *HTTP*: A HTTP-based procedure contains a list of <`post`> and <`get`> elements. Such a procedure is used if the password interfaces of a service can be accessed by using plain HTTP GET or POST commands.
- *HTML*: A HTML-based procedure extends the HTTP procedure by a <`form`> element. The element consists of a list of <`selector`> elements which define identifiers to select HTML input fields and enter values in an HTML form. An HTML-based procedure must be used when HTML forms contain hidden input fields with random values like a session identifier.
- *JavaScript*: A JavaScript-based procedure provides the same functionality as an HTML procedure. The elements of the type `js` only indicate that the password assistant processing the PPD needs to support JavaScript in order to access the password interfaces of the service. This is necessary when services use JavaScript to manipulate the input of an HTML form before submitting it. For instance, services hash the password by JavaScript on the client-side before submitting it.
- *Extended JavaScript*: This type extends the HTTP procedure and additionally defines a <`javascript`> element which contains plain JavaScript code. In case that the aforementioned types are not applicable, developers can use this type to implement the procedure using the complete functionality of JavaScript. A password assistant needs to browse the password interface and execute the JavaScript in order to perform the procedure.

Listing 7.1 shows an example for an HTML-based login and password change procedure. The `<htmlPasswordChangeRoutine>` has an attribute `login` which refers to the login procedure. This allows to reuse procedures and makes the creation of PPDs easier and more efficient.

Assertions

It is essential that PAS can verify the correct execution of a procedure, e.g., verify that a login was successful. Therefore, a procedure contains one or more assertions which a password assistant needs to verify. PPML defines the following types of assertions:

- *Existence of a cookie*: A certain cookie is set after performing the procedure.
- *Non-existence of a cookie*: A certain cookie is not set after performing the procedure.
- *Content*: The service responses with a certain message, e.g. “the password is wrong”.
- *Location*: The response from the service redirects to a certain URL.

The procedures of PPML are a powerful framework to describe password interfaces and procedures of services. They are highly flexible and support the wide range of different technologies and methods of password implementations used by services.

Application

A PPD covers all situations of password usage at a service, namely registration, login, password change, and password reset. It facilities a comprehensive assistance in creating and managing passwords. The information about the login simplifies the daily usage of passwords. It also improves the security of automatic logins, because password assistants do not need to use heuristics to find the login forms, which are vulnerable to phishing attacks [207]. The information about the password change facilitates an automatic password change. Apart from that, also the details about the registration and password reset improves the ease of use of passwords.

Finding the registration form of services is challenging. Often users need to click first on *Sign in* and then on buttons like *Create account*. Using a password assistant, users simply select the service in the assistant, e.g by entering its URL. The password assistant opens the registration form and also directly generates an optimal password for the account.

The information about the password reset are useful in various situations. First, it can be used for password changes if the actual password change procedure cannot be realized for a service. Second, it can be used to cope with a compromised account when the attacker already changed the password and the user cannot access his account anymore.

```
<ppd url="https://www.example.com" version="1.0" ppmlVersion="1.0">
  <characterSets>
    <characterSet name="Letters">
      <characters>abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ</characters>
    </characterSet>
    <characterSet name="Numbers">
      <characters>0123456789</characters>
    </characterSet>
  </characterSets>
  <properties>
    <characterSettings>
      <characterSet name="Letters" />
      <characterSet name="Numbers" />
    </characterSettings>
    <minLength>10</minLength>
    <maxLength>20</maxLength>
  </properties>
  <service>
    <register>
      <location>https://www.example.com/register</location>
    </register>
    <login>
      <location>http://www.example.com/login</location>
      <retryCounter>3</retryCounter>
      <procedures>
        <htmlLoginRoutine>
          <get>
            <url>https://www.example.com/login/</url>
            <assert>
              <url><prefix>https://www.example.com/login/</prefix></url>
            </assert>
          </get>
          <form>
            <selector>
              #main form[action^="https://www.example.com/login/"]
            </selector>
            <element>
              <selector>#username</selector><value>{{username}}</value>
            </element>
            <element>
              <selector>#password</selector><value>{{password}}</value>
            </element>
            <assert>
              <select><selector>#userMenu</selector></select>
```

```
            </assert>
          </form>
        </htmlLoginRoutine>
      </procedures>
    </login>
    <passwordChange>
      <procedures>
        <htmlPasswordChangeRoutine login="htmlLoginRoutine">
          <get>
            <url>https://www.example.com/account/</url>
            <assert>
              <url><prefix>https://www.example.com/account/</prefix></url>
            </assert>
          </get>
          <form>
            <selector>
              #main form[action^="https://www.example.com/account/"]
            </selector>
            <element>
              <selector>#password</selector>
              <value>{{password}}</value>
            </element>
            <element>
              <selector>#newPassword</selector>
              <value>{{newPassword}}</value>
            </element>
            <element>
              <selector>#confirmNewPassword</selector>
              <value>{{newPassword}}</value>
            </element>
            <assert>
              <select><selector>div.success</selector></select>
            </assert>
          </form>
        </htmlPasswordChangeRoutine>
      </procedures>
    </passwordChange>
    <passwordReset>
      <location>https://www.example.com/account/recovery</location>
    </passwordReset>
  </service>
</ppd>
```

Listing 7.1: PPD for a fictitious service using HTML procedures.

7.3 Tool-based creation of password policy descriptions

In the previous section, we introduced PPDs to describe the wide diversity of password interfaces and procedures at services in a uniform way. To enable password assistants to use PPDs in practice, the PPDs for existing services are required. However, manually creating the PPDs for services is inconvenient.

In this section, we solve this problem. We present the Password Policy Description Recorder (PPDR), an application that records password interfaces and procedures of a service and creates the corresponding PPD automatically. The PPDR makes the creation of PPDs as easy as possible and is particularly designed for non-technical users. Users just need to perform a password procedure, e.g. changing an account password, in a web browser to create PPDs. In the following, we explain the creation of PPDs using the PPDR in detail.

The PPDR is realized as a browser extension. It guides users through the PPD generation process and records their interactions with a service's website. The general creation process of a PPD describing the password change procedure consists of six steps (cf. Figure 7.4).

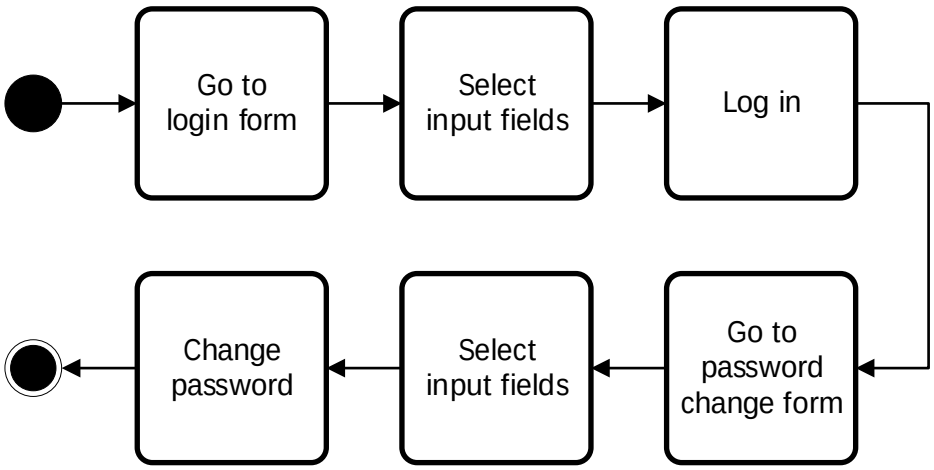


Figure 7.4: Execution steps of the PPD generation procedure.

The PPDR shows messages to users with detailed instructions for each step. Users need to confirm each step by pressing a button. In the following, we describe the six steps which users need to perform:

1. Browse to the login form at the service's website.
2. Mark the username and password input field of the login form.
3. Enter username and password and log in to the account.
4. Browse to the password change form of the service.

5. Mark the input fields (old, new, confirm password) of the password change form.
6. Enter the old/new/confirm password and change the password of the account.

The PPDR records all links and input fields that are clicked and selected by a user while performing these steps. Based on this information the PPDR creates the PPD. In the following, we provide further technical details about the PPDR.

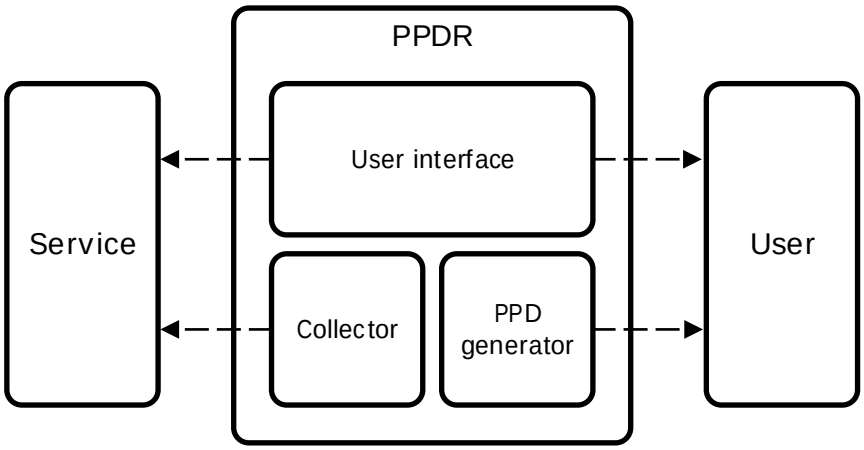


Figure 7.5: Architecture of the PPDR application.

As depicted in Figure 7.5, the PPDR consists of three main components.

- *User interface*: It shows the instructions to the user and controls the progression of the steps. Moreover, it automatically selects the input fields of the login and password change form. This simplifies Step 2 and 5 so that the user only needs to verify the selection and the PPD generation becomes even more convenient. The selection is done by highlighting the input fields with different colors. Incorrect selection can be corrected by the user.
- *Collector*: It records the user interactions with the website. In case the user clicks on a link or selects an input field, the collector tries to determine the HTML `id` attribute of the link/field. This provides the information for the procedures defined in the PPD.
- *PPD generator*: It generates the PPD of the service based on the information provided by the collector. The password procedure is generated as an extended JavaScript procedure (e.g. <`extenedJSPasswordChangeRoutine`>, cf. Section 7.2). After the generation, the PPD generator delivers the PPD to the user.

In Section 7.6.1, we present the generation of PPDs of popular services using our PPDR.

7.4 Distribution of password policy descriptions

To use PPDs in practice, they must be available to password assistants. This can be realized using the same two approaches that we already developed for PRDs (cf. Section 4.4). First, a central service that distributes PPDs. Second, an alternative decentralized approach in which services themselves provide their PPDs to password assistants.

7.4.1 Service-independent centralized solution

In order to distribute PPDs, we realized the Password Policy Description Distribution Service (PPDDS), which is available at [H26]. Its implementation is based on the PRDDS that we introduced in Section 4.4.1.

Like the PRDDS, the PPDDS allows users to submit PPDs of unknown services. Thus, PPDs must be created only once and can be made available through the PPDDS to all Internet users. To prevent an attacker from adding manipulated PPDs, a submitted PPD is manually evaluated and automatically verified by four sanity checks. One attack for instance is changing the URL of the login procedure so that usernames and passwords are send to the attacker's server.

The first sanity check verifies the password requirements stated in the PPD. This is done in the same way as described in Section 4.4.1. The second check verifies that the URLs for the password interfaces and procedures stated in the `<service>` element fit to the URL of the service associated with the PPD (the `url` attribute in the `<ppd>` element). This mitigates the risk of URLs pointing to other servers. The third sanity check verifies the URLs against the *Google Safe Browsing* service [99]. This detects URLs pointing to malware or phishing websites. In the last sanity check the TLS certificates used by the servers available under the URLs are evaluated. They must be the same. This detects an attacker that has control of a subdomain of the service. For instance, the actual service is available at `https://example.org` and the attacker is in control of `https://attacker.example.org`. A prominent example for this case are blogs hosted by WordPress. The PPDDS also checks the validity of the certificates as well as their trustworthiness by using the *ICSI Certificate Notary* service [125].

The PPDDS also provides a history and change log of each PPD. Erroneous PPDs can be reported by users through a feedback system.

7.4.2 Privacy-preserving decentralized solution

Such as for PRDs, we propose to use the *well-known location* scheme [171] to make the PPD of a service available to password assistants. The PPD of a service is then available at the URL `https://www.example.org/.well-known/ppd.xml`. As discussed in Section 4.4.2, this avoids the privacy and trust issue comparison to the centralized solution.

However, such a decentralized distribution relies on the cooperation of services and would entail a presumably long transition time. This gap is bridged by our PPDDS. In practice, password assistants should first try to receive a PPD directly from a service. If unavailable, password assistants should try to fetch the PPD from the PPDDS.

7.5 Strategies for autonomous password changes

To achieve the goal of complete automation, strategies for autonomous password change are required. Password changes are necessary on a regular basis and immediately after password compromise. We present two password change strategies to realize this.

First, we present a proactive strategy for autonomous password changes in Section 7.5.1. It tackles undetected password breaches at services. More precise, an attacker has stolen the password database from a service unnoticed and tries to obtain the passwords through an offline attack. Second, we describe in Section 7.5.2 a reactive strategy that changes passwords immediately after password compromise . It addresses detected password breaches at services and unauthorized access to user accounts.

7.5.1 Proactive password change strategy

Password breaches are often not immediately detected by services themselves or just not made public. Users often only become aware of password breaches after the stolen passwords are put up for sale on the black market. To invalidate stolen passwords before an attacker can compromise, i.e. successfully guess, and exploit them, we present a proactive password change strategy. It regularly changes a password after a certain time interval. In the following, we describe two solutions to select appropriate intervals for password changes.

7.5.1.1 Time intervals based on password security

The time that an attacker needs to guess a stolen (salted and hashed) password by an offline attack depends on two factors: First, the attacker's guessing power, i.e. how fast he can guess passwords. Second, the security level of the password. We select the time interval to change a password depending how long it withstands an offline brute-force attack. In this way, a password is changed before an attacker can guess it. Note that a fixed time interval for all passwords, e.g. 90 days as proposed in the literature [58, 190], is not suitable. As we have shown in Section 3.4.2, the various password requirements of services lead to different security levels of passwords. Therefore, some passwords must be changed more frequently than others.

We calculate the days how long a password withstands an attack by $T = S_{max}/G \cdot I$, where S_{max} is the maximum security level of the service's password requirements, G is the current guessing power of the attacker, and I the rate of increase of the attacker's guessing power, e.g. due to faster hardware. As described in Section 3.4.2.2, S_{max} is calculated by $L_{max} \cdot \log_2(C)$, where L_{max} is the maximum password length and C is the cardinality of the character set allowed by the service. Both information is obtained by the PPD of the service.

In the following, we describe the selection of proper values for the attacker's guessing speed and guessing improvement. Then, we demonstrate our metric in practice and determine the time intervals for password changes.

Determining the attacker's guessing power

Best practice for offline attacks is guessing passwords in parallel on multiple graphic cards [111]. By doubling the number of graphic cards, an attacker can double his guessing power and thus halve the time to guess a password. Consequently, the overall attacker's guessing power depends on how many graphic cards an attacker can afford.

The number of password guesses that are possible with a current graphic card can be obtained from benchmarks (cf. [101]). For instance, using a graphic card for approximately 1.000$ an attacker can perform approximately 10^{15} SHA-1 guesses per day. Note that we consider that services use SHA-1 to hash passwords (cf. Section 3.1). This guessing power allows him to guess passwords consisting of letters and numbers with a length of 9 characters within two weeks. However, an attacker with a budget of 1.000.000$ can afford 1000 graphic cards and thus would be able to guess such passwords in 3 hours.

We consider two attackers. First, a basic attacker with a budget of 10.000$ and consequently a guessing power of $G = 10^{16}$ hashes per day. Second, a powerful attacker with a budget of 1.000.000$ and a guessing power of $G = 10^{18}$.

Determining the attacker's increase of guessing power

We use Moore's law to approximate the rate of increase of the attacker's guessing power [44]. Consequently, the guessing power doubles every 18 months and the improvement factor is $I = 2^{(12/18)} \approx 1.5$ per year. To verify the applicability of Moore's law for our metric, we compared its predictions with real benchmarks [101] of the last 2.5 years. It turns out that it is indeed a suitable approximation for the attacker's improvement regarding offline brute-force attacks. The results are illustrated in Figure 7.6 in a simplified form. The blue line shows the guessing power according to the benchmarks and the red line the estimated increase using Moore's law.

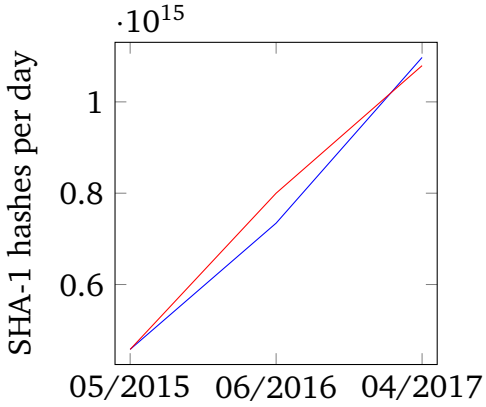


Figure 7.6: Estimation of the attacker's increase of guessing power.

Time intervals in practice

Based on our metric, we calculate time intervals for exemplary PPDs. Table 7.1 provides the time intervals for passwords consisting of letters and numbers with different lengths.

Password length	Days (basic attacker)	Days (powerful attacker)
< 9	< 1	< 1
9	1	< 1
10	96	8
11	5963	501
> 11	$> 10^5$	$> 10^4$

Table 7.1: Time intervals for proactive password changes.

The results show that passwords with 9 or less characters should be changed every day. With regard to the results of our password requirements survey presented in Section 3.4, this would apply for approximately 450 services. Passwords with 10 characters should be changed every week or every three months depending on the attacker's guessing power. This applies for approximately 1000 services in our survey. Finally, for passwords with 11 or more characters a password change every year is sufficient.

Password changes on a daily basis are feasible as we can assume that users turn on at least one of their devices each day so that a password assistant can change passwords. While more frequent password changes are technically possible, this might raise usability issues when users get logged out after a password change or services consider this as suspicious account activity.

Password changes on a yearly basis for passwords with more than 12 characters are not necessary from a mathematical point of view. Nevertheless, as we described in Section 3.2.1, it is reasonable to regularly change them due to the following reasons:

- Hash functions used for password storage might become insecure.
- The secure channel between users and services might become insecure.
- Passwords might be compromised by other attacks (e.g. phishing) in which the security level of passwords is irrelevant.

Against this background, passwords with more than 12 characters like attack-resistant passwords would be changed every year. And, of course, immediately in case of password breaches and suspicious account activity. Our solution can do this automatically for users without demanding any efforts or interactions by them.

7.5.1.2 Time intervals based on user preferences and service conditions

Another solution is to let users select the time interval of password changes. We suggest to allows users to configure the time interval in two ways:

- *Easy mode*: Password change on a daily, weekly, monthly, and yearly basis.
- *Expert mode*: Password change on an arbitrary time interval.

This allows users to keep in control of password changes and it considers users' personal attitudes. Examples include the users' trust in services, the account value, and the attitude to security. Moreover, it enables users to cope with services requiring regular password changes.

7.5.2 Reactive password change strategy

It is essential to immediately change passwords when accounts are compromised. This is the objective of the reactive password change strategy. We present two solutions to enable password assistants to detect compromised accounts of their users. First, we present in Section 7.5.2.1 a central notification service where password assistants receive reports about password breaches. Second, we describe in Section 7.5.2.2 a decentralized approach in which the users' password assistants themselves monitor accounts to detect compromise. In case compromised accounts are detected, password assistants automatically and autonomously change the passwords of the affected accounts.

7.5.2.1 Notification service

We introduce the Password Breach Notification Service (PBNS), a central service that maintains a list of password breaches. Password assistants observe this list to get notified about password breaches at services. In case an account of a user is affected by a breach, the user's password assistant automatically and autonomously performs a password change. Such a password assistant running on a user's mobile device can watch for password breaches around the clock. The necessary information to perform the password change are provided by PPDs which can be retrieved from the PPDDS. The entire setting and general data flow is depicted in Figure 7.7.

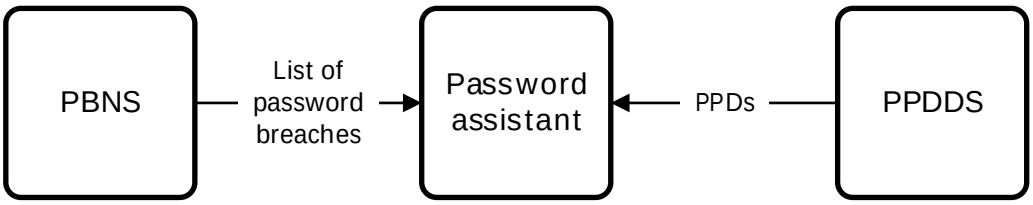


Figure 7.7: Data flow of the password breach notifications.

List of password breaches

For each password breach the list contains three information. First, the URL of the affected service. Second, the date on which the password breach has happened, encoded according to ISO 8601 [126]. Third, a link to a detailed description of the breach provided by the PBNS.

Listing 7.2 provides an example. The list contains password breaches of three real Internet services happened in May 2017. Based on the information about breached available at [122, 159], the entire list would currently contain approximately 250 services.

```
<services>
  <service>
    <url>http://www.dafont.com</url>
    <date>2017-05-18</date>
    <description>
      https://pbns.passwordassistance.info/2017/05/18/dafont
    </description>
  </service>
  <service>
    <url>https://www.zomato.com</url>
    <date>2017-05-17</date>
    <description>
      https://pbns.passwordassistance.info/2017/05/17/zomato
    </description>
  </service>
  <service>
    <url>http://bell.ca</url>
    <date>2017-05-15</date>
    <description>
      https://pbns.passwordassistance.info/2017/05/15/bell
    </description>
  </service>
</services>
```

Listing 7.2: Password breach list.

While requesting the list from the PBNS, a password assistant can specify a time interval. The list returned by the PBNS then only contains password breaches that happened within this time interval. This keeps password assistants informed about password breaches in an efficient way. Instead of always downloading the entire list, password assistants can only request a sublist of recent breaches that happened after the last request. This significantly reduces network load and processing time. The implementation of the PBNS is future work.

Detecting password breaches

In practice, reports about password breaches are mainly received from the media. The operator of the PBNS observes prominent news portals for such reports. In addition to this, the PBNS provides a reporting system. It allows users to inform the PBNS about password breaches. The operator of the PBNS verifies such reports and if necessary announce a password breach through the notification list.

Besides users, the reporting system is also used by our account monitor, that we present in the next section, to automatically report account compromise. This facilitates an early alert system that detects password breaches long before news reports or stolen passwords are put up for sale on the black market.

While other approaches (e.g. [122, 144]) request users to register their usernames, our solution protects the users' privacy. The PBNS does not know the services for which users have accounts. In our solution, the monitoring is done by the users' password assistants which know the services/usernames anyway. Moreover, our solution directly notifies the password assistants, thus, passwords are changed immediately. Other approaches (e.g. [122, 144, 157]) inform users. Consequently, valuable time is lost before passwords are changed. A typical situation is when users are on holiday.

7.5.2.2 Account login monitor

To detect compromised accounts of individual users, we present the Account Login Monitor (ALM). It monitors the logins of online accounts. In the event of a login, the ALM checks if the login was done by the user. If not, the ALM triggers a password change to invalidate the former account password immediately.

The ALM is capable of recognizing a targeted attack against a single user. Such attacks as well as attacks against a small number of users often are not become public and therefore not listed by the PBNS. In general, the ALM further reduces the time span between account compromise and password change.

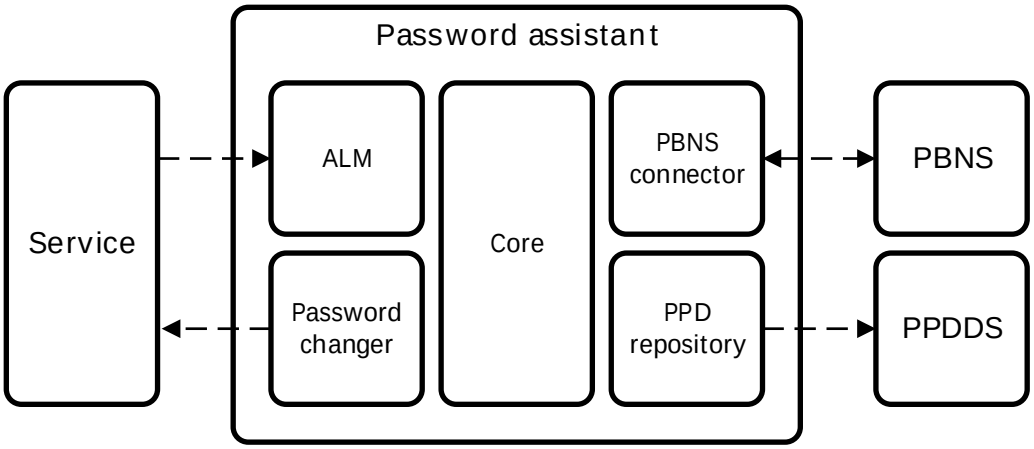


Figure 7.8: Architecture of a password assistant with ALM.

The ALM is directly integrated into a password assistant. A conceptual architecture of such a password assistant is depicted in Figure 7.8. It consists of five main components which we briefly describe in the following:

- *Core*: It manages the components and in particular keeps records of password changes.
- *Account Login Monitor*: It monitors the logins of the user's account at the service. In case an unauthorized login is detected, the ALM triggers a password change. We describe two solutions to realize the detection of unauthorized logins later in this section.
- *Password changer*: It changes passwords of accounts. A password change is triggered by the ALM. The password changer obtains the necessary information to change passwords, including services' password requirements, interfaces, and procedures, from PPDs.
- *PBNS connector*: It implements the notification and reporting feature of the PBNS. In case the ALM detects a compromised account, the PBNS connector sends the URL of the affected service to the PBNS. As described in Section 7.5.2.1, this information is used to rapidly detect password breaches. In addition, the PBNS connector periodically receives the list of recent password breaches from the PBNS and checks if an account of the user is affected. In such a case, the PBNS connector triggers the password changer.
- *PPD repository*: It manages the PPDs of the services for which the user has accounts. In case the user adds an account, the repository requests the corresponding PPD of the service from the PPDDS (cf. Section 7.4). The repository also checks for updates.

Detecting unauthorized logins

We describe two general solutions to realize the ALM and detect unauthorized logins.

The first realization of the ALM makes use of account activity logs such as the *last login* field. It states the date and time of the last login. The ALM logs in to an account and memorizes the point in time. After a random period of time, it logs in again and checks the last login field. If a login was perform in the meanwhile, the ALM contacts the core component of the password assistant (cf. Figure 7.8) and checks if the user recently performed a login. If not, someone else obviously accessed the account and the ALM triggers a password change.

The second realization of the ALM makes use of login notifications send by services via email. Many popular services such as Google allow users to activate such notifications. The ALM monitors the user's mailbox to detect login notification. When a login notification arrives, the ALM contacts the core component of the password assistant and checks if the user recently performed a login. If not, it triggers a password change.

In case of PALPAS (cf. Chapter 5) it is possible to make such login information available to all user devices. The PALPAS server stores an additional time stamp for each account data object which indicates the last access. In case the ALM receives a login notification, it computes the related account data identifier (cf. Section 5.1.3) and requests the time stamp. Based on the time stamp it decides whether the user recently logged in to the account. In case of a login from another device, the PALPAS client of that device would have requested the corresponding account data in order to generate the account password. Therefore, the time stamp could not be older than a few minutes.

It depends on the service if the ALM can be realized by means of last login fields or email notifications. To cope with the different implementations of the notification mails, locations of last login fields,or other types of realizing a detection of unauthorized logins, a uniform description is required. This can be provided by the next version of PPML (cf. Section 7.2). The implementation of the ALM is also future work.

7.6 Implementation and practical evaluation

In this section, we present an implementation and practical evaluation of our solution. We demonstrate that our concept of automatically changing passwords is applicable in practice.

We start in Section 7.6.1 by an evaluation of our PPDR and show that it can be used to generate PPDs for existing services. Then, we present a password assistant in Section 7.6.2 that makes use of PPDs and thus is capable of automatically changing passwords for users.

7.6.1 Creation of password policy descriptions

We used our PPDR to create PPDs of 25 popular services selected from the Alexa Top 100 list [7]. For 21 services the PPDR was able to create PPDs. Table 7.2 provides an excerpt of these services and shows which technology types of procedures are supported by them.

In case of the other 4 services the PPD creation using the PPDR failed because of two reasons: First, the login form of the service is embedded in an iFrame. Second, the `id` attributes of input fields are not static so that the PPDR was unable to determine proper selectors for the fields for the procedures. For such services, the PPDs must be created manually.

Service	Procedure type
Amazon	HTML
Dropbox	ExtJS
eBay	HTML
Facebook	ExtJS
Google	HTML
LinkedIn	HTML
Netflix	HTML
Paypal	ExtJS
Twitter	HTML
Wikipedia	JS

Table 7.2: Created PPDs.

7.6.2 Usage of password policy descriptions in password assistants

In this section, we present the Automatic Password Changer (APACHA). It is a password assistant that solves the password change problem. APACHA can process PPDs and is thereby capable of automatically changing passwords on behalf of users. A password change can be triggered by users as well as by proactive strategy as described in Section 7.5. APACHA does not store passwords on user devices. It implements the PALPAS scheme that we presented in Chapter 5.

The architecture of APACHA is depicted in Figure 7.9. It consists of seven main components which we briefly describe in the following.

- *Core*: It manages APACHA and in particular keeps track of regular password changes.
- *User interface*: It provides a graphical user interface for the user. The interface provides a list of all online accounts that are managed by APACHA as well as a form to add new accounts to APACHA. Each account in the list has a button that triggers the password change and a button to configure the proactive password change strategy of the account.
- *Storage*: It stores necessary data on the user device such as the PALPAS secret as well as secret authentication key and certificate for the PLS. The storage is protected by a user-chosen master password.
- *PPD repository*: It manages the PPDs of the services for which the user has an account. In case the user adds an account to APACHA, the repository requests the PPD of the service from the PPDDS (cf. Section 7.4). The repository also checks for updates.

- *PLS connector*: It establishes the communication to the PLS and performs the authentication. It obtains the secret authentication key and certificate for the authentication from the storage component. The PLS connector is also used to retrieve the account data, authentication tokens, and one-time-pad keys from the PLS.
- *PALPAS password generator*. It implements the PALPAS password generation procedure. For the login at a service, it retrieves the corresponding account data from the PLS (through the PLS connector). In case of a password change, it generates a new salt and corresponding password and updates the account data at the PLS.
- *Password changer*: It takes as input the PPD of a service and executes the password change procedure defined in the PPD. For the login at the service, the password changer obtains the current password from the PALPAS password generator. It also triggers the password generator to generate a new password for the password change. After performing and verifying a password change it informs the PLC connector to update the salt at the PLS so that the new password is available to other devices.

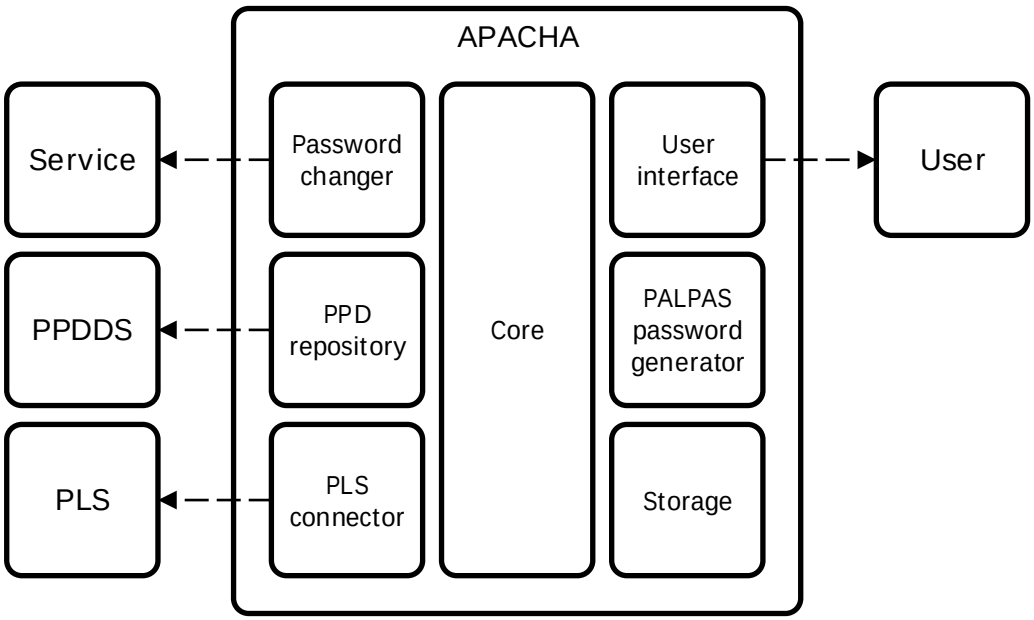


Figure 7.9: Architecture of the APACHA client.

We implemented APACHA in Java. The password changer uses the Web Engine of JavaFX to interact with the services' password interfaces. This enables APACHA to support all types of procedures (HTTP, HTML, JavaScript, and Extended JavaScript, cf. Section 7.2). The implementation is available at [H26].

Besides the password change problem, APACHA also solves the problem of migrating from weak to secure passwords. When users decide to establish secure passwords they often already have a lot of accounts. This is a major obstacle because it is very time-consuming to manually replace all passwords by attack-resistant ones. APACHA does this for users automatically. Users only need to provide the username and the password of an account as well as the URL of the corresponding service. APACHA obtains the PPD of the service, automatically logs in to the account, and replaces the weak password by an attack-resistant one. This allows users to easily eliminate their weak passwords and establish secure passwords for their accounts.

7.7 Easy-to-use passwords

In this section, we describe the concept of easy-to-use passwords. It makes the usage of passwords on devices that does not belong to users as easy and safe as possible. Easy-to-use passwords can be realized by password assistants like APACHA (cf. Section 7.6.2).

Today, users access their accounts mainly from their own devices. In particular, smartphones allow this everywhere and anytime. However, there are still situations in which users need to access their accounts from devices that do not belong to them. For instance, a user wants to access his video streaming account from a friend's computer or his cloud storage account to print out some documents. Another example includes accessing accounts from public computers at airports to safe the battery of one's smartphone. These situations pose two issues:

- Manually entering passwords on another device is very inconvenient and error-prone. Brute-force-resistant passwords are very long and might contain ambiguous characters which lead to typographical errors.
- There is the risk of malware on the other device so that an attacker obtains the users' passwords and gets access to their accounts.

Based on PAS' feature of automatic password changes these issues can be solved. A PPD-enabled password assistant can temporarily replace the account password by an easy-to-use password. Such a password is short and less complex so that it can be easily entered on another device where a user wants to access his account. After service usage, the password assistant automatically changes the password in order to invalidate a possibly compromised password and replaces it by an attack-resistant one. The password assistant can obtain the necessary information to generate a valid easy-to-use and attack-resistant password from the service's PPD.

In the following, we describe the application flow of easy-to-use passwords (cf. Figure 7.10). A user (U) who has a user device (UD), e.g. his smartphone, with a PPD-enabled password assistant. To simplify the description, we consider the UD and the password assistant as a single entity and just use the term UD in the following explanation. U wants to access his account at a service (S) from another device (D). Instead of entering his current attack-resistant password at the D, he makes use of easy-to-use passwords provides by PAS.

1. U selects the S on the UD.
2. The UD generates an easy-to-use password. It logs in to the user's account and replaces the current account password by the easy-to-use password.
3. The UD provides the username and the easy-to-use password to U.
4. U now enters his username and the easy-to-use password on the D.
5. The D logs in to the user's account at S using the username and the easy-to-use password. U can now access his account on the D.
6. When U finishes the service usage on the D, he selects S on the UD again.
7. The UD generates a new attack-resistant password. It logs in to the user's account again and changes the account password. The easy-to-use password is now invalid and the D or a possible attacker cannot access the user's account anymore.

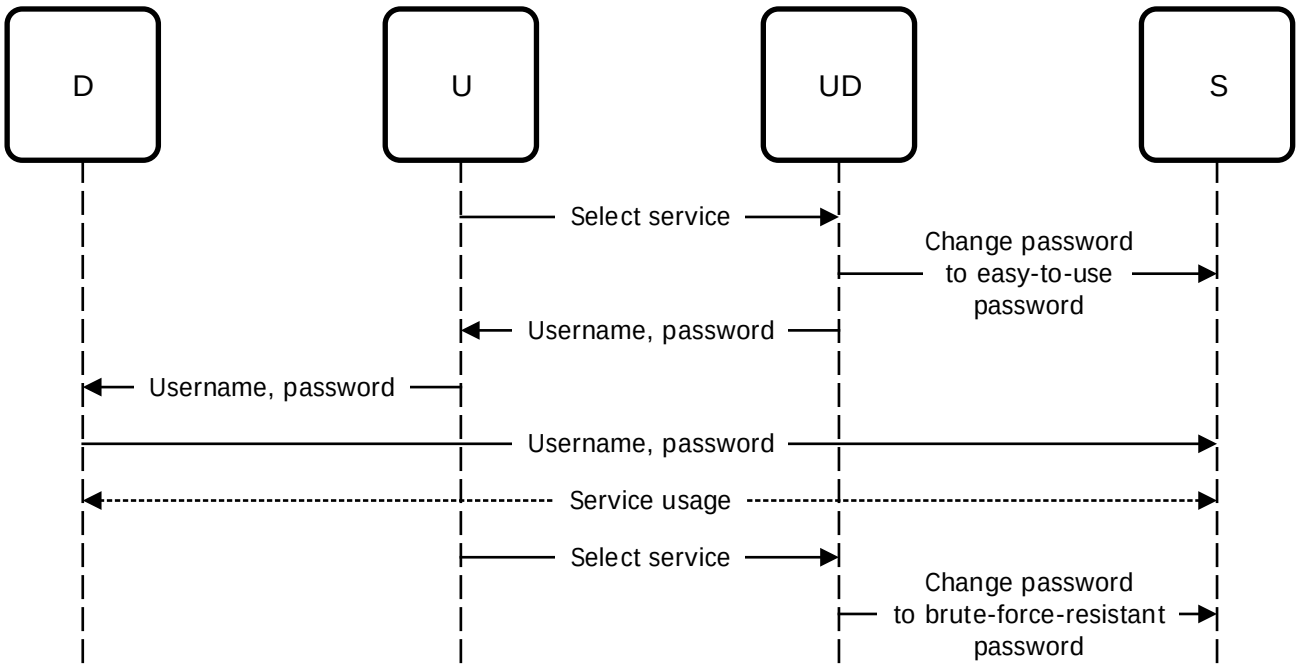


Figure 7.10: Data flow of easy-to-use passwords.

Easy-to-use passwords has many benefits. From the usability perspective, it meets the users' preference of a portable password assistant using a mobile device [131]. It also does not require users to install any software (e.g. a password manager/assistant) on the other device, which is often not possible at all, e.g. in case of public computers at airports.

From the security perspective, users only need to enter the password of the account that they want to access on the other device and not a master password of a password manager or credentials for an online password manager. To this end, there is only the risk that an attacker obtains the account password and not the entire password portfolio of a user. But, with easy-to-use passwords this password becomes immediately invalid after service usage as it is changed by the password assistant.

An attacker might change the account password to keep access to the user's account, but also this attack is immediately detected when the password assistant changes the password at the end of the service usage. Finally, easy-to-use passwords are completely transparent to services so it works with all services for which automatic password changes based on PPDs are possible.

7.8 Conclusion

In this chapter, we solved the password change problem. PAS enables password assistants to automatically and autonomously change password on a regular basis and immediately after password compromise. Existing approaches are limited to popular services and certain platforms and browsers. PPDs are the first universal solution to realize an automation of password changes at arbitrary services. They are independent from platforms and programming languages. This facilitates an easy integration of PPDs into password assistants running on different platforms and devices. Furthermore, PPDs can be easily extended to support new technologies. Using the PPDR, also the generation of PPDs is very simple and can be done by users themselves. By making the PPDs available through the PPDDS, they must be created only once and can be made available to all Internet users and their password assistants.

Moreover, existing approaches only simplify password changes, but left the responsibility to actually perform password changes to users. PAS solves this by two sophisticated password change strategies that facilitates automatic password changes in an intelligent way. Passwords are changed on a regular basis with respect to the security level of passwords as well as immediately after PAS detects a compromise of users' passwords. This minimizes the time period for an attacker to exploit stolen passwords. In this way, PAS significantly improvements the password security of Internet users while requiring no interactions and efforts by users at all.

The practicality of our solution and its capability to solve the password change problem have been demonstrated. We generated PPDs for existing services by using our PPDR. Moreover, we developed APACHA which makes use of PPDs and demonstrates that a PPD provides all necessary information to automatically change passwords of online accounts at real services. APACHA is designed as a stand-alone application. But, many users already use password managers and might not be willing to switch to APACHA, even if their current password manager does not support automatic password changes. To solve this problem, APACHA can act as a simple password changer in the background while users continue to use their current password manager. Many password managers support extensions or have APIs that allow to access the passwords that they manage. This enables an integration of APACHA into such applications. In this way, users benefit from automatic and autonomous password changes realized by APACHA and can continue to use their current password manager. Of course, other applications can also directly use the individual components of PAS such as PPDs and their retrieval from the PPDDS.

Easy-to-use passwords are an additional example for the strength of PAS. Besides making passwords easier to use, easy-to-use passwords allow to maintain secure passwords even when users use their passwords on other devices, where they have no control of the platform and device security. Like all components of PAS are entirely realized on the user-side, easy-to-use passwords are widely applicable in practice because they are transparent to services.

In this chapter, we presented the fourth and last part of PAS. It fully automates the third password task of changing passwords. In Chapter 4 we described the first part which fully automates the first password task of generating passwords. In Chapter 5 and 6 we presented the second and third part of PAS which perform the second password task of preserving passwords. PAS as a whole thereby provides the first ubiquitous solution that performs all password tasks. It solves all password problems and addresses all conditions, concerns, and needs of users. In this way, PAS enables users to practically use secure passwords for their online accounts at Internet services for the first time.

8 Conclusion

In this thesis, we presented the Password Assistance System (PAS). It enables users to realize secure passwords for their accounts at services on the Internet. We presented the various security requirements, service conditions, and usage requirements of secure passwords. Moreover, we described the complex and extensive password tasks of users to achieve secure passwords for their accounts. The three tasks are generating, preserving, and changing passwords. Manually performing these tasks is practically impossible for users.

PAS performs the first password task of generating attack-resistant, individual, and valid passwords for users automatically. We showed that attack-resistant and individual passwords are essential because services are compromised in practice and users can only expect a basic protection of their passwords. Other than existing approaches, PAS automatically generates passwords in accordance with services' password requirements. The need therefore was demonstrated by our largest ever conducted survey of password requirements. We showed that the requirements are quite different, incomplete, and often missing at all. Further, we showed that there exists no single password-composition rule set that creates valid passwords for all services. We introduced Password Requirements Descriptions (PRD) to handle the wide diversity of the password requirements. Moreover, we introduced solutions to automatically create PRDs and distribute them. So far, PAS provides PRDs for 185,696 services. Finally, we complemented PAS with an optimal password-composition rule set to solve the incompleteness of password requirements. Based on this rule set, password assistants generate attack-resistant passwords with the best possible acceptance rate even when requirements are unavailable. In addition, existing password generators can be significantly improved regarding security and acceptance rate at the same time by using our optimal password-composition rules. Up to now they make use of suboptimal rules, which generate weak and/or invalid passwords.

Furthermore, PAS extensively supports users in the second password task of preserving passwords. We introduced the Passwordless Password Synchronization (PALPAS) scheme which automatically generates attack-resistant, individual, and valid passwords and preserves them for users. Moreover, it protects the preserved passwords, makes them available on all user devices, and automatically synchronizes changes. But, PALPAS does not store passwords, neither at user devices nor at servers on the Internet. On contrast to other approaches, passwords cannot be stolen from servers. PALPAS' revocation mechanism additionally provides protection in

case of device theft. We also presented PASCO (PALPAS RECOVERY), a backup solution that provides recovery of the PALPAS data. Backups do not have to be updated when users' password portfolios change. PASCO also allows an emergency access to the passwords of users for trusted persons. Like PALPAS, PASCO is equipped with a revocation mechanism which facilitates invalidation of backups even when users do not have physical access to them. Based on PALPAS and PASCO, PAS provides the first solution for the password preservation problem. By achieving the confidentiality, availability, recoverability, and accessibility of the passwords, PAS address all concerns and needs of users regarding the preservation of passwords.

Moreover, PAS performs the third password task of changing passwords for users automatically. In contrast to existing approaches, it does not only simplify password changes, it completely takes over this extensive and time-consuming task. We introduced Password Policy Descriptions (PPD) to cope with the different password interfaces and procedures of services. Moreover, PAS is equipped with a proactive and a reactive password change strategy to realize complete automation. It changes passwords on a regular basis with respect to the security level of passwords as well as immediately after PAS detects a compromise of users' passwords. Finally, with easy-to-use passwords we outlined a further application of PAS to complement our goal of making the usage of passwords as easy as possible by means of automation.

We evaluated the practicality of PAS and it became apparent that it is ready to be used in practice. None of the PAS' components demands changes to services, they can be operated independently, and used with arbitrary services. With PPDs there exist the first practical solution that allows to handle the different services' password implementations. PPDs can be easily extended to cope with changes and new technologies coming up in the future. Moreover, users can create their own PPDs to use PAS with any service they want. We implemented the components of PAS as stand-alone applications, showed their integration in existing applications, and even new applications such as AutoPass [156] already build on PAS.

Future work

There still exist further challenges and interesting research topics related to this thesis. One challenge is imposed by the Internet of Things (IoT). Similar to the constantly increasing number of online accounts, users will have more and more connected physical devices, such as connected cars and smart homes. User authentication in the IoT is again build on passwords. This leads to the same problems which we already encounter with user accounts on the Internet: weak and reused passwords. Extending the scope of PAS and the functionality of password assistants to cover the new challenge of users to generate and manage secure passwords for the IoT should be subject of further research.

Bibliography

[1] Steven Van Acker, Daniel Hausknecht, Wouter Joosen, and Andrei Sabelfeld. Password Meters and Generators on the Web: From Large-Scale Empirical Study to Getting It Right. In *Proc. CODASPY*, 2015. Cited on page 50.

[2] Anne Adams and Martina Angela Sasse. Users Are Not The Enemy. *CACM*, 42(12), 1999. Cited on page 22.

[3] Naveen Agarwal, Scott Renfro, and Arturo Bejar. Yahoo!'s Sign-in Seal and current anti-phishing solutions, 2007. Cited on page 44.

[4] AgileBits Inc. 1Password. `https://1password.com`. Cited on page 59.

[5] Mahdi Nasrullah Al-Ameen, Matthew K. Wright, and Shannon Scielzo. Towards Making Random Passwords Memorable: Leveraging Users' Cognitive Ability Through Multiple Cues. In *Proc. CHI*, 2015. Cited on page 55.

[6] Ahmed AlEroud and Lina Zhou. Phishing environments, techniques, and countermeasures: A survey. *COMSEC*, 68, 2017. Cited on pages 11 and 17.

[7] Alexa Internet. Alexa Top Sites. `http://www.alexa.com/topsites`. Cited on pages 75, 87, 88, and 157.

[8] Alexa Internet. Top Sites in United States. `http://www.alexa.com/topsites/countries/US`. Cited on pages 40, 66, 75, 87, 88, and 139.

[9] Bander AlFayyadh, Per Thorsheim, Audun Jøsang, and Henning Klevjer. Improving usability of password management with standardized password policies. In *Proc. SAR-SSI*, 2012. Cited on page 53.

[10] Adil Alsaid and Chris J. Mitchell. Preventing phishing attacks using trusted computing technology. In *Proc. INC*, 2006. Cited on page 116.

[11] American National Standards Institute (ANSI). Coded Character Sets – 7-Bit American Standard Code for Information Interchange (7-Bit ASCII). ANSI X3.4-1986, 1986. Cited on page 36.

[12] Anonymizer Inc. Anonymizer. `https://www.anonymizer.com`. Cited on page 80.

[13] Apache Software Foundation. Apache UIMA. `https://uima.apache.org`. Cited on page 73.

[14] Apple Inc. Apple Media Advisory – Update to Celebrity Photo Investigation, 2014. `https://www.apple.com/newsroom/2014/09/02Apple-Media-Advisory/`. Cited on page 19.

[15] Jean-Philippe Aumasson. Password Hashing Competition. `https://password-hashing.net`. Cited on page 20.

[16] Michael Bachmann. Passwords are Dead: Alternative Authentication Methods. In *Proc. JISIC*, 2014. Cited on page 2.

[17] Daniel V. Bailey, Markus Dürmuth, and Christof Paar. Statistics on Password Re-use and Adaptive Strength for Financial Accounts. In *Proc. SCN*, 2014. Cited on page 55.

[18] Gregory V. Bard. Spelling-Error Tolerant, Order-Independent Pass-Phrases via the Damerau-Levenshtein String-Edit Distance Metric. In *Proc. ACSW Frontiers*, 2007. Cited on page 52.

[19] Elaine B. Barker and John M. Kelsey. Recommendation for Random Number Generation Using Deterministic Random Bit Generators. NIST Special Publication 800-90A Rev. 1, 2015. Cited on pages 7, 111, and 116.

[20] Erick Bauman, Yafeng Lu, and Zhiqiang Lin. Half a Century of Practice: Who Is Still Storing Plaintext Passwords? In *Proc. ISPEC*, 2015. Cited on page 32.

[21] Andrey Belenko and Dmitry Sklyarov. "Secure Password Managers" and "Military-Grade Encryption" on Smartphones: Oh, Really? *Blackhat Europe*, 2012. Cited on page 56.

[22] Steven M. Bellovin. Frank Miller: Inventor of the One-Time Pad. *Cryptologia*, 35(3), 2011. Cited on pages 8, 99, 119, and 135.

[23] Francesco Bergadano, Bruno Crispo, and Giancarlo Ruffo. Proactive Password Checking with Decision Trees. In *Proc. CCS*, 1997. Cited on page 50.

[24] Akashdeep Bhardwaj, G. V. B. Subrahmanyam, Vinay Avasthi, and Hanumat Sastry. Ransomware: A Rising Threat of new age Digital Extortion. *CoRR*, abs/1512.01980, 2015. Cited on page 58.

[25] Robert Biddle, Sonia Chiasson, and Paul C. van Oorschot. Graphical passwords: Learning from the first twelve years. *CSUR*, 44(4), 2012. Cited on page 2.

[26] Alex Biryukov, Daniel Dinu, and Dmitry Khovratovich. Argon2: New Generation of Memory-Hard Functions for Password Hashing and Other Applications. In *Proc. EuroS&P*, 2016. Cited on page 20.

[27] Christopher M. Bishop. *Pattern recognition and machine learning*. Springer, 2006. Cited on page 7.

[28] Matthew A. Bishop. *Computer Security: Art and Science*. Addison-Wesley, 2003. Cited on page 10.

[29] Jeremiah Blocki, Manuel Blum, and Anupam Datta. Naturally Rehearsing Passwords. In *Proc. ASIACRYPT*, 2013. Cited on page 55.

[30] BlueKrypt. Cryptographic Key Length Recommendation. `https://www.keylength.com`. Cited on page 20.

[31] Bert den Boer and Antoon Bosselaers. Collisions for the Compressin Function of MD5. In *Proc. EUROCRYPT*, 1993. Cited on page 21.

[32] Hristo Bojinov, Elie Bursztein, Xavier Boyen, and Dan Boneh. Kamouflage: Loss-Resistant Password Management. In *Proc. ESORICS*, 2010. Cited on page 56.

[33] Joseph Bonneau. Statistical Metrics for Individual Password Strength. In *Proc. SPW*, 2012. Cited on page 37.

[34] Joseph Bonneau. The Science of Guessing: Analyzing an Anonymized Corpus of 70 Million Passwords. In *Proc. SP*, 2012. Cited on pages 1, 37, and 49.

[35] Joseph Bonneau, Elie Bursztein, Ilan Caron, Rob Jackson, and Mike Williamson. Secrets, Lies, and Account Recovery: Lessons from the Use of Personal Knowledge Questions at Google. In *Proc. WWW*, 2015. Cited on pages 17 and 25.

[36] Joseph Bonneau, Cormac Herley, Paul C. van Oorschot, and Frank Stajano. The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes. In *Proc. SP*, 2012. Cited on pages 1, 2, 10, and 11.

[37] Joseph Bonneau and Sören Preibusch. The Password Thicket: Technical and Market Failures in Human Authentication on the Web. In *Proc. WEIS*, 2010. Cited on pages 2, 19, 21, 29, and 49.

[38] Joseph Bonneau and Stuart E. Schechter. Towards Reliable Storage of 56-bit Secrets in Human Memory. In *Proc. USS*, 2014. Cited on page 55.

[39] Joseph Bonneau and Ekaterina Shutova. Linguistic Properties of Multi-word Passphrases. In *Proc. FC*, 2012. Cited on page 52.

[40] John G. Brainard, Ari Juels, Ronald L. Rivest, Michael Szydlo, and Moti Yung. Fourth-factor authentication: somebody you know. In *Proc. CCS*, 2006. Cited on page 10.

[41] Johannes Braun. *Maintaining security and trust in large scale public key infrastructures*. PhD thesis, Darmstadt University of Technology, Germany, 2015. Cited on page 9.

[42] Johannes Braun, Florian Volk, Jiska Classen, Johannes A. Buchmann, and Max Mühlhäuser. CA trust management for the web PKI. *JCS*, 22(6), 2014. Cited on page 16.

[43] Alan S. Brown, Elisabeth Bracken, Sandy Zoccoli, and King Douglas. Generating and remembering passwords. *ACP*, 18(6), 2004. Cited on page 1.

[44] Bostjan Brumen and Viktor Taneski. Moore's Curse on Textual Passwords. In *Proc. MIPRO*, 2015. Cited on pages 20 and 151.

[45] Johannes A. Buchmann. *Introduction to Cryptography*. Springer, 2002. Cited on pages 7 and 8.

[46] William E. Burr, Donna F. Dodson, Elaine M. Newton, Ray A. Perlner, W. Timothy Polk, Sarbari Gupta, and Emad A. Nabbus. Electronic Authentication Guideline. NIST Special Publication 800-63-2, 2013. Cited on page 50.

[47] James H. Burrows. Password Usage. Federal Information Processing Standards Publication (FIPS) 112, 1985. Cited on page 31.

[48] Erik Cambria and Bebo White. Jumping NLP curves: A review of natural language processing research. *CIM*, 9(2), 2014. Cited on page 73.

[49] John Campbell, Wanli Ma, and Dale Kleeman. Impact of restrictive composition policy on user password choices. *BIT*, 30(3), 2011. Cited on page 29.

[50] Xavier de Carné de Carnavalet and Mohammad Mannan. From Very Weak to Very Strong: Analyzing Password-Strength Meters. In *Proc. NDSS*, 2014. Cited on pages 35 and 50.

[51] Xavier de Carné de Carnavalet and Mohammad Mannan. A Large-Scale Evaluation of High-Impact Password Strength Meters. *TISSEC*, 18(1), 2015. Cited on page 50.

[52] Luca Casati and Andrea Visconti. Exploiting a Bad User Practice to Retrieve Data Leakage on Android Password Managers. In *Proc. IMIS*, 2017. Cited on page 56.

[53] Claude Castelluccia, Chaabane Abdelberi, Markus Dürmuth, and Daniele Perito. When Privacy meets Security: Leveraging personal information for password cracking. *CoRR*, abs/1304.6584, 2013. Cited on pages 1 and 13.

[54] Claude Castelluccia, Markus Dürmuth, and Daniele Perito. Adaptive Password-Strength Meters from Markov Models. In *Proc. NDSS*, 2012. Cited on pages 14 and 50.

[55] Rahul Chatterjee, Joseph Bonneau, Ari Juels, and Thomas Ristenpart. Cracking-Resistant Password Vaults Using Natural Language Encoders. In *Proc. SP*, 2015. Cited on page 56.

[56] Wendy Chen and Weide Chang. Applying Hidden Markov Models to Keystroke Pattern Analysis for Password Verification. In *Proc. IRI*, 2004. Cited on page 14.

[57] Hsien-Cheng Chou, Hung-Chang Lee, Hwan-Jeu Yu, Fei-Pei Lai, Kuo-Hsuan Huang, and Chih-Wen Hsueh. Password cracking based on learned patterns from disclosed passwords. *IJICIC*, 2013. Cited on page 13.

[58] Peter Cisar and Sanja Maravic Cisar. Password – A Form of Authentication. In *Proc. SISY*, 2007. Cited on page 150.

[59] Luke St. Clair, Lisa Johansen, William Enck, Matthew Pirretti, Patrick Traynor, Patrick D. McDaniel, and Trent Jaeger. Password Exhaustion: Predicting the End of Password Usefulness. In *Proc. ICISS*, 2006. Cited on page 22.

[60] Richard Clayton. Insecure real-world authentication protocols (or why phishing is so profitable). In *Proc. SPW*, 2005. Cited on page 116.

[61] Cloud Security Alliance. What Rules Apply to Government Access to Data Held by US Cloud Service Providers. `https://cloudsecurityalliance.org/download/government-access-to-data-held-by-us-cloud-service-providers/`. Cited on page 57.

[62] Codenomicon. The Heartbleed Bug. `http://heartbleed.com`. Cited on pages 2, 17, 21, and 60.

[63] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280, 2008. Cited on pages 9, 16, and 113.

[64] Jean-Sébastien Coron, Paul C. Kocher, and David Naccache. Statistics and secret leakage. In *Proc. FC*, 2000. Cited on page 135.

[65] Quynh H. Dang. Secure Hash Standard (SHS). Federal Information Processing Standard Publication (FIPS) 180-4, 2012. Cited on page 11.

[66] Anupam Das, Joseph Bonneau, Matthew Caesar, Nikita Borisov, and XiaoFeng Wang. The Tangled Web of Password Reuse. In *Proc. NDSS*, 2014. Cited on pages 1, 19, 20, and 55.

[67] Dashlane Inc. Best Password Manager, Free Form Filler, Secure Digital Wallet. `https://www.dashlane.com`. Cited on pages 53, 57, 59, and 60.

[68] Dashlane Inc. Dashlane Security Whitepaper. `https://www.dashlane.com/download/Dashlane-Security-Whitepaper-V2.8.pdf`. Cited on page 60.

[69] Antoine Delignat-Lavaud, Martín Abadi, Andrew Birrell, Ilya Mironov, Ted Wobber, and Yinglian Xie. Web PKI: Closing the Gap between Guidelines and Practices. In *Proc. NDSS*, 2014. Cited on page 16.

[70] Matteo Dell'Amico, Pietro Michiardi, and Yves Roudier. Password Strength: An Empirical Analysis. In *Proc. INFOCOM*, 2010. Cited on pages 13 and 14.

[71] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, 2008. Cited on pages 9 and 133.

[72] Thai Duong and Juliano Rizzo. Here come the XOR ninjas, 2011. Cited on pages 17 and 21.

[73] Markus Dürmuth, Fabian Angelstorf, Claude Castelluccia, Daniele Perito, and Chaabane Abdelberi. OMEN: Faster Password Guessing Using an Ordered Markov Enumerator. In *Proc. ESSoS*, 2015. Cited on page 14.

[74] Morris Dworkin. Automated Password Generator (APG). Federal Information Processing Standard Publication (FIPS) 181, 1993. Cited on page 53.

[75] Morris J. Dworkin, Elaine B. Barker, James R. Nechvatal, James Foti, Lawrence E. Bassham, E. Roback, and James F. Dray Jr. Advanced Encryption Standard (AES). Federal Information Processing Standard Publication (FIPS) 197, 2001. Cited on page 8.

[76] D. 3rd Eastlake, J. Schiller, and S. Crocker. Randomness Requirements for Security. RFC 4086, 2005. Cited on page 7.

[77] Serge Egelman, Andreas Sotirakopoulos, Ildar Muslukhov, Konstantin Beznosov, and Cormac Herley. Does my password go up to eleven?: The impact of password meters on password selection. In *Proc. CHI*, 2013. Cited on pages 29 and 50.

[78] Federal Bureau of Investigation (FBI). Incidents of Ransomware on the Rise – Protect Yourself and Your Organization. `https://www.fbi.gov/news/stories/incidents-of-ransomware-on-the-rise/`. Cited on page 58.

[79] Adrienne Porter Felt, Matthew Finifter, Erika Chin, Steve Hanna, and David Wagner. A survey of mobile malware in the wild. In *Proc. SPSM@CCS*, 2011. Cited on page 58.

[80] David A. Ferrucci and Adam Lally. UIMA: an architectural approach to unstructured information processing in the corporate research environment. *NLE*, 10(3-4), 2004. Cited on page 73.

[81] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, 1999. Cited on page 142.

[82] Dinei Florêncio and Cormac Herley. A Large Scale Study of Web Password Habits. In *Proc. WWW*, 2007. Cited on pages 22 and 48.

[83] Dinei Florêncio and Cormac Herley. Where do security policies come from? In *Proc. SOUPS*, 2010. Cited on pages 30, 36, 49, 60, and 78.

[84] Dinei Florêncio, Cormac Herley, and Paul C. van Oorschot. An Administrator's Guide to Internet Password Research. In *Proc. LISA*, 2014. Cited on pages 11, 19, 53, and 60.

[85] Dinei Florêncio, Cormac Herley, and Paul C. van Oorschot. Password Portfolios and the Finite-Effort User: Sustainably Managing Large Numbers of Accounts. In *Proc. USS*, 2014. Cited on pages 22 and 55.

[86] Roman Fojtik. Salt Synchronization Service. Bachelor thesis, Darmstadt University of Technology, Germany, 2016. Cited on page 112.

[87] Alain Forget, Sonia Chiasson, Paul C. van Oorschot, and Robert Biddle. Improving text passwords through persuasion. In *Proc. SOUPS*, 2008. Cited on page 51.

[88] Christian Forler, Stefan Lucks, and Jakob Wenzel. Catena: A Memory-Consuming Password Scrambler. *IACR*, 2013. Cited on page 20.

[89] Lorenzo Franceschi-Bicchierai. Another Day, Another Hack: 117 Million LinkedIn Emails And Passwords. `https://goo.gl/YWaMkR`. Cited on pages 19, 21, and 39.

[90] David Freeman, Sakshi Jain, Markus Dürmuth, Battista Biggio, and Giorgio Giacinto. Who Are You? A Statistical Approach to Measuring User Authenticity. In *Proc. NDSS*, 2016. Cited on page 19.

[91] Steven Furnell. An assessment of website password practices. *COMPSEC*, 26(7-8), 2007. Cited on pages 31 and 49.

[92] Steven Furnell. Assessing password guidance and enforcement on leading websites. *CFS*, 2011(12), 2011. Cited on pages 31 and 49.

[93] Ravi Ganesan, Chris Davies, and Bell Atlantic. A new attack on random pronounceable password generators. In *Proc. NCSC*, 1994. Cited on page 53.

[94] Paolo Gasti and Kasper Bonne Rasmussen. On the Security of Password Manager Database Formats. In *Proc. ESORICS*, 2012. Cited on page 56.

[95] Shirley Gaw and Edward W. Felten. Password management strategies for online accounts. In *Proc. SOUPS*, 2006. Cited on pages 1, 53, and 55.

[96] Nethanel Gelernter, Senia Kalma, Bar Magnezi, and Hen Porcilan. The Password Reset MitM Attack. In *Proc. SP*, 2017. Cited on pages 17 and 25.

[97] Maximilian Golla, Benedict Beuscher, and Markus Dürmuth. On the Security of Cracking-Resistant Password Vaults. In *Proc. CSS*, 2016. Cited on page 56.

[98] Google Inc. Angular. `https://angular.io`. Cited on page 89.

[99] Google Inc. Google Safe Browsing. `https://safebrowsing.google.com`. Cited on page 148.

[100] Google Inc. Minor updates to your Google sign-in experience, 2015. https://support.google.com/mail/forum/AAAAK7un8RUoAsE-6wmaSU/?hl=en. Cited on page 44.

[101] Jeremi M. Gosney, 2017. https://gist.github.com/epixoip. Cited on pages 38, 51, 150, and 151.

[102] Paul A. Grassi, Michael E. Garcia, and James L. Fenton. Digital Identity Guidelines. NIST Special Publication 800-63-3, 2017. Cited on page 10.

[103] Kristen K. Greene and Yee-Yin Choong. Must I, can I? I don't understand your ambiguous password rules. *IMCS*, 25(1), 2017. Cited on pages 44 and 50.

[104] Hana Habib, Jessica Colnago, William Melicher, Blase Ur, and Sean Segreti. Password Creation in the Presence of Blacklists. In *Proc. USEC*, 2017. Cited on page 50.

[105] J. Alex Halderman, Brent Waters, and Edward W. Felten. A Convenient Method for Securely Managing Passwords. In *Proc. WWW*, 2005. Cited on pages 22, 54, and 117.

[106] William G. J. Halfond, Jeremy Viegas, and Alessandro Orso. A Classification of SQL-Injection Attacks and Countermeasures. In *Proc. ESSoS*, 2006. Cited on page 34.

[107] Aaron L. F. Han, Derek F. Wong, and Lidia S. Chao. Password Cracking and Countermeasures in Computer Security: A Survey. *CoRR*, abs/1411.7803, 2014. Cited on page 11.

[108] Christian Happ, André Melzer, and Georges Steffgen. Trick with treat – reciprocity increases the willingness to communicate personal data. *CHB*, 61, 2016. Cited on pages 11 and 17.

[109] S. M. Taiabul Haque, Mahdi Nasrullah Al-Ameen, Matthew Wright, and Shannon Scielzo. Learning system-assigned passwords (up to 56 bits) in a single registration session with the methods of cognitive psychology. In *Proc. USEC*, 2017. Cited on page 55.

[110] Harris Interactive; Various sources (Dashlane). Which of the following online security precautions did you take within the past 30 days?, 2015. http://www.statista.com/statistics/418676/us-online-security-precuations/. Cited on pages 2 and 60.

[111] Hashcat. hashcat – advanced password recovery. https://hashcat.net. Cited on pages 12, 38, 51, and 150.

[112] Martin E. Hellman. A cryptanalytic time-memory trade-off. *TIT*, 26(4), 1980. Cited on page 11.

[113] Cormac Herley. More Is Not the Answer. *SP*, 12(1), 2014. Cited on page 2.

[114] Cormac Herley and Paul C. van Oorschot. A Research Agenda Acknowledging the Persistence of Passwords. *SP*, 10(1), 2012. Cited on pages 1 and 2.

[115] Cormac Herley, Paul C. van Oorschot, and Andrew S. Patrick. Passwords: If We're So Smart, Why Are We Still Using Them? In *Proc. FC*, 2009. Cited on page 1.

[116] Erwin Hess, Norbert Janssen, Bernd Meyer, and Torsten Schütze. Information Leakage Attacks Against Smart Card Implementations of Cryptographic Algorithms and Countermeasures – A Survey. In *Proc. EUROSMART*, 2000. Cited on page 135.

[117] Josef Horalek, Filip Holík, Oldrich Horák, Lukás Petr, and Vladimir Sobeslav. Analysis of the use of rainbow tables to break hash. *JIFS*, 32(2), 2017. Cited on page 11.

[118] Shiva Houshmand and Sudhir Aggarwal. Building better passwords using probabilistic techniques. In *Proc. ACSAC*, 2012. Cited on page 51.

[119] Shiva Houshmand, Sudhir Aggarwal, and Umit Karabiyik. Identifying Passwords Stored on Disk. In *Proc. IFIP WG 11.9*, 2015. Cited on page 13.

[120] Andreas Hülsing. *Practical forward secure signatures using minimal security assumptions*. PhD thesis, Darmstadt University of Technology, Germany, 2013. Cited on page 8.

[121] Daniel Humphries. 67 Percent of Internet Users Haven't Changed Passwords After Heartbleed. `http://intelligent-defense.softwareadvice.com/67-percent-havent-changed-passwords-after-heartbleed-0414/`. Cited on pages 2 and 60.

[122] Troy Hunt. Have I been pwned? `https://haveibeenpwned.com`. Cited on pages 11, 19, 32, 39, 49, 153, and 155.

[123] Philip Inglesant and Martina Angela Sasse. The true cost of unusable password policies: password use in the wild. In *Proc. CHI*, 2010. Cited on pages 29 and 50.

[124] InsidePro Software. PasswordsPro. `http://www.insidepro.com`. Cited on pages 12 and 51.

[125] International Computer Science Institute (ICSI). ICSI Certificate Notary. `https://notary.icsi.berkeley.edu`. Cited on page 148.

[126] International Organization for Standardization (ISO). Data elements and interchange formats – Information interchange – Representation of dates and times. ISO/IEC 8601, 2004. Cited on page 153.

[127] International Organization for Standardization (ISO). Identification cards – Integrated circuit cards – Part 8: Commands for security operations. ISO/IEC 7816-8, 2004. Cited on page 131.

[128] Iulia Ion, Rob Reeder, and Sunny Consolvo. "...No one Can Hack My Mind": Comparing Expert and Non-Expert Security Practices. In *Proc. SOUPS*, 2015. Cited on pages 1, 2, and 51.

[129] Blake Ives, Kenneth R. Walsh, and Helmut Schneider. The domino effect of password reuse. *CACM*, 47(4), 2004. Cited on pages 1 and 55.

[130] B. Kaliski. PKCS #5: Password-Based Cryptography Specification Version 2.0. Cited on pages 20, 55, and 111.

[131] Ambarish Karole, Nitesh Saxena, and Nicolas Christin. A Comparative Usability Evaluation of Traditional Password Managers. In *Proc. ICISC*, 2010. Cited on pages 1, 57, 60, and 162.

[132] Keisuke Kato and Vitaly Klyuev. Strong passwords: Practical issues. In *Proc. IDAACS*, 2013. Cited on pages 11 and 51.

[133] Mark Keith, Benjamin Shao, and Paul John Steinbart. The usability of passphrases for authentication: An empirical field study. *IJMMS*, 65(1), 2007. Cited on page 52.

[134] Mark Keith, Benjamin Shao, and Paul John Steinbart. A Behavioral Analysis of Passphrase Design and Effectiveness. *JAIS*, 10(2), 2009. Cited on page 52.

[135] Patrick Gage Kelley, Saranga Komanduri, Michelle L. Mazurek, Richard Shay, Timothy Vidas, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, and Julio Lopez. Guess Again (and Again and Again): Measuring Password Strength by Simulating Password-Cracking Algorithms. In *Proc. SP*, 2012. Cited on pages 1, 12, 13, 37, 49, 50, 51, and 52.

[136] Amin Kharraz, William K. Robertson, Davide Balzarotti, Leyla Bilge, and Engin Kirda. Cutting the Gordian Knot: A Look Under the Hood of Ransomware Attacks. In *Proc. DIMVA*, 2015. Cited on page 58.

[137] Johannes Kiesel, Benno Stein, and Stefan Lucks. A Large-scale Analysis of the Mnemonic Password Advice. In *Proc. NDSS*, 2012. Cited on page 52.

[138] Saranga Komanduri, Richard Shay, Patrick Gage Kelley, Michelle L. Mazurek, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, and Serge Egelman. Of passwords and people: measuring the effect of password-composition policies. In *Proc. CHI*, 2011. Cited on pages 29, 48, 49, 50, 52, and 55.

[139] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-Hashing for Message Authentication. RFC 2104, 1997. Cited on page 111.

[140] H. Krawczyk and P. Eronen. HMAC-based Extract-and-Expand Key Derivation Function (HKDF). RFC 5869, 2010. Cited on page 111.

[141] Cynthia Kuo, Sasha Romanosky, and Lorrie Faith Cranor. Human selection of mnemonic phrase-based passwords. In *Proc. SOUPS*, 2006. Cited on pages 1, 12, 48, and 52.

[142] Stanley A. Kurzban. Easily Remembered Passphrases: A Better Approach. *SIGSAC*, 3(2-4), 1985. Cited on page 52.

[143] Martin Landi. Stars' nude photo attack may have been down to password codes, 2014. `http://www.independent.ie/business/technology/news/stars-nude-photo-attack-may-have-been-down-to-password-codes-30552629.html`. Cited on page 19.

[144] LastPass Corporate. LastPass – The Last Password You Have to Remember. `https://lastpass.com`. Cited on pages 53, 57, 59, 60, 116, and 155.

[145] LastPass Corporate. LastPass Security Notification, 2011. `https://blog.lastpass.com/2011/05/lastpass-security-notification.html/`. Cited on page 57.

[146] LastPass Corporate. LastPass Security Notification, 2015. `https://blog.lastpass.com/2015/06/lastpass-security-notice.html/`. Cited on page 57.

[147] Legion of the Bouncy Castle Inc. Bouncy Castle Crypto APIs, 2017. `https://www.bouncycastle.org`. Cited on pages 110 and 113.

[148] Michael D. Leonhard and V. N. Venkatakrishnan. A Comparative Study of Three Random Password Generators. In *Proc. EIT*, 2007. Cited on page 53.

[149] Zhiwei Li, Warren He, Devdatta Akhawe, and Dawn Song. The Emperor's New Password Manager: Security Analysis of Web-based Password Managers. In *Proc. USS*, 2014. Cited on page 56.

[150] David Llewellyn-Jones and Graham Rymer. Cracking PwdHash: A Bruteforce Attack on Client-Side Password Hashing. In *Proc. PASSWORDS*, 2016. Cited on page 54.

[151] Ijlal Loutfi and Audun Jøsang. Passwords are not always stronger on the other side of the fence. In *Proc. USEC*, 2015. Cited on page 2.

[152] Stefan Lucks and Jakob Wenzel. Catena Variants – Different Instantiations for an Extremely Flexible Password-Hashing Framework. In *Proc. PASSWORDS*, 2015. Cited on page 20.

[153] Jerry Ma, Weining Yang, Min Luo, and Ninghui Li. A Study of Probabilistic Password Models. In *Proc. SP*, 2014. Cited on pages 13, 14, and 48.

[154] Majestic. Top Million Root Domains List. `https://majestic.com/reports/majestic-million`. Cited on page 87.

[155] Fatma Al Maqbali and Chris J. Mitchell. Password Generators: Old Ideas and New. In *Proc. WISTP*, 2016. Cited on page 54.

[156] Fatma Al Maqbali and Chris J. Mitchell. AutoPass: An Automatic Password Generator. In *Proc. ICCST*, 2017. Cited on pages 57, 95, and 166.

[157] Peter Mayer, Hermann Berket, and Melanie Volkamer. Enabling Automatic Password Change in Password Managers Through Crowdsourcing. In *Proc. PASSWORDS*, 2016. Cited on pages 60 and 155.

[158] Michelle L. Mazurek, Saranga Komanduri, Timothy Vidas, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, Patrick Gage Kelley, Richard Shay, and Blase Ur. Measuring password guessability for an entire university. In *Proc. CSS*, 2013. Cited on pages 13, 49, and 50.

[159] David McCandless. World's Biggest Data Breaches. `http://www.informationisbeautiful.net/visualizations/worlds-biggest-data-breaches-hacks/`. Cited on pages 19, 32, 39, 49, and 153.

[160] Daniel McCarney, David Barrera, Jeremy Clark, Sonia Chiasson, and Paul C. van Oorschot. Tapas: design, implementation, and usability evaluation of a password manager. In *Proc. ACSAC*, 2012. Cited on pages 22, 23, 56, and 136.

[161] Andrew Mehler and Steven Skiena. Improving Usability Through Password-Corrective Hashing. In *Proc. SPIRE*, 2006. Cited on page 52.

[162] William Melicher, Blase Ur, Sean M. Segreti, Saranga Komanduri, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. Fast, lean, and accurate: Modeling password guessability using neural networks. In *Proc. USS*, 2016. Cited on pages 1 and 14.

[163] Frank Miller. *Telegraphic code to insure privacy and secrecy in the transmission of telegrams*. CM Cornwell, 1882. Cited on pages 8, 99, 119, and 135.

[164] George A. Miller. The magical number seven, plus or minus two: some limits on our capacity for processing information. *Psychological review*, 63(2), 1956. Cited on page 22.

[165] Dennis Mirante and Justin Cappos. Understanding password database compromises. *TR-CSE-2013-02*, 2013. Cited on pages 11, 19, 20, 32, 39, and 49.

[166] Bodo Möller, Thai Duong, and Krzysztof Kotowicz. This POODLE bites: Exploiting the SSL 3.0 Fallback, 2014. Cited on pages 17 and 21.

[167] Robert Morris and Ken Thompson. Password Security – A Case History. *CACM*, 22(11), 1979. Cited on pages 2, 11, 12, and 49.

[168] Chanathip Namprempre, Phillip Rogaway, and Thomas Shrimpton. Reconsidering generic composition. In *Proc. EUROCRYPT*, 2014. Cited on pages 100 and 111.

[169] Arvind Narayanan and Vitaly Shmatikov. Fast dictionary attacks on passwords using time-space tradeoff. In *Proc. CCS*, 2005. Cited on pages 1 and 14.

[170] Gilbert Notoatmodjo and Clark D. Thomborson. Passwords and Perceptions. In *Proc. AISC*, 2009. Cited on page 55.

[171] M. Nottingham and E. Hammer-Lahav. Defining Well-Known Uniform Resource Identifiers (URIs). RFC 5785, 2010. Cited on pages 80 and 149.

[172] M. Nystrom and B. Kaliski. PKCS #10: Certification Request Syntax Specification Version 1.7. RFC 2986, 2000. Cited on page 113.

[173] Philippe Oechslin. Making a Faster Cryptanalytic Time-Memory Trade-Off. In *Proc. CRYPTO*, 2003. Cited on page 11.

[174] OneLogin Inc. Security Incident, 2017. `https://www.onelogin.com/blog/may-31-2017-security-incident`. Cited on page 57.

[175] Cecilia Pasquini, Pascal Schöttle, and Rainer Böhme. Decoy password vaults: At least as hard as steganography? In *Proc. SEC*, 2017. Cited on page 56.

[176] C. Percival and S. Josefsson. The scrypt Password-Based Key Derivation Function. RFC 7914, 2016. Cited on page 20.

[177] Alexander Peslyak. John the Ripper. `http://www.openwall.com/john/`. Cited on pages 12 and 51.

[178] Alexander Peslyak. Openwall wordlists collection. `http://www.openwall.com/wordlists/`. Cited on page 12.

[179] Pew Research Center. Heartbleed's Impact. `http://www.pewinternet.org/2014/04/30/heartbleeds-impact/`. Cited on pages 2 and 60.

[180] Benny Pinkas and Tomas Sander. Securing passwords against dictionary attacks. In *Proc. CCS*, 2002. Cited on page 19.

[181] Progress Software Corporation. NativeScript. https://www.nativescript.org. Cited on page 89.

[182] Niels Provos and David Mazières. A Future-Adaptable Password Scheme. In *Proc. USENIX, FREENIX Track*, 1999. Cited on page 20.

[183] Quantcast. Quantcast Top Million U.S. Web Sites. https://www.quantcast.com/top-sites. Cited on page 87.

[184] Ashwini Rao, Birendra Jha, and Gananand Kini. Effect of grammar on security of long passwords. In *Proc. CODASPY*, 2013. Cited on pages 48 and 52.

[185] Mudassar Raza, Muhammad Iqbal, Muhammad Sharif, and Waqas Haider. A survey of password attacks and comparative analysis on methods for secure authentication. *WASJ*, 19(4), 2012. Cited on pages 11, 17, and 21.

[186] Dominik Reichl. KeePass Password Safe. http://keepass.info. Cited on pages 53 and 91.

[187] Phillip Rogaway and Thomas Shrimpton. Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance. In *Proc. FSE*, 2004. Cited on page 8.

[188] Tanja Römer and Jean-Pierre Seifert. Information leakage attacks against smart card implementations of the elliptic curve digital signature algorithm. In *Proc. ESMART*, 2001. Cited on page 135.

[189] Blake Ross, Collin Jackson, Nick Miyake, Dan Boneh, and John C. Mitchell. Stronger Password Authentication Using Browser Extensions. In *Proc. USS*, 2005. Cited on pages 54 and 117.

[190] Karen Scarfone and Murugiah Souppaya. Guide to Enterprise Password Management. NIST Special Publication 800-118, 2009. Cited on page 150.

[191] Stuart E. Schechter, A. J. Bernheim Brush, and Serge Egelman. It's no secret: measuring the security and reliability of authentication via "secret" questions. In *Proc. SOUPS*, 2009. Cited on page 17.

[192] Stuart E. Schechter, Cormac Herley, and Michael Mitzenmacher. Popularity is everything: A new approach to protecting passwords from statistical-guessing attacks. In *Proc. HotSec*, 2010. Cited on page 50.

[193] Mario Schlipf. Passwort-Richtlinien. Bachelor thesis, Darmstadt University of Technology, Germany, 2014. Cited on page 91.

[194] Mario Schlipf. Password Policy Crawler. Master thesis, Darmstadt University of Technology, Germany, 2015. Cited on page 74.

[195] David Schmidt and Trent Jaeger. Pitfalls in the automated strengthening of passwords. In *Proc. ACSAC*, 2013. Cited on page 51.

[196] Stan Schroeder. Mark Zuckerberg's Twitter and Pinterest accounts hacked, possibly due to an awfully weak password. http://mashable.com/2016/06/06/mark-zuckerberg-twitter-hacked/. Cited on page 21.

[197] Tobias Seitz, Manuel Hartmann, Jakob Pfab, and Samuel Souque. Do Differences in Password Policies Prevent Password Reuse? In *Proc. CHI Extended Abstracts*, 2017. Cited on page 49.

[198] Richard Shay, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, Alain Forget, Saranga Komanduri, Michelle L. Mazurek, William Melicher, Sean M. Segreti, and Blase Ur. A Spoonful of Sugar?: The Impact of Guidance and Feedback on Password-Creation Behavior. In *Proc. CHI*, 2015. Cited on pages 29 and 50.

[199] Richard Shay, Patrick Gage Kelley, Saranga Komanduri, Michelle L. Mazurek, Blase Ur, Timothy Vidas, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. Correct horse battery staple: exploring the usability of system-assigned passphrases. In *Proc. SOUPS*, 2012. Cited on page 51.

[200] Richard Shay, Saranga Komanduri, Adam L. Durity, Phillip (Seyoung) Huh, Michelle L. Mazurek, Sean M. Segreti, Blase Ur, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. Can long passwords be secure and usable? In *Proc. CHI*, 2014. Cited on pages 13, 39, 49, 50, 51, and 52.

[201] Richard Shay, Saranga Komanduri, Adam L. Durity, Phillip (Seyoung) Huh, Michelle L. Mazurek, Sean M. Segreti, Blase Ur, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. Designing Password Policies for Strength and Usability. *TISSEC*, 18(4), 2016. Cited on pages 39, 49, 50, 51, and 52.

[202] Richard Shay, Saranga Komanduri, Patrick Gage Kelley, Pedro Giovanni Leon, Michelle L. Mazurek, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. Encountering stronger password requirements: user attitudes and behaviors. In *Proc. SOUPS*, 2010. Cited on pages 1, 48, 49, and 55.

[203] Chao Shen, Tianwen Yu, Haodi Xu, Gengshan Yang, and Xiaohong Guan. User practice in password security: An empirical study of real-life passwords in the wild. *COMPSEC*, 61, 2016. Cited on pages 22 and 49.

[204] David Silver, Suman Jana, Dan Boneh, Eric Yawei Chen, and Collin Jackson. Password Managers: Attacks and Defenses. In *Proc. USS*, 2014. Cited on page 56.

[205] Eugene H. Spafford. OPUS: preventing weak password choices. *COMPSEC*, 11(3), 1992. Cited on page 50.

[206] Peder Sparell and Mikael Simovits. Linguistic Cracking of Passphrases Using Markov Chains. *IACR*, 2016. Cited on page 52.

[207] Frank Stajano, Max Spencer, Graeme Jenkinson, and Quentin Stafford-Fraser. Password-Manager Friendly (PMF): Semantic Annotations to Improve the Effectiveness of Password Managers. In *Proc. PASSWORDS*, 2014. Cited on pages 33, 53, and 143.

[208] Elizabeth Stobert and Robert Biddle. The Password Life Cycle: User Behaviour in Managing Passwords. In *Proc. SOUPS*, 2014. Cited on pages 1, 22, 23, 29, 48, 49, and 136.

[209] Elizabeth Stobert and Robert Biddle. Expert Password Management. In *Proc. PASSWORDS*, 2015. Cited on page 2.

[210] Ben Stock and Martin Johns. Protecting users against XSS-based password manager abuse. In *Proc. ASIACCS*, 2014. Cited on page 56.

[211] Viktor Taneski, Marjan Hericko, and Bostjan Brumen. Password security – no change in 35 years? In *Proc. MIPRO*, 2014. Cited on pages 2 and 49.

[212] Viktor Taneski, Marjan Hericko, and Bostjan Brumen. Impact of security education on password change. In *Proc. MIPRO*, 2015. Cited on pages 2, 19, 49, and 60.

[213] The Tor Project Inc. TOR Project: Anonymity Online. `https://www.torproject.org`. Cited on page 80.

[214] Christian Thöing. PWGen. `http://pwgen-win.sourceforge.net`. Cited on page 53.

[215] TorGuard. TorGuard: online privacy protection services. `https://torguard.net`. Cited on page 80.

[216] Harshal Tupsamudre, Vijayanand Banahatti, and Sachin Lodha. POSTER: Improved Markov Strength Meters for Passwords. In *Proc. CCS*, 2016. Cited on pages 14 and 50.

[217] United States Computer Emergency Readiness Team (US-CERT). Alert (TA16-091A) – Ransomware and Recent Variants. `https://www.us-cert.gov/ncas/alerts/TA16-091A`. Cited on page 58.

[218] Blase Ur, Felicia Alfieri, Maung Aung, Lujo Bauer, Nicolas Christin, Jessica Colnago, Lorrie Faith Cranor, Henry Dixon, Pardis Emami Naeini, Hana Habib, Noah Johnson, and William Melicher. Design and Evaluation of a Data-Driven Password Meter. In *Proc. CHI*, 2017. Cited on page 50.

[219] Blase Ur, Jonathan Bees, Sean M. Segreti, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. Do Users' Perceptions of Password Security Match Reality? In *Proc. CHI*, 2016. Cited on page 55.

[220] Blase Ur, Patrick Gage Kelley, Saranga Komanduri, Joel Lee, Michael Maass, Michelle L. Mazurek, Timothy Passaro, Richard Shay, Timothy Vidas, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. How Does Your Password Measure Up? The Effect of Strength Meters on Password Creation. In *Proc. USS*, 2012. Cited on pages 13, 29, and 50.

[221] Blase Ur, Fumiko Noma, Jonathan Bees, Sean M. Segreti, Richard Shay, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. "I Added '!' at the End to Make It Secure": Observing Password Creation in the Lab. In *Proc. SOUPS*, 2015. Cited on pages 39, 48, 49, and 55.

[222] Blase Ur, Sean M. Segreti, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, Saranga Komanduri, Darya Kurilova, Michelle L. Mazurek, William Melicher, and Richard Shay. Measuring Real-World Accuracies and Biases in Modeling Password Guessability. In *Proc. USS*, 2015. Cited on page 37.

[223] Wali Ahmed Usmani, Diogo Marques, Ivan Beschastnikh, Konstantin Beznosov, Tiago João Vieira Guerreiro, and Luís Carriço. Characterizing Social Insider Attacks on Facebook. In *Proc. CHI*, 2017. Cited on pages 11 and 17.

[224] Rafael Veras, Christopher Collins, and Julie Thorpe. On Semantic Patterns of Passwords and their Security Impact. In *Proc. NDSS*, 2014. Cited on pages 1, 13, 35, and 48.

[225] Rafael Veras, Julie Thorpe, and Christopher Collins. Visualizing semantics in passwords: the role of dates. In *Proc. VIZSEC*, 2012. Cited on page 48.

[226] Kim-Phuong L. Vu, Robert W. Proctor, Abhilasha Bhargav-Spantzel, Bik-Lam (Belin) Tai, Joshua Cook, and E. Eugene Schultz. Improving password security and memorability to protect personal and organizational information. *IJMMS*, 65(8), 2007. Cited on pages 29, 50, and 51.

[227] W3Techs. Usage of content languages for websites. `https://w3techs.com/technologies/overview/content_language/all`. Cited on page 78.

[228] Ding Wang, Debiao He, Haibo Cheng, and Ping Wang. fuzzyPSM: A New Password Strength Meter Using Fuzzy Probabilistic Context-Free Grammars. In *Proc. DSN*, 2016. Cited on page 50.

[229] Ding Wang and Ping Wang. The Emperor's New Password Creation Policies: An Evaluation of Leading Web Services and the Effect of Role in Resisting Against Online Guessing. In *Proc. ESORICS*, 2015. Cited on pages 19, 31, 35, 49, and 50.

[230] Ding Wang, Zijian Zhang, Ping Wang, Jeff Yan, and Xinyi Huang. Targeted Online Password Guessing: An Underestimated Threat. In *Proc. CCS*, 2016. Cited on page 20.

[231] Luren Wang, Yue Li, and Kun Sun. Amnesia: A Bilateral Generative Password Manager. In *Proc. ICDCS*, 2016. Cited on page 56.

[232] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding Collisions in the Full SHA-1. In *Proc. CRYPTO*, 2005. Cited on page 21.

[233] Rick Wash, Emilee J. Rader, Ruthie Berman, and Zac Wellmer. Understanding Password Choices: How Frequently Entered Passwords Are Re-used across Websites. In *Proc. SOUPS*, 2016. Cited on pages 1 and 55.

[234] Matt Weir, Sudhir Aggarwal, Michael P. Collins, and Henry Stern. Testing metrics for password creation policies by attacking large sets of revealed passwords. In *Proc. CCS*, 2010. Cited on pages 1, 13, 35, and 37.

[235] Matt Weir, Sudhir Aggarwal, Breno de Medeiros, and Bill Glodek. Password Cracking Using Probabilistic Context-Free Grammars. In *Proc. SP*, 2009. Cited on pages 1 and 13.

[236] World Wide Web Consortium. Javascript Web Cryptography API. `https://www.w3.org/TR/WebCryptoAPI`. Cited on page 89.

[237] Jeff Jianxin Yan, Alan F. Blackwell, Ross J. Anderson, and Alasdair Grant. Password Memorability and Security: Empirical Results. *SP*, 2(5), 2004. Cited on pages 22 and 51.

[238] Weining Yang, Ninghui Li, Omar Chowdhury, Aiping Xiong, and Robert W. Proctor. An Empirical Study of Mnemonic Sentence-based Password Generation Strategies. In *Proc. CCS*, 2016. Cited on page 52.

[239] Hwei-Ming Ying and Noboru Kunihiro. Decryption of frequent password hashes in rainbow tables. In *Proc. CANDAR*, 2016. Cited on page 11.

[240] Saman Taghavi Zargar, James Joshi, and David Tipper. A Survey of Defense Mechanisms Against Distributed Denial of Service (DDoS) Flooding Attacks. *COMSUR*, 15(4), 2013. Cited on page 114.

[241] Emanuel von Zezschwitz, Alexander De Luca, and Heinrich Hussmann. Survival of the Shortest: A Retrospective Analysis of Influencing Factors on Password Composition. In *Proc. INTERACT*, 2013. Cited on pages 1 and 55.

[242] Yinqian Zhang, Fabian Monrose, and Michael K. Reiter. The security of modern password expiration: an algorithmic framework and empirical analysis. In *Proc. CCS*, 2010. Cited on pages 13, 55, and 60.

[243] Leah Zhang-Kennedy, Sonia Chiasson, and Paul C. van Oorschot. Revisiting password rules: facilitating human management of passwords. In *Proc. ECRIME*, 2016. Cited on page 55.

[244] Rui Zhao and Chuan Yue. Toward a secure and usable cloud-based password manager for web browsers. *COMPSEC*, 46, 2014. Cited on page 1.

[245] Dominik Ziegler, Mattias Rauter, Christof Stromberger, Peter Teufl, and Daniel M. Hein. Do you think your passwords are secure? In *Proc. PRISMS*, 2014. Cited on page 56.

[246] Herbert Spencer Zim. *Codes and secret writing*. W. Morrow, 1948. Cited on page 13.

[247] Moshe Zviran and William J. Haga. Password Security: An Empirical Study. *JMIS*, 15(4), 1999. Cited on pages 1, 19, 31, 48, 49, and 60.