KAuth: A Strong Single Sign-On Service based on PKI

Panayiotis Charalambous, Marios Karapetris, and Elias Athanasopoulos
University of Cyprus, Cyprus
{pchara20,mkarap01,eliasathan}@cs.ucy.ac.cy

Keywords: Authentication, Passwords, PKI

Abstract: We deploy PKI for human authentication. We use a publicly available infrastructure, namely Keybase, for managing public-key pairs across devices. In addition, Keybase offers us several features for identifying users in social networks and a login-to-Keybase process which is password-less, meaning that authentication takes place using digital signatures produced by an Elliptic Curve (EC) cryptosystem. By using Keybase, we minimize the required cryptographic keys to the absolute minimum: one. We transform Keybase to a Single Sign-On (SSO) service which can vet users for using other services, exactly as it happens now with very popular, but entirely password-based, services. We implement two authentication schemes based on Keybase, KAuth and KAuth+, and we evaluate them using a state-of-the-art methodology.

1 Introduction

Human-to-machine authentication is still based on text-based passwords, despite the many different proposals for better authentication systems and the clearly negative stance of IT vendors against passwords (Wired.com, 2013; PayPal, 2013). This has significant implications, since passwords are reused (Florencio and Herley, 2007), are leaked due to services' vulnerabilities (Ars Technica, 2013) and not due to user mistakes (Juels and Rivest, 2013; Kontaxis et al., 2013), are phished (Dhamija et al., 2006), and the overall user experience is severely degraded (Mail, 2015).

Cryptology has built several tools for building strong authentication. Unfortunately, such techniques have been used so far for machine-to-machine authentication (Dierks, 2008) or ad hoc for password hardening (Everspaugh et al., 2015; Schneider et al., 2016; Lai et al., 2017), where a cryptographic service is used to add layers of encryption thus making password cracking more difficult. Although cryptography has progressed, the techniques provided are marginally applied to human authentication, since they are still deemed as user-unfriendly and cryptographic systems, for instance based on a public-key infrastructure (PKI), are not deployed.

So far, although PKI can offer strong authentication, for human-to-machine authentication PKI is still considered unfriendly due to the following major problems:

P1 *Key maintenance*. Cryptographic keys have to be present during authentication, while users frequently use several devices to access services. Moving cryptographic keys from device to device, especially upon buying a new one, is considered a tough process even though passwords can be memorized, or easily recovered using password reminders.

P2 *Key revocation*. Compared to changing a password by following an e-mail link, finding the services that are associated with a particular (leaked) cryptographic key and revoking the key is challenging.

In this paper, we attempt to deploy PKI for human authentication by attacking both aforementioned problems, *P1* and *P2*. In particular, for solving *P1* we use a publicly available infrastructure, namely Keybase (Keybase, 2018), for managing public-key pairs across devices. In addition, Keybase offers us several features for identifying users in social networks and a login-to-Keybase process which is password-less, meaning that authentication takes place using digital signatures produced by an Elliptic Curve (EC) cryptosystem. Furthermore, for solving *P2*, we minimize the required cryptographic keys to the absolute minimum: one. We transform Keybase to a Single Sign-On (SSO) service which can vet users for using other services, exactly as is happening now with very popular, but entirely password-based, services (Miculan and Urban, 2011; Google, 2018; Twitter, 2018). Our proposed system, KAuth, uses PKI to authenti-

cate users, without suffering from *P1*, and once a user is authenticated, they can proceed and enjoy a third-party service. In the case of private key leakage, a user can simply revoke their key which is known *only* to our system, without being affected by *P2*.

Why Keybase? We build KAuth on Keybase for two major reasons. First, Keybase offers several options for cryptographic operations. We, also, assume that in the future Keybase can incorporate additional cryptographic ciphers. Second, Keybase offers most of the features through a user-friendly environment, such a web browser. For instance, when a user needs to authenticate with Keybase, a passphrase is used to derive a cryptographic key that will carry out an EdDSA signing process. The whole process is implemented in the web browser and resembles a typical password-base authentication routine, but it is not.

Is Keybase secure? Keybase is a relatively new platform and it is likely to suffer from vulnerabilities that are not exploited, yet. For instance, Keybase uses Elliptic Curves for user authentication, which are much more under-researched than RSA. In this paper, we use Keybase mostly as a reference implementation and we argue that cryptographic primitives can be offered in a user-friendly way, while realizing a much more stronger authentication to users.

In fact, our vision is that authentication should be provided with *options* and users should be able to be *selective*. Nowadays, many authentication proposals are never implemented because they are deemed non friendly. Our philosophy is that users do not fall all under the same catergory and many may be willing to sacrifice convenience for more security. Having said that, we view our prototype more as complementary to other SSO implementations and not as a competitor. For instance, currently deployed SSO services can be inspired from Keybase and our work, and integrate (optional) cryptographic-based authentication schemes in addition to their typical password-based authentication.

Contributions

1. We design and implement KAuth, a system which provides strong PKI human-to-machine authentication.

2. We evaluate KAuth with an established framework (Bonneau et al., 2012) and show how KAuth can defend users against several password-related attacks, such as phishing and password leakage, without severely affecting the user's experience. In fact, the user is hardly aware that PKI is in place.

2 KAuth Architecture

One of the most popular and widely adopted mechanisms that is really similar to our approach is Facebook Connect (Miculan and Urban, 2011). Identical to KAuth, if a user wants to access a third-party website using Facebook Connect, they follow the exact same procedure. Due to the massive integration of this feature by developers, this has led to many different implementations, each one of them offering different functionalities. What stays the same for all implementations is how Facebook validates user credentials however this is where KAuth tries to make a difference. Our system has a simple architecture as there are two big parts that end up working together. On one side, the OAuth server handles the token requests, waits for the Keybase login procedure to be completed, and then serves as a resource server, providing an interface to the API of Keybase.

OAuth Server The OAuth server is split into 3 controllers. The token controller, authorizer controller and resource controller. The token controller is responsible to generate an authorization code for the client website. This authorization code is sent to the authorizer controller by the Client website, with the Client ID and Client Secret, and is exchanged for an access token. This access token is then passed to the resource controller, in order to make API calls to Keybase and access the user's data. This architecture defined by the library we used to apply the OAuth protocol (Brent Shaffer, 2014).

Keybase Login System The Keybase Login System operates in two modes. The basic version where the user has to enter a username and a passphrase, and the + version where they have to enter username, password and also sign a message using their PGP key.

KAuth Login System This login system procedure starts by requesting a salt using the user's username or email. Then, the passphrase and the salt (unhexed) are entered as parameters in the scrypt function which generates a 256 byte stream. The last 32 bytes of this stream, are handled as a private key. This private key is used to sign a JSON blob whose structure is defined by Keybase. An EdDSA signature is generated and then packaged into a Keybase-style signature (Keybase, 2018). The result is sent to the Keybase server as the pdpka5 parameter.

KAuth+ Login System The KAuth+ login system differs from the Basic one as it executes an additional

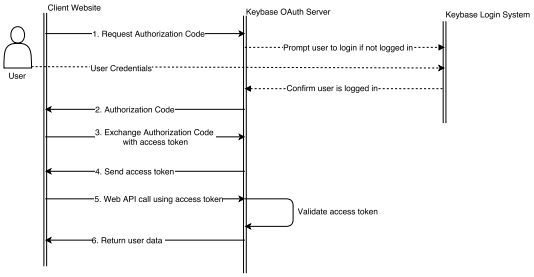

Figure 1: The flow a user follows to login at a third party website and they choose to do so using KAuth or KAuth+.

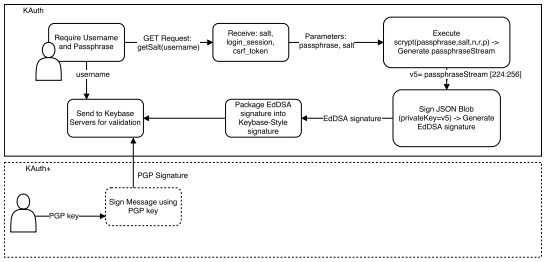

Figure 2: Login validation for KAuth and KAuth+.

action. It follows all steps of the basic login system, but also uses the user's private key to sign a message. This signed message is sent to the server, gets validated and completes the Login procedure, if successful.

3 Implementation

For the implementation of our system we have created a client website that offers the KAuth and KAuth+ service, an OAuth2 server offering the Authorization Code flow and a simple website that works as an interface for the user to connect to Keybase. The only difference between the two versions of our scheme, is the login procedure where in one case only the user's Keybase username and passphrase are required, while in the other version, the username/-passphrase as well as a private key are required. Both Client side and Server side (OAuth and Keybase) of the system are hosted on the Heroku Platform.

Client Website First we registered our client website on the OAuth server in order to determine which data the client website would ask permission for (referred to as scope). The OAuth server generated a Client ID and Client Secret required for all OAuth communication between the client website and the OAuth server. This was handled by the token controller. The client website was written in PHP and had two main pages. The main welcoming page and the login page. The login page offered the user the option to login using a username and password or to "Login with KAuth". Choosing to login with KAuth, the user was redirected to the Authorize controller of the OAuth server. This controller redirected the user to our login page of Keybase Website (if they hadn't already) and then asked for permission to access the data that was defined in the scope. If the user authorized the client website to access their data, the OAuth server sent an Authorization Code back to the client website in a GET request. Then the client website exchanged the Authorization code as well the Client ID and Client secret for an access token using POST. Optionally they could have requested a refresh token so that this procedure did not need to be repeated every time. This token would be used each time the client website wanted to use any of the user's Keybase data.

OAuth2 server For the implementation of the OAuth2 protocol, we used Brent Shaffer's open source OAuth2 PHP library (Brent Shaffer, 2014). This library offers all OAuth grant types although we used the Authorization code grant. The library supports many database schemas but we used PostgreSQL since this is the one that fit our needs best with Heroku.

Keybase Login procedure Regarding the login procedure of Keybase, we used an open source nodeJS module provided by Keybase (Keybase, 2018). This module handled everything during login for our basic version. For the extended version, there was an additional procedure where the user had to sign a message generated by Keybase, using their private key. Every time a user was redirected to our Keybase login page they were presented with a form requiring a username and passphrase. When they submitted the form, the nodeJS script was called and replicated the official Keybase login procedure as described on their API (Keybase, 2018). Recall that even though the login procedure regarding the username and passphrase, looked like a text based authentication scheme, it was actually much more complicated. As soon as the user entered their username and passphrase and the script was called, a two round login protocol (as called by Keybase) was started. The first round consisted of a GET request to the Keybase API requesting the user's salt which was sent along with a csrf token and a login session token. For the second round, the salt retrieved was used to generate a passphrase Stream using the scrypt function which worked as a Key derivation function. The user's

Table 1: Comparative evaluation of KAuth and KAuth+ and other similar password-replacement schemes.
●=Offers the benefit; ○=almost offers the benefit; no circle=does not offer the benefit
↑=better than passwords; ↓=worse than passwords; no arrow= no change

			Usability									Deployability						Security										
Category	Scheme	Reference	Memory-wise Effortless	Scalable-for-Users	Nothing-to-Carry	Physically-Effortless	Easy-to-Learn	Efficient-to-Use	Infrequent-Errors	Easy-Recover-From-Loss		Accessible	Negligible-Cost-per-User	Server-Compatible	Browser-Compatible	Mature	Non-Proprietary	Resilient-to-Physical-Observation	Resilient-to-Targeted-Impersonation	Resilient-to-Throttled-Guessing	Resilient-to-Unthrottled-Guessing	Resilient-to-Internal-Observation	Resilient-to-Leaks-from-Other-Verifiers	Resilient-to-Phishing	Resilient-to-Theft	No-Trusted-Third-Party	Requiring-Explicit-Consent	Unlinkable
	Web passwords				●		●	●	○	●		●	●	●	●	●	●		○						●	●	●	●
	KAuth		○	●	●	○	○	●	●	○		●	●		●		●	○	●	●	●		●			●	●	
			↑	↑		↑			↑	↓								↑	↑	↑	↑		↑			↑		
	KAuth+		○	●		○		●	○	○		●	●				●	●	●	●	●		●			●	●	
			↑	↑	↓	↑			↓	↓				↓				↑	↑	↑	↑		↑			↑	↑	↓
Federated	OpenID		○	●	●	○	○	●	●	●		●	●		●	●	●	○	○	○	○		●		●		●	
			↑	↑		↑	↓		↑					↓				↑		↑	↑		↑			↓		↓
	Facebook Connect		○	●	●	○	●	●	●	●		●	●		●	●		○	○	○	○		●		●			
			↑	↑		↑			↑					↓			↓	↑		↑	↑		↑			↓	↓	↓

passphrase was also used in this function. Scrypt returned a 256 byte output. The last 32 bytes of the output were interpreted as an EdDsa key. This key was used to sign a JSON blob of a certain form given in Keybase's API. Then the EdDSA signature was packaged as a Keybase-style signature also mentioned in the Keybase API. This was posted to the server as a pdpka5 parameter (pdpka stands for Passphrase-Derived Public Key Authentication). This completed the login request. If the server replied with a session cookie, this meant that the login was successful. For our KAuth+ version the same procedure was followed but after a successful username/passphrase procedure, the user was redirected to a wizard-like page which asked them to download and run a script, and then upload the signature that was generated. If the signature was valid and generated less than 1 minute ago, the login procedure was completed successfully.

4 Evaluation

In evaluating KAuth, we note that it looks like a typical username/password scheme but what happens on the client before contacting the server is what makes it different. The passphrase never leaves the user's device and all validation happens using Public Key Infrastructure with Elliptic Curves. We discuss the evaluation of the 2 systems (KAuth and KAuth+) separately. The evaluation is based on a state-of-the-art evaluation framework (Bonneau et al., 2012). This framework uses 25 properties split into 3 catergories, Usability, Deployability and Security. The framework is created for evaluation of other schemes compared to web passwords.

KAuth: Usability Evaluation The scheme is Quasi-Memorywise-Effortless as users have to remeber their Keybase passphrase. It is scalable-for-users since OAuth2 gives you the option to have access and refresh tokens stored for a certain amount of time, in order to skip the authorization process the next time you want to log in. The scheme can remember arbitrarily many passwords since the client must give permission to each client only once. It also satisfies the Nothing-to-Carry and Quasi-Physically-Effortless requirements as the user only needs to type the passphrase on Keybase's login page once per session. It is Quasi-Easy-to-Learn since the user must choose to login with Keybase and enter their Keybase credentials. A bad interface at the third party client could harden the process for a user. We rate our system Efficient-to-Use and Infrequent-Errors in that it is presented as a simple password authentication to the user or can occur semi-automatically if the user has been logged in with cached login cookies in Keybase (The user still needs to grant the application access to their information if it is the first time). Our system is Quasi-Easy-Recovery-from-Loss. If someone loses their passphrase they can recover their account if they have Keybase installed and logged in on any device. If they are not logged in Keybase in any device, they can still recover their account using a reset link though they will lose all their keys and data. Like OpenID and Facebook Connect, KAuth offers all Usability benefits at a satisfying level.

KAuth: Deployability Evaluation KAuth is Accessible and Negligible-Cost-per-User. Anyone who can

use passwords, is able to use KAuth. It not Server-Compatible since Keybase must offer OAuth2 services (Client registration/interaction). It is Browser-Compatible since a user can use KAuth on any device without having to install any plugins or other software. We rate the system as not Mature, since no implementation of such authorization has been deployed in large scale before. Finally, it is Non-Proprietary as Keybase and OAuth2 are open-source and free to use.
KAuth: Security Evaluation Our scheme is Quasi-Resilient-to-Physical-Observation as an attacker can target the infrequently typed passphrase. It is Resilient-to-Targeted-Impersonation as someone impersonating a user cannot get access to their account using personal information. Due to Keybase's strong authorization method, the scheme is Resilient-to-Throttled-Guessing and Resilient-to-Unthrottled-Guessing. It is not Resilient-to-Theft as a loss of a user's username and passphrase grants access to their account. In addition, KAuth is No-Trusted-Third-Party, Requiring-Explicit-Consent but not Unlinkable. The scheme is not Resilient-to-Internal-Observation as malware on the user's device can log their key presses and capture the input of the passphrase. It is Resilient-to-Leaks-from-Other-Verifiers but not Resilient-to-Phishing as it involves redirection to Keybase's login page. Regarding the extended version of our approach, there are differences mainly at usability and security benefits.

KAuth+: Usability Evaluation For the same reasons as KAuth the system is rated as offering the Quasi-Memory-Wise-Effortless, Scalable-for-Users and Efficient-to-Use benefits. It is rated Quasi-Infrequent-Errors as the procedure of signing a message to log in might look complex to some users. Due to the nature of private keys, the scheme does not satisfy the Nothing-to-Carry and Easy-to-Learn benefits as the user must carry a device that has their private key on to use for the second step of the login procedure. We rate it Quasi-Physically-Effortless as the user must choose their private key although this process can be automated using a default key location/name or by an automated procedure through the Keybase client. Regarding recovery from loss, we rate our system as Quasi-Easy-Recovery-from-Loss as users can still recover their account using a reset link though they will lose all their keys and data.
KAuth+: Deployability Evaluation From a Deployability standpoint, KAuth+ offers the same deployability benefits as KAuth with the exception of the Browser-Compatible benefit. It is not Browser-Compatible since the user has to install the Keybase client in order to handle their keys.

KAuth+: Security Evaluation The system is Resilient-to-Physical-Observation due to the signing process. It is also Resilient-to-Targeted-Impersonation, Resilient-to-Throttled and Unthrottled guessing, Resilient-to-Leaks-from-other-Verifiers, Phishing and Requiring-Explicit-Consent. It is not Resilient-to-Theft since the private key and the passphrase can be lost, and not Resilient-to-Internal-Observation as there is malware that can intercept the input of the passphrase as well as the (even encrypted) private key from the user's device. KAuth+ also offers the no-Trusted-Third-Party benefit. When it comes to security It is evident that KAuth+ offers stronger security with a marginal reduction of usability compared to KAuth. It might also offer stronger security than other password-replacing schemes. Its password-less nature defeats a lot of attacks and might make it a good alternative to authentication methods.

5 Related Work

Passwords Researchers have analyzed a corpus of 70 million passwords and have concluded that they provide little entropy, in particular 10 bits of security against an online, trawling attack, and only about 20 bits of security against an optimal offline dictionary attack (Bonneau, 2012). Additionally to little entropy, researchers have identified significant password reuse (Florencio and Herley, 2007; Das et al., 2014). Towards minimizing passwords, researchers have built a password-based authentication system on top of password reminders (Tzagarakis et al.,). KAuth does not rely on passwords, but on *passphrases*. The user needs to memorize a strong secret, but the secret is never transmitted in the network. Moreover, KAuth can use a private PGP key, which is not derived from a user secret to sign a second message for authenticating the user.

SSOs The large interest in SSO services has led to doubts about its security benefits. Various research papers have studied the security of such mechanisms (Cao et al., 2014) and have addressed vulnerabilities (Owano, 2014). Also, there are tools like SSOScan (Zhou and Evans, 2014) that scan websites with SSO integrations searching for vulnerabilities using the Facebook SSO APIs. SSOScan was run on 1600+ websites that used Facebook SSO and the results showed that more than 20% suffered from at least one of the five major SSO vulnerabilities. KAuth works in a similar way but offers a unique mechanism that strengthens the user's account security.

6 Conclusion

In this paper we presented an SSO service based on PKI that builds on Keybase. KAuth validates a user using their Keybase username and passphrase and KAuth+ validates using the user's username, passphrase but also their PGP key. The results show that our approaches offer similar benefits regarding usability with OpenID and Facebook Connect, nevertheless, we are better in terms of security. KAuth and especially KAuth+ achieve better scores than most similar approaches, and a significant improvement on security compared to web passwords.

REFERENCES

Ars Technica (2013). Twitter detects and shuts down password data hack in progress. http://arstechnica.com/security/2013/02/twitter-detects-and-shuts-down-password-data-hack-in-progress/.

Bonneau, J. (2012). The science of guessing: Analyzing an anonymized corpus of 70 million passwords. In *SP, 2012*.

Bonneau, J., Herley, C., Oorschot, P. C. v., and Stajano, F. (2012). The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. SP '12.

Brent Shaffer (2014). OAuth2 Server Library for PHP. https://bshaffer.github.io/oauth2-server-php-docs.

Cao, Y., Shoshitaishvili, Y., Borgolte, K., Kruegel, C., Vigna, G., and Chen, Y. (2014). Protecting web-based single sign-on protocols against relying party impersonation attacks through a dedicated bi-directional authenticated secure channel. In *RAID 2014*.

Das, A., Bonneau, J., Caesar, M., Borisov, N., and Wang, X. (2014). The tangled web of password reuse. In *NDSS, 2014*.

Dhamija, R., Tygar, J., and Hearst, M. (2006). Why phishing works. In *CHI '06*.

Dierks, T. (2008). The transport layer security (tls) protocol version 1.2.

Everspaugh, A., Chatterjee, R., Scott, S., Juels, A., and Ristenpart, T. (2015). The pythia prf service. In *USENIX Security Symposium*.

Florencio, D. and Herley, C. (2007). A large-scale study of web password habits. WWW '07.

Google (2018). Google+ Sign-in. https://developers.google.com/+/web/signin/.

Juels, A. and Rivest, R. L. (2013). Honeywords: Making password-cracking detectable. CCS '13.

Keybase (2018). Keybase. https://keybase.io.

Kontaxis, G., Athanasopoulos, E., Portokalidis, G., and Keromytis, A. D. (2013). Sauth: Protecting user accounts from password database leaks. CCS '13.

Lai, R. W. F., Egger, C., Schröder, D., and Chow, S. S. M. (2017). Phoenix: Rebirth of a cryptographic password-hardening service. In *USENIX Sec 2017*.

Mail, D. (2015). Do YOU suffer from password rage? A third of people have thrown a tantrum after forgetting login details. http://www.dailymail.co.uk/sciencetech/article-3115754/Do-suffer-password-rage-people-thrown-tantrum-forgetting-login-details.html.

Miculan, M. and Urban, C. (2011). Formal analysis of facebook connect single sign-on authentication protocol. In *SOFSEM 2011*.

Owano, N. (2014). Math student detects OAuth, OpenID security vulnerability. https://techxplore.com/news/2014-05-math-student-oauth-openid-vulnerability.html.

PayPal (2013). PayPal Leads Industry Effort to Move Beyond Passwords. https://www.thepaypalblog.com/2013/02/paypal-leads-industry-effort-to-move-beyond-passwords/.

Schneider, J., Fleischhacker, N., Schröder, D., and Backes, M. (2016). Efficient cryptographic password hardening services from partially oblivious commitments. CCS '16.

Twitter (2018). Sign in with Twitter. https://dev.twitter.com/docs/auth/sign-twitter.

Tzagarakis, G., Papadopoulos, P., Chariton, A. A., Athanasopoulos, E., and Markatos, E. P. Øpass: Zero-storage password management based on password reminders. In *EuroSec 2018*.

Wired.com (2013). Google Declares War on the Password. http://www.wired.com/wiredenterprise/2013/01/google-password/all/.

Zhou, Y. and Evans, D. (2014). Ssoscan: Automated testing of web applications for single sign-on vulnerabilities. In *USENIX Security 14*.