

Accessing password-protected resources without the password

Andreas Pashalidis

Abstract

Sometimes it is desirable to access password-protected resources, but undesirable to disclose the password to the machine in use. In such situations, providing the password is a task that can be delegated to a remote proxy server. This server has to engage the user in a challenge-response mechanism that does not require him to disclose his password to the local machine; if the user responds correctly, then the proxy must recover his password and fetch the protected resource for him. In this paper, we propose three schemes that are suitable for use in this environment and that (a) do not require the proxy server to permanently store a copy of the user's password, and (b) may be implemented without requiring support from the resource provider. We also briefly describe 'Keep Your Password Secret' (KYPS), which is a system that uses one of the schemes, and that has been in use for nearly two years..

1 Introduction

Sometimes it is necessary to access password-protected resources, for example certain web pages, using a machine that is not particularly trusted, such as a public terminal at an Internet café or an airport. In such a situation, the user typically has no alternative other than disclosing his password to the machine, and trusting that there is no hardware or software that enables unauthorised parties to retrieve it. Fortunately, it is possible to relieve the user from having to trust the terminal in this respect, by delegating the task of providing the password to a remote and trusted proxy server. This proxy server must ensure that (a) the user's password is not disclosed to unauthorised parties, and that (b) noone can abuse its service in order

to impersonate a legitimate user.

Such a proxy typically works by issuing a challenge to the user whenever a password-protected resource is requested. If the user's response to the challenge is correct, then the proxy recovers his password, fetches the resource, and forwards it to the user. Of course, the challenge-response scheme must be replay-resistant, and must not require the user to disclose his password (or any information that would enable an attacker to compute it).

In this paper, we propose three schemes that (a) enable the proxy to construct the password from the user's response to the challenge, i.e. that do not require the proxy to maintain a copy of the user's passwords, and (b) may be implemented without requiring support from the resource provider. Our schemes do not disclose *any* information about user's passwords to the machine in use and consist of two phases, namely a registration phase and a login phase. At registration, the user must execute a special 'registration program' that must be obtained from a trusted source, e.g. from the proxy website.¹ We further briefly describe 'Keep Your Password Secret' (KYPS) which is a web proxy that uses one of the schemes.

Related Work: The schemes described in this paper belong to the body of work on authenticating towards a remote server using an untrusted device. Many systems have been proposed (see, for example, [3, 5] and the references therein); most relevant to our work are the systems described in [2, 3, 9] because, in contrast to other works, they do not require support from resource providers, and they have been implemented as web proxies. The first system [9] can employ any challenge-response scheme for authenticating the user. However, it keeps a copy of the user's passwords in its database. While the sec-

¹This program could be, for example, a downloadable executable, a java applet, or a script on a webpage.

ond system [2] does not suffer from this drawback, two out of its three variants leak the user’s passwords to the local machine in one way or another (see section 3.3.2 of [2]), and the third variant is not easy to use (see section 3.4 of [2]). The schemes described in this paper do not leak any information about the user’s passwords and are relatively easy to use. It should moreover be noted that both [9] and [2] require changes to the local browser configuration in order to work, which limits their applicability; KYPS, however, does not require such changes. The third system, which was independently developed and is called Universal Replay-Resistant Secure Authentication (URRSA) [3], is similar to KYPS in many respects and, in fact, uses the same challenge-response mechanism.

The rest of the paper is organised as follows. Sections 2 and 3 present our schemes, section 4 describes KYPS and compares it to URRSA, and section 5 concludes and provides directions for further research.

2 A scheme based on one-time pads

This section describes our first scheme, which we call ‘one-time pad system’ (otps). It is based on a set of one-time pads that the user and the proxy share at registration.

Registration phase: The registration program asks the user to specify the set of password-protected resources for which he wishes to delegate password provision to the proxy. For each specified resource, the program then does the following.

1. It asks the user to enter his username and password for the resource. We denote his password by Π , and its bitlength by $|\Pi|$.
2. It asks the user how many one-time passwords (OTPs) he would like to generate, say t . The program then generates t one-time pads, each $|\Pi|$ bits long and independently and uniformly at random from $\{0, 1\}^{|\Pi|}$. These pads, denoted by p_1, p_2, \dots, p_t , are then sent to the proxy.
3. Finally, it generates a list of OTPs by computing $c_i = \Pi \oplus p_i$ for all $1 \leq i \leq t$, and prints this list using a suitable encoding, such as e.g. base64 [4]. An

www.example.com				
S/N	OTP	S/N	OTP	...
0	KkJ103G	26	fKtJ14X	...
1	VJaKSgn	27	Rtc6f60	...
2	w912mrT	28	qLaf92d	...
3	c2Ms6kL	29	mNsdgrR	...
...

Table 1: An example of an OTP list

example of how such a list may look like, is shown in Table 1.

Login phase: First, the user contacts the proxy and indicates which password-protected resource he wishes to access, and his username for accessing that resource. Then the proxy proceeds as follows.

1. It selects a pad from list of unused ones of that user (initially (p_1, p_2, \dots, p_t)). Say the server selects p_i . It then asks the user to supply the OTP with serial number i . We denote the user’s response by \hat{c}_i .
2. It recovers a password by computing $\hat{\Pi} = p_i \oplus \hat{c}_i$. Using the user’s username and $\hat{\Pi}$ as the password, the proxy now tries to fetch the password-protected resource on behalf of the user.
3. It deletes $\hat{\Pi}$ and removes p_i from the user’s list of unused pads.

Remark 1. If $\hat{c}_i \neq c_i$, then $\hat{\Pi} \neq \Pi$ and fetching the password-protected resource fails. Detecting, however, whether or not $\hat{\Pi}$ is wrong would require the proxy to interpret the response from the resource provider. Also note that the proxy does not authenticate the user; it merely maps a masked version of his password back to its clear-text version.

Remark 2. The bitlength of each pad reveals the length of the user’s password, and, since pads are stored at the proxy, the *length* of passwords is potentially exposed. However, a simple variation of the scheme where the length of pads have a (sufficiently large) *fixed* length, avoids this unnecessary potential exposure. KYPS uses a fixed pad length of 144 bits.

Remark 3. The user can obtain a new OTP list without having to disclose his password to the local machine, as

follows. First, he executes `otps` with the proxy, as usual. As a result of this, the proxy recovers the user’s password, and can now generate a new list of one-time pads, compute new OTPs in the same way the registration program does, and return the OTPs to the user. It should be noted, however, that the user should perform this task using a trusted machine because knowledge of the new OTPs enable an attacker to fetch the password-protected resource. That is, while this trick protects against keystroke loggers, it does not protect against an attacker examining the content accessed by the user. Moreover, if the user makes a mistake (which the proxy is unable to detect), then the new OTP list is useless.

Security analysis: As a result of using the scheme, a set of challenge-response pairs is disclosed to the local machine. Assuming that all pads are indeed generated independently and uniformly at random, knowledge of such pairs does not enable an attacker to learn anything about the user’s password (because they are information-theoretically independent from it [10]), and to use the services of the proxy (because they do not reveal any information about the correct response to any of the proxy’s future challenges). In particular, the probability that any given string in $\{0, 1\}^{|\Pi|}$ being the correct response to a future challenge, is uniform over all possible responses. Thus, the scheme does not enable an attacker to gain an advantage compared to password guessing, even for small passwords. However, the local machine gets to know the *length* of the user’s password. This is a potential weakness, since it enables an attacker to identify those passwords that are small enough to be guessed. A simple variation of `otps` where too short passwords are padded with a random string (which is removed by the proxy upon recovery) addresses this concern.

3 Two schemes based on one-way functions

This section describes two similar schemes, namely one that is based on a hash function, called the ‘hash-based system’ (`hbs`), and one that is based on a hash function and an encryption scheme, called the ‘encryption-based system’ (`ebs`). Let \parallel denote concatenation, let H denote a second pre-image resistant hash function with

output size h , and let E and D denote the encryption and the decryption algorithm of a semantically secure encryption system. `hbs` is based on a puzzle solving algorithm $\text{ps}(\rho, \tau, s)$ that, on input a ‘reference’ bitstring $\rho \in \{0, 1\}^*$, a ‘target’ bitstring $\tau \in \{0, 1\}^h$, and an integer number $1 \leq s < h$, outputs either a ‘solution’ bitstring $\sigma \in \{0, 1\}^s$ such that $H(\sigma \parallel \rho) = \tau$, or the failure symbol \perp if no such σ can be found. Note that the `ps` algorithm is somewhat similar to the puzzle solvers employed in certain schemes that address denial of service attacks (see, for example, [1]). In terms of security, `hbs` requires H to only have the one-wayness property; second pre-image resistance is, however, also required in order to avoid failures. The security of `ebs`, on the other hand, depends on the semantic security of E and of the one-wayness property of H . For a definition of the terms ‘one-wayness’, ‘second pre-image resistance’, and ‘semantic security’ the reader is referred to [7].

Registration phase: The registration program asks the user to choose a passphrase, denoted by \mathbb{P} in the sequel. Using, for example, suitable heuristics (see, for example, [11]), the program should check whether or not \mathbb{P} is long and complex enough in order for offline dictionary attacks on $H(\mathbb{P})$ to be infeasible; if it is not, then it should issue a warning to the user, informing him that choosing a too simple passphrase might enable an attacker that compromises the proxy to extract his passwords. The program could also offer the option of generating a suitable passphrase for the user. The program then asks the user to specify the set of password-protected resources for which he wishes to delegate password provision to the proxy. For each such resource, the program then does the following.

1. It asks the user to enter his username and password for the resource. We denote the password by Π and its k characters by $\pi_1, \pi_2, \dots, \pi_k$.
2. In `hbs`, it generates a ‘mask’ that hides the password using the passphrase. That is, for all $1 \leq i \leq k$, it generates $\mu_i = H(\pi_i \parallel H(\mathbb{P}))$. In `ebs`, it computes an encrypted version of the password $\Phi = E_{H(\mathbb{P})}(\Pi)$.
3. In `hbs`, it sends the mask $\mu_1, \mu_2, \dots, \mu_k$, to the proxy. In `ebs`, it sends the encrypted password Φ to the proxy.

www.example.com				
S/N	TAN	S/N	TAN	...
0	283742	26	975316	...
1	134841	27	186544	...
2	099471	28	210042	...
3	918823	29	722931	...
4	898234	30	541932	...
...

Table 2: An example of a TAN list

At registration, the user and the proxy must also agree on a suitable one-time password scheme. There exist a number of choices for such a scheme, e.g. a list of ‘transaction numbers’ (TANs), which is a technique that has been used in the context of online banking for some time. (An example of how such a list looks like, is shown in Table 2.) The registration program should enable the user to print his TAN list and, as an option, also include his passphrase in the printout.

Remark 4. In order to prevent outsiders from discovering the mask (and the TANs, if a TAN-based one-time authentication mechanism is used), the registration program must communicate with the proxy using a secure channel. This is important because otherwise an attacker that is listening on the network can try to recover password by means of an offline dictionary attack on $H(P)$. The success probability of such an attack depends on the quality of P and is, in practice, likely to be significantly higher than the probability of breaking a properly established secure channel.

Login phase: First, the user contacts the proxy and indicates which password-protected resource he wishes to access, and his username for accessing that resource. The proxy then executes the one-time authentication mechanism and, in addition, asks the user to provide his passphrase, denoted \hat{P} .

1. The proxy tries to recover the user’s password. That is, in hbs , for all $1 \leq i \leq k$, it recovers the i th password character as $\pi_i = ps(P, \mu_i, s)$ where s denotes the bitlength of a single character. If, at any stage, ps returns \perp , then this step is considered to have failed. Note that, due to the assumed second pre-image resistance of H , and if $\hat{P} = P$, then the probability that ps returns anything other than π_i or \perp , is negligible.

In ebs , on the other hand, the password is recovered by computing $\hat{\Pi} = D_{H(\hat{P})}(\Phi)$. If decryption fails, this step is considered to have failed.

2. If either the preceding step or the one-time authentication mechanism has failed, then the proxy’s next steps are subject to policy. The proxy could, for example, reply with a suitable error message, and ask the user to correct his entries for a couple of times before blocking the account for some time.
3. Using the user’s username and password, the proxy now fetches the password-protected resource on behalf of the user. On completion, it deletes his passphrase and his password (and removes the used TAN from the user’s list of unused TANs if a TAN-based one-time authentication mechanism is used).

Security Analysis: Due to the assumed one-wayness property of the hash function and the semantic security of the encryption scheme, and provided that P is sufficiently long and complex, the information disclosed by the registration program (i.e. the mask $\mu_1, \mu_2, \dots, \mu_k$ in case of hbs , or the encrypted password Φ in case of ebs) does not enable the proxy to recover any character of the user’s password.

As a result of using hbs or ebs , the user’s passphrase and a set of challenge-response pairs is disclosed to the local machine. These items of information, however, do not enable an adversary (a) to learn anything about the user’s password (because they are independent from it), and (b) to use the services of the proxy (because they do not reveal any information about the correct response to any of the proxy’s future challenges).

Remark 5. In the context of a TAN-based mechanism, the printed TAN list does not contain any information about the user’s password; a stolen TAN list is useful to an attacker only if he knows the user’s passphrase, too. Therefore, if the passphrase is not printed on the list, then losing the TAN list does not represent a significant security risk to the user. In other words, the system provides two-factor authentication in this case.

The user can be given the choice to have a single passphrase for all his passwords, or a separate one for each. The tradeoff involved is that, while the former alternative is more user-friendly, someone who gets to know

the passphrase *and* the mask $\mu_1, \mu_2, \dots, \mu_{k-1}$ (in `hbs`) or the encrypted password (in `ebs`), such as an attacker that has compromised the proxy and that observes the user's passphrase during a login phase, will be able to recover all the user's passwords. The latter alternative, while more difficult to use, only affects the password that is masked/encrypted by that passphrase.

Remark 6. Assuming a s -bit character encoding and that `ps` operates by testing all strings in $\{0, 1\}^s$ in turn, and by outputting the first solution it finds, and \perp if no solution has been found after after all 2^s strings have been tried, recovering a k -character password requires $k2^{s-1}$ hash function evaluations on average, and $k2^s$ evaluations in the worst case. The mask $\mu_1, \mu_2, \dots, \mu_k$, on the other hand, takes up kh bits. One can trade off space and time complexity of `hbs` by dividing the binary representation of the password into s' -bit chunks (where $s' \neq s$), and treat each chunk as a 'character'. In the extreme case where each bit of the password is treated separately, the amount of necessary hash function evaluations is reduced to sk (average case) and $2sk$ (worst case), at the expense of exchanging a mask that takes up hsk bits at registration time. As an example, consider a hash function with output size 160 bits, a 9-character password, and an 8-bit character encoding. The number of hash function evaluations in order to recover the password in this case is $9 \cdot 2^7 = 1152$ on average, and $9 \cdot 2^8 = 2304$ in the worst case, and $160 \cdot 9 = 1440$ mask bits have to be exchanged at registration. Treating each bit of the password separately yields a system where the number of required hash function evaluations is $8 \cdot 9 = 72$ (average case) and $2 \cdot 8 \cdot 9 = 144$ (worst case), at the expense of exchanging $160 \cdot 8 \cdot 9 = 11520$ mask bits at registration.

Remark 7. While `otps` requires the user to carry an OTP list, `hbs` and `ebs` do not necessarily require the user to carry anything; if the TAN-based one-time authentication mechanism described above is replaced, for example, with a challenge-response scheme that is based on something the user knows (and assuming that he memorises his passphrase), then there is no need for the user to carry anything. For examples of such mechanisms, see e.g. [9]. Furthermore, in all three schemes, the user is not required to even remember his passwords during the login phase: in `otps`, it suffices if the user has the OTP list, and in `hbs` and `ebs` it suffices if he knows his passphrase

and if he can correctly respond to the challenge-response scheme in use. This is a usability advantage for users that would otherwise have to remember a large number of different passwords. Moreover all three schemes provide a form of 'forward secrecy' because, even if an attacker compromises the proxy at some point in time, he will be unable to reconstruct previously provided passwords. While this holds unconditionally for `otps` [10], for `hbs` and `ebs` this holds only for those passphrases \mathbb{P} that are unknown to the attacker and that are indeed long and complex enough in order for offline dictionary attacks on $H(\mathbb{P})$ to be infeasible.

4 Implementation

'Keep Your Passwords Secret' (KYPS) is a Java implementation of a 'reverse' web proxy service that enables users to log into websites using `otps`-based OTPs (see Table 1) instead of their passwords.² Since its initial deployment in early 2007, the service has migrated from a plain socket-based implementation to one based on servlets. Currently, there exist two variants of the service, namely one that fully encrypts Uniform Resource Locators (URLs), denoted *K1*, and one that only partially encrypts URLs, denoted *K2*. Both variants, collectively denoted simply by KYPS, have been tested with all popular browsers and major websites including Google Mail, Yahoo Mail, GMX, and Paypal. KYPS has been deployed at <http://kyps.net> and, to date, has been used by several hundred users from different countries. In the following, we describe the main differences between KYPS and URRSA [3].

- In contrast to URRSA, KYPS encrypts cookies using a symmetric encryption scheme with a key only known to the service, and irrevocably deletes this key after a specified period of inactivity. This behaviour addresses the threat posed by cookie theft, and results in effective logout after the inactivity period has passed.

²KYPS uses a *pseudorandom* number generator (PRNG) for the generation of the pads. Thus, the pads are not generated independently from each other, as required by the theoretic construction. However, since KYPS uses a *secure* PRNG (i.e. one that passes certain tests [8]), it is safe to assume that, in practice, it is infeasible to extract any information about the password from the OTPs.

- In contrast to KYPS, URRSA supports the sending of OTPs to the user's mobile phone using the Short Messaging Service (SMS). A similar extension is, however, also planned for KYPS.
- Both URRSA and KYPS use a lossless compression scheme in order to reduce the bitlength of plaintext passwords, as first proposed in [3]. However, URRSA and KYPS differ in their encoding of OTPs. In particular, URRSA uses a restricted set of capital letters that avoids characters that are easy to confuse, and KYPS uses regular base64 encoding, albeit using a font where the appearance of similar characters, e.g. '0' and 'O', still differs significantly. While avoiding confusion, the OTPs returned by URRSA are, on average, lengthier than those returned by KYPS. Determining which approach is preferable in terms of usability is an interesting research question.
- In contrast to URRSA, KYPS does not interpose the website's login page during the login phase, but directly logs the user into the website. While a usability advantage, this only works if the service can detect both the username and the password field in the remote login form. If it fails to do this, then the affected website is not supported. URRSA, on the other hand, only needs to correctly identify the password field of the login form. However, we have found that the majority of websites use easy-to-identify username fields in their login forms; hence, only few websites are affected by this issue in practice.
- In contrast to URRSA, K1 encrypts URLs. This addresses the threat posed by the theft of special authentication values that are sometimes encoded in URLs.
- Some web page functionalities (i.e. certain javascripts) that are supported by URRSA are not supported by K1. This is because, in order to encrypt all URLs on a webpage, K1 has to identify them and convert them into absolute form prior to encryption. However, some websites use dynamically construct URLs using javascript; not all URLs that are constructed in this manner are currently not detected by K1, causing the affected webpages to lose some or all of their functionality.
- K2 uses a reverse proxying method that is inspired by the DNS-assisted method described in [6, 3], and which circumvents the need to identify and convert relative URLs on into absolute form. Unlike URRSA, K2 encrypts the DNS name of the remote server, and prepends the encryption to its own domain name *in base32 encoding* [4]. The resulting new scheme has both advantages and disadvantages compared to the reverse proxying method of URRSA. Advantages include the fact that, if KYPS is invoked using HTTPS (the secure HTTP variant), then the proxying method ensures that a wildcard SSL/TLS certificate applies to all subdomains that are generated in this way, and that, as a result, no certificate mismatch warnings are generated by the browser. Moreover, due to the garbled appearance of part of the DNS name, the method makes it easier for users to distinguish between being directly connected to a website, and being connected via the proxy. The main disadvantage is the fact that the scheme requires the proxy to identify and translate entire DNS names, and this is currently not supported for webpages that dynamically construct DNS names using javascript. As a result, the affected webpages are likely to lose some or all of their functionality. URRSA, on the other hand, is able to handle even webpages with dynamically generated DNS names because its algorithm does not need to identify entire DNS names, but only their top-level domain (TLD) part (see [6]).

5 Conclusion

We proposed three schemes that enable users to access password-protected resources without disclosing any information about their passwords to the device they are using. The schemes, which can be implemented without the need for support from the resource provider, delegate the provision of the password to a trusted proxy. They do not, however, require the proxy to permanently store a copy of the password. Furthermore, we briefly described two variants of KYPS, which is a service that uses one of the schemes and that enables one to log into username/password-based web accounts using OTPs. KYPS does not require the user to make any changes to

the browser configuration, and is therefore suitable for login from untrusted devices such as an Internet café or a public terminal.

In the future, we intend to extend it as follows. Firstly, we intend to extend the number of challenge-response mechanisms it supports, and enable users to choose their preferred method during registration. Secondly, we intend to examine other encodings and entry modes for OTPs such as graphic representations, dictionary words, and selection as opposed to keyboard entry. Thirdly, we intend to provide a ‘personal edition’ of the server software. This software, will enable users with flatrate Internet access to turn their home computer into a trusted ‘password provision proxy’ and use it in order to safely log into their web accounts using foreign machines when away from home. Finally, we intend to deploy the service in a corporate environment in order to enable remote workers to login using foreign machines without the risk of password compromise. We believe that deploying its own KYPS service for this purpose is an attractive alternative in a corporate environment because it works ‘out-of-the-box’, i.e. without the need for potentially costly integration with existing infrastructure.

Acknowledgements

The author would like to thank Markus Dichtl who suggested otps during discussions in 2006, Cormac Herley for helpful discussions, feedback, and clarifications on the operation of URRSA, and Bernd Meyer for feedback on some of the schemes in this paper.

References

- [1] T. Aura, P. Nikander, and J. Leiwo. DOS-resistant authentication with client puzzles. In B. Christianson, B. Crispo, and M. Roe, editors, *8th International Workshop on Security Protocols, Revised Papers*, volume 2133 of *LNCS*, pages 170–177. Springer, 2001.
- [2] D. Florencio and C. Herley. KLASSP: Entering passwords on a spyware infected machine using a shared-secret proxy. In *ACSAC 2006, Proceedings*. ACSAC, 2006.
- [3] D. Florencio and C. Herley. One time password access to any server without changing the server. In *ISC 2008, Proceedings*, volume 5222 of *LNCS*, pages 401–420. Springer, 2008.
- [4] S. Josefsson. *RFC 4648: The Base16, Base32, and Base64 Data Encodings*. IETF, October 2006.
- [5] M. Mannan and P. C. van Oorschot. Using a personal device to strengthen password authentication from an untrusted computer. In S. Dietrich and R. Dhamija, editors, *FC 2007 and USEC 2007, Proceedings*, volume 4886 of *LNCS*, pages 88–103. Springer, 2007.
- [6] Z. Mao and C. Herley. *MS-TR: A Robust Link Translating Proxy Server Mirroring the Whole Web*, 2008.
- [7] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, Boca Raton, 1997.
- [8] National Institute of Standards and Technology. *FIPS PUB 140-2: Security Requirements for Cryptographic Modules*, May 2001.
- [9] A. Pashalidis and C. J. Mitchell. Impostor: A single sign-on system for use from untrusted devices. In *IEEE Globecom Conference*. IEEE Press, 2004. (<http://impostor.sf.net>).
- [10] C. E. Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28:656–715, 1948.
- [11] E. R. Verheul. Selecting secure passwords. In M. Abe, editor, *CT-RSA 2007, Proceedings*, volume 4377 of *Lecture Notes in Computer Science*, pages 49–66. Springer Verlag, Berlin, 2007.