On Storing Private Keys in the Cloud

Jonathan Anderson and Frank Stajano

University of Cambridge
Computer Laboratory
15 JJ Thomson Avenue, Cambridge, UK
{jonathan.anderson,frank.stajano}@cl.cam.ac.uk

Abstract. Many future applications, such as distributed social networks, will rely on public-key cryptography, and users will want to access them from many locations. Currently, there is no way to store private keys "in the cloud" without placing complete faith in a centralised operator. We propose a protocol that can be used to share secrets such as private keys among several key recovery agents, using a weak password, in a way that prevents insiders from recovering either the private key or the password without significant collusion. This protocol will enable the safe storage of private keys online, which will facilitate the advent of secure, decentralized, globally-accessible systems.

1 Introduction

Today, online services such as social networks can rely on passwords for authentication because they are provisioned in a centralised way: user passwords, or hashes thereof, can be tested against values stored on the provider's servers. The integrity of the personal information stored in these networks is attested to by the service provider, which implicitly asserts that "you should believe that Alice said this because I know that the person who typed it in knew Alice's password."

We have proposed that in the future, distributed social networks will provide such attribution via public-key cryptography, taking centralised providers out of the identity verification game [1]. The trouble is, public-key cryptography requires the secure storage of private keys, and a system that requires users to carry a Trusted Computer Base (TCB) in order to post status updates is unlikely to compete well with traditional, familiar, token-less authentication via password.

Other online services have a similar problem. Webmail providers may typically be trustworthy, but as Chinese dissidents have discovered, they sometimes co-operate "regularly and efficiently" with authorities [2] in order to operate in nations with widely varying standards of free speech and tolerance for political diversity. Many online services allow or require users to register an e-mail address as a backup authentication channel in case they lose their password, but acquiring access to this e-mail account gives an adversary the keys to the proverbial kingdom: if she can read your e-mail, she can reset all of your other passwords. Online services could use stronger means of authenticating users, e.g.

B. Christianson and J. Malcolm (Eds.): Security Protocols 2010, LNCS 7061, pp. 98–106, 2014.

SSH keys or client certificates, but the problem is, again, that forcing users to carry a TCB for authentication is likely not a smart business strategy.

One could propose a system in which private keys are stored encrypted on a server, but this only solves the problem if the user can remember a cryptographically strong key with which to decrypt their private key.

The problems that we wish to solve are:

Problem 1. We wish to allow users to recover very sensitive information, such as private keys, from an untrusted online service without needing to remember cryptographically strong secrets.

We discuss the authentication properties of this problem in Section 2, in particular justifying the use of weak passwords in a system that uses public keys. We propose a concrete threat model in Section 3 in the context of a distributed social network, describing security threats from both insiders and outsiders.

A related problem is *plausible deniability*:

Problem 2. We wish for the key recovery service to provide outsiders and malicious insiders with as little information as possible about the users of the service, including their identities and knowledge of whose keys are stored where.

We propose a series of cryptographic protocols in Section 4 which answer Problem 1, allowing our user to recover his private key from untrusted sources, and move towards answering Problem 2, requiring progressively less trust in the other principals involved.

2 Passwords

The most widely-deployed authentication system for online services is also the least expensive: "something you know", typically a username and (likely weak) password. Authentication credentials are revealed to the computer that they are typed on and the centralised service's provider, both of which have the ability to violate the user's security expectations and must thus be trusted absolutely.

Since we propose using public-key cryptography for message integrity, it is tempting to require users to carry some form of Trusted Computing Base, perhaps storing private keys on a mobile phone. If, however, we want to enable the use case, "Alice wants to use an online service on a friend's computer with a proper screen," then either:

1. the computer and Alice's phone must be tethered e.g. via Bluetooth, or
2. Alice's private key must be made available to the computer.

The first of these options may be acceptable for services such as online banking, but for photo sharing or other social applications, only the second, with its uninterrupted workflow, is likely to encourage widespread adoption. Alice is going to supply the local machine with her private key, so there is little reason for her to carry it around with her. Rather, she should be able to retrieve it from "the

cloud," using nothing but what she knows to authenticate herself, and what she knows is likely to be a very weak password.

It is true, then, that we require Alice to gauge the trustworthiness of the computer that she is sitting in front of. This is, on the face of it, a ludicrous requirement, but it is a requirement which is currently accepted by many, many users: despite the existence of key-loggers and other malware, people use online services via computers owned by friends, airport lounges and other Internet cafés. This decision is made, implicitly or explicitly, based on the user's perception of the computing environment, sensitivity to risk, education in security issues, etc.

In short, to use computers is to trust computers. We do not attempt to solve this problem, but we do reduce the number of principals which the user must trust. In the current model, there is an implicitly trusted service provider at the other end of the TLS tunnel whose security practices may be unknown. We replace absolute faith in this single principal with limited trust in a number of principals, reducing the trust that we are required to place in any one.

3 Principals

Bob, B, is a user of a distributed online social network. This network has no central, trusted authority to map Bob's digital identity onto his real-world one, so his client software proves who he is using Bob's private key.

Wishing to access the network from abroad without carrying any hardware, Bob prepares a private key K_B^{-1}, which he advertises to his peers along with a set of constraints[1]. He then enlists friends Alice, Alexa, Alicia, etc. $(A_0, A_1, A_2, \dots)$ to act as "key recovery agents" on his behalf: each will run code on their computers that stores some portion of the private key, which will only be given out to Bob later if he authenticates himself with the weak password k_B [2].

3.1 Insiders

We assume that Alice, Alexa, etc. are not particularly malicious — they will try to guard secret information that they are entrusted with and they will not collude *en masse* — but they not particularly trustworthy either. In particular, any or all of these key recovery agents may be susceptible to curiosity or malware.

[1] For instance, Bob could prepare one key per calendar week and declare them valid for signing instant messages but not sharing photos. This would ensure that, if a key is compromised by an untrustworthy computer, the damage which can be done is limited.

[2] We assume that Bob is able to recover his public key, K_B, and those of his agents ($K_{A_0}, K_{A_1}, \dots$) from a public source, but that the only secret information which can be used for authentication across sessions, computers, etc. is k_B. This means that the public keys which Bob recovers can be used to provide confidentiality of communications, but not authentication of principals.

Curiosity. We assume that, if Bob gives his private key and/or password to Alice in the clear, she might look at it out of sheer curiosity. In fact, Alice might even be willing to mount a dictionary attack if it is easy enough to do. This definition of curiosity is different from the classical sense of "honest-but-curious:" an honest-but-curious participant will complete a protocol faithfully, but may gossip with the other participants after the protocol run in order to learn additional information [3]. We assume that an agent will not collude with $k-1$ of n others in order to learn additional secrets such as Bob's private key or password.

Malware. We assume that many, or even all, agents may be infected with malware that can read any stored information and eavesdrop on any network communication. In our favour, however, we assume that the malware which infects the various agents is not capable of collusion.

Malice. We assume that, without resorting to large-scale collusion, one agent may attempt impersonating Bob to another agent in order to learn secrets such as Bob's private key K_B^{-1} or password k_B.

3.2 Outsiders

We assume that there is a malicious outsider, Eve, who can observe all communication between Bob and his agents, and that she might attempt to impersonate either Bob or a subset of agents. Her goal may include learning Bob's password and/or private key or ascertaining which agents Bob has stored his key with, or even simply whether or not Bob uses the system.

4 Protocols

We present several protocols, beginning with a "straw man" and building incrementally towards the final goal of key recovery (Problem 1) which thwarts attacks from both insiders and outsiders (Problem 2). Each protocol is more complex than its predecessor, but also mitigates more attacks.

4.1 Trusted Storage

The first, essentially trivial protocol we consider is to store the private key on a trusted server in the clear. This is analogous to a common practice in backup authentication: if Bob loses his Flickr password, he can have a reset token sent to his webmail account, which is hosted by a trusted provider. Since Bob is able to download Alice's public key from a public source, the protocol is simply:

$$
\begin{aligned}
B &\to A : \{B, K_t\}_{K_A} \\
A &\to B : \{n\}_{K_t} \\
B &\to A : \{k_B, n\}_{K_A} \\
A &\to B : \left\{K_B^{-1}\right\}_{K_t}
\end{aligned}
$$

in which K_t is a temporary key used by Bob to provide confidentiality until he recovers his private key K_B^{-1}, and n is a nonce selected by Alice, which prevents Eve from performing a replay attack should she later learn the value of K_t.

Attacks. This protocol prevents outsiders from conducting successful dictionary attacks, since Alice can simply limit the rate of incoming password guesses. Eve's dictionary attack must be performed online, so Bob's password can be as weak as an English word.

There are, however, two very obvious attacks against the system: firstly, insider Alice can simply read Bob's private key and password in the clear, and secondly, outsider Eve can impersonate Alice to Bob in order to learn Bob's password—Bob can download a K_A which is attributed to Alice, but without a shared secret, he has no way to verify that it is actually Alice's key.

4.2 Semi-trusted Storage

A slight improvement on the previous scheme is the "semi-trusted storage" scheme, in which Alice does not hold Bob's password and private key in the clear, but rather a cryptographic hash of the password with her own public key, $h(K_A|k_B)$, and the private key encrypted using the weak password, $\left\{K_B^{-1}\right\}_{k_B}$. The protocol is very similar to that of the trusted storage scheme:

$$
\begin{aligned}
&B \to A : \{B, K_t\}_{K_A} \\
&A \to B : \{n\}_{K_t} \\
&B \to A : \{h(K_A|k_B), n\}_{K_A} \\
&A \to B : \left\{\left\{K_B^{-1}\right\}_{k_B}\right\}_{K_t}
\end{aligned}
$$

After receiving message 3, then, Alice verifies that $h(K_A|k_B)$ matches her stored copy, and returns $\left\{K_B^{-1}\right\}_{k_B}$ to him in message 4.

Attacks. This protocol prevents a truly disinterested Alice from reading Bob's password and key, but it does little to stop a curious Alice or a clever Eve: either can mount an offline dictionary attack, against either the private key (Alice) or the password hash (Alice or Eve). Since the password is assumed to be weak, we expect that such an attack would succeed without very much effort.

4.3 Threshold Encryption

A logical extension to this protocol is for Bob to spread his private key across several agents, Alice, Alexa, Alicia, etc. using a standard k of n secret sharing scheme [4]. In this case, Alice stores $h(K_{A_i}|k_B)$, which is a version of $h(K_A|k_B)$ above, but personalized to her, and instead of the private key K_B^{-1} she stores D_i,

which is one portion of the key K_B shared according to the thresholding scheme. The protocol between Bob and any of his agents is now:

$$
\begin{aligned}
&B \rightarrow A_i : \{B, K_t\}_{K_{A_i}} \\
&A_i \rightarrow B : \{n\}_{K_t} \\
&B \rightarrow A_i : \{h\left(K_{A_i} | k_B\right), n\}_{K_{A_i}} \\
&A_i \rightarrow B : \{D_i\}_{K_t}
\end{aligned}
$$

Attacks. This addition to the protocol prevents Alice (or Alexa, Alicia, etc.) from reading Bob's private key or performing a dictionary attack against its encrypted form, but there is nothing to prevent her or impostor Eve from attacking the weak password by brute force and, once successful, impersonating Bob to other key recovery agents in order to assemble Bob's public key.

4.4 Collision-Rich Password Hashing

We can improve on this protocol further by using collision-rich functions [5]. In this case, Bob authenticates to Alice via a hash function that intentionally has many collisions. This means that Bob's authentication to Alice is weaker, but two important purposes have been served:

1. Alice does not need to know k_B or even $h\left(K_{A_i} | \, k_B\right)$, but merely (see the Insider Dictionary Attack, below) $h\left(K_{A_i} | \, k_B\right) \bmod N$, and
2. Eve, should she impersonate Alice, will not be able to perform a successful offline dictionary attack[3] (see the Outsider Dictionary Attack, below).

The protocol is now:

$$
\begin{aligned}
&B \rightarrow A_i : \{B, K_t\}_{K_{A_i}} \\
&A_i \rightarrow B : \{n\}_{K_t} \\
&B \rightarrow A_i : \{h\left(K_A \mid k_B\right) \mod N, h\left(n\right)\}_{K_{A_i}} \\
&A_i \rightarrow B : \{D_i\}_{K_t}
\end{aligned}
$$

where N is a number chosen by Bob when he enlists Alice's help as a key recovery agent.

Attacks. Having eliminated the most straightforward attacks, we introduce more interesting ones.

[3] Collision-rich hash functions frustrate dictionary attack because the input can be grouped into equivalence classes, any member of which will produce the same output as any other member. Which member is actually the correct password must be determined by online dictionary search, which will be expected to fail if the classes are large.

Insider Dictionary Attack. Using this protocol makes it more difficult for Alice to obtain Bob's password via offline dictionary attack, since many passwords generate the same hash. If the number of possible passwords is x, then the number of password candidates that Alice will be able to learn from an offline dictionary attack is:

$$\begin{cases} x & N \geq x \\ \frac{x}{N} & N < x \end{cases}.$$

Since Alice's hash takes both Bob's password and Alice's identity and input, the hashes stored at each key recovery agent are independent. Thus, if Alice attempts to impersonate Bob to Alicia, she will have no likelihood information about the $\frac{x}{N}$ candidate passwords.

Still, there are $n-1$ other agents that Alice can go to and try to confirm Bob's password. If we allow Alice α guesses at each, she will expect to guess Bob's password if

$$\alpha \cdot (n-1) \geq \frac{x}{2N}$$
$$N \geq \frac{x}{2\alpha \cdot (n-1)}.$$

If we assume that Bob uses $n = 9$ recovery agents with $\alpha = 5$ attempts allowed and a dictionary of 10,000 equally-probable passwords[4], then Alice would expect to be able to guess his password if $N \geq 125$.

Large-N Attack. If Eve impersonates Alice, she could send a large value of N to Bob. He should ignore values which lie outside of some reasonable range related to the strength of his password.

Outsider Dictionary Attack. If N is small—in order to minimize the success probability of insider attack—there is a non-trivial chance that Eve may be able to obtain Alice's portion of the shared secret by impersonating Bob and authenticating to Alice with a randomly-selected $\mathbf{x} \leq N$. This does not give Eve access to Bob's key, however, only one portion of it. In order to recover Bob's key, she needs k portions of the secret. This is still difficult, because having successfully guessed $\mathbf{x} \leq N$, Eve is now effectively in the same position as Alice, attempting to employ the Insider Dictionary Attack.

Impostor Identity Disclosure Attack. If Eve impersonates Alice, she will not be able to learn Bob's password easily, for the reasons given in the section on

[4] Of course, in a natural-language or chosen-password dictionary, all words are not equally probable. If the dictionary is small, however (e.g. 10,000 words / 4-digit PINs), random assignment is possible. If user-chosen passwords are essential for memorability reasons, password strengthening techniques might be of use, even if they are of little help against a conventional offline dictionary attack.

the Insider Dictionary Attack. Neither will she be able to perform a completely successful middle-person attack, since the hash $h(K_A|k_B)$ is bound to Alice's public key. She would, however, learn that Bob has stored his private key with Alice by virtue of the fact that he has attempted authentication, so Problem 2 is still not satisfied.

4.5 Collision-Rich Identity Hashing

In order to prevent identity disclosure in the face of the Impostor Identity Disclosure Attack, we can add one more layer of complexity: collision-rich hashing of Bob's identity. To prevent such hashes from acting as *de facto* persistent identifiers while allowing agents to perform their service for many clients, Bob should first send Alice a collision-rich hash which takes both his identity and Alice's as input, using a very low modulus of his choice. If Alice cannot disambiguate him using this hash, he can try again using a different modulus, reducing the size of the set of possible client identities to one (possibly after several iterations).

$$
\begin{aligned}
&B \to A_i : \{h(A_i|B) \bmod N, N, K_t\}_{K_{A_i}} \\
&A_i \to B : \text{try again} \\
&B \to A_i : \{h(A_i|B) \bmod N', N', K_t\}_{K_{A_i}} \\
&A_i \to B : \{n\}_{K_t} \\
&B \to A_i : \{h(k_B|A) \bmod N'', h(n)\}_{K_{A_i}} \\
&A_i \to B : \{D_i\}_{K_t}
\end{aligned}
$$

Attacks. This protocol gives the same (low) probability of successful dictionary attack by insider or outsider, and it also counters the Impostor Identity Disclosure Attack, for the same reasons. It does not, however, mitigate the potential for traffic analysis: a technically competent adversary could observe that Bob has connected to Alice.

5 Related Work

Lomas and Christianson [5] have used collision-rich hash functions to provide assurance of OS kernel integrity when booting across an untrusted network.

SRP [6] is a protocol which allows clients to authenticate to servers using zero-knowledge proofs, but it provides little protection against a malicious server unless the user's password is strong enough to resist an offline dictionary attack.

6 Conclusion

We have developed a protocol which allows a user to store a private key "in the cloud," using the services of several key recovery "agents" who are expected to

be curious and perhaps even malicious, but who are trusted not to collude *en masse*. The user is able to authenticate himself to these agents using a very weak password, yet the agents are unable to learn what the password is, and malicious outsiders are unable to even verify that a particular user avails himself of the recovery service.

Using such a key storage service enables the development of online services that rely on public key cryptography without requiring users to carry key material with them. We hope that these future services, such as distributed social networks, will allow people to do the things that they want to do online without needing to place absolute faith in any one software developer or service provider.

Acknowledgements. We would like to thank Joseph Bonneau, Bruce Christianson and Michael Roe for their helpful review of this work.

References

1. Anderson, J., Diaz, C., Bonneau, J., Stajano, F.: Privacy-Enabling Social Networking Over Untrusted Networks. In: The Second ACM SIGCOMM Workshop on Social Network Systems (WOSN 2009), pp. 1–6 (May 2009)
2. Blakeley, R.: Yahoo in second Chinese blogger row. TimesOnline (January 2006)
3. Beimel, A., Chor, B.: Secret Sharing with Public Reconstruction. In: Coppersmith, D. (ed.) CRYPTO 1995. LNCS, vol. 963, pp. 353–366. Springer, Heidelberg (1995)
4. Shamir, A.: How to share a secret. Communications of the ACM 22 (November 1979)
5. Lomas, M., Christianson, B.: Remote Booting in a Hostile World: To Whom Am I Speaking? IEEE Computer, 50–54 (1995)
6. Wu, T.: The Secure Remote Password Protocol. In: The 1998 Internet Society Network and Distributed System Security Symposium, pp. 97–111 (1998)