Passwords for both Mobile and Desktop Computers: ObPwd for Firefox and Android[*]

Mohammad Mannan
Concordia Institute for Information Systems Engineering
Concordia University
Montreal, Canada

P.C. van Oorschot
School of Computer Science
Carleton University
Ottawa, Canada

Abstract

Many users now access password-protected accounts and websites alternately from desktop machines, and mobile devices (e.g., smartphones, tablets). The input mechanisms of the mobile devices are often miniature physical or virtual on-screen keyboards, posing challenges for users trying to type passwords with mixed-case and special-characters expected by websites and more easily entered on desktop keyboards. We begin with a review of these challenges and existing proposals addressing cross-device password entry, including some password managers. We then bring the issues into focus with detailed discussion of the interoperation challenges, and implementation details, and interface details of the object-based password "ObPwd" mechanism, as implemented for the Android platform, plus compatible browser-based and stand-alone implementations for desktop environments. ObPwd generates a password from a user-selected digital object (e.g., image), does not require changes to server-side software, and avoids the text-input challenges of mobile devices. We also briefly evaluate ObPwd using a recently proposed evaluation framework for password authentication schemes. A major goal is to increase attention to the cross-device password authentication problem.

1 Introduction and Motivation

We all wish passwords would go away. Their faults are many, and well-documented elsewhere. However, for the present time, the Internet world continues to have a deep investment in them. Millions of websites continue to demand passwords.

Almost all users with a few years of experience are familiar with the task of typing passwords on full-size Qwerty keyboards. For many accounts and websites, users are encouraged or required to choose passwords with mixed-case letters, digits, and special characters. (How such policies affect the overall security and usability requires an entirely separate discussion.) In recent years, the flood of new devices in the marketplace has included smartphones, tablet PCs, Internet-enabled TVs, and game consoles. Many of these offer only a limited on-screen keyboard, e.g., touch/stylus-based, remote control, or miniature physical keyboards. When these devices are used for web browsing, passwords must be input to access password-protected sites. Many applications ("mobile apps")

[*]July 12, 2012. Author's copy. A version of this paper appears in USENIX ;login: 37(4):28-37, August 2012. Contact author: `mmannan@ciise.concordia.ca`.

also require password input, some of which are also accessible on desktops as independent or web applications.

The authentication task is now more complicated than when using a full-size keyboard: text password input is more error-prone and frustrating in terms of locating the keys and entering the characters—especially if entering special characters requires multiple keys to reach alternate (shifted) keyboards. Such text-unfriendly input interfaces may also influence user-chosen passwords to be even weaker than otherwise. Indeed, Jakobsson et al. [9] reported from a survey of 50 smartphone users that 88% of device passwords are digit-only. (Arguably, user-choice is influenced by available screen-lock options and the default option presented to users.) They also reported that 46% of users enter a password once or more per day on mobile devices, and that 56% mistype a password at least one in ten times. Users even stated that mobile-device password entry is more annoying than lack of coverage and poor voice quality. On the other hand, a new smartphone dynamic is underway largely due to the input problem: mobile apps often require users to enter an app-password only once (e.g., upon installation or first run), which is then saved by the app, and thereafter the stored password is used to authenticate the user. This is analogous to a desktop browser permanently remembering passwords for websites. However, the general password input problem remains, especially for browser-accessed services in mobile devices.

Others also have recognized the problem of text-password input in mobile devices. Several graphical and image-based schemes have been proposed; e.g., Windows 8 picture password, mobile Blue Moon authentication (http://mobile-blue-moon-authentication.com). Another idea as discussed later, is a password scheme involving multiple real words [7, 8] and leveraging widely available auto-correct and auto-suggest features on mobile devices. However, using these same passwords becomes challenging in the desktop world if similar auto-correction functionality is unavailable. For further reading on a similar topic, i.e., discussion of need for scheme that allows users to alternate between mobile phone and desktop keyboard see Bicakci and van Oorschot [2].

More generally, user-friendly remote authentication mechanisms custom-designed for mobile devices may not find wide acceptance, as the same online services will often be accessed from multiple devices with different input mechanisms. A major design criteria that must not be overlooked, and which is more challenging than initially apparent, is that such authentication mechanisms must also be suitable back on a desktop computer with a standard keyboard since users alternate access devices. Consequently, the design of a user-friendly authentication systems must suit a wide variety of devices including those with input-constrained and conventional keyboards. We consider this challenge herein. As a specific example, we revisit a password mechanism called object-based password (*ObPwd*) originally designed in the context of the desktop world, and its implementation adopted to the mobile world.

The ObPwd mechanism constructs text-compatible passwords from digital objects such as pictures, generating strong passwords without requiring that users type mixed-case or special characters (see Fig. 1 in Section 4). A previous publication [3] outlines the basic idea and detailed results of a user study and security analysis. Our main focus here is an illustrative example, with concrete implementation details and design choices, of one solution to this challenge of designing a password scheme supporting *password entry modes across devices with a variety of input mechanisms*—to stimulate further innovation and better solutions. In what follows, we describe the implementation of ObPwd adapted to the mobile space (on the Android platform), as well as in the desktop PC environment. Beside the basic design, we also emphasize the *domain-salted* variant of ObPwd that generates unique passwords from the same object (e.g., a picture) for different websites; see

Section 4. This variant is particularly interesting in terms of addressing the wide-spread password reuse behavior among users that enables passwords leaked from low-security websites to be used in more sensitive sites.

2 Password Entry Challenges from non-standard Keyboards

Numerous usability issues arise when conventional text passwords must be entered on mobile devices which do not have standard physical keyboards. We review a few of these here.

Multitude of devices with different keypad layouts. No standard keyboard layout is followed across mobile devices. Common text entry methods include (a) multi-tap: multiple letters are mapped to the same physical key; (b) T9: text on 9 keys, a predictive text entry method allowing words to be entered by a single keypress; (c) full Qwerty keyboard; and (d) on-screen alternate keypads. Even different versions of devices from the same manufacturer may vary significantly in keypad layout (e.g., Nokia E7 vs. Nokia E63). Layout across devices from different classes of the same vendor (e.g., Nokia E vs. N series), and across devices from distinct vendors, differ sufficiently to cause vendor lock-in for some users.

Keyboard forms. Various form factors are available, with no clear winner. Examples include: physical keypad, touch-based, stylus, or pointer-based on-screen keypad (e.g., on the Wii game console). Forms other than physical keypads typically lack tactile feedback during input. Some on-screen keypads offer feedback via audio and visual channels, and vibrating the device. Interestingly, such user-friendly feedback may also leak information to nearby attackers when these devices are used in public places (e.g., through shoulder-surfing, or video recording [13]).

Multiple steps for keyboard shifting. In the default layout, small keypads (whether physical or touchscreen) offer a limited number of characters. Accessing additional characters requires tapping or pressing special keys (e.g., using shift to switch between keypad modes). Thus inputting a strong password with mixed case letters and digits requires more keystrokes than characters in a password.

Cold weather issues and fat-fingers. Most modern touchscreens use a capacitive sensing panel which operates by completing an electric circuit with the human body through the fingers. This hinders providing input to these devices with gloved fingers, as common gloves are made of electrically insulating materials. *Fat fingers* ("working men's fingers") may also hinder input precision in mobile keypads due to small keypad size and visual interference.

3 Existing Proposals Supporting Cross-Device Password Entry

Here we mention a few of the existing proposals providing password authentication which is compatible across devices with varying text-input mechanisms. Security and usability problems with passwords are well-known, and many techniques have been proposed to replace passwords altogether, e.g., biometrics, graphical passwords, and security tokens; detailed surveys for these are beyond the scope of this article. Additional discussion on proposals supporting mobile device password entry is provided in Appendix A.

Stand-alone password managers. Password managers are becoming more popular, for reasons including: convenient access to passwords, need to maintain numerous accounts, browser integration, and online access. Several of the existing password managers (both stand-alone applications and web services) which support multiple devices may alleviate the password input problem to some extent. Below we discuss two example password managers.

KeePass. KeePass (http://keepass.info) is an open source password manager for multiple desktop platforms. It saves several website/application specific items such as the site URL, userid and password in an encrypted database file on a user's local PC; the encryption key is derived from a user-chosen master password, and optionally a user-selected or randomly generated file. The database files can additionally be locked with the user's Windows user account. The saved passwords can be copied into the clipboard, and then pasted into the intended application or website. A saved URL can be launched through the default system browser directly from the KeePass application; the KeeFox extension for Firefox can automate password entry to the website. KeePassSync is a KeePass plugin that offers synchronization of password databases between devices using online storage providers (e.g., Amazon S3). The database files can also be manually transferred. Third-party developed apps (e.g., iKeePass, KeePassDroid, and KeePassMobile for iOS, Android, and J2ME devices respectively) enable users to access KeePass databases from mobile devices.

LastPass. LastPass (http://lastpass.com) is a free online password manager available in most major desktop and mobile platforms. Passwords and other user data (e.g., notes, form data) are encrypted locally using a user-chosen master password; the encrypted result is saved on the LastPass server. LastPass and similar online password managers offer two distinct features to alleviate password problems in general: (a) portability – all user passwords are accessible from anywhere with a browser and Internet connection; and (b) backups – passwords are backed up without involving the user. However, these highly appreciated features come with side-effects. For example, the encrypted password list may be vulnerable to dictionary attacks. This vulnerability depends on the strength of the user-chosen master password; historical experience of user-choice issues makes this worrisome. (Some password managers facilitate generating random passwords as site passwords, but the master password itself remains user-chosen. Solutions such as Kamouflage [4] which store decoy passwords along with user passwords can also be adopted to frustrate dictionary attacks on stolen encrypted password storage.) These online password managers offer an attractive target to attackers: compromising such a server allows access to a large number of user accounts. Indeed in May 2011, LastPass was possibly compromised in such an attack [16]. When users were forced to reset their master password after the attack, some users were stuck as the reset process required logging into their pre-registered email address—the password of which was also saved with LastPass. Such account lock-out appears to be an intrinsic problem with online password managers that use email for account recovery.

A general drawback of password manager services is the tangible privacy concern: despite assurances that user data is stored in encrypted form, the service providers may be compelled to make user data 'available' to government agencies; e.g., see the recent changes in DropBox privacy policy [6].

Karole et al. [11] conducted a usability study comparing three widely used password managers: LastPass (online), KeePassMobile (phone-based), and Roboform2Go (USB-based). Overall, LastPass was least preferred, and specifically the non-technical users in the study favored the phone-based manager. The authors attributed this finding to the fact that users prefer control over their passwords, rather than trusting a third party. However, they also reported that the usability of phone-based managers is not at par with user expectations. Another study [1] on the security of smartphone-based password manager apps reported several implementation weaknesses including easy verification of master password and hard-coded encryption keys.

Browser-based password synchronization. Synchronization functionality is built into Firefox 4 (older versions can use the Sync addon), and Firefox Mobile (available on Android OS and Maemo/Nokia N900). For example, Firefox Sync saves user bookmarks, passwords, open tabs, form data and browsing history—for access from PCs and mobile devices. Saved content is also accessible from iOS (iPhone, iPad, iPod) via the Firefox Home application. User data is encrypted locally, and then stored and shared via a Mozilla hosted server; users can set up their own server to be used with Sync (e.g., if they do not trust Mozilla with their data).

Tapsure is a Firefox Mobile addon (https://addons.mozilla.org/en-us/mobile/addon/tapsure/) that enables users to input saved text passwords by tapping a rhythm/pattern on the phone's touchscreen. Users can save a password entered on a website (through usual input methods), by tapping a personal pattern on the screen; the same password can be accessed from any sites that use it via Tapsure. Tapsure uses Firefox's built-in password manager for storing passwords. The tapping pattern serves as an easy-to-use unlock password that enables access to the saved text password. Tapsure differs from browser password managers in a subtle but important way. A Tapsure-saved password can be accessed from any site when the specific tapping pattern is entered, thus facilitating easier input of reused passwords. In contrast, browser password managers save pairs of userid-password for individual sites which are made available only at the specific sites (from the saved password list, which is optionally encrypted under a user-chosen master password).

4 Cross-platform/Cross-device ObPwd Implementations

We outline here the basic ObPwd scheme and several implementations in different platforms. The implementation details may help understand challenges such as design and user interface issues in cross-platform/cross-device implementations. Implementations are publicly available as an OS-agnostic Firefox browser extension, and as stand-alone applications in Android, Microsoft Windows, Mac OS X, and Linux. An initial Firefox extension and prototypes on other major platforms have been well-received, and public feedback has resulted in modifications and upgrades. To our knowledge, the technology is free of patents. ObPwd FAQ and download page is available at: http://www.ccsl.carleton.ca/~mmannan/obpwd/.

While we explain specific implementation choices made in order to convey a concrete sense of the functionality and interfaces available, different design choices could be made for reasons of preference, usability and security, matching different intended contexts and use environments.

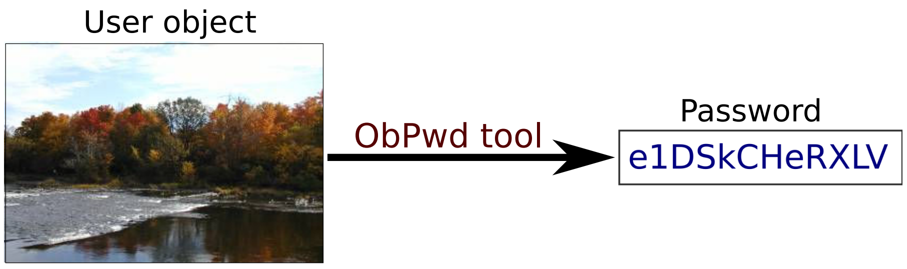

Figure 1: ObPwd basic mechanism

The basic ObPwd mechanism. The core functionality in all ObPwd tools is to output a text password from user-selected content; see Figure 1. The current implementations use SHA-1 to hash password objects. The hash output is mapped to a base-64 character set, then converted to an alphanumeric password (default 12 characters) by known techniques [14]; for example, assuming a local file is used as the password object, $pwd = Hash2Text(Hash(fileContent))$. Up to the first $n = 100,000$ bytes are used from an object; 160 bytes are required. Variants of the basic mechanism are discussed elsewhere [3]. One of these variants, discussed later in Section 5, is the *domain-salted variant*, which involves using a local file and the website domain as a salt (i.e., $pwd = Hash2Text(Hash(URLdomain||fileContent)))$. This domain-salted variant is provided by both the Firefox addon and the Android app implementations mentioned above, starting with versions 1.0.1 and 1.0.

ObPwd Firefox extension (desktop). This extension can be activated from the browser context menu (i.e., right-/secondary-click menu). Under the "Object-based Password (ObPwd)" menu, several sub-menus appear (depending on the right-click context): (i) "Get ObPwd from Local File" brings up a file dialog box for selecting a local file as a password object; (ii) "Get Unique ObPwd: Local file + Domain" offers choosing a local file, and then the domain name of the current page is used with the file content to generate a site-specific password; (iii) "Get ObPwd from Selected Text" generates a password using the selected text block on the web page (if there is any selected text string); (iv) "Get ObPwd from Image" generates a password from the selected image (i.e., the one right-clicked on, if any); (v) "Get ObPwd from Link" generates a password from the content as pointed by the URL right-clicked on, if any. Certain types of relatively stable HTTP and HTTPS links are supported by default (e.g., pdf, mp3, avi, txt, jpg, zip, wav), but not several common URL extensions (e.g., html, php, asp) which commonly host dynamic content—e.g., news page content may change as user comments are added, precluding regeneration of the original password.

Configuration preferences support changing the default password output length (6 to 20 characters, default 12) and including special characters. If a password is generated with certain preferences, the same preferences must be selected to re-create that password (irrespective of where the password is used).

When a password object (an image, highlighted text, URL, or a local file) is selected, the extension generates a password from the underlying content and displays the password in a dialog box in plaintext to enable users to record it for backup in a secure way. If the OK button is clicked, the password is copied to the system clipboard allowing pasting anywhere by the user. Note that by default, for security and privacy reasons, Firefox clipboard data is not accessible to JavaScript embedded on a site. The password is inserted directly into a password input box on a login page, if the extension is activated from such a box (i.e., the context menu is brought up by right-clicking on the password box). This auto-filling both automates the password copy-paste step, and protects the password from shoulder-surfing.

ObPwd Android app (mobile devices). Installing the ObPwd app adds a menu item (labelled "ObPwd") to the "Share" menu of Gallery, the default media app for Android devices. Users can browse their media files stored on the device and from Picasa web albums (if the user's Google account is linked to Picasa). When the user selects an image or video, and chooses the ObPwd app from Gallery's Share menu, then the user is asked if the domain of the last visited website should be used in the password generation (i.e., whether to use the *domain-salted variant*). Then the corresponding text password is displayed. The password display dialog offers two choices: "Copy to clipboard" (which copies the password to the clipboard), and "Quit" (which quits the ObPwd app). If copied, the password can be pasted to any password field (e.g., in websites and other apps), without requiring typing the password.

5 Usability, Limitations and Evaluation

Discussion on usability and features of the basic ObPwd. A hybrid in-lab/at-home user study using the ObPwd desktop extension was conducted involving 32 participants (see [3] for full details). Participants were asked to use 11 new passwords (8 test websites and 3 real-world sites) in a span of 7–10 days. The study reported encouraging results in terms of several usability factors. The login success rate was more than 90% (on the first attempt) in a return-to-lab session. The average login time was about 20 seconds—which despite being longer than text password logins in the desktop environment, was reflected in a positive affect by participants: they reported that they enjoyed browsing their password objects both when creating passwords and logging in. This result is atypically positive compared to other new password proposals. Whether similar positive results occur for other ObPwd platforms and/or implementations requires further user studies; we presently have only anecdotal praise from users of the Android app, but these are self-selected, technically-savvy users not representative of the general population.

The user interfaces of ObPwd tools described above differ depending on the device/environment, reflecting hardware and software interfaces which vary significantly across these devices. Instead of implementing a separate image/video browsing interface on Android, the implementations described rely on the default Gallery app for object selection. This provides a familiar interface to users, but on the down side, the implementation shortcut overloaded the "Share" menu to initiate the ObPwd app. "Share" is an unfortunate name for this function menu, since security is defeated if users openly share their password objects (as facilitated by several other applications in the Share menu). Indeed, ObPwd security relies heavily on the assumption that users' password objects remain private, as opposed to, for example, publicly posted photos.

Some features of the ObPwd scheme may favor its adoption. Personal digital content is now easily available on both desktop and mobile platforms. As ObPwd requires no server-side changes, users can immediately benefit upon installing freely available implementations. ObPwd passwords are typically as strong as system-generated passwords, with respect to guessing attacks (see [3] for further discussion), yet the password objects are user-chosen. On the other hand, many visible and invisible barriers exist to installing any new authentication mechanism intended to replace passwords; widespread adoption of ObPwd, or any other alternative, is likely to occur only if adopted by a major platform vendor or browser provider.

ObPwd is a hybrid mechanism both in terms of authentication method (part what-you-know, part what-you-have-access-to), and input type (involving media such as image/video/music-based). It is not a password manager in a traditional sense (i.e., does not store passwords), but empowers users to better manage several strong passwords (as apparent from our user testing) by taking advantage of the positive attachment users already have with their personal content.

Discussion on the ObPwd domain-salted variant. For generating ObPwd passwords, we assume the *domain-salted variant* in the evaluation below (see also Section 4). This variant is arguably the safest, and, from a user interface viewpoint, indistinguishable from that used in the user study [3]; we note however that this variant itself and and the ObPwd Android app have not been formally user-tested. We also assume that a single local file is used as the password object for all websites; i.e., the password object is used as a master password from which unique,

	Usability								Deployability						Security										
	U1: Memorywise-Effortless	U2: Scalable-for-Users	U3: Nothing-to-Carry	U4: Physically-Effortless	U5: Easy-to-Learn	U6: Efficient-to-Use	U7: Infrequent-Errors	U8: Easy-Recovery-from-Loss	D1: Accessible	D2: Negligible-Cost-per-User	D3: Server-Compatible	D4: Browser-Compatible	D5: Mature	D6: Non-Proprietary	S1: Resilient-to-Physical-Observation	S2: Resilient-to-Targeted-Impersonation	S3: Resilient-to-Throttled-Guessing	S4: Resilient-to-Unthrottled-Guessing	S5: Resilient-to-Internal-Observation	S6: Resilient-to-Leaks-from-Other-Verifiers	S7: Resilient-to-Phishing	S8: Resilient-to-Theft	S9: No-Trusted-Third-Party	S10: Requiring-Explicit-Consent	S11: Unlinkable
Web passwords (desktop)			●		●	●	○	●	●	●	●	●	●	●		○						●	●	●	●
Web passwords (mobile)			●		●			●	○	●	●	●	●	●		○						●	●	●	●
ObPwd (desktop)	○	●			●	○	●	●		●	●		○	●	●	●	●	●		●	●	●	●	●	●
ObPwd (mobile)	○	●			●	○	○	●		●	●			●	●	●	●	●		●	●	●	●	●	●

Table 1: Comparing conventional text passwords (web passwords) to ObPwd passwords.
Key: ● (offers the benefit); ○ (almost offers the benefit); blank (benefit not offered).

site-specific passwords are generated (cf. PwdHash [14]). This assumption is rooted in the current practice of reusing the same or few text passwords across many accounts. We argue that access to site-specific passwords does not allow an attacker or a malicious site operator to (easily) create passwords for other sites since this will require guessing the file-content of the password object (recall that $pwd = Hash2Text(Hash(URLdomain||fileContent))$). In effect, the best attack remains to be the exhaustive search; note that, for an attacker not in possession of the password object, exhaustive search requires on the order of 2^{70} guesses in default settings (see [3] for details). In contrast, a compromised site-specific password generated from a user-chosen master password (e.g., $pwd = Hash2Text(Hash(URLdomain||masterPassword))$) may reveal the master password under offline dictionary attacks. Thus, from a security viewpoint, ObPwd is analogous to using a random string stored on the user's machine. However, from a usability viewpoint, in contrast to ObPwd's use of a user-chosen object, a random string is neither user-friendly nor recognizable to users, and provides no positive feedback.

Comparison and usability-deployability-security evaluation. We compare and evaluate ObPwd against basic text passwords, using the UDS (usability, deployability, security) framework of 25 baseline properties for user authentication schemes [5]. For context, we show the UDS rating for (web passwords, desktop) as the first row of Table 1 herein. Appendix C explains how we came up with the ratings that appear in Table 1. We use separate table rows for implementations of each mechanism for desktop (assuming a full-size keyboard) and mobile platforms (e.g., mobile phones with small hardware or touch-based keypads, and tablets), as the usability ratings in particular differ.

6 Concluding Remarks

The ubiquity of traditional keyboards in desktop systems has played an important role in the proliferation of text passwords as the primary mode of user authentication on the Internet. For user authentication from mobile devices, the opportunity exists to exploit device-specific features such as multi-touch, GPS, accelerometer, and camera to improve both security and usability (e.g., see [9]). However, a critical requirement is that the authentication of users who alternate between desktop and mobile systems must be accommodated. Greater customization of authentication schemes, such as allowing user-selection of per-login authentication modes, may be the path to better support the emerging multi-device/multi-platform usage scenarios. The system side could automatically detect the type of device the user is on, and offer a different login interface or variation based on that. The interaction between user authentication and evolving password managers (and their support across platforms and devices, including cross-device password synchronization) is likely to become an increasingly important part of the user authentication equation. Another important but unexplored aspect of cross-device authentication is: whether current user behavior is affected by input difficulties, and if so, to what extent; for example, do users significantly change which sites they visit depending on which access device they use specifically to avoid accessing sites that require password input?

The implementations discussed herein are an illustrative example intended to motivate further discussion and innovation addressing the problem of user-friendly password authentication from alternating computing devices supporting divergent user input capabilities. No doubt, better mechanisms will appear over time. Their chances of deployment success in practice will be much higher if designed from the start keeping in mind cross-device requirements as discussed herein. We hope this article motivates and expedites further progress.

References

[1] A. Belenko and D. Sklyarov. "Secure password managers" and "military-grade encryption" on smartphones: Oh, really? Blackhat Europe 2012. Online manuscript available at: http://www.elcomsoft.com/WP/BH-EU-2012-WP.pdf.

[2] K. Bicakci and P. van Oorschot. A multi-word password proposal (gridWord) and exploring questions about science in security research and usable security evaluation. In *New Security Paradigms Workshop (NSPW'11)*, Marin County, CA, USA, Sept. 2011.

[3] R. Biddle, M. Mannan, P. van Oorschot, and T. Whalen. User study, analysis, and usable security of passwords based on digital objects. *IEEE Transactions on Information Forensics and Security (TIFS)*, 6(3):970–979, Sept. 2011.

[4] H. Bojinov, E. Bursztein, X. Boyen, and D. Boneh. Kamouflage: Loss-resistant password management. In *European Symposium on Research in Computer Security (ESORICS'10)*, Athens, Greece, Sept. 2010.

[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*, Oakland, CA, USA, May 2012.

[6] BusinessInsider.com. DropBox: We'll turn your files over to the government if they ask us to. News article (Apr. 18, 2011). http://www.businessinsider.com/dropbox-updates-security-terms-of-service-to-say-it-can-decrpyt-files-if-the-government-asks-it-to-2011-4.

[7] W. Cheswick. Rethinking passwords. Invited talk at USENIX LISA 2010. http://www.usenix.org/event/lisa10/tech/slides/cheswick.pdf. See summary in ;login: The USENIX Magazine, 36(2):68-69, Apr. 2011.

[8] M. Jakobsson and R. Akavipat. Rethinking passwords to adapt to constrained keyboards. In *Mobile Security Technologies (MoST) Workshop*, San Francisco, CA, USA, May 2012.

[9] M. Jakobsson, E. Shi, P. Golle, and R. Chow. Implicit authentication for mobile devices. In *USENIX Workshop on Hot Topics in Security (HotSec'09)*, Montreal, Canada, Aug. 2009.

[10] M. Jakobsson, E. Stolterman, S. Wetzel, and L. Yang. Love and authentication. In *Conference on Human Factors in Computing Systems (CHI'08)*, Florence, Italy, Apr. 2008.

[11] A. Karole, N. Saxena, and N. Christin. A comparative usability evaluation of traditional password managers. In *International Conference on Information Security and Cryptology (ICISC'10)*, Seoul, Korea, Dec. 2010.

[12] K. Kostiainen, J.-E. Ekberg, N. Asokan, and A. Rantala. On-board credentials with open provisioning. In *ACM Symposium on Information, Computer and Communications Security (ASIACCS'09)*, Sydney, Australia, Mar. 2009.

[13] R. Raguram, A. White, D. Goswami, F. Monrose, and J.-M. Frahm. iSpy: Automatic reconstruction of typed input from compromising reflections. In *ACM Computer and Communications Security (CCS'11)*, Chicago, IL, USA, Oct. 2011.

[14] B. Ross, C. Jackson, N. Miyake, D. Boneh, and J. C. Mitchell. Stronger password authentication using browser extensions. In *USENIX Security Symposium*, Baltimore, MD, USA, 2005.

[15] J. Strauss, C. Lesniewski-Laas, J. M. Paluska, B. Ford, R. Morris, and F. Kaashoek. Device transparency: A new model for mobile storage. *ACM SIGOPS Operating Systems Review*, 44(1), Jan. 2010.

[16] TheNextWeb.com. LastPass potentially hacked, users urged to change master passwords. News article (May 5, 2011). http://thenextweb.com/apps/2011/05/05/lastpass-potentially-hacked-users-urged-to-change-master-passwords/.

Appendix

A Other Proposals Supporting Password Entry on Mobile Devices

The Blue Moon authentication scheme [10] is a preference-based secondary login system designed to be used when users forget account passwords. Its primary goal is to replace common password reset methods such as personal verification questions (PVQs) by using personal preference based questions; the underlying assumption is that preferences are more stable than long-term memory. During setup, users select items they like and dislike from several categories (e.g., sports, music, food). For authentication, users must correctly categorize previously selected items as ‘liked’ or ‘disliked’. Items can be presented as text or image; an image-based implementation is available for mobile devices (http://mobile-blue-moon-authentication.com/). For both the text-based and image-based schemes, users need only select displayed items, e.g., via mouse pointer or touch, rather than by typing. Thus Blue Moon seems appropriate for primary web logins in both desktop and mobile platforms, if adopted by site maintainers.

To ease text entry, many smartphone platforms offer a predictive text entry feature where the system auto-fills or suggests a list of commonly used words once a user has typed the first few characters. At times, auto-correction may produce amusing results; see e.g., http://damnyouautocorrect.com/. Cheswick [7] has recently revived the circa 1980’s or earlier idea of *multi-word passwords* to combine this and users’ existing preference to choose dictionary words as passwords; see Bicakci and van Oorschot [2] for a summary of old and new variations of multi-word password proposals. The basic idea of multi-word passwords [7] is as follows: instead of a non-dictionary password with special characters, users (must) choose multiple common words as their password. Users need type only a few characters per word of their multi-word password, enabling easy-to-input, high-entropy passwords for mobile devices. For example, a system-assignment of 3 words from a fixed 1024-word list provides 30 bits of entropy; the password distribution, and thus entropy, should be expected to be skewed if selection from the list is user-chosen. Predictive text is also easily implemented on desktop platforms—desktop browsers now commonly integrate dictionaries to help users fill forms. Multi-word passwords thus appear to offer convenient password entry on both platforms—if adopted by websites—although we are aware of no user studies that explore their memorability, or usability in general.

Password patterns on a 9-dot grid used for screen-unlocking of Android phones are a simplified form of graphical passwords, used for local device authentication. This authentication mechanism is not presently compatible with desktop machines which typically (at present) lack touch-sensitive screens, though compatible mouse-driven interfaces could easily be implemented. The PIN-level security seems mainly of interest for casual security appropriate for screen-locking rather than remote authentication to websites.

Jakobsson et al. [9] developed a model for *implicit authentication* (IA), in which users are authenticated based on passively collected usage data from their mobile devices (e.g., phone calls, SMSs, GPS coordinates, emails, and calender events). During authentication, IA outputs a score comparing the user’s recent usage behavior with a pre-established *user model* (calculated from the user’s past usage data); this score is then used to make authentication decisions. Note that IA mechanisms are already being used in the desktop world by certain industry sectors; see e.g., RSA adaptive authentication (http://www.rsa.com/node.aspx?id=3018).

To secure the increasing amount of sensitive data on mobile devices against malicious apps, the on-board credential (ObC [12]) system has been developed by Nokia for devices running on Symbian and Maemo systems. ObC secures user credentials by relying on trusted hardware such as Trusted Platform Module (TPM), and M-Shield.

B Syncing Objects and the Case for Allowing Multiple Passwords

Password objects must be copied (or made available via other methods including portable memory cards) to all devices/platforms from which the user wishes to use ObPwd; and synced if passwords are updated. Existing sharing and sync mechanisms can facilitate the availability of the same password-generating object on these platforms. For example, Digital Living Network Alliance (DLNA) certified devices including TVs, DVD players, game consoles, computers, and mobile devices can easily share digital content such as photos, video and music files when connected in a home network. Millions of such devices are currently in use; see dlna.org. Advanced sync techniques for generic user content have also been proposed, e.g., *device transparency* [15]. Note that, to prevent exposure of password objects which is equivalent to leaking real passwords, the sync mechanisms must be over secure channels (e.g., physical USB connection, encrypted connection over Bluetooth or wifi). In the absence of such guarantees, this makes reliance on generic syncing tools dangerous if ObPwd is used.

Although sync mechanisms are readily available in modern devices, we expect syncing of password objects would remain a usability and/or deployability obstacle for many users. Here we briefly sketch an alternative that will require back-end support. Beyond the current practice of supporting one valid password per account, multiple passwords could be allowed for accessing each account. Services can make available new interfaces that allow entering alternate passwords (e.g., as part of a "change password" dialogue, a new option is "allow an alternate password, e.g., from a mobile phone"); by entering the original password (or any later registered ones), users can authorize the use of the alternate password (with a customary second-time password entry for confirmation). Then users can use the same or different password objects from their multiple devices without syncing the objects. To some extent, this is akin to services which allow users to register alternate email addresses in case the primary address becomes inaccessible. Allowing multiple passwords may also encourage the use of more device-specific password mechanisms. However, we re-iterate that this proposal breaks the "drop-in" feature of current ObPwd, which we believe is a huge enabling factor. The use of alternative passwords (instead of syncing) also increases the cognitive load for users, and for example increases the chances of errors due to password interference, i.e., confusing passwords between accounts.

C Detail of Usability-Deployability-Security Evaluation

Evaluation of (web passwords, mobile). We first discuss how the original ratings [5] for regular web passwords change for a mobile platform (e.g., smartphone), as summarized in row 2 of Table 1 with label (web passwords, mobile). We note the following ratings as downgrades from the desktop version: not-*Efficient-to-Use*/U6 and not-*Infrequent-Errors*/U7 (password entry is less efficient and more error-prone on mobile keyboards); and *Quasi-Accessible*/D1 (e.g., motor-impaired and blind users may have additional challenges). The rating *Nothing-to-Carry*/U3 is unchanged (the mobile platform is the primary device, not an auxiliary device needed for login to a primary device).

Evaluation of (ObPwd, desktop). We rate (ObPwd, desktop) as *Quasi-Memorywise-Effortless*/U1 (users must remember where to find the password object file in their file system but need not remember precise syntax details of the password characters); *Scalable-for-Users*/U2 (one password object generates unique passwords for different websites). We rate it not-*Nothing-to-Carry*/U3 in order to highlight the following device dependency: a desktop user may need to carry storage media containing their object files, for the reason of not having access to their digital objects on all login devices which they may wish to use (e.g., consider a friend's machine); available syncing mechanisms (as discussed earlier) should not be used on borrowed machines. We rate it not-*Physically-Effortless*/U4, by this benefit's strict definition, as login requires more than the press of a button. It is *Easy-to-Learn*/U5. It is *Quasi-Efficient-to-Use*/U6 as locating a single object file becomes easier with repeated use (similar to typing the same password). Note that, in our user study [3], the ObPwd login task on average took almost twice as long as text password login; however, users were selecting different object files for their eight test accounts. We rate it *Infrequent-Errors*/U7 (typing errors are eliminated; users need to locate only one password object across accounts). We rate it *Easy-Recovery-from-Loss*/U8 (if a user forgets the object file's path, or loses the object file, recovery is possible by the same mechanism as for lost regular text passwords).

For the primary use case of image-based objects, we must rate ObPwd non-*Accessible*/D1 (blind users will find it problematic). It is *Negligible-Cost-per-User*/D2 and *Server-Compatible*/D3 but not-*Browser-Compatible*/D4 (additional software must be installed). The desktop version is *Quasi-Mature*/D5 (has been implemented for various platforms, has a small user base beyond academic users, and has been formally user-tested but only one variant in small scale). It is *Non-Proprietary*/D6 (no patents are known to the scheme's designers; freely available for download).

The scheme is both *Resilient-to-Physical-Observation*/S1 and *Resilient-to-Targeted-Impersonation*/S2 (an attacker would also need an exact copy of the object file), and since in these passwords are essentially random from the viewpoint of an attacker with access to the object file [3], it is also *Resilient-to-Throttled-Guessing*/S3 and *Resilient-to-Unthrottled-Guessing*/S4. Due to the generation involving an essentially random string being salted with a site domain, the scheme is also *Resilient-to-Leaks-from-Other-Verifiers*/S6 and *Resilient-to-Phishing*/S7. The scheme matches web passwords for the remaining security benefits S8-S11. Notably, S1-S4, S6, and S7 all improve over basic web passwords. For S8 (*Resilient-to-Theft*), we grant the benefit based on strict definition, but note that password object files may be stolen if backed up onto portable media to allow device-independence, or made available to attacks through insecure syncing mechanisms (as discussed earlier); here we rate the scheme assuming that neither occurs, and note that consequently, the scheme has the disadvantage of being device-dependent (which also results in the penalty of not having the benefit *Nothing-to-Carry*/U3).

Evaluation of (ObPwd, mobile). For the ObPwd mobile version (see Table 1, last row), the following benefits are worth some comments relative to the desktop version: only *Quasi-Infrequent-Errors*/U7 as using the ObPwd app requires users to switch to the Gallery app to generate a password, and then paste the password to the requesting website. However, these ratings are based on anecdotal comments rather than a formal user study. Regarding the rating not-*Nothing-to-Carry*/U3 here: syncing mechanisms in the mobile version may also be not available at all, are not yet proven easy to set up and use by all users with existing mobile phones, or not used by choice if a secure version is not available, in which case a user may need to carry storage media containing their object files. We rate the mobile version as not *Mature*/D5 (a version for Android only is available for public download but is more recent than the desktop version, has not been formally user-tested, and has a smaller user base).

The issue of not being *Browser-Compatible*/D4 would disappear for ObPwd (as for any other proposal) if the method were widely adopted by browser vendors, and likewise for the missing benefit of being not *Mature*/D5 – but, the ratings measure the status quo, not what could be. Being not *Resilient-to-Internal-Observation*/S5 remains an important drawback, but is strongly related to the existing infrastructure for password verification—as a side effect of aiming to be *Server-Compatible*/D3, which is desirable at least in the short term to avoid imposing server-side changes on the password-supporting subset of the approximately one hundred million currently active web sites.