SecureSafe: A Highly Secure Online Data Safe

Industrial Use Case

Marc Rennhard
Zurich University of Applied Sciences, Switzerland
marc.rennhard@zhaw.ch

Michael Tschannen
DSwiss AG, Switzerland
michael.tschannen@dswiss.com

Tobias Christen
DSwiss AG, Switzerland
tobias.christen@dswiss.com

Abstract

In this paper, we present the core security architecture of SecureSafe, a cloud service that provides a highly secure online storage for sensitive data. The architecture combines best practices in cryptography and security protocols with novel approaches to offer high security while providing good usability and performance. More than two years of successful operation with many satisfied customers and no security incidents demonstrate the soundness of the architecture. In addition to granting insight into the security architecture of a field-proven service to provide inputs for other services that have similar requirements, there's another motivation for this paper: We believe that to increase trust in cloud services, service providers should be more open about internal security details – although this is in contrast to what typical cloud service providers do today.

Categories and Subject Descriptors D.4.6 [**Operating Systems**]: Security and Protection – Authentication, Cryptographic controls

General Terms Design, Security

Keywords Cloud Security, Security Architecture, DoubleSec

1. Introduction

In this section, we first talk about privacy challenges in the cloud and how trust in cloud computing can be increased. We then briefly introduce SecureSafe and describe the main contributions of this paper.

1.1 Privacy Challenges in Cloud Computing

All systems that contain information about individuals, their location, or their transactions, carry the risk that unauthorized users can search for or reveal facts that are unknown or unknowable to others and hence violate the privacy of individuals.

So-called cloud services (in particular SaaS, but also PaaS, and IaaS) increase this risk due to their inherent characteristics. Because the business model is most often built on low revenue margins (or is even offered for free), the service provider needs to attract large amounts of users, which increases the value that potential attackers see in the service and which increases the impact that potential administrative errors or design faults can have.

1.2 Trust in Cloud Computing

While the consumer is unaware of the increased risks to which cloud services expose him, the consumer is intuitively afraid that giving away data means losing control over the data. In the Internet, each online storage service promises highest security and claims to be trustworthy. But how should a consumer understand that the best among these service providers achieve security to his data that goes far beyond what he can achieve himself, but that others expose him to undesired risks? The user trusts his provider based on its size, its number of years in business, and other attributes such as responsiveness and professional look and feel. Other aspects such as personal recommendations or reference statements by institutional users also increase the trust. However, the real building stones of trustworthiness such as the strength of the security architecture or the sincerity of the personnel cannot be judged by most end-users but only by institutional users and security experts. Unfortunately, it is still rarely seen that service providers make available internal details about how security is exactly provided. If anything, providers talk about buzzwords such as "high security data centers" or "128-bit encryption" and the like, without providing more details.

1.3 Contributions

In this paper, we present the core security architecture of SecureSafe[1], a highly secure online storage for sensitive data (e.g. documents and passwords). The system was built to enforce the claim that not even physical access to the data storage system would allow access to the customers' data. In analogy to the vaults in the cellars of the banks, we refer to the system with the term "data safe" to reflect that the system provides much stronger security controls than typical home usage security scenarios would allow. SecureSafe is offered as a service to consumers worldwide and serves users from well over 70 countries. While desktop usage (where SecureSafe is accessed through the web browser) is steadily growing, it is the mobile usage[2] that drives most of the growth. User surveys show that highest security com-

MPM'12, April 10, 2012, Bern, Switzerland.

[1] For more details about the services offered by SecureSafe, refer to http://www.securesafe.com.

[2] SecureSafe currently supports the iOS platform (iPod, iPhone, iPad) with custom apps, Android will soon be supported as well.

bined with usability is the most important reason for the users to choose SecureSafe. The high security level also motivates Swiss banks to trust SecureSafe, which offer direct electronic delivery of bank statements into the SecureSafe of their customers.

The motivation for this paper is twofold. First, we explicitly want to present details about the security architecture of SecureSafe, to increase trust in the service. This is in contrast to what typical cloud service providers do today. Second, and more importantly for the security community: We believe we have designed and implemented a sound security architecture by combining best practices and novel approaches in a reasonable way to achieve both high security and good usability while preserving good performance for all operations. In this respect, the paper should provide a good use case of applied cryptography in a practical and successful system that can provide valuable inputs for other security-sensitive services.

The remainder of this paper is organized as follows: In Section 2, we present core aspects of the security architecture of SecureSafe. In Section 3, we present DoubleSec, a novel authentication scheme to provide user-friendly two-factor authentication for accessing SecureSafe with mobile devices. In Section 4, we conclude our work.

2. Security Architecture

In this section, we describe the core security architecture of SecureSafe. We focus on user authentication and the cryptographic measures employed by SecureSafe to provide a high level of security while maintaining good usability.

2.1 Requirements

The two most important aspects when designing SecureSafe were providing both great usability and high security. It is well known that achieving both at the same time is not easy. For instance, user certificates on smartcards are widely assumed to provide stronger authentication than passwords, but user certificates are harder to use because they, e.g., require software installation and are not well suited to be used on mobile devices such as smartphones.

The two key aspects with respect to security of SecureSafe are user authentication and data protection on the server, and we will focus on them in this section.

With respect to user authentication, it was clear that we must use a password-based approach, as anything else is neither widely known nor accepted among users. But we all know the various security problems with passwords such as password re-use, password guessing, phishing attacks, dictionary attacks etc., so we had to look for a solution that maintains the simplicity of passwords while avoiding or at least reducing its shortcoming.

With respect to data protection, one central question is whether the service provider has access to the data or not. Most likely, users expect that their data is cryptographically protected on the server and that an intruder that gets physical access to the servers cannot easily access the data. In addition, it should be difficult for personnel of the service provider to access the users' data. Ideally, only the user can provide the necessary key material to access and decrypt his own data. As we don't want to put the burden of any additional key management on the user, the keys should be derived from his credentials, i.e. his password. But this directly leads to another problem: What happens if the user looses his password? Simply telling the user that "we are sorry, all data is lost" is not an option, so a mechanism to still being able to access the own data after the password has been lost should be incorporated.

As one can see from this discussion, offering a service that is both secure and usable is very challenging. Nevertheless, we attempted to achieve both and specified the following requirements:

- Secure, password-based login
- Secure transmission of sensitive data
- Secure server-side storage that prevents SecureSafe or third party personnel from gaining access to any user's data or account
- Secure and tamper-proof account recovery
- Extensibility: Features such as document sharing among SecureSafe users should be possible to be integrated in the future without significant changes to the architecture
- Usage of well-known and proven cryptographic standards
- Performance-optimized use of cryptography, in particular with respect to minimizing the number of public key computations

In the remainder of this section, we describe how SecureSafe manages to fulfill all these requirements to provide high security and good usability. Note that as the basis for communication security, the well-established Transport Layer Security protocol (TLS, the successor of SSL) with server-side certificates is used and we won't go into the details of this.

2.2 The Login Protocol

One common way to categorize attacks targeting the user credentials is the following:

- Attacks targeting the user or the client computer
- Attacks targeting the communication channel
- Attacks targeting the server

Attacks of the first category are hard to mitigate completely because users tend to write down, re-use or insecurely use their credentials (e.g. on a compromised computer). Two-factor authentication helps against some of these threats and we'll come back to this later. The other two, however, can be addressed effectively by employing appropriate protocols for the login process. One such protocol is Secure Remote Password (SRP) [1], which is a password-based authentication protocol and a popular choice by the IETF for strong password protocols. With respect to the security of data stored in the cloud, SRP has some very interesting characteristics:

- No password-equivalent storage: SRP does not store simple password-equivalent data (e.g. hashes of passwords) on the server. This makes offline dictionary attacks (getting access to the "password" database and cracking them offline) very expensive.
- No password-equivalent information sent over the wire: SRP does not send any password-equivalent information during login. Instead, SRP is a so-called "verifier-based" protocol where users don't provide their passwords to the service during login, but just prove knowledge of them. This provides strong resistance against both passive and active attacks where the attacker can read or manipulate all traffic exchanged during authentication.
- Session key establishment: In addition to providing authentication, SRP establishes a strong shared secret during each login process.
- Performance: Despite its increased complexity compared to simple challenge-response based protocols, SRP can be used even on (computationally relatively weak) mobile devices in the sense that the user does not experience a significant delay during login.

- Mutual authentication: SRP both authenticates the server against the client and the client against the server.

Due to the fact that SRP only needs a salt[3] and a verifier[4] to be stored on the server to authenticate a user, it is even possible for a user to register an account without the server having to see the password at any time. From a security perspective, this behavior is not only desirable, but also effective to ensure maximal privacy for the user's data. As such, it is possible to derive a secure encryption key from the password and use this key to encrypt a user's data, without the cloud storage provider (its personnel) being able to derive the key (as it does not know the password) and decrypt the data. Thanks to the fact, that SRP does not exchange any secret information during login, it can even be used over insecure protocols or within a broken TLS session (i.e. where a MITM is splitting the TLS session and intercepting and manipulating messages).

Despite all its advantages, SRP is still password-based and suffers from the same problem as all password-based protocols: It provides only single-factor authentication, meaning an attacker may be able to get a user's password by a social engineering attack (e.g. phishing). Considering the privacy protection SecureSafe wants to provide, authentication based only on passwords is therefore not sufficient. Consequently, we use two-factor authentication where the second login code is delivered to the user by SMS. This approach is often identified as mTAN (mobile TAN), although we use it only for login and not to confirm transactions as often used in e-banking systems. Once the SRP-based password authentication has successfully completed, SecureSafe sends the login code to the user's mobile phone (the number of which is registered in the user's profile). The user receives the code, enters it in the login form and sends it to server. If the server receives the correct logon code, the user is completely authenticated and is granted access to his account.

2.3 Server-Side Storage

From a privacy-perspective, the service provider should store the users' data in a way that prevents it (i.e. its personnel) from getting access to the data. The obvious solution to this would be to encrypt documents always on the client and only send the protected documents to the server. However, SecureSafe was always designed to be available also on mobile devices such as smartphones, where performing cryptographic operations of possibly large documents still implies a significant performance hit, which would drastically reduce usability of the service.

But there's another important argument against such an approach. Even if all data were encrypted on the client before being sent to the server, it would not really increase protection from a malicious service provider as the user cannot verify what the software exactly does[5] each time when using the service. Likewise, it does not protect from a malicious software developer who manages to include code in the application without being noticed that allows him to access the users' data. As a result, even if a service provider claims to encrypt all data on the client with keys that are never accessible by the server, the user still has to trust the service provider.

The only way to implement such a client encryption in a trustworthy way would be by performing the client-side encryption by software and keys completely outside of the SecureSafe service[6]. However, requiring such an additional software or key management component is too cumbersome for the vast majority of users and is therefore unacceptable from a usability point of view.

We came up with the following approach that protects the users' documents on the server whenever possible while providing good performance and usability when a user wants to access his documents:

- All documents are encrypted with user- and document-specific keys while they are stored on the server. This makes it virtually impossible for personnel (e.g. malicious operators) or a thief to get access to the users' data in case they manage to get physical access to the servers.
- When a user submits a document, it is encrypted on the server. If he requests a document, it is decrypted on the server and delivered to the user.
- Using appropriate key management, the necessary keys to encrypt or decrypt a document of a specific user are only available to the server during an active session of that user. This minimizes the time window any user's keys are available to the SecureSafe application on the server. Note that the keys are only available to the application (in memory) itself and not to any administrator of the application, which minimizes the risk of key exposure.

The last step implies that the user must provide access to the cryptographic keys. The most natural choice is – as already discussed in Section 2 – to base this on the user's password, which implies that without knowing a user's password, access to his data is not possible.

When designing SecureSafe, we had two additional requirements: Despite the cryptographic protection, sharing documents between users should be possible and the cryptographic operations should be performed in an efficient manner[7]. While the first requirement directly leads to the usage of public key cryptography, the second one does the opposite: public key operations are computationally much more expensive than secret key operations. To achieve both goals, we have combined both technologies in an efficient way.

Taking into account all these requirements, we came up with the key management[8] structure as in Figure 1:

- Symmetric User Key: This key is derived from the user-chosen password. It isn't stored anywhere and has to be computed on the client during each login process. The protocol SecureSafe uses for this is PBKDF#2 [2], a "Password-Based Key Derivation Function", which allows generating a reasona-

[3] Used to make pre-compiled dictionary attacks infeasible.

[4] Used to verify the user's password during login.

[5] E.g. whether the software really encrypts the data on the client or whether the password is not simply sent to the server along with the data.

[6] Of course, even with this approach, one still requires, i.e. has to trust, that the additional third party components does not contain any malicious components.

[7] In practice, that just means reducing public key operations to a minimum.

[8] For all secret key operations, AES with 256 bit keys is used. For public key operations, RSA-2048 is used.

bly strong secret key even if the underlying password is of limited strength[9].

- Public Key pair: Using this key pair would not be necessary for single-user cloud-storage only, but it significantly facilitates secure document sharing. The private part of this key pair is encrypted with the user key and stored on the server.
- Symmetric Master Key: This key was introduced for performance reasons so that the document-specific symmetric keys don't have to be encrypted asymmetrically. This key is encrypted with the user's public key and stored on the server. It can only be unlocked with the user's private key.
- Symmetric Document Keys: These keys are unique per document and are used for document encryption. All keys are encrypted using the symmetric master key and stored on the server, along with the encrypted documents.

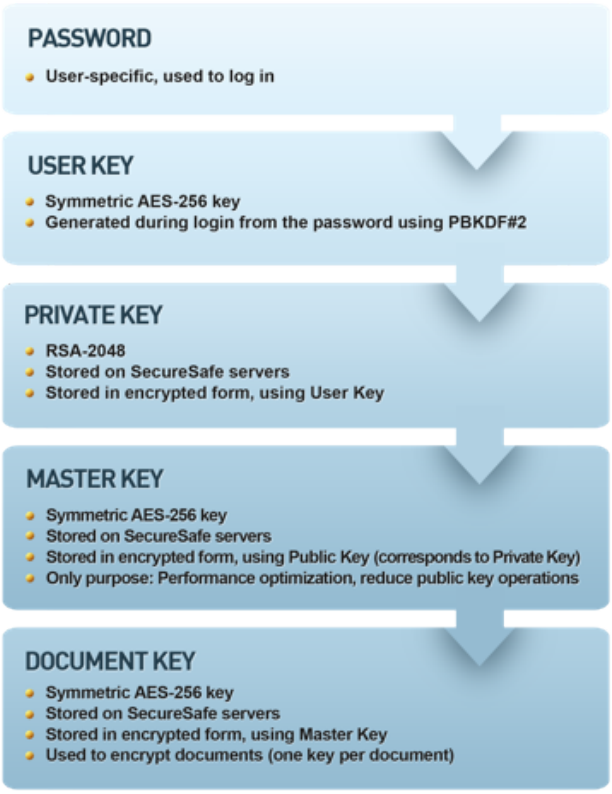

Figure 1. Key Management

At first glance, this key chain appears more complex than necessary. However, in practice, it is has proven to be extremely efficient for several reasons:

- Deriving a symmetric encryption key from the user's password makes sure that access to the data is only possible with the knowledge of the password, which is only available during active sessions.
- The usage of a public key enables document sharing, more details follow below.
- The symmetric master key makes sure that expensive public key operations are reduced to a minimum.
- Symmetric per-document keys make document-specific settings and document-specific sharing options possible.

2.4 Sharing and Document Provisioning

As described above, we have introduced a "public key cryptography layer" to enable document sharing. If a SecureSafe user A wants to share a document with another user B[10], this works as follows:

1. While using the service, user A specifies that a document should be made available to user B.
2. SecureSafe generates a new document key and encrypts the document[11].
3. The document key is not encrypted using B's master key (which is not possible at that time because the master key is only accessible during active sessions of B), but with B's public key.
4. During the next active session of B, the document key is decrypted using B's private key and re-encrypted with B's master key – which means the document is now inserted in B's normal key chain.

The same mechanism can also be used for document provisioning. Here, any external entity can put documents into the safe of a SecureSafe user B by sending them to the service, which replaces step 1. Steps 2 – 4 remain unchanged. A typical application of this is a secretary that pushes documents into the safe of her manager.

To summarize, the key structure used by SecureSafe is powerful with respect to enabling features that are usually hard to achieve in a truly secure way. Of course, sharing and document provisioning are always possible if the service provider does either not encrypt the stored data or uses an internally accessible encryption key (i.e. an arbitrary key that is not derived from the user's password), however these approaches do not really ensure the privacy of the stored data because it would be much easier for personnel (or a thief getting physical access) to access the users' data.

2.5 Account Recovery

When encrypting all data based on a key derived from the user's password, we have the problem of account recovery if the user forgets his password: There is no way for SecureSafe to recover a user's account as it does not know its users' passwords, which prohibits accessing their data.

One way to make account recovery possible is to give the user an additional token that can be used to unlock the key chain to recover the account at any time by himself. SecureSafe has therefore introduced the so-called *Recovery Code*, a login token with

[9] This is achieved by using salt and repeating a cryptographic computation (e.g. a hash) many times to significantly increase the work factor for an attacker that tries to guess the password.

[10] Note that sharing is not limited to 1:1-relationships – a user A could also share a document with n other users at the same time. All of these n users will then have to get access to the symmetric document key.

[11] This assumes the document is copied for the recipient. One can also do true sharing (only one copy of the document exists on the server) by making the original document key available to the recipient.

high entropy that can solely be used to recover a user's account in case of password loss. The user can print a sheet with the recovery code after account creation or at any later time and lock it away.

Unlocking the key chain with the recovery code works similar as a normal login. In fact, the recovery code is nothing else but the combination of an alternative username and password:

- 8 characters *username*: randomly chosen.
- 27 completely random characters *password*, which guarantees high entropy.

With the username and password encoded in the recovery code, it is possible to perform an SRP login[12] because when the code was created, the corresponding SRP-salt and -verifier values were also created and stored on the server. After the user has entered the recovery code, a fully automated SRP login used to authenticate the user is started in the background. Just like with the usual user login, a shared secret is established during this process.

Using PBKDF#2, an alternative user key (called *Recovery Key*) is generated on the client and sent to the server. SecureSafe is then able to reach the user's account and data because the user's private key is not only encrypted with the user key, but also with the recovery key. Once this is done, the user can pick a new password and the private key is encrypted with the new corresponding user key, which results in the key chain being accessible with the new password.

2.6 Summary

Coming back to the security requirements defined at the beginning of this section, SecureSafe fulfills them by offering strong authentication, communication security, and secure data storage and access as follows:

1. TLS with server certificates provides the basis for communication security.
2. The client uses SRP for user authentication, additionally a strong per-session secret is established. This secret is used to additionally encrypt sensitive pieces of information (e.g. the user key) to maintain security even if the underlying TLS session is broken.
3. The client sends the second login factor (mTAN), encrypted with the prior established shared secret, to the server.
4. Based on the password, the client computes the user key (using PBKDF#2) and sends it, encrypted with the shared secret, to the server.
5. The server decrypts the user key using the shared secret.
6. The server decrypts the user's private key using the user key.
7. The server decrypts the user's master key using the decrypted private key.
8. When the user requests a document, the server decrypts the document key using the master key, decrypts the document, and sends the document to the client.
9. To store a document on the server, the server receives the document, creates a document key and encrypts the document, and encrypts the document key with the master key.
10. After logout, the server forgets the user key and all other decrypted user-specific keys, which prevents access to the user's documents. This means the server only has access to the necessary key material during active sessions of a user.
11. In case of password loss, the user can recover his account in a secure way by providing the recovery code he can print and lock away after account creation.

The requirements with respect to performance and extensibility could also be fulfilled, mainly by relieving the client from performing computationally expensive operations (e.g. bulk data encryption or decryption) and by using a layer of public key cryptography in the key chain while keeping the number of public key operations to a minimum.

3. DoubleSec

As mentioned in Section 1, SecureSafe can not only be used with a "real" web browser, but also with mobile devices by means of the SecureSafe app. To maintain overall security, user authentication when using the mobile app should also employ two-factor authentication as otherwise, an attacker could use this mobile access to possibly compromise a user's account with less effort.

At first glance, this seems trivial: simply use the mTAN-based approach for the second login factor also on the mobile device. However transferring the mTAN-approach to mobile apps – especially on smartphones – doesn't work well, mainly because it would be cumbersome or even impossible to be used as it requires the user to switch between apps[13]. In addition, even if switching between apps is supported (which is usually the case today with modern smartphones), the user has to memorize the received login code before switching back to the SecureSafe app and entering it, which is not user-friendly at all. As a result, we had to come up with an authentication scheme that is better suited for mobile apps from a usability perspective while providing a security level comparable to login codes received by SMS.

We name the solution we came up with to achieve this *DoubleSec*. DoubleSec makes use of a field-proven idea that has been used by SSH (SecureShell) over years, which is called *baby duck* (or *key-continuity*) security model [3][4]. Basically, the baby duck security model says that "if we can securely authenticate one session (e.g. the first one) and derive a shared secret, the secret can be re-used to authenticate later sessions". Our proposal initializes the first shared secret using a code received by SMS[14] and then subsequently calculates each of the following tokens using a keyed one-way function, e.g. HMAC(previous_secret, key).

The first shared secret (which we name DoubleSec-token) is calculated as follows:

$$S(1) = HMAC(mTAN, SessionKey)$$

As there is no previous secret, the code received by SMS (mTAN in the formula above) is used. Since we already have derived a shared secret between client and server during SRP login, we are using this secret as the key in the HMAC computation (Session Key). The computed token S(1) is then sent to the

[12] Note that this isn't a normal login, i.e. the user won't be logged in afterwards, but just the way SecureSafe authenticates the user during account recovery.

[13] Between the mobile app and the SMS inbox.

[14] Using the mTAN approach once for initialization is acceptable from a usability point of view.

server in encrypted form[15] using again the shared secret (SessionKey):

encrypt(SessionKey, S(1))

The server decrypts this and verifies if the correct S(1) was delivered. If this is the case, the first DoubleSec authentication has successfully been completed and both server and client store the current DoubleSec-token (S(1)). During subsequent logins, the new DoubleSec-token S(n) is computed based on the previous token S(n-1) as follows:

S(n) = HMAC(S(n-1), SessionKey)

As can be seen from the formula above, the new token can only be computed if one has access to the previous token – an attacker not being able to get access to the mobile device will therefore not be able to compute this token. Transmission of S(n) from the client to the server works in exactly the same way as above with the first token:

encrypt(SessionKey, S(n))

3.1 Security Analysis and Comparison with mTAN

As mentioned above, the two-factor authentication solution for mobile devices should not be weaker than the "main two-factor authentication solution" to not make attacks easier. For SecureSafe, this means that DoubleSec should provide comparable attack-resistance as a login code received by SMS (mTAN).

With two-factor authentication, the second login factor is usually "something the user has", assuming the first factor is a password, i.e. "something the user knows". This is the case when mTAN is used because possessing (or having access to) the mobile device makes it trivial to receive and submit the login code. Exactly the same is true with DoubleSec: To get S(n), the user must know S(n-1), which is only possible (under practical assumptions) by having access to the mobile device. As a result, both mTAN and DoubleSec require possession of the device to successfully perform authentication.

The fact that the mobile app itself has access to the DoubleSec-token[16] also has no impact: DoubleSec (just like mTAN and similar approaches) intends to secure access to the user's account and not the user's mobile device. However, there exists a small difference with respect to the "critical app" on the mobile device[17]: With DoubleSec, it is the mobile app itself (the SecureSafe app in our case); with mTAN, it is the SMS app.

With mTAN, an attacker is able to log in only once if he manages to gain access to the second login factor (i.e. the code received by SMS), assuming he already knows the user's password. With DoubleSec, the attacker is able to log in multiple times because by gaining knowledge of S(n-1), he will be able to compute S(n) as well as S(n+1), S(n+2)[18], etc. However the next time the legitimate user tries to log in, a loss of synchronization (the tokens on client and server do not match) will be detected, which leads to a re-initialization of the user's mobile device (by mTAN) by agreeing on a new S(1).

From a usability perspective, DoubleSec exceeds mTAN on mobile devices because there is no need for the user to manually switch between apps and to enter a login code – in fact, providing the second authentication happens completely transparent (with the exception of DoubleSec initialization) for the user as using the "correct" mobile device is enough. Additionally, it has lower operational costs for the service provider because there is no need to send SMS messages to the customers for each login attempt.

One shortcoming of DoubleSec in comparison to mTAN is that transaction confirmation, which is increasingly used in e-banking solutions, is not possible with this solution. However, if the only requirement is providing a second authentication factor during login, DoubleSec can be used as a very nice usability-optimized mTAN-derivate with comparable security.

4. Conclusions

SecureSafe demonstrates that offering a highly secure cloud service while providing good usability and performance is feasible. We achieved this by designing a security architecture that combines and makes use of best practices in cryptography and security protocols. We also came up with with a novel, user-friendly two-factor authentication scheme (DoubleSec) for mobile devices. More than two years of successful operation (since June 2009) with high customer satisfaction and no security incidents demonstrate the soundness of the overall approach.

We motivate other service providers to be more open about the internal workings and in particular about the security architecture of their cloud services. This will allow unbiased analysis by experts and provides private and institutional customers with information about the true security a service offers. Especially in the case of young companies that cannot count on reputation gathered during long-term operation, this can greatly help to increase trust.

Acknowledgments

This work was partly funded by the Swiss Confederation's innovation promotion agency CTI (projects nr. 9536.1 PFES-ES, 11304.1 PFES-ES, and 12565.1 PFES-ES).

References

[1] T. Wu. *The SRP Authentication and Key Exchange System*. RFC 2945, September 2000.

[2] B. Kaliski. *PKCS #5: Password-Based Cryptography Specification Version 2.0*. RFC 2898, September 2000.

[3] Frank Stajano and Ross Anderson. *The Resurrecting Duckling: Security Issues in Ad-Hoc Wireless Networking*. In Proceedings of the 7th International Workshop on Security Protocols, Springer-Verlag Lecture Notes in Computer Science No.1796, April 2000.

[4] Frank Stajano. *The Resurrecting Duckling – What Next?* In Proceedings of the 8th International Workshop on Security Protocols, Springer-Verlag Lecture Notes in Computer Science No.2133, April 2000.

[15] This is done for additional security because an attacker being able to intercept one DoubleSec-token (e.g. by breaking the underlying TLS session) is able to calculate the next valid DoubleSec-token (assuming he also knows the user's password).

[16] Access control mechanisms provided by the operating system of modern mobile devices are considered adequate to protect the stored DoubleSec-token from being accessed by other apps.

[17] Assuming we face malware trying to gain access to the second login factor.

[18] This is possible because we assume the attacker knows the password, which allows him to complete the SRP login and derive the shared secret (SessionKey), which allows computing S(n) when knowing S(n-1).