UbiKiMa: Ubiquitous authentication using a smartphone, migrating from passwords to strong cryptography*

Short paper

Maarten H. Everts
TNO
maarten.everts@tno.nl

Jaap-Henk Hoepman
Radboud University Nijmegen
jhh@cs.ru.nl

Johanneke Siljee
TNO
johanneke.siljee@tno.nl

ABSTRACT

Passwords are the only ubiquitous form of authentication currently available on the web. Unfortunately, passwords are insecure. In this paper we therefore propose the use of strong cryptography, using the fact that users increasingly own a smartphone that can perform the required cryptographic operations on their behalf.

This is not as trivial as it sounds. Services will not migrate to new forms of authentication if few users have the means to use it. Similarly, users will not acquire the means if there are few services that accept them. Moreover, enabling one's smartphone to seamlessly sign in at a website when browsing on an arbitrary PC is non-trivial.

We propose a system, based on a smartphone app, that can be used to sign in with username and password to arbitrary websites using an arbitrary PC or laptop. We describe the protocol and implementation to achieve this without the need for typing usernames and passwords. Furthermore, we propose an authentication protocol based on public key cryptography, integrated in the same smartphone app. This allows websites to seamlessly migrate towards a much more secure authentication method on the web, independently of each other.

A prototype of our system has been developed.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection; K.6.5 [**Management Of Computing And Information Systems**]: Security and Protection

Keywords

identity management; passwords; strong authentication

*This research is supported by the research program Sentinels (www.sentinels.nl) as project 'Mobile Identity Management' (10522). Sentinels is being financed by Technology Foundation STW, the Netherlands Organization for Scientific Research (NWO), and the Dutch Ministry of Economic Affairs.
This research was performed within the Privacy & Identity Lab www.pilab.nl and funded by SIDN.nl.

DIM'13, November 8, 2013, Berlin, Germany.

ACM 978-1-4503-2493-9/13/11 ...$15.00.
http://dx.doi.org/10.1145/2517881.2517885.

1. INTRODUCTION

Passwords are the only ubiquitous form of authentication currently available on the web. Unfortunately, passwords are insecure, so they need to be replaced by a stronger method of authentication [9, 4].

The ultimate goal of our work is to replace the use of passwords and to ensure the ubiquitous use of public key cryptography for authentication. This is a challenge for several reasons. (1) Service providers (aka relying parties (RP)) need to support this new form of authentication. This requires them to change systems, which incurs a cost. They will not invest unless they see an advantage. (2) Users access services through a multitude of browsers and devices, being office desktops, home PCs or tablets. (3) Users prefer a consistent user experience. (5) Users resist solutions that require them to install new software or update existing software. In certain cases they are not even in the position to do so, for example at PCs at work, or in Internet cafés.

We are inspired by the recent surge in password management apps for smartphones as the ideal wedge to break this status quo. These days, many users own smartphones that are capable of performing strong cryptographic operations, and that have a graphical user interface. We propose a system that explicitly supports a migration path from passwords to secure cryptographic authentication. It allows users and relying parties to increase their security gradually at every step. These steps can be made independently, allowing each party to decide to move forward when they see the advantage.

In the first step users decide to use their smartphone to manage their passwords and to authenticate at relying parties. Their benefit is manifold: their passwords are stored in one place, need no longer be memorised, and are always with them. In addition, account passwords no longer need to be stored within a browser at work or at a PC shared with others. Finally, the authentication app supports the generation of strong passwords. All this increases security for the user.

In the second step users install a bookmarklet[1] in their browser. By clicking on the bookmarklet (and confirming sign-on on their phone) the smartphone app logs the user in automatically. Passwords no longer need to be entered manually, increasing the user experience. All that is required is the installation of a bookmarklet and registration at a proxy. The way the proxy is used ensures minimal privacy consequences[2].

[1]We were inspired by the bookmarklet used by www.instapaper.com.
[2]Not all devices may allow the installation of a bookmarklet. In the future, pre-installed browser extensions or eventually native browser support are foreseen.

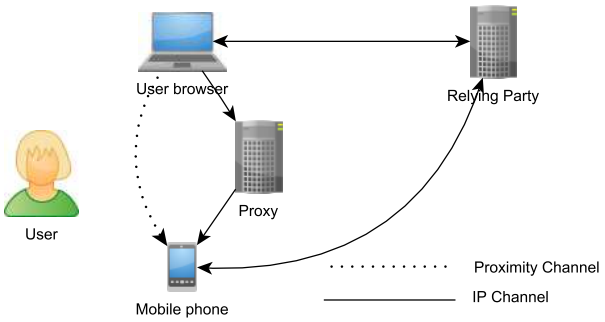

Figure 1: System components.

All migration steps so far cover the user side of the equation. At any point, and in parallel to the above sketched path, relying parties may choose to support the public key based form of authentication. All code at the client side supports this option from the start. Whenever a relying party upgrades, users are offered the option to upgrade to the new (much more secure) form of authentication automatically.

To summarise, our contribution is the following. We propose the use of a smartphone to authenticate users on the web in such a way that that it supports a gradual yet seamless transition towards an authentication scheme that is more secure than passwords. To this end we introduce in Section 6 a protocol that, using a proxy, a bookmarklet, and a password management app on a smartphone, allows users to sign in at arbitrary websites, without the need to remember and retype account names and passwords. The same app (and associated stored account data) can be used anywhere, at any PC and browser that allow such a bookmarklet to be saved. In addition we propose in Section 4 an authentication protocol based on public key cryptography, that is integrated within the same smartphone app, for a secure form of authentication at websites. The protocol is based on the modified Needham-Schroeder protocol [8, 6, 5] and uses QR codes to link the browsing session with the smartphone app. We analyse the security of this protocol in Section 5. Because the same app supports both forms of authentication, websites can independently decide to migrate towards supporting this stronger form of authentication.

Finally, a prototype of our system has been developed and will be made available as open source at the UbiKiMa website (http://ubikima.com).

2. RELATED WORK

Similar to our approach in Section 6, Bursztein *et al.* [2] also propose the use of bookmarklets to automate signing-in process using a user's smartphone. However, they rely on the possibility to transfer browser session state from one session to another. This is an undocumented feature at best, if not a downright bug in the server side implementation of the authenticated session. It does allow them, however, to always keep the account information local to the user smartphone. We, on the other hand, do not rely on such awkward assumptions, and thus are the first to propose a truly universal mechanism that supports secure login to every website from any terminal using an off the shelf phone.

Grosse and Upadhyay [4] describe an approach where the user's credentials (based on public key cryptography) are stored on a separate, external device. They use a USB token with capacitive touch-sensitive area for user confirmation (for which no additional device drivers need to be installed). The Relying Party (RP) sends a hash of the website's URL, which is forwarded to the USB token. Similar to what we propose, the token generates a new self-signed public-private key pair for each new RP during registration to prevent tracking of users over different RPs. This solution requires the user to buy special additional hardware, whereas our solution only requires hardware that many users already own.

Several QR-based schemes have been proposed before [3, 12]. TIQR [13] is an open source mobile authentication solution that shares two important properties with our approach. First, it also uses the phone as the carrier for the user's credentials and second, they also use a QR codes. As such, the flow of actions by the user is similar. A key difference however is that TIQR authentication protocol is based on pre-shared secrets whereas we use public-key cryptography. In addition, because our approach features mutual authentication, it is less susceptible to phishing attacks.

There are a number of authentication schemes that do use public-key cryptography, but typically these do not support the use of a separate device (i.e., a smartphone) as the holder for the key material. The WebID [10] protocol for authentication for example is based on user certificates that reside in the browser. The certificates can be self-signed, but the public key is (also) stored at a so-called Web-ID Provider, which hosts a profile page for the user. A WebID certificate contains the URI that directs the RP to the user's profile on the WebID Provider, for validation of the certificate. Since the RP visits the user profile on the WebID Provider directly, the WebID Provider knows exactly which user visits which RP at which time, posing a privacy problem.

A similar approach is called Mozilla Persona [7], which implements Mozilla's BrowserID protocol. In this case the user generates a public-private key pair and has one of her identity providers (IdPs, in the case of Persona one or more email providers) sign the public key. A Persona certificate contains (among other) the user's email address and the IdP's domain name, for validation of the certificate. Similarly to WebID, the RP visits the IdP directly. But in this case the IdP's public key can also be cached at the RP, in which case look-ups are not necessary, improving user privacy. Other issues are 1) a possible weakness to man-in-the-middle attacks since the RP does not generate a challenge, 2) that the IdP controls the identity and can create valid assertions for it at any time, and 3) that anyone who controls the browser can let the browser sign assertions and subsequently log in at the user's account at the RP. Persona tries to resolve this by letting the user click on a login button at the RP and select her email address, but these steps can be scripted and therefore do not authenticate the user, only the browser.

3. PRELIMINARIES

Before moving on to explain our authentication solution we present a system model that is relevant for both the ideal situation that uses public key cryptography (Section 4) and our approach for supporting legacy passwords systems (Section 6).

3.1 System model

We wish to allow a user browsing the web to sign in at a website using his smartphone as a means of authentication.

We assume the user to browse the web using an arbitrary PC and an arbitrary browser, not necessarily his own. Nevertheless, the PC/browser has to be trusted at least partially. No authentication scheme is secure if the end-point itself is corrupted. We

assume that the user is able to add a bookmark, including bookmarks that contain JavaScript code (so-called bookmarklets).

The smartphone belongs to the user and is trusted[3]. The smartphone has an Internet connection (WiFi, or cellular data). Ideally, there also is a means to communicate directly with the smartphone from the PC using a local channel which, at a minimum, allows the PC to send messages to the smartphone. This channel could be implemented using Bluetooth, Near Field Communication (NFC), or 2-dimensional barcodes such as QR codes (by using the smartphone camera to read the message embedded in the QR code displayed on the PC display). The latter option is preferred as it requires no special set-up steps, and can be supported by all PCs and all camera-equipped smartphones.

Due to firewall restrictions that are often present, we cannot assume that the PC and smartphone can connect and communicate over the Internet directly. We do assume however that both can connect to an external server over HTTP. This allows both devices to register at a proxy, to which they can send messages to be forwarded to the other party. The model is depicted in figure 1.

3.2 System Requirements

The system must implement mutual authentication: the user needs to authenticate at the relying party, and vice versa. Moreover, it must prevent (cf. . [1]): 1) unauthorised access to the user's security credentials, 2) abuse of the service provided by the RP 3) unauthorised access to the user's account, and 4) unwanted collection of private information of the user. Moreover, while taking the security and privacy requirements into account, the system should be easy and convenient to use.

3.3 Notation

Throughout the paper we use the following notation. We have a User U who is accessing a service offered by a Relying Party R with his browser. The user also owns a smartphone M. The Relying Party typically consists of a standard web server as well as a separate component that handles the authentication protocol. All these entities may have one or more cryptographic key pairs. Entity a has public key K_a and private key k_a. A message m encrypted using public key K_a is denoted $\{m\}_{K_a}$.

4. THE IDEAL CASE: AUTHENTICATING USING PUBLIC KEY CRYPTOGRAPHY

Our protocol is based on the modified Needham-Schroeder protocol [8, 6, 5]. Relying party R has public key K_R and private key k_R for a suitable public key cryptosystem. For privacy reasons, users have multiple key pairs for authenticating at different relying parties. This prevents them from being tracked across services.

The smartphone M of user U keeps a database DB_U of tuples $\langle K_R, \mathrm{URI}_R, \mathrm{Name}_R, K_U, k_U\rangle$ that record, for every relying party R the user knows about, the following information: the public key K_R, the endpoint URI_R at which the relying party is listening for the authentication protocol, a user-friendly representation of the relying party's name Name_R, and the public key K_U and private key k_U to be used for authenticating at this particular relying party. The Relying Party R keeps a database DB_R of tuples $\langle K_U, \mathrm{Account}_U\rangle$ that record, for every user U, the following information: the public key K_U and its account $\mathrm{Account}_U$. The account essentially is a pointer to where all user data for this user can be found. These databases are initially empty and filled by registering users at relying parties using the protocol.

In the following, $\mathscr{R}$ and $\mathscr{S}$ are sets from which random nonces and session identifiers are selected, respectively. The domain for account names and relying party names is not further specified. These can be considered arbitrary binary strings with proper length encoding. In general, a web server may be engaged in several access requests in parallel. Therefore, R stores information about ongoing requests in a pool Pool_R. Data in this pool is only kept for a short amount of time (in the order of minutes). Initially $\mathrm{Pool}_R = \emptyset$.

4.1 Registration/authentication protocol

To register at or authenticate to a certain relying party R, the following protocol is run between a user U with her smartphone M and the relying party R. Registration and authentication are handled in a single protocol because both protocols (when developed separately) turned out to be remarkably similar, and it improves the user experience as the user does not have to remember whether she registered already at a service.

- U browses to the entry page of R using SSL/TLS< and clicks on the access button on that page.
- R generates a random session identifier $sid \in_r \mathscr{S}$, and returns $sid \| \mathrm{URI}_R$ to the browser of U. Also, it internally associates sid with this particular SSL/TLS session, and it stores an entry $\langle \bot, sid, \bot, \bot\rangle$ for sid in the request pool Pool_R. (If such an entry already exists for sid, a new session identifier is generated.)
- U forwards this data to the mobile device M (see 4.2 for options).
- M connects to R over URI_R and retrieves $K_R \| \mathrm{Name}_R$ from R. M then checks whether it has an entry in its database DB_U for K_R, and whether that entry corresponds to the following tuple: $\langle K_R, \mathrm{URI}_R, \mathrm{Name}_R, K_U, k_U\rangle$. If the entry exists, but appears to contain the wrong data, the protocol aborts. Multiple entries for K_R are not permitted. If the entry exists, it sets $type :=$ `authenticate` else $type :=$ `register`, to record the type of the run.
- If $type =$ `authenticate`, M asks the user U whether she wishes to access his account at relying party Name_R. Otherwise M asks the user U whether she wishes to create a new account at relying party Name_R. If not, the protocol aborts.
- If $type =$ `register` then M generates a key pair K_U, k_U.
- M generates a random $r_U \in_r \mathscr{R}$.
- M sends $\{\mathtt{t1} \| type \| sid \| r_U \| K_U\}_{K_R}$ to R at endpoint URI_R.
- R decrypts the message, and verifies tag `t1` and verifies that it has an session open with sid in the request pool Pool_R. If $type =$ `register`, then R verifies that it does not have an entry for K_U in DB_R. If $type =$ `authenticate` then this tuple must exist. In all other cases R aborts.
- R generates random $r_R \in_r \mathscr{R}$ and updates the tuple for session sid in Pool_R to $\langle r_R, sid, K_U, type\rangle$. (If a tuple with r_R exists, a new r_R is generated.) R returns $\{\mathtt{t2} \| r_R \| r_U \| K_R\}_{K_U}$ to M.
- M decrypts the message using k_U, verifies tag `t2` and verifies r_U and K_R. If $type =$ `register` then M creates entry $\langle K_R, \mathrm{URI}_R, \mathrm{Name}_R, K_U, k_U\rangle$ in its database DB_U.
- M sends r_R to R (unencrypted). It notifies the user that authentication/registration was completed successfully.

[3]As it stores the user credentials, we have no option but to trust it. One can store the credentials in the SIM, or a secure SD card, if so desired.

- R looks up a matching tuple $\langle r_R, sid, K_U, type \rangle$ in Pool_R. If such an entry does not exists, the protocol fails. If it does exist, the web server is informed that the session at *sid* is associated with public key K_U. If $type =$ `register`, the web server creates a fresh account Account_U, and adds $\langle K_U, \text{Account}_U \rangle$ to DB_R. (It will complain if such an entry already exists for K_U.) If $type =$ `authenticate`, it looks up the tuple $\langle K_U, \text{Account}_U \rangle$ in DB_R, and associates the SSL/TLS session with Account_U. The user is signed in to this account. (R will complain if such a tuple does not exist for K_U.)

All entities in the protocol wait only a short amount of time for a message that they expect to arrive in response to an earlier message they sent. If that message does not arrive in time, an error message is displayed and the protocol is aborted.

4.2 Bootstrapping communication

In the the initial phase of our authentication protocol a session identifier (sid) and a URI are to be transferred from the Relying Party to the Mobile through the user's browser to bootstrap further communication between the Mobile M and the Relying Party. By design, in our scenario the phone and the PC running the browser are physically close to each other, allowing us to use a simple QR-code as a proximity channel for this message. As stated before, compared to other proximity channels such as NFC, QR-codes require little setup and given the ubiquity of cameras in smartphones QR-code scanning is generally available.

Alternatively, the Relying Party could initiate the communication with the phone directly. This however requires the user to share with the Relying Party information on how to reach his phone (either during setup or at each authentication). There are however number of privacy, security and usability issues with such an approach, which are further discussed in Section 5.

4.3 Concrete implementation

An important consideration when implementing our authentication scheme is the choice of the public-key encryption scheme. For our prototype we opted for Elliptic Curve Cryptography (ECC) using the 256 bit NIST curve `prime256v1`. For the encryption of the messages t1 and t2 we use a combination of asymmetric and symmetric cryptography in the form of Hashed ElGamal for the key encapsulation mechanism, AES for the block cipher and CCM as the authenticated mode of operation. The AES key is 128 bit long and the values *sid* r_R and r_U are all 128 bit random values. The (random) nonce for CCM is 13 bytes long.

5. SECURITY ANALYSIS

In this section we perform a security analysis on the authentication protocol described in the previous section by analysing the possible attack scenarios.

5.1 Malware on the browser

If the user PC or browser is compromised by malware, the attacker can take over the entire session after successful authentication to steal information, insert transactions, etc. No authentication protocol protects against such attacks and therefore our protocol does not have special measures to prevent it either.

However, our protocol does offer protection against a man-in-the-browser attack that tries to obtain the user's credentials. Replay attacks are prevented and the private key never leaves the smartphone.

5.2 Malware on the phone

The protocol does not protect against malware on the smartphone. To reduce the risk of malware on the smartphone, all the keys could be encrypted based on a user-supplied pin code. This prevents malware (or a malicious user with a lost or stolen phone) from reading the private keys directly. Alternatively, storing the private keys in a secure element (SIM or other hardware) prevents an attacker from transferring credentials. However, without the possibility for secure pin code entry, a secure element does not prevent an attacker from *using* the credentials once they are unlocked, or trying to intercept the user's pin code.

Also note that if a piece of malware on the smartphone can read or manipulate the *sid* value sent from the browser, this may lead to problems. See section 5.5.

5.3 Malicious or compromised RP

A malicious RP (or an attacker with information stolen from a compromised RP) could attempt to stage a phishing attack to lure a user to authenticate at another legitimate RP for him. This will fail because (a) our Needham-Schroeder based protocol features mutual authentication, (b) the smartphone will check for the right combination of $\langle K_R, \text{URI}_R, \text{Name}_R \rangle$, and (c) a separate user key pair is used for each RP. An additional advantage of authentication based on public key cryptography is that the credentials stored at at the (malicious or compromised) RP are public keys, which are typically harder to brute-force than hashes of passwords.

User privacy is protected by having a unique key pair for each RP, which prevents multiple RPs from colluding to link users among themselves. Furthermore, By using a proximity channel to bootstrap communication between the smartphone and the RP so there is no need to store information at the RP on how to contact the smartphone.

5.4 Browser-RP connection

We assume that the channel between the RP and the browser is secured using SSL/TLS while the protocol runs. This ensures that a MitM between RP and the browser cannot read, modify or inject messages, and therefore the protocol is protected against this kind of attack.

5.5 Browser-phone connection

The information sent over the proximity channel is better protected against unauthorised reading and manipulation than in case of an IP-connection between browser and the smartphone. Therefore we do not use additional measures such as signatures and encryption to protect this information. However, sending the session identifier *sid* unprotected incurs certain risks.

Suppose an attacker manages to intercept *sid* during registration. With the intercepted *sid* he can send an alternative `t1` message in which he registers his own public key. The attacker will be able to send the right value for r_R back to the RP, the session will appear to be authenticated and a *new* account will be associated with it. However, the smartphone will not receive a valid `t2` message, and will raise a warning message about a possible session hijacking attempt. Next, users should close the browser session. For this reason, a future account migration protocol should require the user to first register his public key with the RP, and then to authenticate the in old way (using username and password). Doing it the other way around would allow the attacker to hijack the account using the method outlined above.

Also, if an attacker controls the browser or the channel from browser to the smartphone, he can swap the user's *sid* with the

sid from an authentication session he started himself, tricking the user into signing the attacker in at the RP. This could be mitigated by printing the *sid* in the browser window and on the GUI of the mobile app (provided the browser itself is not compromised). However, *any* authentication protocol will suffer from a similar attack: you simply cannot protect yourself from an attacker that sits within your PC or browser.

Finally, using a QR-code to initiate the authentication process has an additional advantage in that it requires deliberate user action.

5.6 Phone-RP connection

The communication between the smartphone and the RP does not require a secure channel such as TLS/SSL because the important messages (t1 and t2) are encrypted directly. The response to the `hello` message contains public knowledge (K_R and Name_R), manipulation of which will be detected by the smartphone. The final message containing the random value r_R is not reusable by an attacker. Blocking or manipulating it will only result in a DoS attack against the user.

Using a QR-code to send *sid* via the browser to the smartphone instead of directly from the RP prevents the risk of an attacker intercepting *sid* on the Phone-RP link (see section 5.5). Alternatively, not sending any information at all between the smartphone and the RP could be achieved by sending all messages via the browser/PC.

6. PASSWORD-BASED AUTHENTICATION

The most common way of authenticating on the web is still a username/password combination. We provide a migration solution towards our authentication protocol (see Section 4) in which the smartphone is used as a portable password manager. The username/password combinations are only stored on the phone, and thus the Relying Party does not have to modify its login procedure.

On a high level (and from a user's perspective) an authentication now has the following steps:

- The user visits a website where she needs to authenticate with a username and a password.
- In the browser, the user clicks on a 'Authenticate'-button provided by an authentication helper (see Section 6.2).
- The user receives a notification on her smartphone that an authentication request has been received.
- On her smartphone, the user verifies that the request is for the correct website and taps "OK".
- Within moments her username and password are filled in on the website and submitted.
- The user is logged in.

6.1 Technical perspective

Below we discuss how such a solution can be implemented. As illustrated in Figure 1, there are four actors in this use case: (the user at) the browser, (the same user at) the smartphone, the Relying Party, and the proxy. This proxy is needed to be able to contact the smartphone from the browser. However, we will make sure the proxy will not be able to determine for which RP the credentials are requested, nor will it be able to obtain the credentials themselves.

6.1.1 Setup

The setup consists of three steps. First, the user generates a public/private key pair (K_u, k_u) in the app on the smartphone. Next, the user creates an account at the proxy by asking the app to submit to the proxy server: (a) the public key (K_u) and (b) information on how to contact the phone (the details differ for each smartphone platform). The proxy server responds with a message containing a new account-id for the user. This account-id is used a later stage by the authentication helper in the browser for the actual authentication steps and needs to be transferred somehow to the browser (see Section 6.2). The password database in the app on the smartphone is (incrementally) filled by the user.

6.1.2 Authentication

Once the setup is completed, the authentication helper in the browser can be used to authenticate to websites. Figure 2 illustrates the steps involved. First, the user goes to a website that uses password-based authentication and clicks on the 'Authenticate' button provided by the authentication helper.

The authentication helper proceeds as follows. First a random session key k_s is generated. This session key is then encrypted to the public key of the user: $\{k_s\}_{K_u}$. The authentication helper looks up the URI of the currently visited page and creates an authentication request for that URI. This request is encrypted using the session key with a symmetric encryption scheme: $\{\text{request}\}_{k_s}$. The encrypted session key ($\{k_s\}_{K_u}$) and the encrypted request ($\{\text{request}\}_{k_s}$) are both sent to the proxy to be forwarded to the phone.

The proxy has the information to contact the phone and simply forwards the messages to the phone. Because the request is encrypted with the session key, the proxy learns nothing. In contrast, the app on the phone has the private key, with which it can decrypt the session key and subsequently decrypt the authentication request. The smartphone matches the URI in the request to a collection of URIs and username/password combinations. If there is a match, phone issues a notification and the user is asked whether she really would like to authenticate to the requested URI. If there is no match, the user is given the option to fill in the username and password. After the user has indicated the authentication should proceed, the app creates an authentication response that contains the username and password. This response is encrypted with the same session key ($\{\text{response}\}_{k_s}$) and simply sent back to the proxy. The proxy knows how to get back to the authentication helper in the browser that sent the original request and passes the authentication response on. Because the authentication helper originally created the session key it can decrypt the response, retrieve the username and password, and using some heuristic fill the username and password fields in the page. Optionally, the login-form can also be automatically submitted.

6.2 Authentication helper

As discussed in the previous section, the authentication helper in the browser acts as the link between the smartphone (having the credentials) and the web page (requesting the credentials). It can be implemented by (a) a browser extension or (b) a bookmarklet. Both options have their advantages and disadvantages. A browser extension is a way to add new features to a browser that can potentially use additional (native) libraries. However, a browser extension does require the user to actively install such an extension in the browser, for which she might not have permissions. A bookmarklet does not have this restriction as it is a special URL containing Javascript code that is to be added to the bookmark-bar of the browser (hence the name). This Javascript code will be executed when the bookmark is clicked upon. Be-

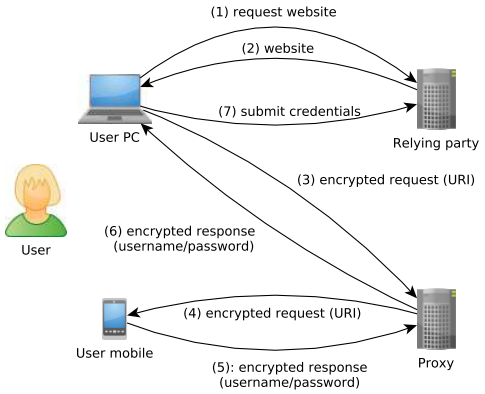

Figure 2: Schematic depiction of the steps involved in using a smartphone as a password manager for browser sessions on a PC.

cause the installation of a bookmarklet does not need special permissions and a bookmarklet allows for easier prototyping, we chose to create a Javascript bookmarklet implementation of the authentication helper for our proof-of-concept. This has some security implications, which is discussed in the next section.

6.3 Proof-of-concept implementation

Our proof-of-concept of the authentication helper is implemented as a bookmarklet. When clicked upon, the Javascript code in the bookmarklet loads additional Javascript libraries to support the next steps of the authentication. The Javascript library used for cryptography is the "Stanford Javascript Crypto Library"[4], and in particular the ECC branch. We note that cryptography in Javascript is usually frowned upon[11] for a number of reasons, including the malleability of the Javascript runtime and the lack of good entropy sources. This means that any host providing Javascript code running on the website can control the cryptographic operations. However, in this case one can argue it is not more or less secure than filling in the password "by hand", because in that case any malicious Javascript code can also obtain the credentials.

We use the 256 bit NIST ECC curve `prime256v1`, and Hashed ElGamal is used to generate and encrypt the session key k_s. This session key is 128 bits long and used together with AES in CCM mode (104 bit nonce) to encrypt the messages.

7. CONCLUSIONS

In this paper we have presented a new authentication protocol based on secure public key cryptography for usage on the web as an alternative to the traditional username/password approach. We leverage the ubiquity of smartphones and propose to use a smartphone app as a (semi-trusted) container for these secure credentials. The same app also supports traditional username/password authentication that can be used on existing websites that do not yet offer the stronger, public key based, approach. This provides a seamless experience for both users and service providers, as each can decide independently of the other when to start the transition to the new, more secure, form of authentication.

[4] http://crypto.stanford.edu/sjcl/

We have a developed a working prototype of the smartphone app, as well as the supporting infrastructure (integration modules for the service provider, a proxy and an authentication helper to be integrated in the browser).

There are several steps to be taken given our current results. First and foremost, the working prototype needs to be streamlined and tested. Then, actual deployment needs to be encouraged. This step will focus on getting as many users as possible to adopt the smartphone app as a convenient and ubiquitously usable password manager. Parallel to that, integration support for service providers in terms of API's and proper documentation will be developed. Finally, key management (that allow users as well as relying parties to update their keys in case of a compromise) protocols need to be developed, and secure backup strategies for key material need to be integrated (especially for the user side).

8. REFERENCES

[1] Gergely Alpár, Jaap-Henk Hoepman, and Johanneke Siljee. The identity crisis. Security, privacy and usability issues in identity management. *CoRR*, abs/1101.0427, 2011.

[2] Elie Bursztein, Chinmay Soman, Dan Boneh, and John C. Mitchell. Sessionjuggler: secure web login from an untrusted terminal using session hijacking. In *WWW*, pages 321–330. ACM, 2012.

[3] Ben Dodson, Debangsu Sengupta, Dan Boneh, and Monica S. Lam. Secure, consumer-friendly web authentication and payments with a phone. In *MobiCASE*, pages 17–38. Springer, 2010.

[4] Eric Grosse and Mayank Upadhyay. Authentication at scale. *IEEE Security & Privacy*, 11(1):15–22, 2013.

[5] Gavin Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Inf. Process. Lett.*, 56(3):131–133, 1995.

[6] Alfred Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[7] Mozilla. Mozilla persona. https://www.mozilla.org/en-US/persona/.

[8] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Commun. ACM*, 21(12):993–999, 1978.

[9] Peter G. Neumann. Risks of passwords. *Commun. ACM*, 37(4):126, 1994.

[10] J. Sayre and H. Story. The WebID protocol and browsers. http://www.w3.org/2011/identity-ws/papers/idbrowser2011_submission_22/webid.html, May 2011.

[11] Matasano Security. Javascript cryptography considered harmful. http://www.matasano.com/articles/javascript-cryptography/.

[12] Guenther Starnberger, Lorenz Froihofer, and Karl M. Göschka. Qr-tan: Secure mobile transaction authentication. In *ARES*, pages 578–583. IEEE Computer Society, 2009.

[13] Roland M Van Rijswijk and Joost Van Dijk. TIQR: a novel take on two–factor authentication. In *Proceedings of the 25th international conference on Large Installation System Administration*, pages 7–7. USENIX Association, 2011.