Thesis Dissertation

KAUTH: A STRONG SINGLE-SIGN ON SERVICE BASED ON PKI.

Panayiotis Charalambous

UNIVERSITY OF CYPRUS

COMPUTER SCIENCE DEPARTMENT

May 2018

UNIVERSITY OF CYPRUS
COMPUTER SCIENCE DEPARTMENT

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

Panayiotis Charalambous

Supervisor
Dr. Elias Athanasopoulos

Thesis submitted in partial fulfilment of the requirements for the award of degree of Bachelor in Computer Science at University of Cyprus

May 2018

Acknowledgments

First and foremost, I have to thank my research supervisor, Dr. Elias Athanasopoulos. Without his assistance and dedicated involvement in every step throughout the process, this thesis project would never have been accomplished. I would like to thank you very much for your support and understanding over this past year.

Getting through my thesis project required more than academic support, and I have many friends to thank for their support. You made all those endless nights in labs, suprisingly a very pleasant time.

Most importantly, none of this could have happened without my family. To my parents and my sister - there is not a single moment that you weren't by my side providing absolutely everything you could. This project stands as a testament to your unconditional love and encouragement.

Summary

Passwords are the dominant form of human-to-machine authentication despite all the problems that are commonly associated with them. For users, remembering a simple secret is much more convenient than other forms of authentication that involve complex protocols. In this thesis project, we attempt to 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.

Results of this thesis project

The results of this thesis project will be published in the proceedings of the SECRYPT conference that will be held in Porto, Portugal in 26-28 July 2018

Contents

Page

1 Introduction **1**

2 Background **4**

2.1 Public key cryptography 4

2.2 Keybase 5

2.3 OAuth 6

3 Architecture **7**

3.1 Traditional Approach: Facebook Connect 7

3.2 KAuth 7

3.2.1 OAuth Server 8

3.2.2 Keybase Login System 8

4 Implementation **12**

4.1 Heroku 13

4.2 Client Website 13

4.3 OAuth2 server 15

4.4 KAuth Login procedure 16

4.5 KAuth+ Login procedure / Wizard 17

4.6 Technologies and tools used 19

4.6.1 Javascript 19

4.6.2 Cascading Style Sheet (CSS) 19

4.6.3 HyperText Markup Language (HTML) 19

4.6.4 Sublime Text 19

4.7 Database Implementation 21

5 Evaluation **24**

5.1 Usability Evaluation 25

5.1.1 KAuth: Usability Evaluation 25

5.1.2 KAuth+: Usability Evaluation . . . 26
5.2 Deployability Evaluation . . . 27
5.2.1 KAuth: Deployability Evaluation . . . 27
5.2.2 KAuth+ Deployability Evaluation . . . 28
5.3 Security Evaluation . . . 29
5.3.1 KAuth: Security Evaluation . . . 29
5.3.2 KAuth+: Security Evaluation . . . 30

6 Discussion and Future Work **31**
6.1 Attacks on Keybase . . . 31
6.2 Incorporating KAuth with the official Keybase . . . 32
6.3 Extended Usability Studies . . . 32

7 Related Work **33**
7.1 Passwords . . . 33
7.2 Single sign-on services . . . 34

8 Conclusion **35**

Bibliography **36**

Chapter 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 [37, 44]. For web browsing especially, users rely on passwords for using the continuously growing Internet services, while experts are still debating on the right password policies [39]. This has significant implications, since passwords are reused [16], are leaked due to services' vulnerabilities [1, 10, 18, 26, 33] not user mistakes [21, 24], are phished [12, 13], and the overall user experience is severely degraded [27, 40]. We stress here that, as far as phishing is concerned, the attack is getting more serious [17] overcoming even several forms of enhanced authentication, such as two-factor authentication (2FA), which is commonly believed to secure passwords. As far as database leaks are concerned, according to Facebook they happen on a *weekly* basis [32]. It is more than clear, that the community needs to actively seek a radical change in the way humans authenticate to services.

Cryptology has built several tools for building strong authentication. Unfortunately, such techniques have been used so far for machine-to-machine authentication [14] or ad hoc for password hardening [15,25,38], 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, altough 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 thesis project, 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 [22], 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) [36] service which can vet users for using other services, exactly as is happening now with very popular, but entirely password-based, services [19, 30, 41]. Our proposed system, KAuth, uses PKI to authenticate 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 greatly 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 project, 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. This thesis project contributes the following.

1. We design and implement KAuth and KAuth+, two systems which provide strong PKI human-to-machine authentication.

2. We evaluate KAuth and KAuth+ with an established framework [5] and show our systems 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 when using KAuth.

Organization. The rest of the thesis is organized as follows. In Section 2 we discuss the basic components which our system uses, such as the OAuth2 protocol (for the SSO part) and Keybase. In Section 3 we discuss the architecture of KAuth and KAuth+ and we provide the technical details in Section 4. We discuss our future steps in Section 6 and we evaluate KAuth in Section 5 using a state-of-the-art methodology [5]. Related work is discussed in Section 7 and we conclude in Section 8.

Chapter 2

Background

Contents

2.1 Public key cryptography **4**

2.2 Keybase **5**

2.3 OAuth **6**

2.1 Public key cryptography

Public key cryptography is an encryption technique that ensures that two sides that want to communicate will not have a 3rd person listening to the conversation, tampering with it or impersonating one of the two parties. This is all possible by using pairs of keys. Each side has their private key which is kept for themselves and a public key that they share with whoever they want to communicate with. To communicate with each other, the senders must encrypt their message using the recipient's public key. Each recipient can decrypt the message using his private key. Typically asymmetric ciphers are suitable for encrypting and decrypting short messages; for longer messages a symmetric cipher can be involved.

Public key cryptography is implemented by a variety of internet standards such as TLS, PGP, GPG, SSL and HTTP. Public key encryption also offers proof as to who wrote a message. If someone wants to send a message that will ensure the recipient that they are the one who wrote it, they can encrypt the message –in practice, the message's cryptographic digest– with their private key. Then the recipient can decrypt the message using the sender's public key, to verify the signature. One issue that public key cryptography poses is key distribution. Searching for keys but also making sure that the key you have found belongs to the person it says can be a hard task. Sharing your public key on your social media or appending it on all your emails might sound a solution but it can be dan-

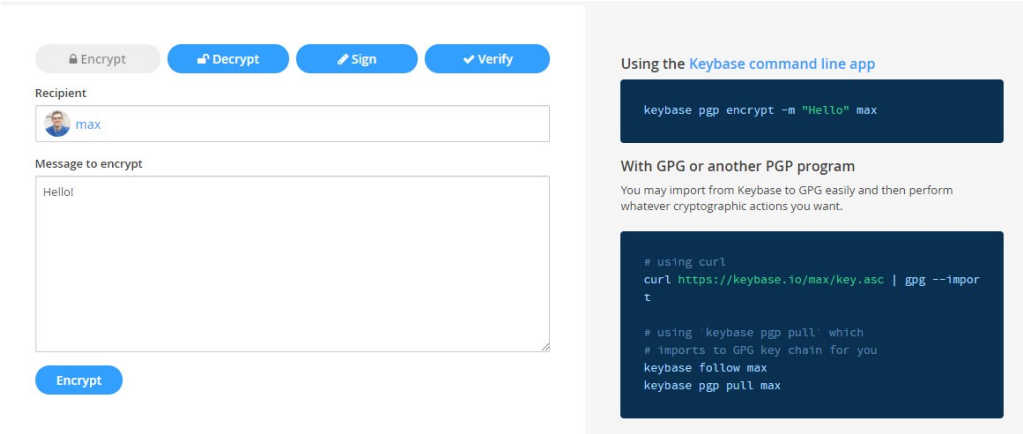

Figure 2.1: Users can do crypto operations such as encryption/decryption and sign/verify using Keybase's website, their client, or their command line tool. The first method requires that the user has their private key encrypted and uploaded on Keybase's servers. If you want to keep your private key fully protected, you have to use the command line tool or the local client.

---

gerous should one (or all) of your accounts get compromised and the public key replaced with a different one.

2.2 Keybase

Keybase's idea is that all your social media networks combined, present your public identity. Therefore, Keybase lets you tie your Keybase account to any other social media account. In addition it enables you to distribute your public key in a way that can be verified that the user making a statement, is the one that holds the Keybase account, the social media accounts and an exact public key. All of this happens automatically using the Keybase client offering centralized management with decentralized trust. Keybase also solves the problem with PGP keys which must be distributed amongst all your devices, using per-device keys. These keys that never leave the device, verify each other.

Keybase now offers an encrypted end-to-end chat system and an encrypted file-system. As seen on figure 2.1, users can do crypto operations such as encryption/decryption and sign/verify using Keybase's website, their client, or their command line tool. The first method requires that the user has their private key encrypted and uploaded on Keybase's servers. If you want to keep your private key fully protected, you have to use the command line tool or the local client. In addition, Keybase also let's you use your keys on GPG or any other PGP program.

2.3 OAuth

OAuth is an authorization framework that assigns the authentication of the user to the service that the user has signed up, authorizing 3rd party applications to access the user's data. There are several authorization flows for web/desktop applications and mobile devices, but we will only be discussing the Authorization code grant.

There are 4 roles in the OAuth framework: The resource owner is the user who authorizes an application to access an account. There is the Resource server that holds the user's credentials, the Authorization server that verifies the user and the Client which is the 3rd party application that wants to access the user's account. The Client receives a Client ID and Client Secret from the service's API (Resource/Authorization server) that he uses to communicate. [31] When a user wants to access the client's services, the user gets a code from the Resource server that passes to the client. Then the client exchanges the code for an access token. This token is then used to make API calls to the resource center, retrieving any useful information which is needed about the user.

Chapter 3

Architecture

Contents

3.1 Traditional Approach: Facebook Connect 7

3.2 KAuth 7

3.2.1 OAuth Server 8

3.2.2 Keybase Login System 8

3.1 Traditional Approach: Facebook Connect

One of the most popular and widely adopted mechanisms that is really close to our approach is Facebook Connect [30]. Identical to KAuth, where if a user wants to access a third-party website using Facebook Connect, they follow the exact same procedure, including a request for token, authorization and username/password validation. Due to the massive community on Facebook, many developers integrate their websites with Facebook Connect. 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.

3.2 KAuth

Our system has a simple architecture as there are two big parts that end up working together. On the 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.

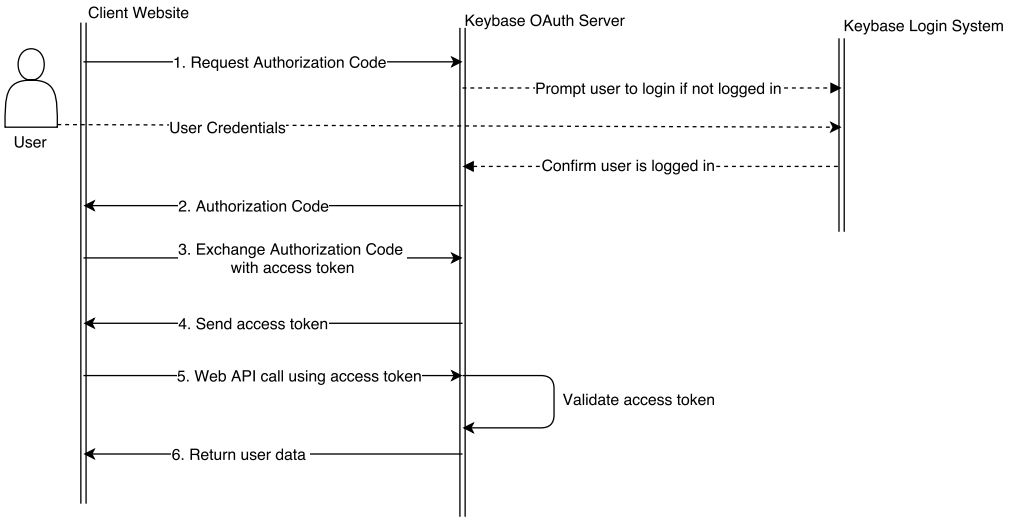

Figure 3.1: When a user wants to login at a third party website and they choose to do so using KAuth or KAuth+, this is the flow that they have to follow. On one side, the OAuth server handles the token requests, waits for the Keybase login procedure to be completed, and then serves as a resource server, providing an interface to the API of Keybase.

3.2.1 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 is defined by the library we used to apply the OAuth protocol [7].

3.2.2 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, passphrase and also sign a message using their PGP private key.

Figure 3.2: Example of the getSalt API call by Keybase official API page.

KAuth Login System This login system procedure starts by requesting a salt using the user's username. This is done through the official Keybase API and an example of that call can be shown in Figure 3.2. 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 (as shown in Figure 3.3). An EdDSA signature is generated and then packaged into a Keybase-style signature [9]. The result is sent to the Keybase server as the pdpka5 parameter. An example of a Keybase-style signature is shown in Figure 3.5

KAuth+ Login System The KAuth+ login system differs from the Basic one as it executes an additional action. It follows all steps of the basic login system, but also uses the user's private key to sign a message. This is done by requiring the user to download and run a script that will generate the signed message and then upload the signed message. This signed message is generated using the user's private PGP key in their device to sign the message. If the message is verified to be produced by the same person that is requesting the login, access is granted.

```
{
  "body": {
    "auth": {
      "nonce": "ab68b24b6bcff3dc6e0cdc558e3e043c",
      "session": "lgG5dGhlbWF4KzhkY2FhNjc4QGdtYWlsLmNvbc5YBShhzQlgwMQgtABibipP7sQIpLv/
hO+akJ5mdrD64QkuhY08VdLwtW0="
    },
    "key": {
      "host": "keybase.io",
      "kid": "0120fffa77faf7c189edbb82a942c5feef831335ced44e2fd3155673b023314719070a",
      "username": "u5c7d0817"
    },
    "type": "auth",
    "version": 1
  },
  "ctime": 1476733025,
  "expire_in": 157680000,
  "tag": "signature"
}
```


Figure 3.3: The client should sign a JSON blob of this form.

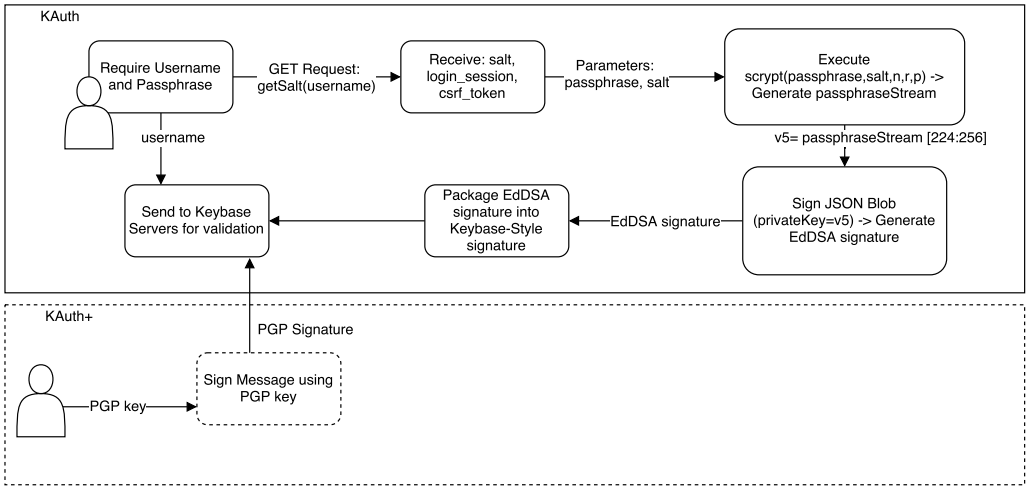

Figure 3.4: To validate a login, KAuth requests a salt using the user's username. The entered passphrase and the salt (unhexed) are entered as parameters in the scrypt function. Part of the result of scrypt is handled as a private key. This private key is used to sign a JSON blob (as shown in Figure 3.3). An EdDSA signature is generated and then packaged into a Keybase-style signature. The result is sent to the Keybase server as the pdpka5 parameter. The KAuth+ login system differs from the basic one as it executes an additional action. It follows all steps of the basic login system, and then requires the user to run a script on their system that will generate a signed message. This signed message is uploaded and verified by Keybase. If all steps succeed, the user is then logged in to Keybase.

Packet Layout

The interior of the packet is a JSON-style object with boolean, integer, and binary fields:

```
{
  "body": {
    "detached": true,
    "hash_type": 10,
    "key": <Buffer 01 20 ... 0a>,
    "payload": <Buffer ... >,
    "sig": <Buffer ... >,
    "sig_type": 32 },
  "tag": 514,
  "version": 1
}
```

The packet is encoded using canonicalized msgpack and then standard Base-64.

Figure 3.5: The structure of a Keybase-style signature as presented by the official Keybase docs file.

Chapter 4

Implementation

Contents

4.1 Heroku 13

4.2 Client Website 13

4.3 OAuth2 server 15

4.4 KAuth Login procedure 16

4.5 KAuth+ Login procedure / Wizard 17

4.6 Technologies and tools used 19

4.6.1 Javascript 19

4.6.2 Cascading Style Sheet (CSS) 19

4.6.3 HyperText Markup Language (HTML) 19

4.6.4 Sublime Text 19

4.7 Database Implementation 21

For the implementation of our system we have created a client website that requests access to users' data from Keybase, 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.

4.1 Heroku

Heroku is the cloud platform that was used to deploy our project. Heroku lets you deploy and run applications supporting many modern languages and frameworks. Heroku apps run on dynos used to run web. There are web dynos that execute web processes, worker dynos that handle background jobs and one-off dynos that are temporary dynos. Heroku's free plan offers 1 web dyno and 1 worker dyno for each app which satisfied our needs.

We created two apps, one for the client website and the other one for Keybase which was consisted of the OAuth server, and the login functionality. Apps in heroku choose a buildpack which basically directs Heroku how to handle the deployed code. For the client website we used the PHP buildpack and for the Keybase server app we used both PHP and NodeJS buildpacks. Our code was deplyed by using Git. Heroku enables their users to deploy their code using Git by associating a Git repository during the app creation.

4.2 Client Website

First we created our client website which went by the name of pchara20-client in Heroku. We registered the web application on the OAuth server as a client, in order to set which data the client website would ask permission for (referred to as scope). The client website received a Client ID and Client Secret that is required for all OAuth communication between the client website and the OAuth server. This was simply done with a mySQL call to our database.

The client website was written in PHP and had two main pages. The main welcoming page and the login page. The login page as shown in Figure 4.1 offered the user the option to login using a username and password or to "Login with Keybase". Choosing to login with Keybase, the user was redirected to the Authorize controller of the OAuth server. This controller redirected the user to our login page of Keybase Website. Depending on the user's account, they were redirected to the KAuth+ wizard if they had KAuth+ enabled. After user validation, they were asked for access permission the data that was defined in the scope as shown in Figure 4.9. 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.

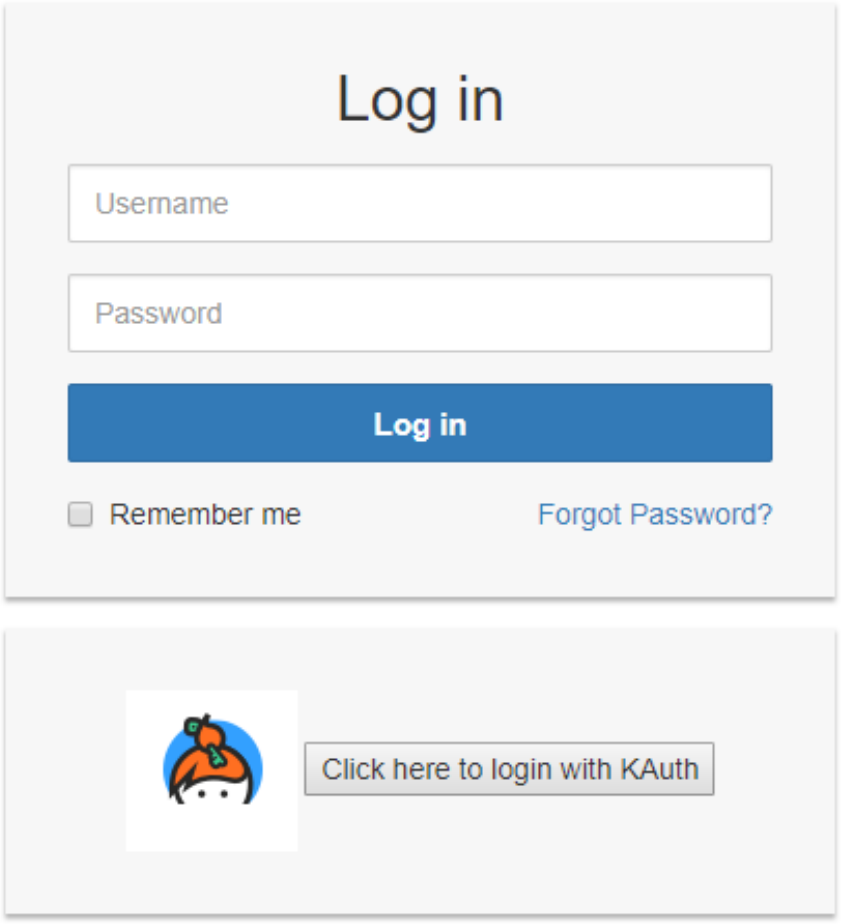

Figure 4.1: The login page offers the user the option to login using a username and password or to "Login with Keybase". Choosing to login with Keybase, the user is redirected to the Authorize controller of the OAuth server. This controller redirects the user to the login page of Keybase Website.

```
passphraseStream = scrypt(passphrase, unhex(salt), N=2^15, r=8, p=1,
dkLen=256)
v4 = passphraseStream[192:224]
v5 = passphraseStream[224:256]
```

Figure 4.2: The user's passphrase and the salt (unhexed) retrieved are entered as parameters in the scrypt function which generates a 256 byte passphrase 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.

4.3 OAuth2 server

For the implementation of the OAuth2 protocol, we used Brent Shaffer's open source OAuth2 PHP library [7]. This library offers all OAuth grant types although we used the Authorization code grant. The library supports many database schemas. We used PostgreSQL since this is the one that fit our needs best with Heroku.

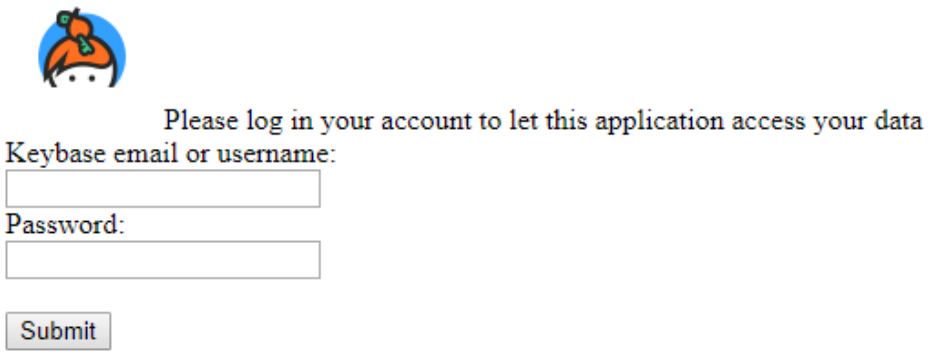

Figure 4.3: Every time a user was redirected to our Keybase login page they were presented with a form requiring a username and passphrase as shown in Figure 4.3. When they submitted the form, the nodeJS script was called and replicated the official Keybase login procedure as described on their API call.

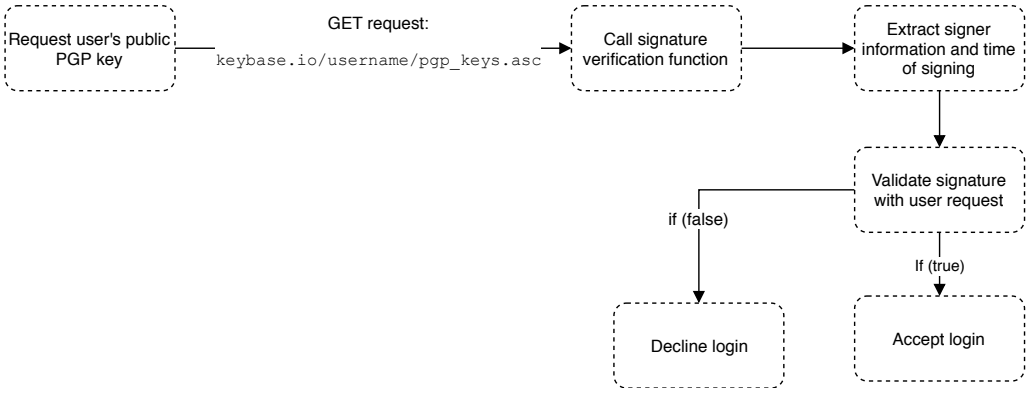

Figure 4.4: For the KAuth+ procedure, we use the uploaded file to verify the users signature. We make a GET request to Keybase's API to get their public keys and then using a PGP Javascript library, we verify the user's identity or reject the login request.

4.4 KAuth Login procedure

Regarding the login procedure of Keybase, we used an open source nodeJS module provided by Keybase [29]. This module handled everything during login for our basic version. Every time a user was redirected to our Keybase login page they were presented with a form requiring a username and passphrase as shown in Figure 4.3. When they submitted the form, the nodeJS script was called and replicated the official Keybase login procedure as described on their API [9]. 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 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) and completed the login request. If the server replied with a session cookie, this meant that the login was successful.

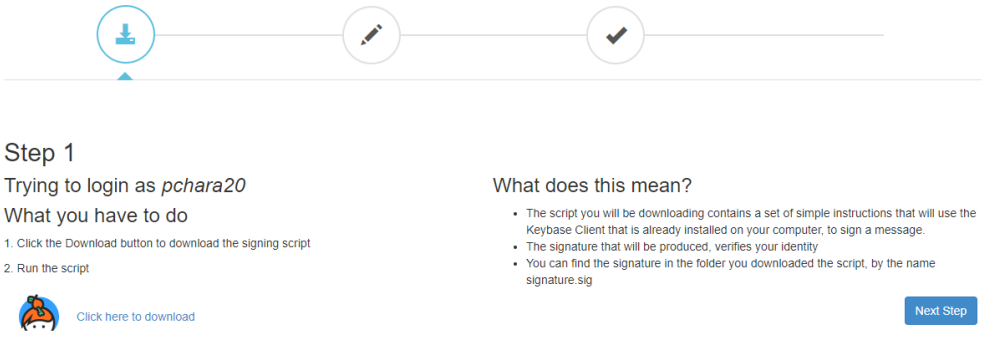

Figure 4.5: For our KAuth+ version, after getting through the basic login procedure, the user was redirected to a wizard-like interface with 3 simple steps. The first step required the user to download and execute a script written in Bash. This script uses Keybase's command line utility (that is required for this to operate) to sign a message. The script automatically generated a file named Signature.sig.

4.5 KAuth+ Login procedure / Wizard

For our KAuth+ version, after getting through the basic login procedure, the user was redirected to a wizard-like interface with 3 simple steps.

The first step required the user to download and execute a script written in Bash. This script uses Keybase's command line utility (that is required for this to operate) to sign a message. The script automatically generated a file named Signature.sig. An example of the output of the signing script is shown in Figure 4.10. The first step of the wizard is shown in Figure 4.5.

In step 2 of the wizard which is shown in Figure 4.6, the user is asked to upload the file generated by the script. When running the signing script, the user can see what is executed to create the signature. An example is shown in Figure 4.11. Then, a GET request is sent to Keybase's API retrieving the user's public key. Then, using Keybase's PGP implementation for Javascript [28] we verify the uploaded file's signature with the public key.

The verification is successful if the signature was produced by the person trying to login from the first login procedure (with username and passphrase) and if the signature was produced earlier than 1 minute before the login attempt. If this was successful, an informative message was displayed to the user and a button that would complete this wizard, logging the user on Keybase. An example of a succesful message signing and login is shown in Figure 4.8. If not, the user is required to repeat the process (or the wrong steps) after being prompted by an error message shown in Figure 4.7. The next

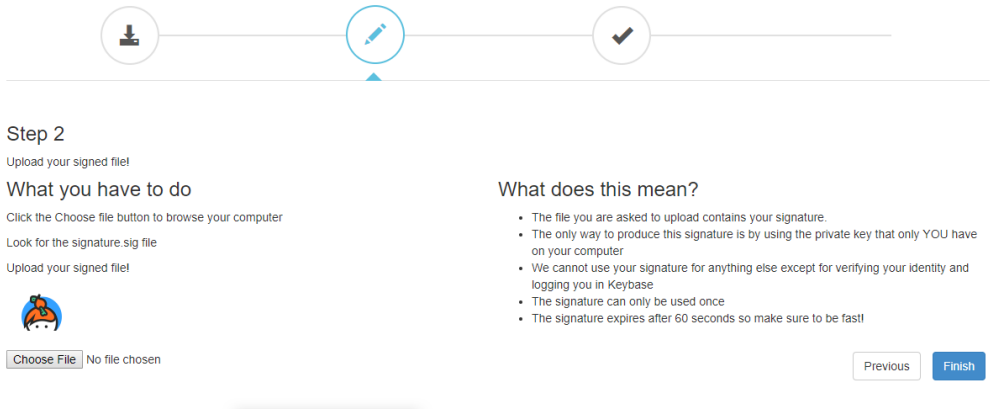

Figure 4.6: In step 2 of the wizard, the user is asked to upload the file generated by the script. Then, a GET request is sent to Keybase's API retrieving the user's public key. Then, using Keybase's PGP implementation for Javascript [28] we verify the uploaded file's signature with the public key. The verification is successful is the signature was produced by the person trying to login from the first login procedure (with username and passphrase) and if the signature was produced earlier than 1 minute before the login attempt

---

step of the validation has to do with permissions given to the client website as shown in Figure 4.9.

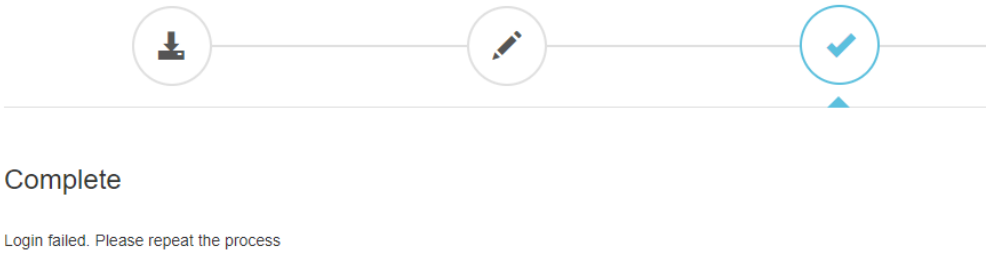

Figure 4.7: If any of the steps of the wizard process are not executed correctly, the appropriate error message is displayed on the screen

4.6 Technologies and tools used

4.6.1 Javascript

JavaScript is a dynamic programming language, which can be embedded in HTML pages, enabling interactive web pages and thus is an essential part in the development of web applications. The vast majority of web applications use it and all major web browsers have a dedicated JavaScript engine to execute it. Our whole system depends on the execution of JavaScript functions not only to provide a nicer user interface but also for implementing key functionalities in the visualization of our gaze plots.

4.6.2 Cascading Style Sheet (CSS)

Cascading Style Sheet is a style sheet language used for describing the presentation of a document written in in a markup language. It is commonly used for styling the user interfaces in HTML. CSS adjusts different styles and methods in the same page and can display or resize the screen depending on the device. In our system CSS complements the JavaScript.

4.6.3 HyperText Markup Language (HTML)

HyperText Markup Language (HTML) is the basic markup language used to create the user interfaces of a web application. It provides a means to design composition documents by structural semantics.

4.6.4 Sublime Text

Sublime Text is the main editor used to write our code. It is a proprietary cross-platform source code editor. It natively supports many programming languages and markup lan-

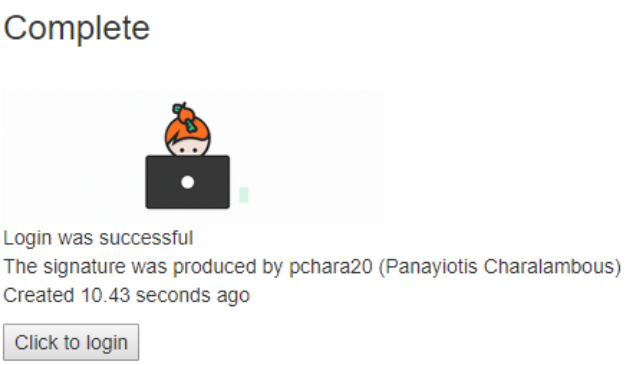

Figure 4.8: The verification is successful is the signature was produced by the person trying to login from the first login procedure (with username and passphrase) and if the signature was produced earlier than 1 minute before the login attempt. If this was successful, an informative message was displayed to the user and a button that would complete this wizard, logging the user on Keybase.

---

guages, and functions can be added easily with the use of plugins.

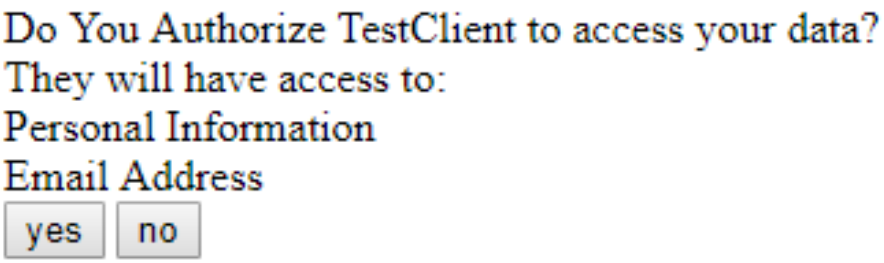

Figure 4.9: After the step of the validation, the user is informed of the scope of the client website, meaning the data that they will access through Keybase's API.

4.7 Database Implementation

For our OAuth2 server we used the schema that was defined by the OAuth2 PHP server library we used. The oauth_clients table is used to keep track of the clients that use the KAuth feature. The oauth_access_tokens table is used to store all access tokens generated for users. Oauth_authorization_codes is used to store the generate authorization codes that are generated and used for the exchange of the access token. Oauth_refresh_tokens table is required to eable the refreshing of the tokens so the OAuth server doesn't have to generate new access tokens for users after a short period of time. The oauth_scopes table is used to define the different scopes offered to the client applications. An EER diagram of the database used in the OAuth2 server can be seen at Figure 4.12

```
-----BEGIN PGP MESSAGE-----
Version: Keybase Go 1.0.48 (windows)
Comment: https://keybase.io/download

xA0DAAoBH/8gr3jcGqcBy+F0AOIAAAAAAMLBXAQAAQoAEAUCWwVpWAkQH/8gr3jc
GqcAAEPWEACK198IgtXXPZw4/XZh8vulIcYCNMLgWeSQFEKiFLljuh03eM0ClLnG
+VETuM/whJdcGM6niSysUlfuOOD2H6QHCGXRY5FS8ycx87zfn0V6BUh5yp3llnF0
P0cZMPgNpsjR5aM8BxwPQZBa3MZKP7qeXBUYzcWZlQ9neTakbEclP+c0QV/YsGfN
XerDjPihI8dmR62f6q4f5LGqRUUsFIMDlNoyVk2pGPA7d2VOyq6sbGc6xRrNrila
DjzJ3RapaNoBWEVTtUiohZiRS4VY/QxP7QmVyNZRYorhTfqwfLGpf/gIw02sUBYX
vn3pWeMEJYMEm3hVnpdTondfUTYHcffS8MGSPt83NUPFkFQZiRJj4ER9SXgedodL
/LKVla0Gh8NJ3GkRjfW87BHfCQnZJeJekOskssxNwfeYmvWMZ85OKjU6l2lMdAxR
drdudBjIAUoEq5SYPfiQJh+5bGOJwze4PzfnI8kTl/k+vKW5en+n9sfR2BvoWP5j
DlldKiUJC5golWTJo2VbYhTuxOSgIDaWnzPJs47plNQJCpvQy6tSwxll9wagMvqE
QgeJg3VK3/DzHoZaMUnIrwzcpEUdORYElPKM3MPE8GEsL79kGiHFQYdvQ7CqIVTa
j/3WBQRc3rl9hJzSw8/fd0z4gleQr47zblOvfHXlZk7q9bw8/jxT7A==
=aE/C
-----END PGP MESSAGE-----
```

Figure 4.10: An example of the signature generated after signing a message.

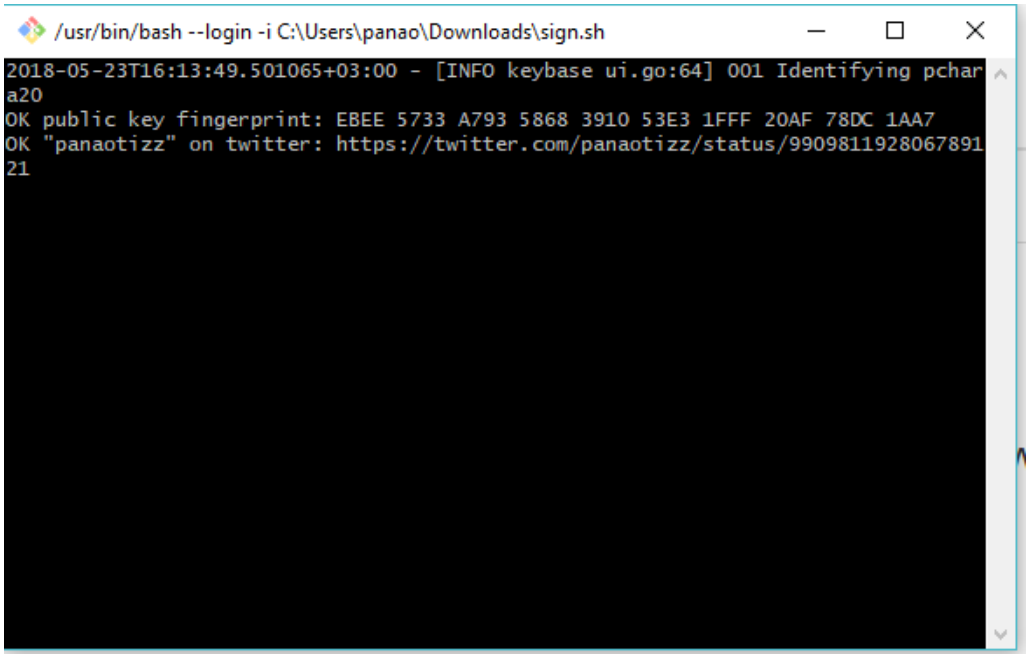

Figure 4.11: The script used to sign a message for the user and prove their identity.

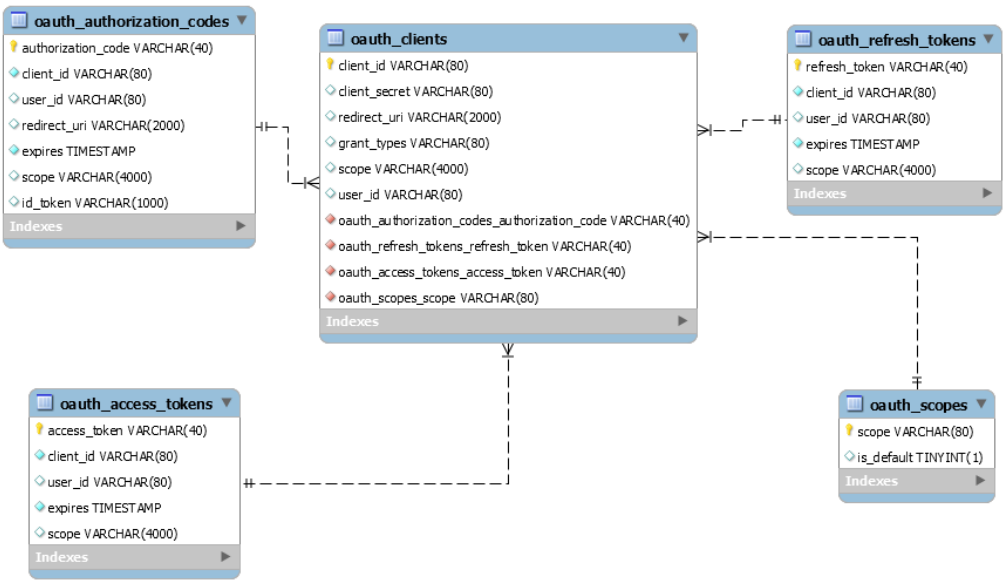

Figure 4.12: The database schema used for the OAuth2 server was instructed by the library we used.

Chapter 5

Evaluation

Contents

5.1 Usability Evaluation 25

5.1.1 KAuth: Usability Evaluation 25

5.1.2 KAuth+: Usability Evaluation 26

5.2 Deployability Evaluation 27

5.2.1 KAuth: Deployability Evaluation 27

5.2.2 KAuth+ Deployability Evaluation 28

5.3 Security Evaluation 29

5.3.1 KAuth: Security Evaluation 29

5.3.2 KAuth+: Security Evaluation 30

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 as they show some differences.

The evaluation is based on the Usability-Deployability-Security evaluation framework that was used to rate other authentication schemes [5]. This framework uses 25 properties split into 3 catergories, Usability, Deployability and Security. These properties are framed as benefits, and systems that satisfiy a property are rated as "offering the benefit". The framework is primarily created for evaluation of other schemes compared to web passwords. That is why each property is rated as "better than passwords", "worse than passwords" or provides "no change". Benefits that can't be applied on a scheme are considered as "offered" instead of "not-applicable", since nothing can go wrong.

Table 5.1: Comparative evaluation of KAuth and KAuth+ and other similar password-replacement schemes regarding usability.
●=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							
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
	Web passwords				●		●	●	○	●
	KAuth		○ ↑	● ↑	●	○ ↑	○ ↑	● ↑	● ↑	○ ↓
	KAuth+		○ ↑	● ↑	↓	↓	○ ↓	○ ↓	● ↓	↓
Federated	OpenID		○ ↑	● ↑	●	○ ↑	○ ↓	●	● ↑	●
	Facebook Connect		○ ↑	● ↑	●	○ ↑	●	●	● ↑	●

5.1 Usability Evaluation

5.1.1 KAuth: Usability Evaluation

The scheme is quasi-memorywise-effortless as users only have to remember their Keybase passphrase. It is scalable-for-users since OAth2 gives you the option to have access and refresh tokens stored, in order to skip the authorization process the next time you want to log in. Also, using the scheme for an arbitrary number of accounts, does not affect the scheme nor increase the burden on the user. It also satisfies the Nothing-to-Carry benefit since they do not have to carry any devices with them in order to use the scheme. We rate KAuth Quasi-Physically-Effortless 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. We suggest that Keybase focuses on providing a user-friendly interface that will satisfy our requirements for rating the system as "Easy to learn and use". 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.

5.1.2 KAuth+: Usability Evaluation

The system is quasi-memory-wise effortless due to the passphrase aswell. It is scalable-for-users since it follows the OAuth2 protocol just like KAuth. It does not offer the "Nothing to carry" benefit as the user must have a device that will include their private signing keys and also have Keybase installed to performed the signing operation. So far, Keybase's app does not offer command line utilities therefore it does not satisfy the benefit. Also, the procedure of signing the message, sets the scheme as not "Physically effortless", and quasi-Easy-to-Learn as the user has to carefully follow the steps to complete the signing process. We rate it as quasi-efficient-to-use as the authentication time exceeds the normal, acceptable time most schemes use. Also, it is rated as offering the infrequent-errors benefit as the procedure of signing a message to log in can't fail if the user executed the authenticating task correctly. Regarding recovery from loss, we rate our system as not offering the benefit since users will have to reset their account and lose all data in case they forget their credentials and lose their keys.

Table 5.2: Comparative evaluation of KAuth and KAuth+ and other similar password-replacement schemes regarding deployability.
●=Offers the benefit; ○=almost offers the benefit; no circle=does not offer the benefit
↑=better than passwords; ↓=worse than passwords; no arrow= no change

			Deployability					
Category	Scheme	Reference	Accessible	Negligible-Cost-per-User	Server-Compatible	Browser-Compatible	Mature	Non-Proprietary
	Web passwords		●	●	●	●	●	●
	KAuth		●	● ↑	↓	●	↓	●
	KAuth+		● ↓	● ↑	↓	↓	↓	●
Federated	OpenID		●	●	↓	●	●	●
	Facebook Connect		●	●	↓	●	●	↓

5.2 Deployability Evaluation

5.2.1 KAuth: Deployability Evaluation

Regarding deployability evaluation KAuth is rated as accessible as anyone who can use passwords, can use KAuth. It is negligible-per-user-cost since the procedure that generates the pdpka5 parameter is executed locally, an increase in the number of users will not affect the scheme's performance. 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 and also Keybase is still in an early stage and not yet widely adopted. Finally, it is non-proprietary as Keybase and OAuth2 are open-source and free to use.

5.2.2 KAuth+ Deployability Evaluation

From a Deployability standpoint, KAuth+ offers half of the benefits. It is accessible since a person who can use passwords, can follow the signing wizard. We rate KAuth+ as offering the negligible-cost-per-user and non-proprietary benefits as Keybase and OAth are open-source and free to use. The scheme is not server-compatible as Keybase has to alter their authentication process to support the signing of the message. Also, it is not browser-compatible since the user has to install the Keybase client or command-line tool in order to handle their keys. We rate the system as not mature as Keybase is a relatively new platform that might suffer from vulnerabilities.

Table 5.3: Comparative evaluation of KAuth and KAuth+ and other similar password-replacement schemes regarding security.
●=Offers the benefit; ○=almost offers the benefit; no circle=does not offer the benefit
↑=better than passwords; ↓=worse than passwords; no arrow= no change

			Security										
Category	Scheme	Reference	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		○ ↑	○	○ ↑	○ ↑		● ↑		●	↓	↓	↓

5.3 Security Evaluation

5.3.1 KAuth: Security Evaluation

Our scheme is quasi-resilient-to-physical-observation as an attacker can target the infrequently typed passphrase. If the user is logged in using Keybase's client, the scheme becomes stronger regarding resistance to physical observation. 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.

5.3.2 KAuth+: Security Evaluation

The system is resilient-to-physical-observation. Even if an attacker observes the input of the passphrase maybe through "shoulder surfing", they still do not have access to the private key which is required for the authentication. It is also resilient-to-targeted-impersonation since knowledge for personal details, cannot grant any advantage in exploiting the system. In addition, the scheme offers the Resilient-to-throttled and unthrottled guessing benefits.It is resilient to theft since the private key can be lost but this does not grant access the the user's account since they also need the passphrase validation. It is 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. Also, it offers the leaks-from-other-verifiers benefit as well as the no-trusted-third-party and requiring-explicit-consent benefits. It is not offering the resilient-to-phishing and unlinkable benefits.

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.

Chapter 6

Discussion and Future Work

Contents

6.1 Attacks on Keybase 31

6.2 Incorporating KAuth with the official Keybase 32

6.3 Extended Usability Studies 32

In this section we discuss the directions we consider for further exploring as part of our future work.

6.1 Attacks on Keybase

A significant part of our future work concerns deeper analysis of Keybase's security. Keybase is a new platform, and even though it seems that they offer strong protection from certain attacks, there might be other attacks that could exploit their system. For instance, using EdDSA [3] with a private key derived from a passphrase is a new paradigm, which needs further evaluation. Attackers could crawl Keybase users and use their public keys in order to brute-force their passphrase. With this in mind we envision research that explores if traditional cryptographic hashes of passwords are more secure than public keys when private keys are derived from a passphrase.

In particular, we have stressed in many places of this project that passwords and passphrases, although they look alike, they incorporate completely different mechanics. Exploring how they compete with each other is something that we plan to investigate in the future; this includes performance results in cracking a password compared to a passphrase, as well as applying existing password-hardening techniques to secure passphrases [21].

6.2 Incorporating KAuth with the official Keybase

Integration of KAuth and the offical source of Keybase would require Keybase offering an OAuth2 API for clients to register their applications. Also if Keybase would want to enhance their login procedure with the signing of the message like KAuth+ proposes, there are changes that need to be done to the existing authentication system.

Our plan is to work with Keybase and port our code to the actual product, offering an additional service to the platform for SSO through PKI. Keybase is open source, therefore our research prototype can be easily integrated with the existing software base of Keybase.

6.3 Extended Usability Studies

So far the evaluation of our system was based on the Usability-Deployability-Security framework [5] and not on actual experiments involving real users. The evaluation methodology we used has been applied to several authentication systems and it is considered as one of the state-of-the-art methodologies for evaluating authentication techniques. Nevertheless, it is useful to validate the outcome of a theoretic evaluation with actual results

Therefore, in order to further analyze the usability aspect of our system and the potential integration with Keybase, we plan to conduct surveys involving actuals users in order to ascertain a complete picture of how usable and user friendly our proposal is. Analyzing how users understand and react to using PKI, will help us define any flaws our approach has and amend the scheme in order to qualify as a feasible password-replacing scheme.

Chapter 7

Related Work

Contents

7.1 Passwords **33**

7.2 Single sign-on services **34**

Human-to-machine authentication is drawing the attention of the research community as several on-line services are solely based on authenticating the user before offering any functionality. We review here related work in passwords –the most popular technique for authenticating humans to machines– and one promising direction for scaling passwords, which is the reduction of many credentials to a single point of authentication, commonly known as Single Sign-on (SSO).

7.1 Passwords

Text-based passwords allow a user to authenticate with a service by providing a string that can be usually memorized. The service receives the password in plain text, cryptographically hashes it (usually the string is concatenated with a *salt*) and checks it with an already computed digest stored in a database. Researchers have analyzed a corpus of 70 millions of 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 [4, 23, 42, 43]. Additionally to little entropy, researchers have identified significant password reuse [11, 16], which raises the probability of an attacker to crack a password hash. Towards stronger passwords, researchers have measured the ability of humans to memorize longer secrets of about 56 bits [6, 34]. KAuth does not rely on passwords, but on *passphrases*. The user needs to memorize a strong secret, but the secret is never transmitted. Instead, the secret is used to derive a cryptographic key that is used for signing a message using EdDSA [3]. Moreover, KAuth can additionally

use a private PGP key, which is not derived from a user secret to sign a second message for authenticating the user.

7.2 Single sign-on services

Single sign-on is an authentication service that enables users to use one set of credentials to access many websites or applications. The user logs in to the main application using their credentials (e.g. username/email and password) and then authorizes other applications to access their data. The wide adoption of SSO services have led to multiple different implementations of it. Some of them include tickets such as the Kerberos-based implementation [2] and others use mark-up languages such as SAML [20]. The large interest in SSO services also led to doubts about its security benefits. Various research papers have studied the security of social login mechanisms [8] and addressed quite a few vulnerabilities that were fixed before the public announcement of these flaws [35]. Also, there are tools like SSOScan [45] that scan websites with SSO integrations searching for vulnerabilities using the Facebook Single sign-on APIs. SSOScan was run on a more than 1600 websites that used Facebook SSO and the results showed that more than 20% of them suffered from at least one of the five known major SSO vulnerabilities. KAuth works in a similar way but offers a different login mechanism that strengthens the user's account security.

Chapter 8

Conclusion

In this thesis project we worked on a PKI based approach for user authentication. Using Keybase, a platform that offers easy handling of keys and strong authentication mechanisms. We build a website that offers KAuth and KAuth+ authentication and a server that uses the OAuth2 protocol along with Keybase's login procedure. KAuth validates a user using their Keybase username and passphrase and KAuth+ validates using the user's username, passphrase but also their PGP key too. The results show that our approaches offer almost the same benefits regarding usability with OpenID and Facebook Connect. It is also evident that all password-replacing schemes lack some deployability benefits that web passwords offer. Security benefits are what our schemes target. KAuth and especially KAuth+ achieve better scores than most similar approaches, and a significant improvement on security compared to web passwords. Our scheme does not aim to become a replacement of the existing methods but become an option for people who want to use PKI for authentication. Because of noawadays situation with data breaches becoming a major issue, security benefits should be considered higher than the others, and our approach assists many of the challenges that existing implementations face.

Bibliography

[1] Ars Technica. Twitter detects and shuts down password data hack in progress, 2013. http://arstechnica.com/security/2013/02/twitter-detects-and-shuts-down-password- data-hack-in-progress/.

[2] S. M. Bellovin and M. Merritt. Limitations of the kerberos authentication system. *SIGCOMM Comput. Commun. Rev.*, 20(5):119–132, Oct. 1990.

[3] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang. High-speed high-security signatures. In *CHES*, 2011.

[4] J. Bonneau. The science of guessing: Analyzing an anonymized corpus of 70 million passwords. In *2012 IEEE Symposium on Security and Privacy*, pages 538–552, May 2012.

[5] J. Bonneau, C. Herley, P. C. v. Oorschot, and F. Stajano. The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, SP '12, pages 553–567, Washington, DC, USA, 2012. IEEE Computer Society.

[6] J. Bonneau and S. Schechter. Towards reliable storage of 56-bit secrets in human memory. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 607–623, San Diego, CA, 2014. USENIX Association.

[7] Brent Shaffer. OAuth2 Server Library for PHP, 2014. https://bshaffer.github.io/oauth2-server-php-docs.

[8] Y. Cao, Y. Shoshitaishvili, K. Borgolte, C. Kruegel, G. Vigna, and Y. Chen. Protecting web-based single sign-on protocols against relying party impersonation attacks through a dedicated bi-directional authenticated secure channel. In A. Stavrou, H. Bos, and G. Portokalidis, editors, *Research in Attacks, Intrusions and Defenses*, pages 276–298, Cham, 2014. Springer International Publishing.

[9] Chris Coyne, Max Krohn. Keybase Login, 2017. https://keybase.io/docs/api/1.0/call/login.

[10] T. N. Daily. Hacker Posts 6.4 Million LinkedIn Passwords, 2013. http://www.technewsdaily.com/7839-linked-passwords-hack.html.

[11] A. Das, J. Bonneau, M. Caesar, N. Borisov, and X. Wang. The tangled web of password reuse. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*, 2014.

[12] R. Dhamija, J. Tygar, and M. Hearst. Why phishing works. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2006.

[13] R. Dhamija and J. D. Tygar. The battle against phishing: Dynamic security skins. In *Proceedings of the 2005 Symposium on Usable Privacy and Security*, SOUPS '05, pages 77–88, New York, NY, USA, 2005. ACM.

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

[15] A. Everspaugh, R. Chatterjee, S. Scott, A. Juels, and T. Ristenpart. The pythia prf service. In J. Jung and T. Holz, editors, *USENIX Security Symposium*, pages 547–562. USENIX Association, 2015.

[16] D. Florencio and C. Herley. A large-scale study of web password habits. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 657–666, New York, NY, USA, 2007. ACM.

[17] N. Gelernter, S. Kalma, B. Magnezi, and H. Porcilan. The password reset mitm attack. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 251–267, 2017.

[18] Gizmodo. The Sony Hack Gets Even Worse as Thousands of Passwords Leak, 2014. https://gizmodo.com/sony-pictures-hack-keeps-getting-worse-thousands-of-pa-1666761704.

[19] Google. Google+ Sign-in, 2018. https://developers.google.com/+/web/signin/.

[20] informationweek.com. SAML: The Secret to Centralized Identity Management. https://www.informationweek.com/software/information-management/saml-the-secret-to-centralized-identity-management/d/d-id/1028656.

[21] A. Juels and R. L. Rivest. Honeywords: Making password-cracking detectable. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 145–160, New York, NY, USA, 2013. ACM.

[22] Keybase. Keybase, 2018. https://keybase.io.

[23] S. Komanduri, R. Shay, P. G. Kelley, M. L. Mazurek, L. Bauer, N. Christin, L. F. Cranor, and S. Egelman. Of passwords and people: Measuring the effect of password-composition policies. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 2595–2604, New York, NY, USA, 2011. ACM.

[24] G. Kontaxis, E. Athanasopoulos, G. Portokalidis, and A. D. Keromytis. Sauth: Protecting user accounts from password database leaks. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 187–198, New York, NY, USA, 2013. ACM.

[25] R. W. F. Lai, C. Egger, D. Schröder, and S. S. M. Chow. Phoenix: Rebirth of a cryptographic password-hardening service. In *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017.*, pages 899–916, 2017.

[26] lwn.net. E-mail discussion at Debian about the wiki.debian.org security breach, 2013. https://lwn.net/Articles/531727/.

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

[28] Max Krohn. Keybase's implementation of PGP in JavaScript, 2014. https://keybase.io/kbpgp.

[29] Max Krohn, Bill Thornton. Keybase Login NodeJS Module, 2016. https://keybase.io/docs/api/1.0/call/login.

[30] M. Miculan and C. Urban. Formal analysis of facebook connect single sign-on authentication protocol. In *Proceedings of the 37th International Conference on Current Trends in Theory and Practice of Computer Science*. Springer, 2011.

[31] Mitchell Anicas. An Introduction to OAuth 2, 2014. https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2.

[32] A. Muffet. Facebook: Password hashing and authentication, 2015. https://video.adm.ntnu.no/pres/54b660049af94, Video.

[33] Neowin.net. IEEE data breach: 100K passwords leak in plain text, 2012. http://www.neowin.net/news/ieee-data-breach-100k-passwords-leak-in-plain-text.

[34] L. N. Nguyen and S. Sigg. Personalized image-based user authentication using wearable cameras. *CoRR*, abs/1612.06209, 2016.

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

[36] A. Pashalidis and C. J. Mitchell. A taxonomy of single sign-on systems. In R. Safavi-Naini and J. Seberry, editors, *Information Security and Privacy*, pages 249–264, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

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

[38] J. Schneider, N. Fleischhacker, D. Schröder, and M. Backes. Efficient cryptographic password hardening services from partially oblivious commitments. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 1192–1203, New York, NY, USA, 2016. ACM.

[39] B. Schneier. Frequent Password Changes Is a Bad Security Idea, 2016. https://www.schneier.com/blog/archives/2016/08/frequent_passwo.html.

[40] R. Shay, S. Komanduri, P. G. Kelley, P. G. Leon, M. L. Mazurek, L. Bauer, N. Christin, and L. F. Cranor. Encountering stronger password requirements: User attitudes and behaviors. In *Proceedings of the Sixth Symposium on Usable Privacy and Security*, SOUPS '10, pages 2:1–2:20, New York, NY, USA, 2010. ACM.

[41] Twitter. Sign in with Twitter, 2018. https://dev.twitter.com/docs/auth/sign-twitter.

[42] E. von Zezschwitz, A. De Luca, B. Brunkow, and H. Hussmann. Swipin: Fast and secure pin-entry on smartphones. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 1403–1406, New York, NY, USA, 2015. ACM.

[43] C. Winkler, J. Gugenheimer, A. De Luca, G. Haas, P. Speidel, D. Dobbelstein, and E. Rukzio. Glass unlock: Enhancing security of smartphone unlocking through

leveraging a private near-eye display. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 1407–1410, New York, NY, USA, 2015. ACM.

[44] Wired.com. Google Declares War on the Password, 2013. `http://www.wired.com/wiredenterprise/2013/01/google-password/all/`.

[45] Y. Zhou and D. Evans. Ssoscan: Automated testing of web applications for single sign-on vulnerabilities. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 495–510, San Diego, CA, 2014. USENIX Association.

List of Figures

2.1 Users can do crypto operations such as encryption/decryption and sign/verify using Keybase's website, their client, or their command line tool. The first method requires that the user has their private key encrypted and uploaded on Keybase's servers. If you want to keep your private key fully protected, you have to use the command line tool or the local client. . . . 5

3.1 When a user wants to login at a third party website and they choose to do so using KAuth or KAuth+, this is the flow that they have to follow. 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. 8

3.2 Example of the getSalt API call by Keybase official API page. 9

3.3 The client should sign a JSON blob of this form. 10

3.4 To validate a login, KAuth requests a salt using the user's username. The entered passphrase and the salt (unhexed) are entered as parameters in the scrypt function. Part of the result of scrypt is handled as a private key. This private key is used to sign a JSON blob (as shown in Figure 3.3). An EdDSA signature is generated and then packaged into a Keybase-style signature. The result is sent to the Keybase server as the pdpka5 parameter. The KAuth+ login system differs from the basic one as it executes an additional action. It follows all steps of the basic login system, and then requires the user to run a script on their system that will generate a signed message. This signed message is uploaded and verified by Keybase. If all steps succeed, the user is then logged in to Keybase. 10

3.5 The structure of a Keybase-style signature as presented by the official Keybase docs file. 11

4.1 The login page offers the user the option to login using a username and password or to "Login with Keybase". Choosing to login with Keybase, the user is redirected to the Authorize controller of the OAuth server. This controller redirects the user to the login page of Keybase Website. 14
4.2 The user's passphrase and the salt (unhexed) retrieved are entered as parameters in the scrypt function which generates a 256 byte passphrase 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. 15
4.3 Every time a user was redirected to our Keybase login page they were presented with a form requiring a username and passphrase as shown in Figure 4.3. When they submitted the form, the nodeJS script was called and replicated the official Keybase login procedure as described on their API call. 15
4.4 For the KAuth+ procedure, we use the uploaded file to verify the users signature. We make a GET request to Keybase's API to get their public keys and then using a PGP Javascript library, we verify the user's identity or reject the login request. 16
4.5 For our KAuth+ version, after getting through the basic login procedure, the user was redirected to a wizard-like interface with 3 simple steps. The first step required the user to download and execute a script written in Bash. This script uses Keybase's command line utility (that is required for this to operate) to sign a message. The script automatically generated a file named Signature.sig. 17
4.6 In step 2 of the wizard, the user is asked to upload the file generated by the script. Then, a GET request is sent to Keybase's API retrieving the user's public key. Then, using Keybase's PGP implementation for Javascript [28] we verify the uploaded file's signature with the public key. The verification is successful is the signature was produced by the person trying to login from the first login procedure (with username and passphrase) and if the signature was produced earlier than 1 minute before the login attempt 18
4.7 If any of the steps of the wizard process are not executed correctly, the appropriate error message is displayed on the screen 19

4.8 The verification is successful is the signature was produced by the person trying to login from the first login procedure (with username and passphrase) and if the signature was produced earlier than 1 minute before the login attempt. If this was successful, an informative message was displayed to the user and a button that would complete this wizard, logging the user on Keybase. 20
4.9 After the step of the validation, the user is informed of the scope of the client website, meaning the data that they will access through Keybase's API. 21
4.10 An example of the signature generated after signing a message. 22
4.11 The script used to sign a message for the user and prove their identity. . . 22
4.12 The database schema used for the OAuth2 server was instructed by the library we used. 23

List of Tables

5.1 Comparative evaluation of KAuth and KAuth+ and other similar password-replacement schemes regarding usability.
•=Offers the benefit; ○=almost offers the benefit; no circle=does not offer the benefit ↑=better than passwords; ↓=worse than passwords; no arrow= no change 25

5.2 Comparative evaluation of KAuth and KAuth+ and other similar password-replacement schemes regarding deployability.
•=Offers the benefit; ○=almost offers the benefit; no circle=does not offer the benefit ↑=better than passwords; ↓=worse than passwords; no arrow= no change 27

5.3 Comparative evaluation of KAuth and KAuth+ and other similar password-replacement schemes regarding security.
•=Offers the benefit; ○=almost offers the benefit; no circle=does not offer the benefit ↑=better than passwords; ↓=worse than passwords; no arrow= no change 29