AutoPass: An Automatic Password Generator

Fatma Al Maqbali and Chris J Mitchell
Information Security Group, Royal Holloway, University of London

***Abstract*—Text password is a very common user authentication technique. Users face a major problem, namely that of managing many site-unique and strong (i.e. non-guessable) passwords. One way of addressing this is by using a *password generator*, i.e. a client-side scheme which generates (and regenerates) site-specific strong passwords on demand, with minimal user input. This paper gives a detailed specification and analysis of AutoPass, a novel password generator scheme. AutoPass has been designed to address issues identified in previously proposed password generators, and incorporates novel techniques to address these issues. Unlike almost all previously proposed schemes, AutoPass enables the generation of passwords that meet important real-world requirements, including forced password changes, use of pre-specified passwords, and passwords meeting site-specific requirements.**

***Index Terms*—authentication, password, password generator, password management, password policy.**

I. Introduction

Despite its well-known shortcomings, text password authentication is widely used. Many attempts have been made to replace password authentication, e.g. using biometrics, tokens and multi-factor authentication [1]. However, single-factor password authentication remains very widely used. Moreover, in recent years the number of widely-used password-protected services has grown significantly, increasing the number of passwords users must remember. There are many issues associated with the use of text passwords [2], including: (a) users may use the same password for multiple accounts; (b) users will often choose guessable passwords, e.g. birthday or pet name; (c) users will often make minimal modifications to an existing password, e.g. by including a serial number, when forced to make a change; (d) many sites enforce complex password policies, e.g. enforcing regular password changes or constraining passwords (e.g. to contain a minimum number of, or include/exclude specific, characters), making memorising passwords even more difficult.

A range of approaches have been devised to help users. Perhaps best known are *password managers*, which store user passwords either locally or in the cloud, and retrieve them automatically when needed; many web browsers have such functionality. A related but distinct approach involves *password generators*, which generate strong and random-looking passwords and regenerate them when necessary. A number of such schemes have been proposed [3], [4], [5], [6], [7], [8]. In a previous paper, [9], we evaluated existing password generator schemes and discussed their strengths and weaknesses; this enabled us to outline a new password generator which we called *AutoPass*. It combines features from existing password generators with novel techniques designed to address identified shortcomings in the existing schemes. In this paper we provide the first detailed specification of AutoPass, and also give a detailed analysis of its properties.

The remainder of the paper is as follows. II defines and gives a general model for password generators (based closely on [9]). III gives a high-level description of AutoPass, building on the previous outline. This is followed in IV by a detailed specification of AutoPass. V provides an analysis of AutoPass, and in particular highlights how it addresses known shortcomings of such schemes. Finally, the paper concludes in VI.

II. A General Model

1) Definition and model: We consider *password generators*, i.e. schemes designed to generate site-specific passwords on demand. This term has also been used to describe schemes that generate random or pseudorandom passwords which the user must then remember; however, we use the term to describe systems intended to be used whenever a user logs in and that generate the necessary passwords on demand and in a repeatable way. A variety of such schemes have been proposed in recent years. The general class of such schemes has been briefly considered previously by McCarney [10] under the name *generative password managers*. We use the following general model for password generators as the basis for describing our novel scheme, AutoPass.

In a password generator, *input values* determine the password for a particular site. Some values must be site-specific so the password is site-specific. The values could be stored (locally or online), based on website characteristics, or user-entered when needed. A *password generation function* combines the inputs to generate a password, and must meet the requirements of the authenticating website. E.g., one web site might forbid non-alphanumeric characters in a password, whereas another might insist that a password contains at least one such character. Thus, to be broadly applicable, a password generation function must be customisable. A *password output method* transfers generated passwords to authenticating sites, e.g. by displaying the generated password to the user, who then types (or copies/pastes) it into the appropriate place. This functionality must be implemented on the user platform, e.g. as a stand-alone application or browser plug-in.

2) Prior art: Password generator schemes conforming to the above model include the following. The *Site-Specific Passwords (SSP)* scheme proposed by Karp [4] in 2002/03 is one of the earliest proposed schemes. SSP generates a password by combining a user master password and a memorable user-chosen name for the web site. *PwdHash*, due to Ross et al. [6], generates a password by combining a user master password, data associated with the web site, and (optionally) a second global password stored on the platform. The 2005 *Password Multiplier* scheme of Halderman, Waters and Felten, [3], computes a password as a function of a user master password, the web site name, and the user name for the web site. Wolf and Schneider's 2006 *PasswordSitter* [7] generates a password as a function of a user master password, the user identity, the application/service name, and some configurable parameters. *Passpet*, due to Yee and Sitaker [8] and also published in 2006, is similar to SSP, i.e. a password is a function of a user master password and a *petname*, i.e. a user-chosen name for the site (with an associated icon, displayed to the user to help reduce the risk of phishing). *ObPwd*, due to Mannan et al. [11], [5], [12], [13] from 2008, generates a password from a user-selected (site-specific) object, e.g. a file, together with optional parameters, e.g. a long-term user password (referred to as a *salt*), and the URL. Finally, *PALPAS* [14] generates passwords complying with site-specific requirements using server-provided password policy data, a stored secret master password (the *seed*), and a client-stored secret (the *salt*) that is synchronised across all user devices using a server.

There are also widely available implementations. *RndPhrase* (https://rndphrase.appspot.com/) is a Firefox add-on/web system that generates passwords as a function of a predefined user-unique salt, host name, and user-entered master password. *PwdHash port* (https://addons.opera.com/en-gb/extensions/details/pwdhash-port/) is an Opera add-on based on PwdHash. *Password generator* (https://goo.gl/SNVtJY), another Android app, generates passwords as a function of a configurable set of parameters including a user salt.

3) Registration and Configuration: We consider schemes that are completely transparent to the authenticating website, so the 'normal' website registration procedure, where the user selects a password and submits it, is used. Thus password generation should occur *before* user registration, since use of a password generator requires a user to modify their existing passwords. This causes usability problems with all previously proposed password generator schemes, as discussed in II-4.

There is a potential need for a password generator to store configuration data, including: *user-specific configuration data*, user- unique values used to help generate all passwords for that user, e.g. a master password; and *site-specific configuration data*, used to help generate passwords for a specific website which are the same for all users, e.g. a password policy. Not all schemes use configuration data, although producing a usable system without at least some user-specific configuration data is challenging. However, configuration data is clearly a major barrier to portability since the configuration data must be kept synchronised across all user platforms, a non-trivial task — see Horsch et al. [15].

4) Issues with Existing Schemes: We next outline three problems that affect almost all previously proposed password generators. These issues motivate the design of AutoPass, which incorporates novel features designed to overcome them.

- **Setting and updating passwords** As noted above, if a user is already using the password generator when newly registering with a website, the user can simply register whatever value the system generates. However, if the user has selected and registered passwords with a range of websites before starting use of the password generator, then all these passwords will need to be changed to whatever the password generator outputs. This could be highly inconvenient if a user has many established relationships, and could present a formidable barrier to adoption of the system.
  Analogous problems arise if a user decides to change a website password, e.g. because the site enforces periodic changes. The only possibility for the user will be to change one of the inputs used to generate the password, e.g. the object (if a digital object is used as an input) or a user site name. Password change could even be impossible if the user does not choose any of the inputs used to generate a password.
- **Using multiple platforms** If a user has multiple platforms, e.g. a desktop and phone, then synchronising locally-stored configuration data is problematic.
- **Password policy issues** A further problem arises from the need to generate passwords in a site-specific form, an issue not satisfactorily addressed by any previously proposed schemes except PALPAS. Some existing schemes have the option for the user to customise a generated password, but the user has to identify the requirements for the website manually and configure the options accordingly. Automatically generating passwords tailored to meet website-specific requirements has been explored by Horsch et al. [14], [15].

Of course, password *managers* (unlike password *generators*, the main focus here) do not, in general, suffer from the first and third of the above problems, since they store whatever passwords are chosen by the user. However, as widely discussed (see, for example, [9]), password managers have their own issues, and we do not discuss them further here.

III. AutoPass: Instantiating the Model

1) AutoPass Components: AutoPass has two main components: the AutoPass server and the AutoPass client. The AutoPass server stores less sensitive user data, e.g. user name and website-specific password policies (specifying the types of password a particular site will accept). The AutoPass client software provides a user interface, and automatically generates site-specific user passwords by combining the specified set of inputs. Some inputs are stored locally and some are stored in the AutoPass server, with which the client software interacts as necessary. Where possible, the generated password is automatically inserted into login forms. Whilst data exchanged between the AutoPass client and server is not highly confidential, some is privacy-sensitive and its integrity is crucial for correct operation. All data exchanged between client and server is therefore protected using a server-authenticated TLS channel established at the beginning of a client session. To make server authentication robust, the client is assumed to use certificate pinning for the server. Figure 1 depicts the AutoPass architecture, showing the main components of the scheme.

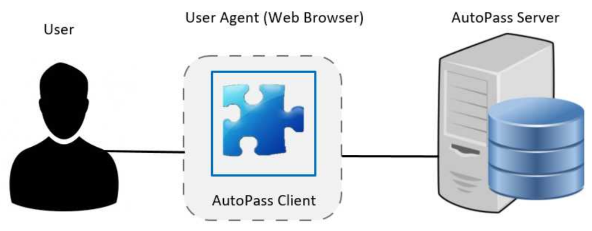

Fig. 1. AutoPass System Architecture

AutoPass conforms to the general model given above. The three main elements, i.e. the input values, the password generation function, and the output method, are as follows.

AutoPass uses the following input types (as used in previous schemes). (a) A **master password**: a long-term strong password selected by the user or generated by the system. It is stored in encrypted form on the AutoPass server as part of the user-specific configuration data. Since it does not need to be remembered by the user, it could, for example, be a 128-bit random value; the precise choice is implementation-dependent. The user should make a written record of this value when it is initially chosen, and store it securely for backup/recovery purposes. (b) The **site name**: the URL of the site for which AutoPass is generating a password. To overcome the issue of changes to URL sub-domains, AutoPass only uses the first part of the URL (i.e. up to the first / character). (c) A **password policy**: specifying the site-specific password requirements, e.g. length constraints and/or minimum numbers of certain classes of character. The policy is specified using the Password Requirements Markup Language (PRML) [15]. (d) A **digital object**: a text fragment, picture, or audio sample, e.g. in the form of a digital file. This is an optional input that potentially adds significant entropy to the password generation process, e.g. for use when generating passwords protecting high-value resources. Such objects need to be available on all user platforms, so objects should be selected with care to avoid causing cross-platform mobility issues.

The process of combining input values to produce the site-specific password has two stages. The first stage involves combining the input values, including the master password and the URL, to produce a bit string. Following Kelsey et al. [16], this computation involves a two-level hash computation, as follows. (a) The master password is submitted to a cryptographic hash-function, e.g. SHA-256, [17], that is iterated n times, where n is chosen to be as large as possible without making the client software too unresponsive. n can be user-dependent, as long it is held in the server to enable it to be synchronised across all user platforms. The output, a 256-bit string, is then cached by the client. Since this value is website-independent, it can be computed once when the client software is started up and cached locally while the client is active. (b) The 256-bit string is concatenated with website-specific inputs (site name and optional digital object) and hashed again to give a site-specific string. This two-level process protects against brute force attacks by slowing them down. Use of a multiply-iterated hash means that any brute force search will involve significantly more computational effort than it otherwise would; however, since the iterated part is only computed once per session, the additional load on the genuine client will be manageable.

The second stage (encoding) involves constructing a password from the output of the first stage (and, possibly, the offset), using the PRML policy specification to ensure the password meets website-specific requirements. How encoding operates is described in III-3 below. The generated password is automatically copied to the target password field. AutoPass uses secure filling techniques to prevent sweeping attacks [18].

We propose to implement AutoPass as a browser add-on, enabling automation of key tasks including fetching the website URL and inserting passwords into login forms. In future, for use with platforms not permitting add-ons, we plan to examine stand-alone applications and web-based functionality.

2) Stored Data: AutoPass needs access to a variety of configuration data, stored at the AutoPass server and client. The configuration data stored at the server is held long-term, i.e. for the lifetime of the user account; data held on the client may be held either short-term, e.g. for the life of a session, or long-term, i.e. while the software remains installed.

The following user-specific configuration data is held at the AutoPass server: (a) the user account name; (b) a user email address; (c) the (encrypted) master password; (d) a hash of the master password; (e) a (salted) hash of the login password (see IV-1). Also, for each website for which a password has been generated, the server holds: the (first part) of the URL; the input types used to generate the password; the password offset (see III-4).

The following site-specific configuration data is held at the AutoPass server: (a) the (first part) of the website URL; and (b) the PRML-encoded password policy of the site (see III-3). Note that the site-specific data could be maintained by a server separate from that used to store the user-specific data. Indeed, since this data is completely non-confidential, it could be provided by a service independent of AutoPass, e.g. the Password Requirements Description Distribution Service (PRDDS) [15], which provides an online interface to meet requests for PRML-based Password Requirements Descriptors (PRDs) for websites identified by their URL.

The following data is held short-term at the client: (a) the login password (see IV-1); and (b) a multiply-iterated hash of the master password (see III-1). The AutoPass client also caches recently downloaded password policies.

3) PRML: The Password Requirements Markup Language (PRML) [15] is an XML-based syntax for specifying password requirements, including minimum and maximum lengths, permissible character set, and minimum required number(s) of specific types of characters. It addresses the diversity of password requirements arising in practice, and enables automatic generation of passwords matching site-specific requirements.

A website's PRML specification is one of the two inputs for the second stage of password generation described in III-1, the other being the bit string output from the first stage. The second stage of password generation operates as follows. (1) The size C of the password character set is derived from the PRML specification; we suppose that a mapping is chosen from the set of integers $\{0, 1, \ldots, C-1\}$ to the characters in the password character set. (2) The length L of the password is chosen to be the minimum of 16 and the minimum length prescribed by the PRML policy. (3) The input bit string is converted to a positive integer by regarding the string as the binary representation of a number, and this number is converted to its C-ary representation $d_t d_{t-1} \ldots d_0$, for some t, where $0 \leq d_i \leq C-1$ for every i $(0 \leq i \leq t)$. (4) The final L digits of the above sequence of numbers, i.e. $d_{L-1} d_{L-2} \ldots d_0$, are converted to characters using the mapping established in step 1. (5) The password is tested to verify it satisfies the other constraints in the PRML specification. If not, then the input bit string is rehashed and the process is recommenced; otherwise the process is complete.

The above procedure assumes the length of the input bit-string is significantly greater than $\lceil L \log_2 C \rceil$. Since a likely value of L is 16, and C is typically at most 64, $\lceil L \log_2 C \rceil$ is likely to be less than 100, i.e. much smaller than the 256-bit output of a modern hash function such as SHA-256. It also assumes that a random password with characters from the specified password set has a reasonable chance of satisfying the PRML requirements. If not, then a more elaborate second stage algorithm could be devised.

4) Password Offsets: As noted in II-4, a major issue with existing password generators is that they do not allow a user to choose a password (e.g. to allow continuing use of a password established prior to use of the system), or to change a password without changing the set of inputs. We propose the use of *password offsets* to support these requirements. A password offset works in the following way. A password $d_{L-1} d_{L-2} \ldots d_0$ is first generated in the normal two-stage way (as described in III-3), and suppose D is the positive integer which has $d_{L-1} d_{L-2} \ldots d_0$ as its C-ary representation. Let the user-chosen password (of length M, say) be encoded as an M-digit sequence $e_{M-1} e_{M-2} \ldots e_0$, where $0 \leq e_i \leq C-1$ for every i $(0 \leq i \leq M-1)$, and suppose E is the positive integer having $e_{M-1} e_{M-2} \ldots e_0$ as its C-ary representation. The password offset is simply $E - D$.

Password generation is unchanged, except that the offset is added; the result will be the C-ary encoding of the desired password. A similar approach can be used for password changes, where a new password can be generated at random (in accordance with the PRML specification) and the password offset is set to the difference between the new password and the value generated using the standard procedure.

Note that the use of a password offset is to some extent like a cloud-based password manager, in that password-related information is stored by a server. Indeed, it can be regarded as a way of combining the best features of both approaches. What distinguishes an offset-based password generator from a password manager is that use of an offset is optional, and users are only expected to employ it if they wish to either continue to use an existing or externally chosen password or make frequent password changes.

IV. Details of Operation

To simplify the description, we assume AutoPass is implemented as a PC browser add-on. Alternative implementation scenarios, e.g. as a stand-alone application on a phone or tablet, will be very similar.

1) First Installation and Account Creation: After client software installation, set-up involves the following steps.

1) When activated, AutoPass asks whether the user has an existing account. If the user indicates a new account is needed, the client collects the following registration details from the user. The *login password* is chosen and entered by the user, who must memorise it. After entry of the login password, the client computes a salted hash of the value, which is sent to the AutoPass server (with the salt) as a means of authenticating the user. The user must select a *user name*. The AutoPass server checks the name is not already in use, and if necessary requests a new value; the name serves as an identifier for the user. The 128-bit *master password* can be generated by the user or the AutoPass client (e.g. as a user choice). If the client software generates it, it is displayed to the user, e.g. as 32 hex characters, and the user is advised to securely retain a copy so that system recovery is possible (see IV-3 below). The login password is used to generate a cryptographic key, e.g. by hashing a concatenation of it and a fixed value; this key is used to encrypt the master password, e.g. using AES [19] in an authenticated encryption mode, prior to server upload. The server retains this encrypted master password. The client also generates a hash of the master password and sends it to the server; this hash value is used for recovery purposes (see IV-3). To allow recovery if a user forgets the user name or login password, an email address should also be collected and submitted to the server.
2) After successful user account creation, the user is requested to log in using his or her newly established user name and login password (see IV-4).

2) Installing AutoPass on a Newly Acquired PC: Once an AutoPass account has been established (as described immediately above), the following step is followed to set up AutoPass on a new machine. We suppose that the client software has been installed on the platform.

As previously, when the AutoPass add-on is activated for the first time, it first asks the user whether he/she has an existing account. In this case the user indicates that he/she already has an account. The AutoPass client then asks the user for his or her user name and login password, and the process continues exactly as in a normal operational session (see IV-4).

3) Recovery: The system needs to provide a recovery mechanism for the case where a user forgets their user name and/or password. The AutoPass client's recovery function can be used in the event of a forgotten user name or password.

If a user forgets their user name, he/she can request a copy from the server by entering their registered email address. The server checks the email address is registered, and emails the user name to the user.

If a user forgets their login password, then it cannot be recovered since neither server nor client retain a copy. However, if the user has a copy of the master password, then recovery is possible. The user is prompted for his/her user name, the master password and a new login password. The new login password is used to generate a key which is used to encrypt the master password, exactly as before. A hash of the new login password, the user name, the encrypted master password, and a hash of the master password are all sent to the AutoPass server. The server authenticates the user by comparing the master password hash with its stored value, and, if successful, replaces the current encrypted master password and login password hash with the new values. Finally, the server communicates success of recovery to the client, which informs the user.

4) Operational Sessions: We next consider what occurs when AutoPass is activated after initial set-up.

1) The user is prompted for user name and login password.
2) The user name is sent to the AutoPass server, which responds with the salt value for its stored copy of the hashed login password for the identified user.
3) The AutoPass client software hashes the combination of salt and entered login password, and the resulting value is sent to the server.
4) The AutoPass server checks that the received hash equals its stored value, and so authenticates the user.
5) The AutoPass server returns the encrypted master password to the client, along with the following information for each site for which the user has created an AutoPass password: the first part of the URL (used as the site identifier); the password policy (in PRML); the set of input types used to generate the password (e.g. whether or not a digital object is used); the password offset, if it exists; any other parameters used to control password generation. This site-specific data is unlikely to change rapidly, so the client can cache the most recently downloaded copy, improving system availability even if the AutoPass server is unavailable.
6) The AutoPass client decrypts the master password using a key derived from the login password, and multiply hashes the master password; the result is cached and the master password can then be deleted.
7) Once activated, the AutoPass add-on will run continuously in the background, examining each web page to see if it is a login page. It does this by using various heuristics, including looking for the string *input type="password"*. The add-on will then work as required, generating passwords automatically, until the session ends, e.g. when the browser is terminated.

When AutoPass is used with a new website, the following procedure is executed.

1) If AutoPass detects a login page, it cross-checks the first part of the site URL with the data from the AutoPass server to determine whether it is a known site. In this case, we suppose that it is a new (unknown) site.
2) The AutoPass add-on communicates with the user (e.g. via a pop-up) to indicate it has detected a login page for a website for which a password has not previously been generated, and asks the user whether it would like AutoPass to manage password generation for this site.
3) If the user declines, then AutoPass goes back to looking for login pages. If the user accepts, AutoPass next asks what types of input the user would like to use to generate the password from amongst those listed in III-1.
4) The user selects the input types; if use of digital objects is selected, the user is also asked to select an object. The AutoPass client assembles the inputs, including the first part of the website URL and the multiply-hashed master password, to be used to generate the site password. The client also offers the user the option to select the password — if the user requests this option then the user is prompted for the pre-chosen value.
5) The password is generated using the procedure specified in III-1 and III-4 and automatically copied to the password field. If the user chose to select the password value, then the appropriate password offset is computed during password generation.
6) The user preferences and the password offset (if appropriate) are sent to the AutoPass server for storage.

When AutoPass is in everyday use, i.e. after a website has already been set-up, the following process occurs (we suppose that the client AutoPass software is already active). If the AutoPass add-on detects a login page, it uses the first part of the site URL to check whether a password has previously been generated for this site — in this case we suppose it has. The AutoPass add-on then assembles the set of inputs to be used to generate the password; if the user preferences for this site indicate that a digital object is to be used, the add-on prompts the user for the object. The AutoPass add-on then generates the password, using the password offset if available, and automatically copies the value to the password field.

V. Evaluation

The AutoPass server must be trusted to some extent by the user, since if it sends incorrect data then correct passwords cannot be generated. It also learns which websites the user interacts with, and hence must be trusted to respect user privacy. However, it cannot learn user passwords, since it only has access to an encrypted copy of the master password, and thus it can be regarded as being 'partially trusted'.

The security and correct operation of AutoPass depends on three key assumptions: (a) the client device is assumed to be uncompromised, since passwords are generated in and used by this device; if, for example, the browser is compromised then clearly generated passwords may be compromised; (b) the AutoPass client is assumed to be without exploitable vulnerabilities; if a corrupted version of the client software is present on the client device, then user passwords may be compromised; (c) the AutoPass server provides correct information (see also V above). Given these assumptions, AutoPass resists the following types of attack.

Active attacks on communications between the AutoPass client and server, including masquerading as the server to the client or vice versa, e.g. as made possible by an untrustworthy wireless access point or DNS poisoning. Such attacks are prevented by use of TLS. The server will be authenticated by a pinned certificate. The client is not explicitly authenticated to the server, but the user is authenticated by checking the login password hash sent over the link.

Attacks on password secrecy by the AutoPass server. The server only has password metadata, a hash of the master password, and an encrypted copy of the master password. If the master password is randomly generated, use of a 128-bit value will prevent direct brute forcing guessing attacks. However, encryption of the master password is based on a key derived from the user-selected login password. If the login password is poorly chosen then it can be brute-forced, meaning that the server could gain access to the master password. Thus it is vital for the user to choose a login password with high entropy. This is a reasonable assumption since it is the only secret the user is required to memorise. Also, the use of password offsets means that if a password (and its offset) are compromised then all future passwords for that particular site can be determined if the offset is known. That is, whilst offsets will not be divulged to any party, a dishonest AutoPass server that (by some means) learns a user's password for a website will be able to determine all future passwords for that site.

Attacks on password secrecy conducted by a valid site against user passwords for other sites. The AutoPass system is completely transparent to authenticating websites. If a website guesses that AutoPass is in use, it could use the password to try to perform a brute force search for the master password. However, if the master password is chosen at random then such a search is infeasible.

Attacks on password secrecy conducted by anyone with access to the AutoPass server database. A party with access to the AutoPass server database will not have access to any user passwords. However, as discussed above, if the unauthorised party obtains the encrypted master password and the login password is poorly chosen, then the attacker might be able to brute-force the login password and learn the master password. This argues in favour of the AutoPass server providing additional encryption of the database, giving protection against compromise of stored user data.

VI. Concluding Remarks

In II-4 we identified three major problems with existing password generators. These issues are all addressed by the AutoPass scheme, as specified here, with the aid of a partially trusted server that does not have the means to recover individual user passwords. Firstly, existing schemes cause major difficulties for users with a large body of existing passwords, since they are obliged to change them all; AutoPass avoids this through the use of password offsets, allowing continued use of existing passwords. Secondly, the use of the AutoPass server allows seamless cross-platform working. Thirdly, the use of the server-provided PRML statements allows passwords to be automatically generated to meet website-specific requirements. Of course, whilst AutoPass works in theory, it remains to verify that the system will work in practice. A prototype implementation is being developed, and will be used to conduct user trials. We plan to report on these trials in a future paper.

References

[1] C. Herley and P. C. van Oorschot, "A research agenda acknowledging the persistence of passwords," *IEEE Security & Privacy*, vol. 10, no. 1, pp. 28–36, 2012.

[2] D. Florêncio, C. Herley, and P. C. van Oorschot, "Password portfolios and the finite-effort user: Sustainably managing large numbers of accounts," in *Proc. 23rd USENIX Security Symposium*. USENIX Association, 2014, pp. 575–590.

[3] J. A. Halderman, B. Waters, and E. W. Felten, "A convenient method for securely managing passwords," in *Proc. WWW 2005*, A. Ellis and T. Hagino, Eds. ACM, 2005, pp. 471–479.

[4] A. H. Karp, "Site-specific passwords," HP Laboratories, Palo Alto, Tech. Rep. HPL-2002-39 (R.1), May 2003.

[5] M. Mannan and P. C. van Oorschot, "Passwords for both mobile and desktop computers: ObPwd for Firefox and Android," *USENIX ;login*, vol. 37, no. 4, pp. 28–37, August 2012.

[6] B. Ross, C. Jackson, N. Miyake, D. Boneh, and J. C. Mitchell, "Stronger password authentication using browser extensions," in *Proc. 14th USENIX Security Symposium*, P. McDaniel, Ed. USENIX Association, 2005, pp. 17–32.

[7] R. Wolf and M. Schneider, "The passwordsitter," Fraunhofer Institute for Secure Information Technology (SIT), Tech. Rep., May 2006.

[8] K.-P. Yee and K. Sitaker, "Passpet: Convenient password management and phishing protection," in *Proc. SOUPS 2006*, L. F. Cranor, Ed. ACM, 2006, pp. 32–43.

[9] F. A. Maqbali and C. J. Mitchell, "Password generators: Old ideas and new," in *Proc. WISTP 2016*, ser. LNCS, S. Foresti and J. Lopez, Eds., vol. 9895. Springer, 2016, pp. 245–253.

[10] D. McCarney, "Password managers: Comparative evaluation, design, implementation and empirical analysis," Master's thesis, Carleton University, August 2013, available at https://danielmccarney.ca/assets/pubs/McCarney.MCS.Archive.pdf.

[11] R. Biddle, M. Mannan, P. C. van Oorschot, and T. Whalen, "User study, analysis, and usable security of passwords based on digital objects," *IEEE Trans. Inf. Forensics & Security*, vol. 6, no. 3, pp. 970–979, 2011.

[12] M. Mannan and P. C. van Oorschot, "Digital objects as passwords," in *Proc. HotSec'08*, N. Provos, Ed. USENIX Association, 2008.

[13] M. Mannan, T. Whalen, R. Biddle, and P. C. van Oorschot, "The usable security of passwords based on digital objects: From design and analysis to user study," School of Computer Science, Carleton University, Tech. Rep. TR-10-02, February 2010, https://www.scs.carleton.ca/sites/default/files/tr/TR-10-02.pdf.

[14] M. Horsch, A. Hülsing, and J. A. Buchmann, "PALPAS — passwordless password synchronization," in *Proc. ARES 2015*. IEEE Computer Society, 2015, pp. 30–39.

[15] M. Horsch, M. Schlipf, J. Braun, and J. A. Buchmann, "Password requirements markup language," in *Proc. ACISP 2016*, ser. LNCS, J. K. Liu and R. Steinfeld, Eds., vol. 9722. Springer-Verlag, 2016, pp. 426–439.

[16] J. Kelsey, B. Schneier, C. Hall, and D. Wagner, "Secure applications of low-entropy keys," in *Proc. ISW '97*, ser. LNCS, E. Okamoto, G. I. Davida, and M. Mambo, Eds., vol. 1396. Springer, 1997, pp. 121–134.

[17] *ISO/IEC 10118–3, Information technology — Security techniques — Hash-functions — Part 3: Dedicated hash-functions*, 3rd ed., International Organization for Standardization, Genève, Switzerland, 2004.

[18] D. Silver, S. Jana, D. Boneh, E. Y. Chen, and C. Jackson, "Password managers: Attacks and defenses," in *Proc. 23rd USENIX Security Symposium*, 2014, pp. 449–464.

[19] *ISO/IEC 18033–3:2010, Information technology — Security techniques — Encryption algorithms — Part 3: Block ciphers*, 2nd ed., International Organization for Standardization, Genève, Switzerland, 2010.