Privacy-aware web authentication protocol with recovery and revocation

Lucas Boppre Niehues[1], **Ricardo Custódio**[1]

[1]Departamento de Informática e Estatística – Universidade Federal de Santa Catarina
Florianópolis – SC – Brazil

lucasboppre@gmail.com, ricardo.custodio@ufsc.br

Abstract. *Password-based authentication is the de facto standard for web services, usually linked to email addresses for account recovery. However, this scheme has several serious known drawbacks. We present a simple pseudonym authentication scheme for users to register and authenticate to online services, by using cryptographic key pairs stored in a mobile device. In contrast to other similar schemes, our proposal requires minimal user interaction, allows for easy account recovery and revocation, and requires no trusted third-party. We created a prototype to evaluate the proposal and concluded that it is viable and has good security, privacy, and usability properties.*

Resumo. *Autenticação baseada em senha é a mais utilizada para serviços web, comumente associada a endereços de email para recuperação das credenciais. Porém, este esquema apresenta várias desvantagens de segurança e privacidade. É apresentado um simples esquema de autenticação pseudônima para serviços online, através de chaves criptográficas assimétricas armazenadas em dispositivo móvel. Em comparação com outros esquemas similares, nossa proposta demanda mínima interação do usuário, permite recuperação e revogação simples das contas, e não exige nenhuma terceira-parte. É criado um protótipo para avaliar a proposta e concluímos que ela é viável e tem boas propriedades de segurança, privacidade e usabilidade.*

1. Introduction

User authentication on the web is a hard problem. It is under competing pressures for usability and security, especially when facing user errors and sophisticated attacks. The de facto standard for general web authentication is the combination of username and password, usually linked to an email address for password recovery purposes [Bonneau et al. 2012].

However, much has been said about the problems of password schemes [Bonneau and Preibusch 2010, Yan et al. 2012]. For example, users have trouble creating and remembering random, high-entropy unique passwords [Bonneau 2012]. Coupled with the difficulty of implementing secure password schemes on the service side, these issues multiply the severity of data breaches.

Numerous alternative authentication schemes have been proposed [Borchert and Günther 2013, Lindemann et al. 2014, Xu et al. 2015]. Although these schemes bring advantages, they usually make sacrifices to security, usability, or privacy. For instance, OpenID and Facebook Login both depend on trusted third parties.

A popular proposal to solve these problems of web authentication is SQRL [Gibson 2016, van Dijk 2014]. However, this scheme has a disaster recovery protocol that is too complex, and the use of deterministic key pairs creates some security concerns.

To remedy these issues we propose a new web authentication protocol with no trusted third parties and no values to memorize, and includes the possibility of account recovery and revocation. Our proposal relies on a mobile device carried by the user to store cryptographic keys for each registered service. This scheme has good security properties and does not require the service to keep any secrets, and neither requires the user to type any name, code or password.

Authentication schemes can also imply privacy concerns. This has been clearly evidenced by the Ashley Madison data breach [Mansfield-Devine 2015], where users were identified by email addresses. Ideally, an authentication scheme should identify users in that service without linking to accounts in different services. We call this property ”pseudonymity” and implement it in our proposed scheme.

This article is organized as follows. The protocol proposed is presented in Section 2. We describe in detail the entire protocol life cycle, considering setup, data storage, the authentication process itself, and disaster recovery (namely account recovery and revocation). In Section 3, we show some scenarios of possible attacks to the proposed protocol. Section 4 compares it to password schemes and to SQRL, which is the closest alternative in literature. Section 5 contains an analysis in terms of strengths and weaknesses of the proposed protocol. We present our conclusions in Section 6.

2. Proposed protocol

This section presents the new privacy aware web authentication protocol. The stages of the protocol life cycle are described, such as setup, data storage, login, the recovery of user data in the event of a disaster, and the revocation, if desired, of the user account in a given service.

The proposed authentication protocol requires minimal user interaction, allows for easy account recovery and revocation, requires no trusted third-party, and implements pseudonymity properties. This is achieved by creating a unique identifier and cryptographic key pair each time the user registers on a new service, plus a disaster recovery key for account recovery and revocation.

The scheme is designed to have three computer systems communicating: the user’s mobile device, the web browser to be authenticated, and the web server hosted by the service. Figure 1 shows the components of the proposed protocol and the corresponding flow of data involved in its execution.

The server authenticates the browser by sending a challenge to be signed by a corresponding private key stored in the mobile device. The offline safe site stores data for recovery and revocation.

2.1. Setup

The first step is to install the mobile application that implements the protocol. When first run, the application generates an asymmetric key pair, referred as *master keys*, and

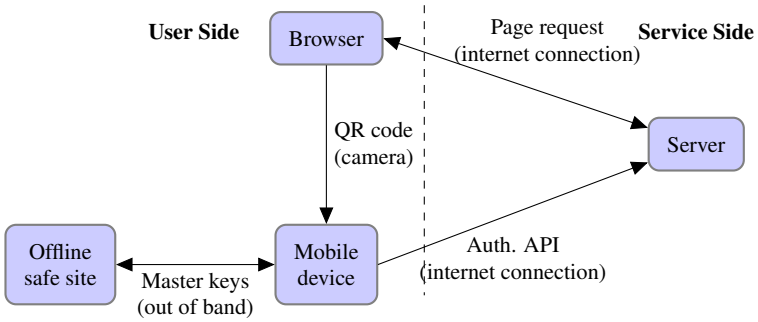

Figure 1. Authentication components.

requests the user to back it up in an offline safe site.

The public key is used to generate user ID, and encrypt the disaster recovery data. The private key is highly sensitive and used only during recovery and revocation. Therefore it is removed from the device once it is backed up.

We label the private key as *disaster recovery key* and the public key as *online master key*. We note that the online master key should be kept secret despite being the ”public” half of the key pair, though it is less sensitive than the private key.

Once the master key pair is created and backed up, the user is ready to register on a service. The protocol steps are shown in Figure 2. For simplicity, we specify this and the following protocols at a high level, without including the exact choice of cryptographic primitives or the message encoding.

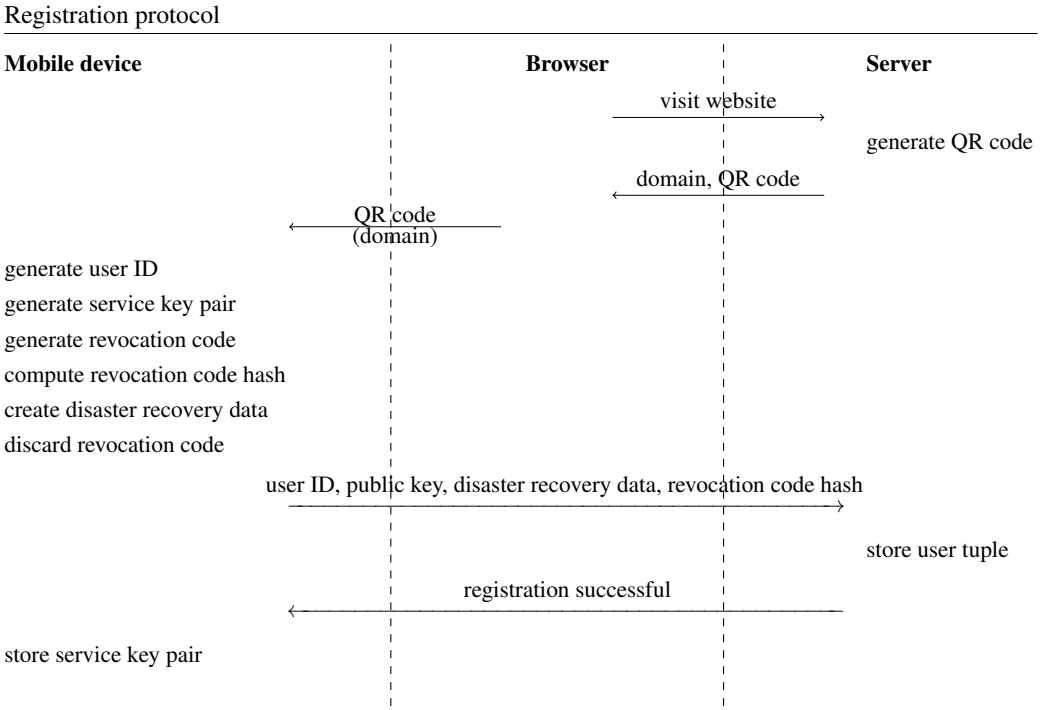

Figure 2. Registration protocol.

The protocol starts with the user visiting the service’s page, which displays a QR

code embedding the service domain name. The mobile device then scans the QR code, acquiring the domain name used to uniquely identify the service. The mobile device then generates the following set of credentials using the online master key and the service's domain name:

User ID: identifies the user on a service. The ID is the hash of the *online master key* concatenated with the *service domain name*, that is: $user\,ID = \text{hash}(online\ master\ key \parallel domain)$.

Revocation code: a large random number, to be presented to the service when revoking a key pair. Stored encrypted inside the *disaster recovery data*. This value is created during the registration process but subsequently discarded, requiring the use of the disaster recovery key to recover it. Its hash is stored in the server to verify revocations.

Service key pair: the asymmetric key pair to be used when authenticating to the service. A copy of the key pair is stored encrypted inside the disaster recovery data.

Disaster Recovery Data: It is the *service key pair* together with the *revocation code* encrypted with the *online master key*.

The user then sends the following set of credentials to the server: $\langle$ *user ID*, *disaster recovery data*, *service public key*, and *revocation code hash* $\rangle$. The mobile device only needs to keep the *domain* and *service private key*.

Before accepting the registration, the server must verify if the *user ID* is actually new. The server must not allow a user to register over another user, or replace the credentials stored.

2.2. Storage

Our proposed authentication scheme involves four different components as shown in Figure 3.

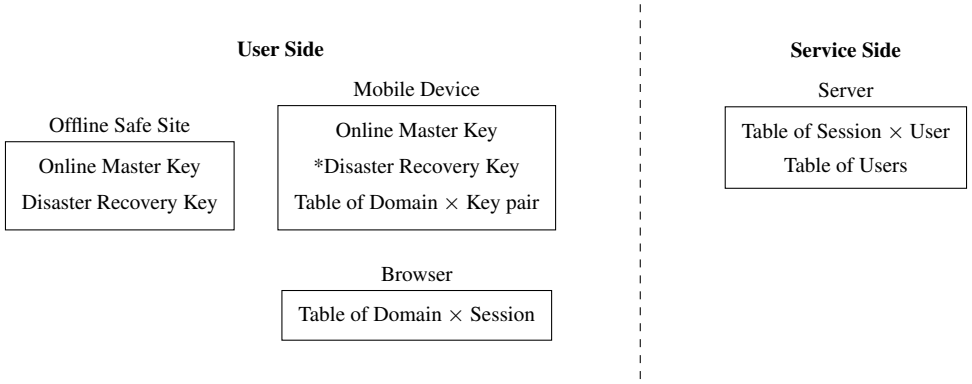

Figure 3. Components of the authentication scheme and the data stored.

The user side has three components: browser, offline safe site, and mobile device. The browser possesses a *session id* cookie for each *domain*. The offline safe site contains the *master key pair* (the *disaster recovery key* and a copy of the *online master key*).

The mobile device stores the *online master key*. The online master key is used to generate the *user ID* based on the *domain*, and to encrypt the *disaster recovery data*

during registration. The online master key should be kept secret to avoid other people registering themselves with it in new services, preventing the original user from using the same id.

The mobile device also stores the table of credentials, which maps the *domain* of each service to the corresponding *key pair*. This table is used during login, using the private key to perform a proof-of-possession protocol. Optionally, it may contain a list of the sessions authenticated by this device, useful to spot attacks and allowing remote log-out.

During revocation and recovery, the mobile device also contains the *disaster recovery key*. Due to its sensitiveness, this key should be discarded from the mobile device after use, and kept only in the *offline safe site*.

The service contains only one component, the web server. To authenticate users it must keep two tables: *users* and *sessions*. The session table associates *hashed session ids* with *user IDs*. This indicates which users are logged-in, and their active sessions (possibly more than one). The session id is sent to the browser as cookie upon its first visit, and discarded from the server afterwards. Note that an attacker with knowledge of a session id can impersonate the corresponding user during that session, which is why only the cryptographic hash of the session id is stored.

The *table of users* maps *user IDs* to their *public key*, *disaster recovery data* and *revocation code hash*. The mobile device supplies all four values during registration, and is subject to change during a revocation. None of these values are secret.

2.3. Login

The login protocol allows the user to authenticate a browser by scanning a QR code using their mobile device camera. Figure 4 shows the messages exchanged during a successful authentication.

The protocol starts when the user points an unauthenticated web browser to the service's website. The server creates a new session id, and responds the browser by sending the session id as a cookie, along with a QR code. The QR code embeds the service domain and a hash of the session id. It is necessary to hash the session id to avoid onlookers from stealing this value.

When scanning the QR code with their mobile device, the user must confirm the service domain. This confirmation prompt is required to protect the user from Man-in-the-Middle attacks. Suppose the user visited the website for a social network, but this service is malicious and wants access to the user's bank service. The malicious server can connect to the bank, posing as a client, and get a QR code for logging in the bank. When the user's browser visits the malicious social network server, it is tricked into displaying the QR code for the bank login, instead of the expected social network QR code. If the mobile device continues with this protocol they will be authenticating the bank session under the malicious server's control. By manually confirming the domain contained in the QR code the user can be sure only the expected service key pair will be used, and no unintended authentications happen.

Once the app has scanned the session id hash and confirmed the domain from the QR code, it fetches the corresponding private key from its local storage. The pri-

vate key then signs the session id hash, and the user ID is derived by hashing the online master key concatenated with the domain. The mobile device then sends the triple $\langle \textit{user ID},\ \textit{session id hash},\ \textit{signature} \rangle$ directly to the server, via Wi-Fi or cellular network, through a TLS channel [Dierks and Rescorla 2008].

The server consults its table of users, indexed by the user ID, and retrieves the public key associated. The public key is used to verify the session id hash signature. If the signature is valid, then it represents the user wishes to allow the session (controlled by the browser) to access their data on this service. The server responds with a confirmation and stores the association $\langle \textit{session id hash},\ \textit{user ID} \rangle$ in the table of sessions. The browser's session is then considered authenticated.

The login may fail if the user ID is not found or if the signature does not match. In this case, the server response to the mobile device indicates the failure, to be displayed to the user. After a successful authentication, the browser will contain a shared secret with the server (session id), the server will have an association between the shared secret and the user (sessions table), and the signature provides unambiguous intent to authenticate that session.

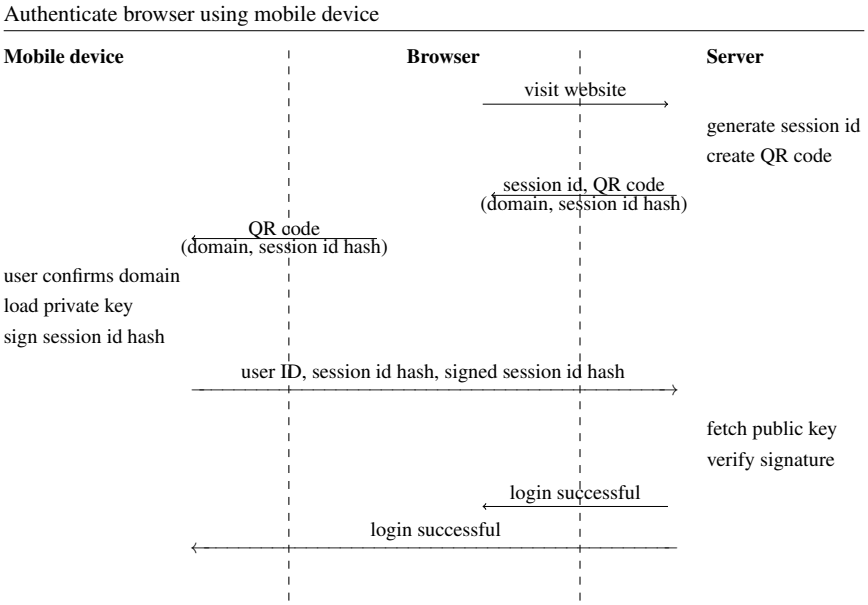

Figure 4. Login protocol.

2.4. Recovery

The authentication protocol depends on data stored in the mobile device. If this data is lost (memory wipe, device destruction or loss) the user must somehow recover access to the services. The recovery process is two step: first, the master key pair must be loaded, then the service key pairs must be recovered from each service. Figure 5 is a diagram of the process.

The master key pair was stored in the offline safe site, set up by the user during the initial enrollment. The user must retrieve this key pair and load the keys in the mobile device. The app can be used to scan a QR code of the keys, or allow the user to type their encoded forms.

Once the online master key and the disaster recovery key (both temporarily loaded into the device) are available, the user inputs the domains of the services they want to recover. For each service the mobile device will generate the *user ID* by hashing the online master key concatenated with the domain, then send a request asking for the *disaster recovery data* associated with this user ID. This value is unique and encrypted, so no authentication is required.

Recall that the disaster recovery data contains the *service key pair* encrypted with the *online* master key, to be decrypted only by the *offline* master key. Since the disaster recovery key is available in the device at the moment, the service key pair is recovered.

During this process, the disaster recovery key is kept loaded in the device. The app must clearly indicate to the user that it's on a recovery state, and that it should be exited as soon as possible. Exiting the recovery state means discarding the disaster recovery key, which brings all components to the same state as before the data was lost.

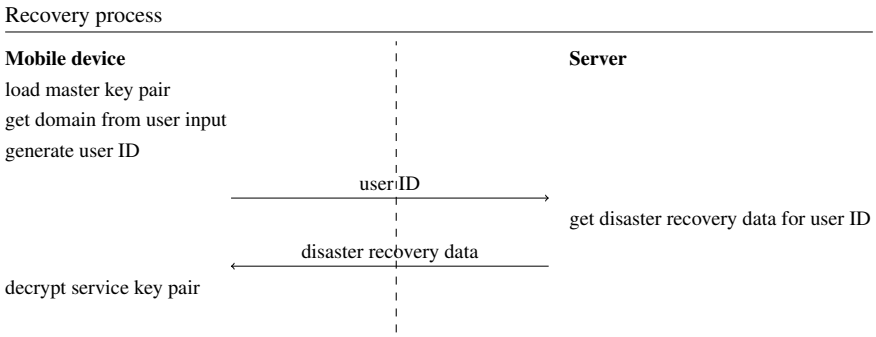

Figure 5. Recovery process.

2.5. Revocation

If the service private keys are *compromised* instead of lost, the user must revoke them. The revocation process is very similar to a recovery followed by a registration. Figure 6 shows the steps required. Due to the similarity of the processes, the steps in common are discussed more superficially.

The revocation process starts with the user loading the *disaster recovery key* in the mobile device and inputting the service domain (if more than one, just repeat the process). The mobile device requests the *disaster recovery data* from the server, and extracts the *revocation code* that was encrypted in it. Up to this point, the process is analogous to a recovery. While the mobile device waits for the disaster recovery data, it also generates the new service key pair, similarly to a registration.

To complete the revocation, the mobile device sends the following tuple to the server: $\langle$ *user ID*, *new public key*, *new disaster recovery data*, *new revocation code hash*, *old revocation code* $\rangle$ (i.e. the same tuple as registration, plus the exposed revocation code). The server can verify the *old revocation code* by hashing it and comparing to the *revocation code hash* stored during registration. If they are the same then the public key, disaster recovery data, and revocation code hash are replaced by the new values received. Additionally, any active session is closed.

At this point, the mobile device has a new set of key pairs, and the server values have been updated. The revocation is complete and the *disaster recovery key* is discarded. The security of this process is based on the security of the *disaster recovery key*, which is why it must be stored in an offline safe site.

It is also possible that the *online master key* is compromised, since it is always present in the mobile device. This key is only used to generate user IDs and disaster recovery data, neither of which allows the attacker to impersonate the user. However, the ability to generate user IDs for services allows the attacker to register themselves on new services, with the same user ID that the legitimate user would have, and a set of credentials that the legitimate user will not be able to revoke. This attack blocks the user from using this new service.

The online master key can be replaced by performing the revocation process and, when the mobile device sends the new credentials, including a ”new user ID” generated from a new master key pair. The server would effectively replace all fields for that user (user ID, public key, disaster recovery data, revocation code hash), but since the *old revocation code* is verified this is a valid process.

The only key compromise not recoverable is the compromise of the *disaster recovery key*. In this case, the attacker is able to recover the private key for all services, or revoke them and lock the legitimate user out. This is an unfortunate consequence, but unavoidable for authentication schemes; there is always some ultimate trust. The protocol has been designed to only use the disaster recovery key during emergencies (recovery and revocation), and to not require constant backups of new keys. This enables the disaster recovery key to be kept in a more secure location that is harder to access.

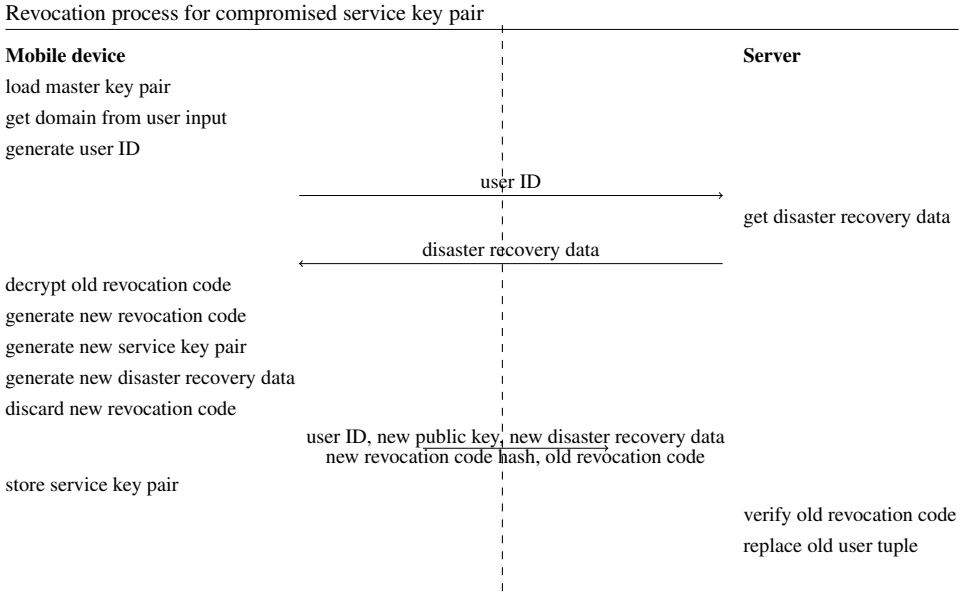

Figure 6. Revocation process to replace credentials.

3. Attack scenarios

Cryptographic schemes are designed with an attacker model in mind, and our proposal is no different. Broadly speaking there are three computer systems (mobile device, browser,

server) plus two environmental factors (network and physical environment of the user). Different assumptions are made for each.

The mobile device is assumed to be secure, but the scheme should have some resilience to physical theft and software compromise. This means that in case one of these events happens, then some security properties may be lost but only temporarily. For example, physical theft may allow the attacker to extract the private keys from the device, and authenticate on the corresponding servers. However, the user is still capable of revoking those keys and taking control back.

The browser is only assumed secure enough to be trusted with one session. Since the user will be using the browser to interact with the service, this assumption must, unfortunately, always be present. Still, the browser is not able to identify the user on unrelated services, or open new sessions by itself in the future.

The server is assumed to be completely untrustworthy for authentication purposes. It may pretend to belong to a different service, forward authentication attempts to steal sessions on a different service, try to identify the user (link accounts), or any other behavior. Additionally, a database leak should not allow attackers to impersonate or identify users.

The network is assumed to also be untrustworthy. The authentication scheme relies on TLS to ensure confidentiality, integrity, and authenticity of the server (domain name matching), even under active attackers. Similarly, the environment around the user is also considered untrustworthy. The scheme should uphold its security properties even if all user interactions are seen by the attacker (e.g. video surveillance recording key presses). Active attacks on the physical environment are not considered, and left as an exercise for spy thrillers.

4. Comparing with other authentication protocols

In this section we compare our proposal to two other authentication protocols: username and password, which is the de facto standard for most of the web; and SQRL, which is the closest protocol to our proposal.

4.1. Username and password

The most common authentication scheme on the web is the username and password combo [Bonneau et al. 2012]. This method requires the user to provide a name, uniquely identifying his account, and a memorized secret to authenticate him. Common additions include registering an email to recover lost passwords, and using the possession of a certain mobile device.

Password authentication schemes have usability problems. For example, account creation is an onerous process, requiring the user to come up with a password that is secure, memorable, unique, and obeys the service's policy, which is often contradictory among different services. Incapable of fulfilling all requirements, users choose low-entropy passwords, which are easy to guess and vulnerable to brute force attacks. Analyzing a dataset of 70 million passwords, [Bonneau 2012] concluded: "As a rule of thumb for security engineers, passwords provide roughly equivalent security to 10-bit random strings against an optimal online attacker trying a few popular guesses for large list of accounts."

Additionally, some groups of people have trouble with the creation, memorization, and typing of passwords. For instance, elderly users face special difficulties because of memory and motor skills disabilities [Renaud 2006], while blind users may have trouble interacting with password forms and complying with policies [Saxena and Watt 2009]. These accessibility challenges are active areas of research and improvements are being made, but due to the myriad of individual implementations, many of the password scheme implementations will certainly remain faulty.

Passwords also imply complexity on the service side. Passwords must not be logged, have to be hashed with a special-purpose algorithm before storage (e.g. bcrypt [Provos and Mazières 1999], scrypt [Percival and Josefsson 2016]), and verified with a constant time algorithm. Updating the hash algorithm creates further difficulties, and more advanced security requirements such as "must be different than the last 10 passwords" are common sources of additional complexity. Worsening the problem, there is a plethora of resources online that teach wrong or outdated practices.

Password based authentication schemes also suffer from several security issues. Weak or reused passwords create trivial security holes, and phishing is rampant [Bonneau et al. 2012]. If a user has forgotten their password, the usual alternatives are receiving a password-resetting link by email (creating a single point of failure for all services in the same inbox), or security questions about public or easily guessable personal data [Stuart Schechter 2009]. And if the service mishandles the passwords it receives, attackers can use data breaches to impersonate legitimate users.

4.2. SQRL

There have been a number of proposals for authentication protocols that seek to mitigate these security and privacy challenges. The best representative of these protocols is the SQRL. SQRL [Gibson 2016] is an authentication system with very similar goals and methods, and that ultimately inspired the creation of our scheme. From the point of view of the user, the setup and authentication processes are virtually identical between the two schemes; however, our proposal and SQRL diverge in the underlying cryptographic techniques used, and therefore security properties achieved. We will briefly describe the SQRL scheme, then compare to our proposal.

The SQRL authentication scheme also relies on users scanning QR codes with their mobile devices, and having the mobile device store service key pairs to perform proof-of-possession protocols. The two main differences to our scheme are in the creation of the service key pairs and the revocation facilities. There are plenty of other smaller differences, but they are usually in the level of detail of the client implementation (e.g. exact algorithm, how to protect secret values with passwords, how URLs are processed), or cryptographic differences that do not result in dissimilar security properties (e.g. SHA-256 vs. SHA-512).

In the SQRL protocol all service key pairs are derived deterministically using as seed the value $\mathrm{HMAC}_{\mathrm{SHA256}}(\mathrm{key} = \textit{master key},\ \mathrm{msg} = \textit{domain})$. The (symmetric) master key is derived from a password and a 256-bit pre-master-key value stored in the device. The password is passed through a suitably slow Key Derivation Function, the result of which is XOR'ed with the pre-master-key. A variation of the pre-master-key is also backed up offline to allow account recovery.

The Key Derivation Function used is PBKDF, with different parameters for the pre-master-key in the device and its variation in the offline backup. The online version is protected with parameters that should take half a second to unlock on a standard mobile device, while the offline backup uses parameters for about 60 seconds of processing time. This protection is to avoid brute force attacks in case an adversary acquires access to the physical device or backup.

When first introduced, SQRL lacked a revocation feature. This led to several complaints, and eventually was patched by adding the concept of an ”id lock”. Six keys are involved in the process, forming three asymmetric key pairs, one pair of which is derived from the other two using Elliptic Curve Diffie-Hellman Key Agreement (DHKA). To aid in their description we will use the following notation:

[M]: stored in the user's *mobile* device.
[S]: stored in the service's *server*.
[B]: stored in offline *backup* (analogous to our *offline safe site*).
[-]: not stored. This value is used to derive other data and then safely discarded.

Grouped by key pair ⟨ public, private ⟩, the keys used in the id lock are:

⟨**Identity Lock [M], Identity Unlock [B]**⟩**:** fixed, created during setup.
⟨**Server Unlock [S], Random Lock [-]**⟩**:** created randomly for each service.
⟨**Verify Unlock Key [S], Unlock Request Signing Key [-]**⟩**:** generated by performing either DHKA(Identity Lock [M], Random Lock [-]) or DHKA(Server Unlock [S], Identity Unlock [B]).

During account creation on a new service, the user generates the ⟨Server Unlock [S], Random Lock [-]⟩ pair. Using the Identity Lock [M] key stored on the client it then generates Verify Unlock Key [S] through DHKA(Identity Lock [M], Random Lock [-]). As indicated by the brackets notation the Random Lock [-] key is discarded, while the Verify Unlock Key [S] and Server Unlock [S] are stored in the server.

To perform a revocation the user first loads the Identity Unlock [B] from offline backup and requests Server Unlock [S]. Together they form DHKA(Server Unlock [S], Identity Unlock [B]), which is able to generate the Unlock Request Signing Key [-], whose public part is the Verify Unlock Key [S] and was stored in the server. Thus only the owner of the offline backup can sign revocation requests, and the server can verify them.

4.3. Comparison of SQRL to our proposal

Both protocols provide pseudonymous authentication in different services without third parties, mainly by using a mobile device to read a QR code, and allow for account management in those services. Also, both protocols require the backup of some important secret key in offline storage.

The main security difference is that SQRL's service key pairs are generated deterministically based on a master key that is stored in the device. We propose random service key pairs, later stored encrypted in the server (namely inside the *disaster recovery data*).

If the device is compromised, SQRL forces the user to revoke the master key. This means revoking all accounts in all services. Our proposed scheme requires revocation of

only the keys that were present in the device. It is still suggested to revoke the master key pair to avoid account collisions, but this is a nuisance instead of a security danger.

Worse still, the compromise of SQRL's master key, that is kept in the device (albeit under a password) compromises all past and future key pairs based on that master key. This means that a one-time attack that goes undetected will expose accounts created after the compromise. An attacker would have total control on all services until the user changes the master key for other reasons, potentially never.

Our equivalent key is the *online master key* (the one kept in the device). If it is compromised, the only attack on future accounts is to pre-register that user ID. This does not compromise those future accounts, and it reveals to the legitimate user that their online master key was attacked. Similarly, service key pairs not stored in that mobile device are also safe. This is because the only way to recover (or take over) an account is having access to the disaster recovery key.

Additionally, SQRL does not allow changing one individual service key pair, since all such key pairs are derived deterministically from the fixed master key. This property discourages the user from using the scheme in more flexible ways, such as sharing less sensitive service key pairs, or storing them in simpler devices (e.g. thumb drives).

The drawback of our proposed method is that if the user wishes to use more than one mobile device (or a mobile device and a desktop client), the synchronization of service key pairs must be done actively. In SQRL this is accomplished by simply sharing the master key, from which all service key pairs are derived. Naturally, the value of this feature must be weighed against the security repercussions.

An important property, unrelated to security, is the simplicity of the scheme. SQRL's scheme uses one symmetric master key, service key pairs, plus three distinct key pairs for revocation, and a non-trivial application of Elliptic Curve Diffie-Hellman Key Agreement. Our proposal has one master key pair, service key pairs, and a revocation code. We believe this simplicity is fundamental for achieving a wider adoption.

5. Analysis of the proposed protocol

An evaluation of the proposed protocol was made, considering several properties of cryptographic protocols commonly used in the analysis of authentication protocols. Table 1 shows the list of desirable properties that our proposal meets.

Table 1: Desirable Properties of our Proposed Protocol.

Property	Analysis
Memoryless	There are no values to be memorized by the human user, secret or otherwise.
Leak resistant	Access to a server's leaked database confers no advantage in impersonating them to the service it was leaked from, or identifying users across services.
Should-surfing and keylogger resistant	Secrets are handled inside the trusted device. The user never interacts directly with any secret.
Pseudonymity	The same user has unrelated identities in each service.
Timeless backups	The user does not have to update the backup after registering or revoking a service. This allows the disaster recovery key to be stored in a way that is safer but harder to access.

Updatable keys	The keys to a service can be changed without changing the master key. This is a trivial property for most schemes, but hard to achieve without updating backups.
No third-party	The scheme does not rely on any trusted third party, even during revocation.
Phishing resistant	It is resistant to Phishing attacks since the domain is explicitly part of the authentication process.
Scalable	Authenticating with a service has the same difficulty independent from how many accounts the user has. Similarly, authenticating one user costs the same for the server regardless of the total number of users registered.
Low friction	Users don't have to type any name, password, or code, and once authorized the login is nearly instantaneous.
Flexible	The user may self-impose a PIN, password or biometric check; detached keys allow usability-security trade-offs; key pairs can be stored with different security levels. This also applies to the disaster recovery key.

Unfortunately, it also has a few downsides. Table 2 shows these drawbacks.

Table 2: Undesirable Properties of our proposed Protocol.

Property	**Analysis**
Device carry	The cryptographic key pairs have to be kept in a personal trusted device. The loss of this device can bring difficult problems.
Backup required	The user has to safely back up the disaster recovery key.
User unfamiliarity	The authentication process has less friction, but it's unfamiliar to users. This increases the rate of errors and makes users more susceptible to social engineering attack.

6. Conclusion

Web authentication is usually based on passwords, and to some extent, emails. These protocols, however, have security, privacy and usability flaws. The requirement of having high-entropy unique passwords is at odds with human memory, enabling various kinds of attacks, and the manual input aspect is responsible for the creation of innumerous types of phishing. The use of an email address for disaster recovery leads to single-point-of-failure scenarios and needlessly reveals the user's identity. Lastly, passwords are a constant source of headaches for developers, administrators and the users themselves.

The SQRL protocol is a popular proposal to this problem, making use of ubiquitous smartphone possession. By using a mobile device to scan a QR code, the protocol enables users to log in without any usernames or passwords, and a much higher level of security. By using an offline backup instead of email account, privacy and single-point-of-failure concerns are also addressed.

However, the SQRL protocol creates cryptographic key pairs deterministically based on a master key, which is kept on the mobile device. This makes the protocol inflexible, and creates security concerns. Additionally, the disaster recovery protocol (revocation + recovery) is extremely complex, complicating its analysis and implementation.

To remedy these flaws, a new protocol for authentication on the web was introduced. From the point of view of a user, our proposal is similar to the SQRL protocol. However, our proposal creates random key pairs and is much simpler during disaster recovery.

Compared with other authentication schemes, we find that our proposal is one of the best candidates for wide adoption on the web. It has vastly superior security properties than pure password schemes, several advantages against more sophisticated schemes, and improves significantly on similar proposals.

The proposed protocol, described in this work, was implemented and verified in practice. The code containing its implementation can be obtained in `https://github.com/boppreh/frango-app`.

References

Bonneau, J. (2012). The science of guessing: analyzing an anonymized corpus of 70 million passwords. In *2012 IEEE Symposium on Security and Privacy*, pages 538–552. IEEE.

Bonneau, J., Herley, C., Van Oorschot, P. C., and Stajano, F. (2012). The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *2012 IEEE Symposium on Security and Privacy*, pages 553–567. IEEE.

Bonneau, J. and Preibusch, S. (2010). The password thicket: Technical and market failures in human authentication on the web. In *WEIS*.

Borchert, B. and Günther, M. (2013). Indirect nfc-login. In *ICITST*, pages 204–209.

Dierks, T. and Rescorla, E. (2008). The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard). Updated by RFCs 5746, 5878, 6176, 7465, 7507, 7568, 7627, 7685.

Gibson, S. (2016). Sqrl. `https://www.grc.com/sqrl/sqrl.htm`.

Lindemann, R., Baghdasaryan, D., and Tiffany, E. (2014). Fido universal authentication framework protocol. *Version v1. 0-rd-20140209, FIDO Alliance, February*.

Mansfield-Devine, S. (2015). The ashley madison affair. *Network Security*, 2015(9):8–16.

Percival, C. and Josefsson, S. (2016). The scrypt Password-Based Key Derivation Function. RFC 7914 (Informational).

Provos, N. and Mazières, D. (1999). Bcrypt algorithm. In *Proceedings of 1999 USENIX Annual Technical Conference*. USENIX.

Renaud, K. (2006). A visuo-biometric authentication mechanism for older users. In *People and Computers XIX—The Bigger Picture*, pages 167–182. Springer.

Saxena, N. and Watt, J. H. (2009). Authentication technologies for the blind or visually impaired. In *Proceedings of the USENIX Workshop on Hot Topics in Security (HotSec)*, volume 9, page 130.

Stuart Schechter, A.J. Brush, S. E. (2009). It's no secret: Measuring the security and reliability of authentication via 'secret' questions. In *Proceedings of the 2009 IEEE Symposium on Security and Privacy*, Berkeley, CA, USA. IEEE Computer Society.

van Dijk, J. (2014). A closer look at sqrl. Technical report, University of Amsterdam.

Xu, F., Han, S., Wang, Y., Zhang, J., and Li, Y. (2015). Qrtoken: Unifying authentication framework to protect user online identity. In *Cyber Security and Cloud Computing (CSCloud), 2015 IEEE 2nd International Conference on*, pages 368–373. IEEE.

Yan, Q., Han, J., Li, Y., and Robert DENG, H. (2012). On limitations of designing usable leakage-resilient password systems: Attacks, principles and usability. In *19th Network and Distributed System Security Symposium (NDSS)*.

