A Protocol to Strengthen Password-Based Authentication

Vazquez Sandoval Itzel, Stojkovski Borce, and Lenzini Gabriele ⋆

SnT/University of Luxembourg
{itzel.vazquezsandoval, borce.stojkovski, gabriele.lenzini}@uni.lu

Abstract. We discuss a password-based authentication protocol that we argue to be robust against password-guessing and off-line dictionary attacks. The core idea is to hash the passwords with a seed that comes from an OTP device, making the resulting identity token unpredictable for an adversary. We believe that the usability of this new protocol is the same as that of password-based methods with OTP, but has the advantage of not burdening users with having to choose strong passwords.

Keywords: Password-based Authentication, Cryptographic Protocols

1 Introduction

Password-based authentication is the most common mechanism for online authentication. It relies on one factor: 'something-you-know', and requests that the legitimate user proves knowledge of a username and a password. The whole security of the method rests, however, on the password, which must be kept secret and be chosen *strong i.e.*, unpredictable. Unfortunately, this is rarely happening.

Security research has extensively commented on the poor quality of people's choices regarding passwords (*e.g.*, see [11, 2, 7, 10]). The problem is subtle. According to [9], people do have certain critical misunderstandings about what makes a password strong and are unaware of the complete attack surface, but their intuitions about what a secure password should look like are usually in line with password-cracking approaches. Despite that, their passwords are commonly short, built from predictable words and phrases which also tend to be semantically related, and are also often reused across different accounts. Regrettably, poor password management practices are also very common, which worsens the situation.

Weak passwords are particularly problematic. Hackers can easily guess them or they can steal password files, a common attack as recent news on the breaches at Reddit[1], Twitter[2] and Yahoo![3] prove. Although login servers protectively

⋆ Authors are supported by the projects: pEp Security SA / SnT "Protocols for Privacy Security Analysis"; FNR-PRIDE "Security and Privacy for System Protection".

[1] https://www.bbc.com/news/technology-45040804

[2] http://www.wired.co.uk/article/twitter-hack-breach-32-million-passwords

[3] https://www.nytimes.com/2016/12/14/technology/yahoo-hack.html

store only the hash of the passwords, poorly chosen passwords are retrievable by off-line dictionary attacks. Attacks on password files are generally a preparation for further intrusions that take advantage of people's reuse of passwords.

This discussion raises an obvious *research question*: if we accept that users choose weak passwords, how can we guarantee that password-based authentication is secure *i.e.*, that it works with unpredictable identity tokens and with password files resilient to off-line dictionary attacks? And, if a solution exists, how can it be achieved without imposing too much burden on users? The goal of this short paper is to answer these questions, to propose a possible solution, and to discuss its security.

1.1 Scope of the Work

There might be several different ways to approach the above research questions and to find answers. Many researchers opt for supporting authentication without passwords (*e.g.*, see [8]); we are not interested in solutions of this kind since we intend to remain within the context of password-based authentication.

There are also methods that use tokens as alternative passwords, like those requesting freshly generated PINs. The security of such methods relies exclusively upon the possession of a personal device. We briefly comment on them in Section 4, mainly to compare their security with that of our solution, but using PINs and similar codes is not the answer we seek for our question.

So, what type of strategies remain in our scope? We see here at least two families of them. One includes techniques and resources that help users generate and memorize hard-to-guess passwords. They can be mnemonic strategies, but they cannot be taught to everyone and, we believe, the practice does not scale. Alternatively, they can be applications, such as the *password vaults* like Schneier's "Password Safe" or the compatible "Password Gorilla". They generate strong passwords on behalf of the user and keep them safe in an encrypted file on the user's personal device, available on demand. The drawback is that the vault's access is password-secured, which introduces a circular problem as the vault's access can be vulnerable to dictionary attacks. Besides, we are unaware of any research that brings evidence of a widespread adoption of password vaults, although their recent integration in some browsers will increase their use.

A second family includes protocols that implement second-factor authentication, and the second factor is often "something-you-have". Authentication is still password-based, but the identity of who is submitting a correct username and password is further verified by proving possession of a personal device (*e.g.*, a token generator, a phone, an account). To this family belongs a multitude of solutions (see for instance the Google 2-step verification[4]), but discussing each and every different instance in this quite crowded family is beyond the ambition of this paper. We can, however, observe one important fact: while the trustworthiness of a user's authentication is strengthened by the second factor, users can

[4] https://www.google.com/landing/2step/

still choose weak passwords and the leak of a password file remains a serious issue.

Nevertheless, it is in this category that we find our main source of inspiration. In particular, we look at protocols where the second factor is a One-Time-Password (OTP) device. Such protocols are common, and the closest to the password-based ones in terms of usability: in addition to the username and password, they require the user to input also a nonce which will be submitted simultaneously.

1.2 Previous Work and Contribution

To the best of our knowledge, the protocol that we describe here is novel, but the motivation of the work is rooted in previous research of ours [4].

There, we studied a password-based authentication system first described by Jewels and Rivest [5]. Its main goal is to make the stealing of a password file *tamper-evident*. The system, called *honeywords system*, has a simple security working principle: legitimate user-chosen passwords are stored together with some decoy words, called honeywords, which are indistinguishable from the password (*e.g.*, indistinguishable as "whitemoon" is from "redsun"). An adversary who stole the password file and retrieved the words by an off-line dictionary attack cannot do better than picking one word at random, revealing that the file has been leaked if he tries to authenticate with a wrongly picked word.

In [4] we reviewed the protocol to make it tamper-evident when the Login Server (LS)'s code is corrupted by the adversary. The resulting protocol seems to suggest a completely new way to authenticate users, which also makes a password file resilient to off-line dictionary attacks. We left for future work to look into the matter; here we develop that idea into a novel password-based authentication protocol.

We anticipate that our solution spares users from having to choose strong passwords provided that they use an OTP device. But, differently from the common use of the device as a proof-of-possession, we use OTP's numbers to generate fresh identification tokens with high entropy, that depend on the password; they are unpredictable (by an adversary) and not vulnerable to dictionary attacks.

2 An Enhanced Password-based Authentication

We assume two roles: the User (U) and a LS. We envision three different protocols: registration, authentication, and update.

Assumptions We assume that U and LS communicate through a secure channel *e.g.*, implemented by a TLS protocol. We also assume that a pre-image resistant hashing algorithm H, such as SHA-512, is applied to passwords and that LS stores hashed passwords as usual. We omit obvious checks like verifying that the username exists. As well, we assume that U holds an OTP device which has been delivered securely, as it is usually the case. The device's output, which changes every time the device is operated, is aligned with the output produced by a corresponding OTP's generator algorithm in the LS. $\texttt{OTP}(n)$ is the number produced by the device when used for the n-th time (equivalently, n times).

Registration U follows this protocol to register to a service and to set his password w (see Figure 1). U operates the OTP device for the first time to get a number, $\texttt{OTP}(1)$ (**1**). The password is hashed using H and then rehashed, this time using $\texttt{OTP}(1)$ as a seed. The token obtained, $h^{r_1}(H(w))$ is sent to LS together with U's id, u (**2**). On reception, LS anticipates the next OTP number, $r_2 = \texttt{OTP}(2)$ (by operating the device twice) and rehashes the identity token it has received from U using that number (**3**,**4**). LS relies on a strategy that we describe next and that does not require LS to know the password. The rehashed token, once more hashed with H, is stored as u's password (**5**). LS also stores r_2 and the registration for U concludes (**6**). Note that here we have assumed that the OTP generates a new number each time it is used. If instead the output of the device depended on the current time, as some OTP systems work, the protocol would have to be modified. This is not a dramatic change, but for space reasons we omit that version.

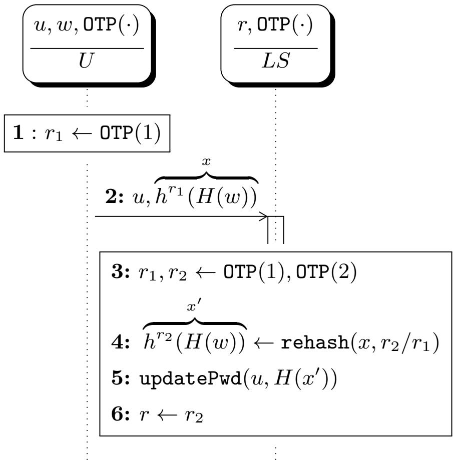
$u, w, \texttt{OTP}(\cdot)$
U
$r, \texttt{OTP}(\cdot)$
LS
$\boxed{\mathbf{1}: r_1 \leftarrow \texttt{OTP}(1)}$
$\mathbf{2:}\ u, \overbrace{h^{r_1}(H(w))}^{x}$
$\mathbf{3:}\ r_1, r_2 \leftarrow \texttt{OTP}(1), \texttt{OTP}(2)$
$\mathbf{4:}\ \overbrace{h^{r_2}(H(w))}^{x'} \leftarrow \texttt{rehash}(x, r_2/r_1)$
$\mathbf{5:}\ \texttt{updatePwd}(u, H(x'))$
$\mathbf{6:}\ r \leftarrow r_2$

Fig. 1. Registration

The hashing is implemented by *cryptographic exponentiation*. For each user id u, LS possesses g_u, a generator of a multiplicative subgroup $\mathbb{G}$ of order q. Then, u's hashed password, $H(w)$, is re-hashed using g_u^r, where $r \in \{1, \cdots, q-1\}$ is a random number. Herein r is obtained by operating the OTP device. The value submitted for authentication is $g_u^{r \cdot H(w)}$, which we denote as $h^r(H(w))$ to stress that it is a hashing and to lighten the notation regarding u's dependency. Using this scheme LS can, for another $r' \in \{1, \cdots, q-1\}$, calculate $h^{r'}(H(w))$ from

$h^r(H(w))$ only by knowing r and r' and not $H(w)$. In fact:

$$h^{r'}(H(w)) = g_u^{r'\cdot H(w)} = g_u^{r\cdot\frac{r'}{r}\cdot H(w)} = (g_u^{r\cdot H(w)})^{\frac{r'}{r}} = h^r(H(w))^{\frac{r'}{r}} \tag{1}$$

Such a feature is at the core of the authentication procedure.

Authentication The protocol's sequence of messages, for a general authentication round n, is shown in Figure 2. U submits a username u, and a token that is the password w, hashed with H and rehashed with the current $\texttt{OTP}(n)$ (**2**). LS retrieves u's token from the password file (**3**) and proceeds with authenticating U. For this, it hashes the received token x, which must match the token y stored in the file (**4**). If the check succeeds, LS uses the OTP to anticipate the next number and calculates the next identity token of u (**5,6**) by using the rehashing as described in equation (1) above; then LS updates the password file and r (**7,8**); otherwise, the access is denied. U receives a response (**9**).

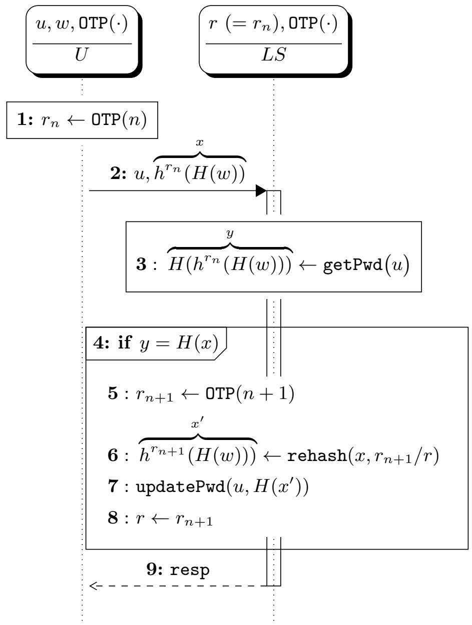
$u, w, \texttt{OTP}(\cdot)$
U
$r\ (= r_n), \texttt{OTP}(\cdot)$
LS
1: $r_n \leftarrow \texttt{OTP}(n)$
2: $u, \overbrace{h^{r_n}(H(w))}^{x}$
3 : $\overbrace{H(h^{r_n}(H(w)))}^{y} \leftarrow \texttt{getPwd}(u)$
4: if $y = H(x)$
5 : $r_{n+1} \leftarrow \texttt{OTP}(n+1)$
6 : $\overbrace{h^{r_{n+1}}(H(w)))}^{x'} \leftarrow \texttt{rehash}(x, r_{n+1}/r)$
7 : $\texttt{updatePwd}(u, H(x'))$
8 : $r \leftarrow r_{n+1}$
9: $\texttt{resp}$

Fig. 2. The n-th ($n > 1$) run of the Authentication Protocol

Update The update protocol allows U to change the password. It combines authentication and registration. We omit the full description for reasons of space.

2.1 Security Analysis

We discuss the security of the protocol in reference to an adversary that either (a) tries to guess U's password, or (b) has eavesdropped U's communication, or (c) has stolen the password file and tries off-line dictionary attacks on it, or (d) has stolen the OTP device and the password file (but not the password).

At the end of round n of the authentication, the value stored by LS is $H(h^{r_{n+1}}(H(w)))$, where H is the common hashing. Since we allow the user to choose the password, we cannot exclude the possibility of w being weak and thus of the intruder guessing it. However, without holding the OTP device the intruder cannot generate the right identity token to get access, nor use LS as an oracle to verify whether the guess is correct. This answers case (a).

Even if the intruder could observe the communication and retrieve one identity token $h^x(w)$ from any previous sessions of any protocol, he cannot reuse that token: identity tokens are one-time valid. Neither can the intruder guess the new token. If we work under CDH assumption, the knowledge of previous identity tokens of the form $h^r(x)$ for some r, does not give any advantage to the attacker even if he combines this knowledge with the password. To generate the next identity token the adversary needs also the OTP number generated by the device. This answers case (b).

Getting possession of the password file does not help the intruder either. First, since the re-hashing is not directly applied to plain text words, but to $H(w)$, the values obtained by the re-hashing function seeded with the OTP number are not retrievable by a dictionary attack. Second, he would need to calculate H's pre-image to extract from the password file the next u's token, but this is not possible since H is pre-image resistant. This answers case (c).

Finally, if the attacker were able to obtain the OTP and the password file, but not the password, he still could not authenticate. Even in the very unrealistic situation where the attacker knows g and the stored LS's r, he cannot generate any next u's identity tokens because he would need H's pre-image for that, *i.e.*, $h^r(H(w))$ which is hashed in the password file. Just for the sake of speculation, we comment that there might exist one remote possibility for the attacker to be able to launch an off-line dictionary attack: steal an OTP ready to be used for the $n+1$ time and get a password file that contains exactly $H(h^{r_{n+1}}(H(w))$. But the intruder cannot know whether he finds himself in this lucky situation.

Aiming to formally prove our claims, we analyzed the protocols in Proverif [1]; the results confirm that access to the system is granted only when there has been a request from a user and the hash of the credentials submitted (user-id, password and `OTP` number) corresponds to the value stored in the password file owned by the LS. In this short paper we omit the part where we describe the analysis and the code, but we plan to add it in an extended version of the paper.

3 Implementation

The most obvious way to implement the protocol's main operation, exponentiation, is by elliptic curve (EC) multiplication. To protect implementations against remote timing attacks [3], the time-cost of the multiplication is usually t_c, a constant that depends on the chosen curve c. Thus, the time-cost of our protocol's implementation is constant in t_c. We do not have measures over our protocol performances (we are currently implementing our solution in `C#` atop the Microsoft .NET framework), but from previous experiences with more complex protocols using exponentiation, as the one documented in [4], we expect the overhead on the user and on the login server to be negligible.

In a practical implementation, we have to consider that a user can accidentally burn some OTP numbers. This problem can be solved by the LS anticipating the next, let us say m, OTPs. Thus, LS has to store for each user a row of values, disposing of the old ones when a valid token is presented.

4 Discussion, Related Work, and Future Work

Our protocol has two main advantages: (1) it releases users from the burden of having to choose strong passwords at the price of handling an OTP device and of minor changes in implementation of the authentication protocol; (2) it makes it less interesting for adversaries to hack the password file, since they cannot use it for further attacks neither in the same nor in other domains.

In proposing our solution we were resolved to keep the use of passwords, which is still the mostly used method for authentication. In current commercial applications, comparable solutions are, however, available. More and more services request users to submit one-time PINs which are generated on (or sent to) their personal devices. The PINs are submitted instead of the password. Other services welcome innovative dongles, like the YubiKey[5], multi-purpose security tokens that can store passwords, generate OTPs, and can play different challenge-response protocols. A formal analysis of the security of such alternative solutions is future work, but, at least informally, it seems that their security is equivalent to that of our protocol: they do not handle tokens that are vulnerable to guess and off-line dictionary attacks. If there is a factor that can make a difference, it is the usability aspect, since the ceremonies and the interfaces that they implement for the authentication differ from one another. We leave for future work to define research questions and to design experiments apt to measure usable security aspects for such a diversified set of authentication procedures.

The closest theoretical work to ours, is that of Lamport [6]. In short, it demands that, at the nth round, LS stores a $(K - n)$-time nested hash of the password while the user authenticates by providing a token that is a $(K-n-1)$-time nested hash of it. Here, K is a shared constant. Lamport's protocol ensures the same advantages (1) and (2) because LS stores a token whose pre-image, if stolen, cannot be calculated. At the same time LS can efficiently perform the authentication check. Lamport's and our solution do probably differ at the level of usability and performance, but we have not made any measurements yet.

[5] https://www.yubico.com/

Limitation We wrote this short paper primarily to share our idea and to open a discussion on it. Since the intuition behind our solution originates from a research we did while addressing another problem (see Section 1.2), we branched from that work and started with the protocol design without having done first an extensive comparison with the state-of-the-art. Thus, although the research reported herein is original (*i.e.*, not published elsewhere) our related work section is limited to a few commercial protocols that use passwords or similar tokens; it lacks consideration of other theoretical ideas that may be close to what we have conceived. The only exception is the Lamport's authentication protocol [6]. Published in 1981, the work has been suggested to us by a reviewer, to whom we are grateful. An extensive comparative analysis of the two protocols in terms of security, performance, and usability is also left for future work.

References

1. Blanchet, B., Smyth, B., Cheval, V.: ProVerif 1.96: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial (2016)
2. Bonneau, J.: The science of guessing: analyzing an anonymized corpus of 70 million passwords. In: Security and Privacy (SP), 2012 IEEE Symposium on. pp. 538–552. IEEE (2012)
3. Brumley, B.B., Tuveri, N.: Remote Timing Attacks Are Still Practical. In: Proc. of the 16th European Conference on Research in Computer Security (ESORICS'11). pp. 355–371. Springer-Verlag (2011)
4. Genç, Z.A., Lenzini, G., Ryan, P.Y.A., Vázquez Sandoval, I.: A Security Analysis, and a Fix, of a Code-Corrupted Honeywords System. In: Proc. of the 4th Int. Conf. on Information Systems Security and Privacy (ICISSP 2018). pp. 83–95 (2018)
5. Juels, A., Rivest, R.L.: Honeywords: Making password-cracking detectable. In: Proc. of the 2013 ACM SIGSAC Conf. on Computer & Communications Security. pp. 145–160. ACM (2013)
6. Lamport, L.: Password Authentication with Insecure Communication. Communication of the ACM **24**(11), 770–772 (1981)
7. Malone, D., Maher, K.: Investigating the Distribution of Password Choices. In: Proc, of the 21st Int. Conf. on World Wide Web. pp. 301–310. WWW '12, ACM, New York, NY, USA (2012)
8. Stajano, F.: Pico: No More Passwords!! In: Proc. of the Security Protocols Workshop. LNCS, vol. 7114. Springer-Verlag (2011)
9. Ur, B., Bees, J., Segreti, S.M., Bauer, L., Christin, N., Cranor, L.F.: Do Users' Perceptions of Password Security Match Reality? Proc. of the 2016 CHI Conf. on Human Factors in Computing Systems (CHI'16) pp. 3748–3760 (2016)
10. Von Zezschwitz, E., De Luca, A., Hussmann, H.: Survival of the Shortest: A Retrospective Analysis of Influencing Factors on Password Composition. In: Human-Computer Interaction (INTERACT 2013). pp. 460–467. Springer-Verlag (2013)
11. Wash, R., Rader, E., Berman, R., Wellmer, Z.: Understanding Password Choices: How Frequently Entered Passwords Are Re-used across Websites. In: Proc, of 12th Symposium on Usable Privacy and Security (SOUPS 2016). pp. 175–188. USENIX Association, Denver, CO (2016)