PALPAS PAsswordLess PAssword Synchronization

Moritz Horsch
Technische Universität Darmstadt
Hochschulstraße 10
64283 Darmstadt, Germany
horsch@cdc.informatik.tu-darmstadt.de

Andreas Hülsing
Technische Universiteit Eindhoven
5600 MB Eindhoven, Netherlands
a.t.huelsing@tue.nl

Johannes Buchmann
Technische Universität Darmstadt
Hochschulstraße 10
64283 Darmstadt, Germany
buchmann@cdc.informatik.tu-darmstadt.de

***Abstract*—Tools that synchronize passwords over several user devices typically store the encrypted passwords in a central online database. For encryption, a low-entropy, password-based key is used. Such a database may be subject to unauthorized access which can lead to the disclosure of all passwords by an offline brute-force attack. In this paper, we present PALPAS, a secure and user-friendly tool that synchronizes passwords between user devices without storing information about them centrally. The idea of PALPAS is to generate a password from a high entropy secret shared by all devices and a random salt value for each service. Only the salt values are stored on a server but not the secret. The salt enables the user devices to generate the same password but is statistically independent of the password. In order for PALPAS to generate passwords according to different password policies, we also present a mechanism that automatically retrieves and processes the password requirements of services. PALPAS users need to only memorize a single password and the setup of PALPAS on a further device demands only a one-time transfer of few static data.**

I. Introduction

The concept of using passwords for user authentication on the Internet is cost-effective for services and easily comprehensible for users. However, the key challenge for users is to choose a strong password for each service and never reuse it for another service. This is essential for the security, but the wide adoption of passwords on the Internet makes it impossible for users to memorize the required amount of different strong passwords. To mitigate this conflict of having user-friendly and secure passwords one possible solution is storing some data locally on the user's device, which can be the passwords themselves or information to compute them. However, this requires an additional protection mechanism to ensure that only the legitimate user is able to access the data, e.g., by a proof of knowledge or possession. Another problem of storing data locally is that it is only available on a single device but nowadays users have multiple devices like smart phones and tablets. Furthermore, the data is not static. Passwords get added, changed, or removed. Thus, the data must be synchronized between devices to ensure that users can access their services at any time and any place. The data can be synchronized manually by directly connecting the devices or in a more convenient and user-friendly way through a synchronization server over the Internet.

There exist many solutions for the generation of individual passwords, secure password storage, and password synchronization, but they have serious drawbacks. The common paradigm of password synchronization is protecting passwords via encryption with a password-derived key and storing them at servers on the Internet. This bears the risk of security breaches at the synchronization servers where adversaries copy the encrypted data and perform offline attacks (cf. [14], [20]). Hence, there is a need for a synchronization scheme that is on the one hand not vulnerable to such attacks and on the other hand user-friendly with respect to the amount of information users need to memorize and the number of actions users need to perform to synchronize their passwords.

The protection of local and/or remote data as well as the user authentication at the synchronization server are usually based on a user-chosen (master) password. This enables various attacks based on web vulnerabilities and phishing attacks in which users are tricked into submitting their password to an adversary (cf. [4], [28]). Hence, there is a need for a solution that strictly separates the duties of providing data and privacy protection as well as user authentication.

Existing approaches to generate random passwords do not consider the password requirements of services, but this is crucial to create strong passwords. Usually, a default character set and password length are used, which hopefully fit the requirements of all services. Otherwise, users need to adapt the password or configure the password generator for the particular service. In both cases, users need to find the password requirements at the service's website or end up in an exhausting trial and error approach to find out which kinds of passwords are accepted. This situation is very inconvenient and leads users to use the first password accepted by the service and not the strongest and most appropriate one. Hence, there is a need for an automatic mechanism that allows password generators to retrieve and process the password requirements of services without requiring any user interaction.

In this work we present PALPAS (PAsswordLess PAssword Synchronization), a novel password tool that creates strong, service-specific passwords and synchronizes them between devices via a central server. However, PALPAS does not store or use any passwords on the server and is therefore not vulnerable to phishing attacks or security breaches.

978-1-4673-6590-1/15 $31.00 © 2015 IEEE
DOI 10.1109/ARES.2015.23

More specifically, we make the following contribution regarding the open issues described above:

- We present a synchronization scheme that shares data between devices through a synchronization server to enable them to compute the same passwords. However, the data is statistically independent of the passwords and hence does not reveal any information about them. Moreover, the user authentication at the synchronization server is not based on passwords but on public key cryptography. Therefore, a security breach at the synchronization server is non-critical, because there are no passwords that could be stolen by an adversary.
- We describe an approach to use different, independent high entropy secrets for password generation, data and privacy protection, and user authentication, which provides a maximum of security on the one hand but is still easily manageable and user-friendly on the other hand.
- We provide a common description of password policies to specify the password requirements of services in a standardized way. We describe a solution to create and distribute the policies. This allows us to finally provide a mechanism which automatically retrieves the password requirements of arbitrary services to create strong and service-specific passwords. It requires no user interaction, except that users need to specify for which service the password should be created.

This paper is organized as follows: In Section II we summarize related work. We describe PALPAS on a conceptual level in Section III and provide more details about its implementation in Section IV. In Section V we present a detailed security analysis of our concept and finally conclude the paper in Section VI. The appendix includes detailed protocol flows.

II. Related Work

Password managers like KeePass [25] and LastPass [19] as well as the browser built-in password managers of Chrome and Firefox [22] encrypt the passwords before storing them at servers on the Internet. The encryption key is derived from a user-chosen master password. A security breach at a synchronization server (cf. [20]) would allow adversaries to steal the encrypted data and to perform offline brute-force attacks. In essence, protecting passwords by another password and storing them on servers cannot be the foundation for a secure password synchronization.

Hash-based approaches like PwdHash [26] allow users to create different passwords for services by hashing a master password and the name or the URL of the service. Unfortunately, an adversary who stole a password can perform a brute-force attack to obtain the master password and thereby generate all user's passwords. Password Multiplier [15] performs additional steps to strengthen the master password, which increases the costs for a brute-force attack. Nevertheless, these hash-based approaches cannot be used to generate a new password for the same service, which is e.g. necessary after a password breach. The authors propose to use an additional user-chosen input for each service, which is included in the hashing, but this has the disadvantage that users then need to memorize this input for each service. Thus, the existing hash-based approaches are not a feasible solution because they still require users to memorize a lot of information.

Approaches which are using hardware tokens [27], [35] or mobile devices [29] for authentication have the disadvantage that users always need to carry an additional device. Furthermore, such solutions require changes on the infrastructure of the service. The weak development of other authentication mechanisms than passwords (cf. [5] for a survey) shows that service-side changes are a major obstacle for the wide adoption of authentication schemes.

Single sign-on (SSO) like Facebook Connect [13] allows users to authenticate themselves with a single password once and access multiple services without being prompted to log in at each service again. This can reduce the number of passwords users have to memorize but the adoption of SSO is still very limited [30]. Furthermore, SSO bears the risk of phishing attacks [36] and studies found out that users have several concerns and misconceptions about SSO and are not feeling comfortable with giving control of their passwords to external services [31], [32]. SSO has also serious privacy issues [10], because the SSO identity provider is aware of where and when a user performs a login. In summary, SSO does not solve the problem of managing many passwords and leads to new problems like the privacy issue.

III. PAsswordLess PAssword Synchronization

In this section we describe PALPAS on a conceptual level. We explain how PALPAS creates passwords for services, how the individual password requirements of each service are met during the password generation, how the passwords are synchronized between several user-devices, and finally how the various data is protected from unauthorized access. Technical details about the implementation, the used encryption schemes, the used key sizes, and so forth are presented in Section IV. Detailed protocol flows can be found in the Appendix.

A. Password Generation

PALPAS does not store a password as typical password managers do. It (re-)computes a password every time the user wants to log in to a service. PALPAS uses completely random passwords but ensures that they comply with the password requirements of a service. It also uses a different password

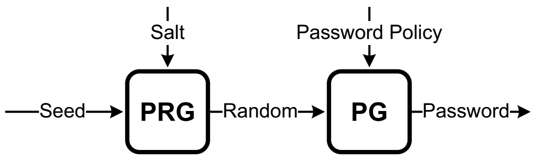

Fig. 1. Password Generation. The Pseudorandom Generator (PRG) generates randomness based on a seed and a salt value. The Password Generator (PG) derives a password from the randomness and ensures that it complies with a password policy specifying the password requirements of the service.

for each service and never reuses it. All that together allows to have an individual and strong password for each service.

As illustrated in Figure 1, a password is computed in two steps: First, a (cryptographically secure) Pseudorandom Generator (PRG) generates a pseudorandom value based on a seed and a salt value. Second, a Password Generator (PG) derives a password from the pseudorandom value and ensures that it complies with a given password policy. The password policy specifies the password requirements of the service. The PRG and the PG are deterministic, thus, using the same seed, salt, and password policy, the same password is generated. We present more details about the implementation of the PRG and the PG in Section IV-C and IV-E.

The seed S is a common secret shared by all user devices. It is randomly generated when the user uses PALPAS the first time and it must be transferred by the user to all his or her devices, but only once. In Section III-E we provide more details how to do this in a convenient way. To protect the seed from unauthorized access it is encrypted using a secret K_{MPW} derived from a user-chosen master password (MPW). It is important to stress that this secret is only used for the local data protection and local user authentication. More precisely, it ensures that only the user is able to access the seed. The secret K_{MPW} is neither stored on a server nor used for any kind of data protection in which the data leaves the device.

The objective of the salt is twofold. First, PALPAS uses a different salt value for each service to create different passwords for the services. Second, changing the salt for a service allows generating a new password for it, which is necessary in case of a password breach or regular required password changes. The salt is chosen by PALPAS and completely random. Consequently, it is statistically independent of the password and its knowledge does not allow to infer any user-, service-, or password-related information.

B. Password Policies

A password policy defines the password requirements of a service. It specifies the minimum and maximum password length, the allowed character sets, and additional constraints of the required characters. For instance, a password policy specifies that passwords must have at least 8 and a maximum of 10 characters, can consist of upper- and lowercase letters and digits, and must contain at least one digit.

PALPAS uses a separate password policy for each service, but instead of putting the burden of creating these policies on the user, we provide an automatic mechanism to process and retrieve the password policies of arbitrary services. We propose a community approach to create the password polices and distribute them through a central service. This has the benefit that it is independent from the services and does not require their contribution. The basic concept is that users create a password policy for a service once and make it available for other users by distributing it through the central service.

We developed a common description for password policies to specify the password requirements of services in a standardized way (cf. Section IV-D for an example). This builds the foundation to automatically retrieve and process password policies without any user interaction. We introduce the Password Policy Service (PPS) where applications like PALPAS can automatically request the policy of an arbitrary service. The PPS provides a form for users where they can enter the password requirements of a service. The PPS then automatically creates the corresponding password policy and publishes it. To ensure the correctness of the password policy the PPS performs multiple verifications. It performs plausibility and sanity checks to ensure, for instance, that the generated passwords have at least a certain threshold of entropy. Moreover, it requires that at least several users enter the same password requirements of a service before a password policy gets published. In addition, the PPS provides a reputation system to allow users to rate the policies and to assist users in deciding whether to trust or not to trust in a certain policy. Applications like PALPAS should also perform sanity checks to ensure that a generated password has a minimum level of entropy.

There is an issue when a service changes its password requirements but that only affects users that change the passwords. It requires that users submit the new password requirements to the PPS to update the password policy. PALPAS retrieves the password policies for the services only once and stores them on the device. In case the user wants to have a new password for a service PALPAS checks if there is a newer password policy to ensure that the password complies with the current password requirements.

In summary, our approach enables applications to automatically retrieve and process password policies, without requiring users to manually search for the password requirements at the service's website and configure the application. The first PALPAS users of a service still have to manually enter the requirements but we expect that after an initialization phase policies for all popular services are available. In Section IV-D we provide more details regarding the implementation of password polices.

C. Password Synchronization

As described in Section III-A a password for a service is computed from the seed, a salt, and a password policy. The seed is available on all user devices. The password policy can be retrieved from the PPS and is stored on the device after the first download. To enable all user devices to compute the same password for a service only the salt needs to be shared between the devices.

We introduce the Salt Synchronization Service (SSS), a central service that synchronizes the salt values between a user's devices. For each service or rather for each password of the user, the SSS stores a separate salt and an associated identifier. The identifier allows PALPAS to request the salt for a particular service. In the interest of simplification, we assume that the identifier is the URL of the service. In case the user wants to log in to a service, PALPAS requests the corresponding salt from the SSS and computes the password. A detailed protocol flow can be found in the Appendix.

TABLE I
SECRETS USED BY PALPAS

Secret	Objective	Generation	Properties
S	Password generation	- Random - First use of PALPAS	- Same for all devices - Manually transfered by the user only once
K_{MPW}	Local data protection and local user authentication	- Derived from master password - Every use of PALPAS	- Same for all devices - Entered on each device - Entered for every use of PALPAS
K_{E}	Remote data and privacy protection	- Random - First use of PALPAS	- Same for all devices - Manually transfered by the user only once
K_{Auth}	Device authentication	- Random - First use of PALPAS on a device	- Different for all devices - Generated by each device - T_{Auth} manually transfered only once

This approach also allows PALPAS to determine if a user already has an account at a particular service by checking if there is a salt for the service stored at the SSS. To synchronize a new password, PALPAS just adds the new salt and the identifier to the SSS. After synchronizing with the SSS, all user devices are able to compute the password and to perform the login. The same applies if the user updates the password for a service, PALPAS stores the new salt value using the old identifier at the SSS. This automatically makes the new password available to all other user devices.

Using the URL of a service as identifier leads to privacy issues as the SSS and potential adversaries would gain information about the services used by the user. Therefore, PALPAS obtains the identifier hashing the URL with a secret value K_{E}. This secret key is randomly generated during the first use of PALPAS. It must also be transferred by the user to all his or her devices when adding a new device. This can be done together with the seed.

Besides the password, a login to a service also demands a username. PALPAS also provide means to synchronize the usernames between the user's devices. Usernames are sensitive information as they might reveal a user's real name on the one hand and give adversaries an advantage in social engineering attacks. On the other hand, knowledge of a username might allow to connect a user with his actions e.g. in forums or on dating sites. Thus, PALPAS protects usernames by encrypting them using K_{E} from above and stores them on the SSS.

In summary, for each service, used by the user, the SSS stores a triple consisting of the salt, the salt identifier, and the encrypted username.

D. User Authentication at the SSS

We need to ensure that only the legitimate user is able to store, update, or delete his or her salt values at the SSS. Common approaches for user authentication at synchronization servers are password-based, which makes phishing attacks very likely. PALPAS uses public key based authentication with an independent secret K_{Auth} (and a corresponding public key) for the user authentication at the SSS. An authentication key pair is automatically generated on each user device. When the user uses PALPAS the first time, PALPAS automatically generates a key pair and uses it to create an account at the SSS. This happens completely transparent to the user. The SSS only stores public keys and hence an attack against the SSS leaks no information usable to hijack a user account. If the user wants to set up PALPAS on a further device, he or she requests an authentication token T_{Auth} from the SSS, which is used by the new device to register its new authentication secret K'_{Auth} at the SSS. This allows to revoke single authentication key pairs in case of loss. The authentication token T_{Auth} is transfered together with seed S and secret K_{E} during set-up.

E. Separation of Duties for Data and Privacy Protection and User Authentication

PALPAS needs to ensure that only the legitimate user is able to compute his or her passwords and to access the necessary data to do so. Furthermore, PALPAS must protect any personal information about the user to ensure his or her privacy. As summarized in Table I, PALPAS uses four different secrets to achieve the goals. The seed ensures that only the legitimate user is able to compute the passwords, even if adversaries get access to the salt values. The secret K_{MPW} protects the data stored on devices and ensures that only the user who knows the MPW can access the local data. Usernames and salt identifiers might reveal personal information about the user; therefore they are protected by K_{E} before storing them on the SSS. Finally, to ensure that only the legitimate user can access (and hence edit) the data stored on the SSS, each device authenticates itself with a different secret K_{Auth}. The seed S, the secret K_{E}, and the secret K_{Auth} are randomly generated with sufficiently high entropy to make brute-force attacks meaningless. Only the secret K_{MPW} is derived from a user-chosen MPW. However, K_{MPW} is only used for local data protection and local user authentication and we use common techniques to make brute-force attacks impractical (see Sec. IV-A). In essence, this provides a maximum of security while keeping PALPAS as user-friendly as possible.

As already mentioned, to set up a new device the user needs to transfer the seed S, the secret K_{E}, and the authentication token T_{Auth} to the new device. All this data can be encoded together into a QR code [8], a Base64 string, or transferred by manual file transfer. In particular the QR code-based approach is today well-known to users and very convenient.

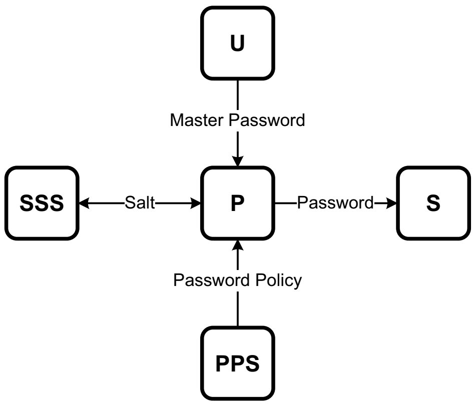

Fig. 2. Overview. The user (U) wants to log in to the service (S). PALPAS (P) prompts the user to enter his or her master password to access the seed. It retrieves the salt value from the Salt Synchronization Service (SSS) and the password policy from the Password Policy Service (PPS) for the service and finally computes the password.

F. Summary

In this section we described how PALPAS works and introduced all involved participants. Figure 2 provides an overview and shows the information flow between the participants. The user (U) needs to enter his or her MPW to enable PALPAS (P) to access the protected seed. To create a password for a service (S) PALPAS determines the identifier of the service and requests the corresponding salt value and username from the Salt Synchronization Service (SSS) as well as the password policy from the Password Policy Service (PPS). Finally, PALPAS computes the password.

IV. Implementation

The previous section described PALPAS on a conceptual level. We now present more technical details regarding the implementation and used technologies.

We implemented PALPAS as a proof-of-concept in Java and the SSS as well as the PPS as a Java web service. The SSS provides a SOAP-based interface for PALPAS to create an account as well as to manage the account and the stored data. The implementation of the user authentication is based on Apache CXF and Bouncy Castle. The PPS provides a RESTful API for applications to retrieve the policies. The source code will be made available under an open-source license at www.palpas.info.

A. Seed Generation and Protection

The seed is generated gathering 256 bits of entropy from a strong random number generator (in our case the Java class `SecureRandom`). It is locally stored, encrypted with secret K_{MPW} using AES-256. The secret key K_{MPW} is derived from the user's MPW using the PBKDF2 function [18].

B. Salt Generation and Synchronization

Using the URL of a service as identifier for the salt would raise privacy issues as argued above. Thus, PALPAS generates identifiers as $id = \text{SHA-256}(K_{\mathrm{E}} \| url)$ to protect a user's privacy. The parameter url represents the URL of the service, in detail, the domain name, for instance http://example.org. The parameter K_{E} is a 256 bit key, generated during set-up and shared by all user devices. Prepending the secret K_{E} adds sufficient entropy to the input to make a brute-force search for a valid preimage, containing a URL intractable.

C. Pseudorandom Generator

We implemented the PRG using AES in CBC mode. The seed S is used as key and the salt as input for the cipher. The ciphertexts form the pseudorandom output. We iteratively increment and then encrypt the salt value until enough random bits for the password generation are produced.

D. Password Policies

We specified an XML-based data format for password policies to provide a standardized description of the password requirements. XML is widely supported by many platforms and programming languages. This ensures an easy adoption of our solution for other password tools. We provide a brief example of a password policy in Listing 1. The example policy allows passwords that have at least 6 and at most 12 characters. The policy specifies a character set of lower- and uppercase letters as well as digits and requires that the password must contain at least one digit. For instance, this policy allows passwords like *Ha1K3A* and *QSJe1Kf4qYt*.

```
<PasswordPolicy>
  <MinLength>6</MinLength>
  <MaxLength>12</MaxLength>
  <CharacterSets>
    <CharacterSet name="LowercaseLetters">
      <Characters>abc...xyz</Characters>
    </CharacterSet>
    <CharacterSet name="UppercaseLetters">
      <Characters>ABC...XYZ</Characters>
    </CharacterSet>
    <CharacterSet name="Digits" minOccurrence="1">
      <Characters>0123456789</Characters>
    </CharacterSet>
  </CharacterSets>
</PasswordPolicy>
```

Listing 1. Password Policy. It specifies the minimum and maximum length of a password as well as the allowed character sets and additional restrictions.

E. Password Generator

The PG maps the random value generated by the PRG to a password which complies with a given password policy (cf. Section III-A). As example we consider a password policy that specifies a password length of 10 characters and a character set of uppercase and lowercase letters as well as digits. Thus, the character set has 26+26+10 = 62 characters in total. To derive a password of length ℓ using a character set of size $|characterset| = \phi$, the PG takes $100 + \lceil log_2(\phi^\ell) \rceil$ bits from the pseudorandom output and determines their decimal value modulo ϕ^ℓ. The resulting value is represented as ℓ base-ϕ numbers. In the above example the PG would take $100 + \lceil log_2(62^{10}) \rceil = 160$ bits, performs the modulo reduction, and splits the result in 10 numbers between 0 and 61. Each base-ϕ number then serves as an index in the character set. For instance, $51 = Z$ in our example character set. Using 100 bits more than required to get a number $> \phi^\ell$ guarantees an only negligibly biased distribution over the character set. More specifically, for each value x in the character set, the probability that x is chosen deviates from $1/\phi$ at most by 2^{-100}. Please note that consequently adding fewer additional bits (e.g. 20 bits) recognizably increases the probability of small indexes to occur. Thus, it would produce biased passwords.

The password policy may specify additional restrictions, for example, that a password has to contain at least one digit. To support this setting while ensuring that passwords are uniformly distributed, we use rejection sampling. We generate a password form the pseudorandom output and check if it fulfills all restrictions. If not, the password is discarded and we use the next block of pseudorandom output from the PRG. The new password is generated and verified. This process is repeated until the password fulfills all restrictions and the verification succeeds. While this process in theory might never terminate, it terminates in practice after a few iterations. E.g., in our running example, the probability that in τ consecutive iterations a password is generated that contains no number is $((1 - 10/62)^{12})^\tau = 0.12^\tau < 2^{-3\tau}$. This probability vanishes exponentially fast in the number of iterations. Please note that a straight-forward approach of considering restrictions during the password generation (e.g., generate $n-1$ characters, check if they contain a digit, if not select a digit for the n-th character) would produce biased passwords.

F. Protecting the Usernames

We use AES-256 in CBC mode with secret K_E to encrypt usernames before storing them on the SSS. Each username is encrypted separately using a different initialization vector (IV) to prevent that the same ciphertexts are produced for the same usernames. Both, ciphertext and IV are additionally protected using a Keyed-Hash Message Authentication Code (HMAC) [23]. Ciphertext, IV, and HMAC are stored on the SSS for each service in addition to the salt value. The encrypted usernames can be retrieved by all a user's devices in the same way as the salts, using the identifier of the service.

Please note that for simplification we only consider that a user has one account at each service (i.e., one username per service) for our prototype. However, our scheme can easily support several user accounts per service storing multiple salt values and encrypted usernames for the same identifier. PALPAS retrieves this information as usually by requesting the data stored on the SSS for a particular identifier. Instead of returning a single salt value and username, the SSS would return a list of salt values with the corresponding encrypted usernames. PALPAS decrypts the usernames and presents a selection dialog to the user.

G. User Authentication

As already mentioned, the user authentication at the SSS is not password-based but uses public-key cryptography with a random authentication secret K_Auth and corresponding public key. In detail, we are using TLS with client authentication for the user authentication at the SSS. Each user's device generates a new key pair comprising a public and a private key. During the account creation at the SSS, PALPAS sends a Certificate Signing Request (CSR) [24] to the SSS. The CSR contains only the public key and no further personal information. The SSS creates an X.509 certificate based on the CSR and transmits it to the device. The public key is bound to the account and stored in a list of trusted keys which are allowed to access and modify the salt values. Each user's device must add its own public key. To add an additional device an already registered device must request an authentication token T_Auth from the SSS. The user must transfer the token to the new device. The new device creates its own key pair and a corresponding CSR that includes T_Auth. Then it sends the token to the SSS. Hence, the authentication token ensures that only the user is able to add new devices.

Authenticating each device by its own public key has the advantage of fine-grained revocation. In case of theft or loss the user can revoke the access for a device by removing the public key from the list of trusted devices. Please see Section V-A for further details.

V. Security Evaluation

In this section we provide a detailed security analysis of PALPAS. We analyze the case of theft of a device, a phishing attack, security breaches at the services and the SSS, and finally a SSS failure.

A. Resistance to Theft or Loss of User Device

Storing passwords or data to compute them, like PALPAS does, bears the risk that devices are lost or get stolen, in particular in case of mobile devices. In that case, an adversaries might get access to the stored data. Although any mobile device is equipped with protection mechanisms such as passwords, PINs, or pattern locks, multiple surveys showed that only two third of the users use them [6], [11]. Even worse, unlock codes are often predictable and can be determined by simple guessing [3], [34]. In essence, locking mechanisms for devices are an important aspect but one cannot rely on them as a protection mechanism for sensitive data.

As already described, PALPAS uses a MPW to derive the secret K_MPW to encrypt the data stored on the device. However, this is only a first line of defense and increases the cost and effort for adversaries to obtain the data. It cannot be excluded that adversaries obtain the data carrying out a brute-force attack, especially if the user used a weak MPW. If encryption would be the only protection mechanism, users had to immediately change all their passwords because it would be just a question of time until adversaries find out the correct MPW. PALPAS has a second line of defense.

PALPAS computes a password using the seed and a salt. Either having the seed or the salt does not enable to compute the password. Thus, even if adversaries obtain the seed from a stolen device they need to get the salt values stored at the SSS to compute the user's passwords for the services. Each user's device uses a different authentication secret K_{Auth}, in detail, a different secret and public key, for the authentication at the SSS. As described in Section IV-G the SSS stores the public key of each device that is authorized to access the salt values. In case of a theft or loss, the user can revoke access for this device. Thus, if the MPW provides at least protection until the device's public key is revoked, the adversary is unable to retrieve the salts using the stolen device and consequently unable to compute the passwords. This gives the user enough time to change the seed on all devices and register the new passwords at all services.

B. Resistance to Phishing Attacks

Although password managers provide means against phishing attacks, they are also vulnerable to such attacks by themselves as soon as they use an online service, e.g., to synchronize passwords. The well-known procedure of luring users to fraud websites to gather usernames and passwords works also fine for password managers or synchronization services that use password-based authentication and provide a login through a web application. Our approach of using public key authentication or rather TLS client authentication is insusceptible against phishing attacks [2], [7], because there is simply no password that can be phished.

C. Resistance to Security Breaches

Numerous examples [1], [9], [12], [20], [33] tremendously showed the danger and impact of security breaches. Therefore, we analyze our approach regarding the threat of encountering a security breach at the SSS and/or at different services.

1) Security Breach at the Service(s): In the first attack scenario we consider that adversaries get access to the password database of a service and in particular to the password of a user. The password was generated by the PG as described in Sections III-A and IV-E. The functionality of the PG and the policy are publicly available. Hence, adversaries are able to invert the PG and obtain (parts of) the pseudorandom output, generated by the PRG. However, it follows from the pseudorandomness of the PRG that knowing the output it is still computationally hard to learn (1) the seed and (2) the output of the PRG on input of the same seed but with a different salt value. Thus, adversaries are unable to compute other passwords from a stolen password.

The same applies in case that adversaries get access to password databases of multiple services and extract the passwords of a user at each service. As already mentioned, adversaries are able to obtain the corresponding PRG outputs, but they are unable to learn the seed or salt values.

Finally, we consider adversaries who are able to access the password database of a service multiple times, in particular after a password change. In detail, for an arbitrary time period the adversaries get a list of passwords generated by PALPAS using the same seed but different salt values. Following the same argumentation as above, adversaries are able to invert the PG and obtain the corresponding pseudorandom output, but neither the seed nor the salts.

In essence, if adversaries are able to steal passwords, they cannot perform an offline attack to obtain information to compute other passwords. Due to the fact that our scheme generates an individual password for each service, adversaries can also not reuse the stolen password for other services.

Please note that in theory adversaries can try all combinations of the seed and the salt and test if they generate the same password as stolen from the service. However, in practice there are a number of obstacles. First, the fact that the seed and the salt are 256 bit random values results in a search space of $2^{256} * 2^{256} = 2^{512}$ bits. Second, even for a strong password policy that uses a 128 character set and length 20 passwords, there exist only 2^{140} possible passwords. Assuming that no further restrictions are applied and every seed-salt combination leads to a valid password, there are on average 2^{372} different seeds and salts that result in the same password. To find the correct seed-salt pair, an adversary has to verify that the seed and the salt are correct. The only way for the adversary to do this is to perform an online attack in which he computes a password for a different service and tests if it is accepted by that service. This is complicated by the fact that adversaries do not necessarily know which services a user uses. More importantly, services block accounts after a certain amount of wrong password attempts. Thus, even for computationally unbounded attackers an online attack is not applicable.

2) Security Breach at the SSS: In the second attack scenario we assume that adversaries are able to access the information stored at the SSS. In detail, the salt values, the encrypted salt identifiers, the encrypted usernames, and the public keys of the user's devices. The public keys for authentication are useless because the adversaries already have access to the SSS. The salts are randomly chosen values, independent of the passwords and do not reveal any information about the seed, the used services, or the corresponding passwords. To be precise, it follows from the pseudorandomness of PRG that only knowing the salt, guessing the output of PRG is infeasible. Therefore, adversaries are not able to generate any passwords. Thus, an offline attack is impossible.

Similar to the first scenario, in theory adversaries could brute-force the seed. This time, they even have the advantage of knowing a list of valid salt values. This reduces the search space to 2^{256} but it still forces adversaries to verify each seed in an online attack. In detail, for every seed and stolen salt the adversaries get a valid PRG output and password. Therefore, they need to verify the password with a service to check if they found the right seed. However, they do not know to which services the salts belong to and services restrict the number of login attempts. Using an additional brute-force attack against the secret K_{E} with complexity 2^{256} would allow an adversary to learn username and service URL. But they still need to test every seed online. Thus, an online brute-force attack is not feasible, even for computationally unbounded attacker.

In summary, a security breach at the SSS might allow adversaries to brute-force the secret K_{E} which affects the user's privacy but does not reveal any information to obtain the passwords of the user.

TABLE II
SECURITY BREACHES

Security Breach	**Seed**	**Salt**	**Salt Identifier**	**Username**	**Password**	**Challenges to obtain the data**
Service	○	○	○	●[a]	●[a]	- Seed and salt must be brute-forced (2^{512} bit) - Various valid seeds and salts - Password must be verified online
SSS	○	●	●[b]	●[b]	○	- Seed must be brute-forced (2^{256}) - Secret K_{E} must be brute-force (2^{256}) - Various valid seeds - Password must be verified online
SSS and Service	○	●	●[b]	●[b]	●[a]	- Seed must be brute-forced (2^{256}) - Secret K_{E} must be brute-force (2^{256}) - Various valid seeds - Password must be verified online

Table 2 summarizes which (protected) data is obtained (depicted by ●) and not obtained (depicted by ○) by adversaries in the respective security breach as well as which obstacles need to be overcome to break PALPAS. [a] Maybe encrypted by the service (out of scope). [b] Encrypted with secret K_{E}.

3) Security Breach at the Service and the SSS: In the third attack scenario we assume a security breach at the service and the SSS. Adversaries would now obtain the password used at the service and the data stored at the SSS. As described in the first scenario, adversaries can compute the PRG output from the password by inverting the PG. However, it follows from the pseudorandomness of the PRG that it is still computationally hard to learn the seed in this scenario.

An offline brute-force attack against the seed now becomes theoretically possible, because adversaries now have a password and a limited list of possible salt values. In case of a 256 bit seed as used by PALPAS such a brute-force attack would take 2^{255} evaluations of the PRG. Moreover, such a brute-force attack would find many different valid seeds: Continuing the example from the first case, the attack would find 2^{116} seeds on average. Hence, again an online phase is required to verify every single seed from this set. This again leads to the same challenges as in scenario 2, i.e., either guessing the service for a salt or running a brute-force attack on the secret K_{E}. We noticed that knowing the service and the username would theoretical allow a known-plaintext attack. However, in practice, e.g. using ciphers like AES, that is not an issue and it leads back to a brute-force attack to obtain K_{E}. In summary, even in case of a security breach at the SSS and the service adversaries are unable to perform an offline brute-force attack.

4) Summary: The three attack scenarios are summarized in Table II. It shows which data adversaries obtain in the case of a security breach in each attack scenario and what adversaries need to do to obtain the seed and break PALPAS. The common approach of encrypting passwords allows adversaries to run offline attacks and to brute-force the encryption key. Furthermore, they can always verify if their attack was successful. Our approach does not allow this, because any possible seed (and salt) generates a valid password for a service. Thus, offline attacks are not possible. Adversaries are forced to choose a seed (and a salt), generate the corresponding password, and check if it is accepted by a service to know if they found the correct seed.

D. Resistance to SSS failure

The password generation is based on the seed and salt values stored on the SSS. Without being able to retrieve the salts form the SSS PALPAS is not able to compute the password. Reasons for an unavailable SSS can be technical problems but also denial-of-service attacks. The reliability and fail-safe stability of the SSS can be ensured by redundancy of the hardware components to make technical problems very unlikely. Also denial-of-service attacks can be tackled by running multiple instances of the SSS on different locations, like common *Content Delivery Networks* do. Furthermore, there exist a lot of research on detection and defense mechanisms of denial-of-service attacks [21], [37]. In summary, there exist multiple technical and organizational means to provide a reliable SSS.

It would have been possible to deploy a client-side solution to cope with the problem. The salt value needs only be changed if the password is changed and we can safely assume that this happens very rarely. Hence, PALPAS could cache the salt values, which would allow generating the password even if the SSS is unavailable. However, this stands in direct contrast to the security advantage in case of theft or loss as described in Section V-A. Therefore, we decided against this solution and left it for future work.

VI. CONCLUSION AND FUTURE WORK

In this paper, we presented PALPAS that generates strong and service-specific passwords and synchronizes them between the user's devices in a secure manner. We provided a detailed security analysis and showed among other things that (1) security breaches at the SSS and/or at the services do not allow adversaries to obtain the passwords, (2) the SSS is not vulnerable to phishing attacks because the user authentication is not based on passwords, and (3) even in case of a device theft the passwords are protected. We also showed that the use of different secrets to provide data and privacy protection as well as user authentication improves the overall security and is still user-friendly. Users only need to remember a single master password. During setup of a new device some data has to be synchronized manually once. This can be done by scanning a QR code. We presented a solution that allows the password requirements of services to be automatically taken into account in the generation of strong passwords without any user interaction. The proposed standardized description, the automatic retrieval, and the distribution of password policies build a foundation and an open solution for various applications to include the password policies and to improve password-based authentication in the Internet.

In the future, we plan to investigate a peer-to-peer approach for the salt synchronization in which the devices exchange the salt values directly. This has the benefit that we do not need a central server for the synchronization as done by the SSS in our case. Furthermore, we plan to analyze a machine-learning approach to generate the password policies automatically. We envision a tool that extracts the password requirements automatically from the services' websites and creates the corresponding password policies.

For future work one might also consider a trade-off between security, in case of theft or loss, and availability, in case of unavailability of the SSS, caching salts. Caching could be limited and the salts could be encrypted locally. PALPAS could also be enhanced by more sophisticated authentication schemes like context-based authentication [16], [17]. The user's context, e.g. the location, could be part of a sophisticated cache erasing algorithm which deletes the cache if the device leaves known locations. Furthermore, the caching could be enhanced by pushing the salt values instead of fetching them on demand. More precise, if a user's device updates the salt at the SSS, the SSS could automatically push the new value to the other devices. This way all devices have the latest salts even if the SSS was unavailable with high probability. A sophisticated caching mechanism could protect the user in case of device theft or loss as well as guaranteeing availability even if the SSS is unavailable.

VII. Acknowledgments

This work was supported in part by the European Union within the 7FP project FutureID (ICT-318424) and by the Netherlands Organisation for Scientific Research (NWO) under grant 639.073.005.

References

[1] Adobe. Important Customer Security Announcement, 2013. http://blogs.adobe.com/conversations/2013/10/important-customer-security-announcement.html.

[2] A. Alsaid and C. Mitchell. Preventing phishing attacks using trusted computing technology. In *Proceedings of INC 2006, Sixth International Network Conference, Plymouth, UK, July 2006*, pages 221–228, 2006.

[3] P. Andriotis, T. Tryfonas, G. C. Oikonomou, and C. Yildiz. A pilot study on the security of pattern screen-lock methods and soft side channel attacks. In L. Buttyán, A. Sadeghi, and M. Gruteser, editors, *Sixth ACM Conference on Security and Privacy in Wireless and Mobile Networks, WISEC'13, Budapest, Hungary, April 17-19, 2013*, pages 1–6. ACM, 2013.

[4] K. Bhargavan and A. Delignat-Lavaud. Web-based attacks on host-proof encrypted storage. In E. Bursztein and T. Dullien, editors, *6th USENIX Workshop on Offensive Technologies, WOOT'12, August 6-7, 2012, Bellevue, WA, USA, Proceedings*, pages 97–104. USENIX Association, 2012.

[5] J. Bonneau, C. Herley, P. C. van Oorschot, and F. Stajano. The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *IEEE Symposium on Security and Privacy, SP 2012, 21-23 May 2012, San Francisco, California, USA*, pages 553–567, 2012.

[6] D. V. Bruggen, S. Liu, M. Kajzer, A. Striegel, C. R. Crowell, and J. D'Arcy. Modifying smartphone user locking behavior. In L. Bauer, K. Beznosov, and L. F. Cranor, editors, *Symposium On Usable Privacy and Security, SOUPS '13, Newcastle, United Kingdom, July 24-26, 2013*, page 10. ACM, 2013.

[7] R. Clayton. Insecure real-world authentication protocols (or why phishing is so profitable). In B. Christianson, B. Crispo, J. A. Malcolm, and M. Roe, editors, *Security Protocols, 13th International Workshop, Cambridge, UK, April 20-22, 2005, Revised Selected Papers*, volume 4631 of *Lecture Notes in Computer Science*, pages 82–88. Springer, 2005.

[8] Denso Wave. Quick Response Code, 2015. http://www.qrcode.com.

[9] eBay Inc. eBay Inc. to ask eBay users to change passwords, 2014. http://blog.ebay.com/ebay-inc-ask-ebay-users-change-passwords/.

[10] S. Egelman. My profile is my password, verify me!: the privacy/convenience tradeoff of facebook connect. In *2013 ACM SIGCHI Conference on Human Factors in Computing Systems, CHI '13, Paris, France, April 27 - May 2, 2013*, pages 2369–2378, 2013.

[11] S. Egelman, S. Jain, R. S. Portnoff, K. Liao, S. Consolvo, and D. Wagner. Are you ready to lock? In G. Ahn, M. Yung, and N. Li, editors, *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, pages 750–761. ACM, 2014.

[12] Evernote Corporation. Security Notice: Service-wide Password Reset, 2013. https://evernote.com/corp/news/password_reset.php.

[13] Facebook. Add Facebook Login to Your App or Website, 2015. https://developers.facebook.com/docs/facebook-login.

[14] P. Gasti and K. B. Rasmussen. On the security of password manager database formats. In S. Foresti, M. Yung, and F. Martinelli, editors, *Computer Security - ESORICS 2012 - 17th European Symposium on Research in Computer Security, Pisa, Italy, September 10-12, 2012. Proceedings*, volume 7459 of *Lecture Notes in Computer Science*, pages 770–787. Springer, 2012.

[15] J. A. Halderman, B. Waters, and E. W. Felten. A convenient method for securely managing passwords. In A. Ellis and T. Hagino, editors, *Proceedings of the 14th international conference on World Wide Web, WWW 2005, Chiba, Japan, May 10-14, 2005*, pages 471–479. ACM, 2005.

[16] E. Hayashi, S. Das, S. Amini, J. I. Hong, and I. Oakley. CASA: context-aware scalable authentication. In L. Bauer, K. Beznosov, and L. F. Cranor, editors, *Symposium On Usable Privacy and Security, SOUPS '13, Newcastle, United Kingdom, July 24-26, 2013*, page 3. ACM, 2013.

[17] R. J. Hulsebosch, M. S. Bargh, G. Lenzini, P. W. G. Ebben, and S. M. Iacob. Context sensitive adaptive authentication. In G. Kortuem, J. Finney, R. Lea, and V. Sundramoorthy, editors, *Smart Sensing and Context, Second European Conference, EuroSSC 2007, Kendal, England, October 23-25, 2007, Proceedings*, volume 4793 of *Lecture Notes in Computer Science*, pages 93–109. Springer, 2007.

[18] B. Kaliski. PKCS #5: Password-Based Cryptography Specification Version 2.0. RFC 2898, Sep 2000. https://tools.ietf.org/html/rfc2898.

[19] LastPass Corporate. LastPass - The Last Password You Have to Remember. https://lastpass.com.

[20] LastPass Corporate. LastPass Security Notification. https://blog.lastpass.com/2011/05/lastpass-security-notification.html/.

[21] A. Mishra, B. B. Gupta, and R. C. Joshi. A comparative study of distributed denial of service attacks, intrusion tolerance and mitigation techniques. In *European Intelligence and Security Informatics Conference, EISIC 2011, Athens, Greece, September 12-14, 2011*, pages 286–289. IEEE, 2011.

[22] Mozilla. Firefox Accounts/Sync Protocol. https://github.com/mozilla/fxa-auth-server/wiki/onepw-protocol.

[23] C. Namprempre, P. Rogaway, and T. Shrimpton. Reconsidering generic composition. In P. Q. Nguyen and E. Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, volume 8441 of *Lecture Notes in Computer Science*, pages 257–274. Springer, 2014.

[24] M. Nystrom and B. Kaliski. PKCS #10: Certification Request Syntax

Specification Version 1.7. RFC 2986 (Informational), Nov 2000. https://tools.ietf.org/html/rfc2986.

[25] D. Reichl. KeePass Password Safe. http://keepass.info.

[26] B. Ross, C. Jackson, N. Miyake, D. Boneh, and J. C. Mitchell. Stronger password authentication using browser extensions. In P. McDaniel, editor, *Proceedings of the 14th USENIX Security Symposium, Baltimore, MD, USA, July 31 - August 5, 2005*. USENIX Association, 2005.

[27] F. Stajano, G. Jenkinson, J. Payne, M. Spencer, Q. Stafford-Fraser, and C. Warrington. Bootstrapping adoption of the pico password replacement system. In B. Christianson, J. A. Malcolm, V. Matyás, P. Svenda, F. Stajano, and J. Anderson, editors, *Security Protocols XXII - 22nd International Workshop Cambridge, UK, March 19-21, 2014 Revised Selected Papers*, volume 8809 of *Lecture Notes in Computer Science*, pages 172–186. Springer, 2014.

[28] B. Stock and M. Johns. Protecting users against xss-based password manager abuse. In S. Moriai, T. Jaeger, and K. Sakurai, editors, *9th ACM Symposium on Information, Computer and Communications Security, ASIA CCS '14, Kyoto, Japan - June 03 - 06, 2014*, pages 183–194. ACM, 2014.

[29] H. Sun, Y. Chen, and Y. Lin. opass: A user authentication protocol resistant to password stealing and password reuse attacks. *IEEE Transactions on Information Forensics and Security*, 7(2):651–663, 2012.

[30] S. Sun, Y. Boshmaf, K. Hawkey, and K. Beznosov. A billion keys, but few locks: the crisis of web single sign-on. In A. D. Keromytis, S. Peisert, R. Ford, and C. Gates, editors, *Proceedings of the 2010 Workshop on New Security Paradigms, Concord, MA, USA, September 21-23, 2010*, pages 61–72. ACM, 2010.

[31] S. Sun, E. Pospisil, I. Muslukhov, N. Dindar, K. Hawkey, and K. Beznosov. What makes users refuse web single sign-on?: an empirical investigation of openid. In L. F. Cranor, editor, *Symposium On Usable Privacy and Security, SOUPS '11, Pittsburgh, PA, USA - July 20 - 22, 2011*, page 4. ACM, 2011.

[32] S. Sun, E. Pospisil, I. Muslukhov, N. Dindar, K. Hawkey, and K. Beznosov. Investigating users' perspectives of web single sign-on: Conceptual gaps and acceptance model. *ACM Trans. Internet Techn.*, 13(1):2, 2013.

[33] Twitter Incorporated. Keeping our users secure, 2013. https://blog.twitter.com/2013/keeping-our-users-secure.

[34] S. Uellenbeck, M. Dürmuth, C. Wolf, and T. Holz. Quantifying the security of graphical passwords: the case of android unlock patterns. In A. Sadeghi, V. D. Gligor, and M. Yung, editors, *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 161–172. ACM, 2013.

[35] K. A. Varmedal, H. Klevjer, J. Hovlandsvåg, A. Jøsang, J. Vincent, and L. Miralabé. The offpad: Requirements and usage. In J. Lopez, X. Huang, and R. Sandhu, editors, *Network and System Security - 7th International Conference, NSS 2013, Madrid, Spain, June 3-4, 2013. Proceedings*, volume 7873 of *Lecture Notes in Computer Science*, pages 80–93. Springer, 2013.

[36] C. Yue. The devil is phishing: Rethinking web single sign-on systems security. In V. Paxson, editor, *6th USENIX Workshop on Large-Scale Exploits and Emergent Threats, LEET '13, Washington, D.C., USA, August 12, 2013*. USENIX Association, 2013.

[37] S. T. Zargar, J. Joshi, and D. Tipper. A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks. *IEEE Communications Surveys and Tutorials*, 15(4):2046–2069, 2013.

Appendix

This appendix provides detailed descriptions of the protocol flows for creating a new account at the SSS, adding a new device, logging in to a service, adding a new password for synchronization, and updating a password. The following diagrams contain the Salt Synchronization Service (SSS), PALPAS (P) running on a user's device, and a service (S).

1) Creating an Account at the SSS: First, P running on the user's device creates a private and a public key as well as a corresponding Certificate Signing Request (CSR). The CSR only contains the public key and no personal information such as the distinguished name, country name, email address, and so forth. P sends the CSR to the SSS and require a new account for the user (cf. Figure 3). Subsequently, the SSS issues a certificate, binds it to the account, and response it to P. The certificate will used in any further communication to the SSS using TLS with client authentication.

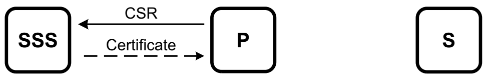

Fig. 3. Account creation at the SSS.

2) Adding a Device: P authenticates to the SSS from an already registered device using a valid certificate and requests an authentication token. The user transfers the token to the new derive and P', running on the new device, generates a key pair and a CSR. Subsequently, it sends the CSR and the token to the SSS. Finally, the SSS creates a new certificate for P' and sends the certificate to it.

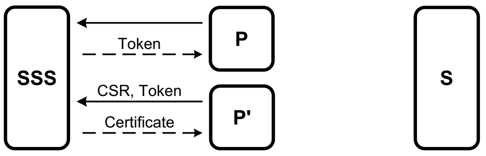

Fig. 4. Adding a Device.

3) Synchronizing a new Password: P generates a new salt and the corresponding password. The user updates the password at the service. P creates the identifier (id) of the service and stores the salt at the SSS.

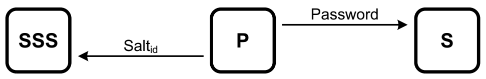

Fig. 5. Synchronizing a Password.

4) Logging in to a Service: To login P determines the identifier of the service and requests the corresponding salt from the SSS (cf. Figure 6). P computes the password and performs the login.

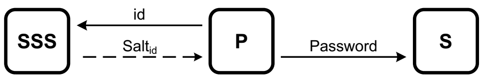

Fig. 6. Login.

5) Changing a Password: P generates a new salt and the corresponding password. The user updates the password at the service (cf. Figure 7). P updates the stored salt at the SSS.

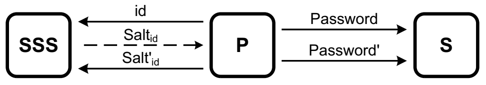

Fig. 7. Password Change.