Strengthening Public Key Authentication Against Key Theft (Short Paper)

Martin Kleppmann[1(✉)] and Conrad Irwin[2]

[1] Computer Laboratory, University of Cambridge, Cambridge, UK
mk428@cl.cam.ac.uk
[2] Superhuman Labs, San Francisco, USA
conrad.irwin@gmail.com

Abstract. Authentication protocols based on an asymmetric keypair provide strong authentication as long as the private key remains secret, but may fail catastrophically if the private key is lost or stolen. Even when encrypted with a password, stolen key material is susceptible to offline brute-force attacks. In this paper we demonstrate a method for rate-limiting password guesses on stolen key material, without requiring special hardware or changes to servers. By slowing down offline attacks and enabling easy key revocation our algorithm reduces the risk of key compromise, even if a low-entropy password is used.

1 Introduction

Although passwords are the prevalent authentication mechanism on the internet today, there are some niches in which public key authentication systems have been successfully adopted. For example, SSH public key authentication [11] is widely used for remote login to servers, TLS client certificates [3] are used in some countries for access to public services [8], and FIDO U2F [10] provides 2-factor authentication for web applications.

In these protocols, a user account is associated with a public key, and a client authenticates itself to a server by computing a digital signature using the corresponding private key. The private key is stored on the client device (perhaps using a cryptographic hardware module), so the signature implements a machine-to-machine authentication protocol (a "something you have" factor). Since the device may be lost or stolen, an additional human-to-machine authentication step is employed to prevent an attacker using the key: for example, a password or biometric information can be used to unlock or decrypt the private key.

However, passwords and biometric identifiers are typically low in entropy, making them susceptible to offline attacks if a device is stolen. Our contribution in this paper is a scheme for storing an RSA private key in a way that makes it harder for an attacker to make use of stolen key material. We build upon the mRSA key-splitting scheme [2,6], which provides instantaneous key revocation, and extend it with a novel protocol for rate-limiting password guesses, which has the effect of slowing down offline attacks against stolen key material. In this

F. Stajano et al. (Eds.): PASSWORDS 2015, LNCS 9551, pp. 144–150, 2016.
DOI: 10.1007/978-3-319-29938-9_9

work we limit our attention to RSA keys, but we hope to extend our approach to support other public-key cryptosystems such as ECC in future.

1.1 Threat Model

In our scenario, a client stores an RSA private key encrypted with a password. The client wishes to authenticate itself to a server as username r. We assume the server already knows which RSA public key belongs to which username. We require that all communication occurs over TLS, and that the client verifies the identity of the server using its existing PKI certificate or a pinned public key.

Our adversary is an active network attacker, but we assume that our use of TLS prevents the attacker from eavesdropping or tampering with messages. The attacker can steal encrypted private key material from a client device (e.g. by stealing the physical device or by compromising it remotely). We assume the attacker can trick the user into accessing fake services, but cannot trick the user into revealing the key encryption password to the attacker. We assume that the user is aware when a device has been lost or compromised, and that they are willing to take steps to revoke it.

In Sect. 2.2 we introduce a semi-trusted service called the *mediator*. We assume that data stored at the mediator is not accessible to the adversary who can steal private key material from clients. The mediator cannot authenticate on the client's behalf, and it need not be trusted as an authority.

2 Revocable Public Key Authentication

In this section we review an existing technique for instant revocation called *mediated RSA* (mRSA) [2,6]. We demonstrate it by example, using a simplified version of the FIDO protocols [10]. We build upon mRSA in Sect. 3 to explain our algorithm for rate-limiting password guesses.

2.1 Basic RSA Authentication

A client has a username r and an RSA private key (n, d), where n is the modulus and d the private exponent. The server knows the corresponding public key (n, e) for r, where e is the public exponent. To authenticate, the client first requests a fresh challenge c from the server. It then constructs an RSA signature s:

$$s = H(c \parallel cb \parallel u \parallel r)^d \mod n \ , \tag{1}$$

where u is the URL of the server, and cb is the TLS channel binding [1] or Origin-Bound Certificate [4] of the connection between server and client. The channel binding prevents MITM and replay attacks. H is shorthand for the EMSA-PSS-Encode operation (hashing and padding) defined in PKCS#1 [5].

The client then uses TLS to send the authentication request (s, c, u, r, n, e) to the server at URL u, which verifies that s is a valid PKCS#1 signature of

$c \parallel cb \parallel u \parallel r$ using the public key (n, e), that c and u are valid for this server, that the channel binding matches, and that (n, e) is a public key for user r.

An adversary who steals the private exponent d can easily impersonate the client. A common solution is to encrypt d with a key derived from a password using a slow KDF such as scrypt [9]. However, password entropy is often low, so this is not sufficient to stop an attacker with significant computing resources.

2.2 The Mediator Service

To prevent theft of the private exponent d, we split it into key fragments using the mRSA method [2,6]. It is based on the identity:

$$s = m^d = m^{d_a + d_b} = m^{d_a} m^{d_b} \mod n \ . \tag{2}$$

The private exponent d is split into d_a, which is an integer drawn from the uniform random distribution $U(0, d)$, and $d_b = d - d_a$. Fragment d_a is encrypted with the user's password and stored on the client device a, while fragment d_b is stored on a remote server called the *mediator*. If the same user has multiple client devices, d can be split in a different way for each device, with the counterpart of each device's fragment stored on the mediator. It would be easy to split d into three or more summands, but we focus on the two-fragment case.

After the key has been split, a client device must work together with the mediator in order to construct a valid signature of the form in (1). When device a wants to generate a signature, it sends a message m to the mediator:

$$m = H(c \parallel cb \parallel u \parallel r) \ . \tag{3}$$

The request is sent over TLS and authenticated as described in Sect. 3.2. The mediator uses its key fragment d_b to calculate a response:

$$resp = m^{d_b} = H(c \parallel cb \parallel u \parallel r)^{d_b} \mod n \tag{4}$$

and returns *resp* to client device a. Now, a can calculate the RSA signature s:

$$s = H(c \parallel cb \parallel u \parallel r)^{d_a} \cdot resp = m^{d_a} m^{d_b} = m^d \mod n \ , \tag{5}$$

and thus authenticate with the server at URL u.

If a device's key fragment is stolen, it can instantly be revoked by deleting the counterpart fragment from the mediator, rendering the stolen fragment useless. This deletion request can be authenticated by another device owned by the same user, as discussed in Sect. 3.2. This implies that a user must enrol at least two physical devices with the mediator, so that the remaining device can revoke a lost device. A paper print-out of the key can serve as last resort in case all devices are lost or destroyed.

The mediator need only be partially trusted. It cannot authenticate as the user without the cooperation of one of the user's physical devices. The user only needs to trust the mediator to be always online, to keep key fragments safe from

attackers who steal devices, and to correctly delete key fragments when the user requires key revocation. The user's privacy is protected by hashing the message $c \parallel cb \parallel u \parallel r$ before sending it to the mediator, so the mediator does not learn which services the user is logging in to, or which usernames they are using.

From the point of view of a server that uses public key authentication, the mediator does not even exist: a server simply verifies the RSA signature, and does not care how that signature was constructed. This is in contrast to federated login systems such as OpenID, where the relying party must trust the identity provider.

3 Rate Limiting Password Guesses

In the original proposal of mRSA [2], requests to the mediator are not authenticated. In this section we show that by adding authentication, we can strengthen mRSA to prevent offline attacks against stolen private key material.

Consider an attacker who has stolen a client device on which key fragment d_a is stored, encrypted with password *pass*. The attacker reads the encrypted fragment $E(d_a, pass)$ from the device, and mounts an offline attack by repeatedly trying a password guess $pass'$ (based on a dictionary or brute force) and computing $D(E(d_a, pass), pass')$ until the correct d_a is found.

However, an offline attack on the password requires the attacker to be able to determine whether a decryption attempt has indeed yielded the correct d_a. The following protocol ensures that an attacker must make a request to the mediator for every decryption attempt in order to determine whether it is correct. This allows the mediator to limit the rate of decryption attempts, giving the user more time to revoke the stolen device, even if the password is fairly weak.

3.1 Key Fragment Encryption

Let k be the RSA key length. A key fragment d_a can be encoded as a k-bit string, using zero padding for the most significant bits, since $d_a < d < n < 2^k$. This k-bit string can then be encrypted into a k-bit ciphertext *efrag*, using a stream cipher and a key derived from a password. For example, we can use the scrypt KDF [9] and AES-128 in CTR mode [7] as stream cipher:

$$efrag = \mathrm{AESCTR}(ctr, \mathrm{scrypt}(pass))_{\{0\ldots k-1\}} \oplus d_a \quad , \tag{6}$$

where *ctr* is a random nonce that is stored in plaintext and incremented by AESCTR for each block of key stream. An attacker who has stolen *efrag* and *ctr* may guess a password $pass'$, and compute a guess d'_a of the key fragment:

$$d'_a = \mathrm{AESCTR}(ctr, \mathrm{scrypt}(pass'))_{\{0\ldots k-1\}} \oplus efrag \quad . \tag{7}$$

If the password guess $pass'$ is incorrect, d'_a is a uniformly distributed pseudorandom number between 0 and 2^k. We deliberately choose *not* to use authenticated encryption, because the MAC would tell the attacker whether the password guess was correct, making an offline attack easy.

Note that d_a is drawn from a uniform distribution $U(0, d)$, whereas d'_a is drawn from $U(0, 2^k)$. Since $d < 2^k$, the distributions are different, which leaks some information: smaller values of d'_a are more likely to be correct than larger ones. Apart from this bias, there are no particular features that distinguish the correct d_a from a random bit string.

To quantify this assertion, we generated 50,000 RSA keys ($k = 2048$ bits) using OpenSSL, and drew a uniformly distributed random d_a with $0 \leq d_a < d$ for each private exponent d. Table 1 shows the bias in the most significant bits of d_a when encoded in k bits. The key fragments had an entropy of 2047.05 bits, implying that 0.95 bits of information are leaked by the bias. This can be used by an attacker to prioritize guesses that are more likely to be correct, but an attacker cannot rule out password guesses from examining d'_a alone.

Table 1. Probability that bit i of d_a is 1, when encoded in $k = 2048$ bits

i	2047	2046	2045	2044	2043	2042	2041	2040	2039
Probability	0.070	0.240	0.340	0.406	0.446	0.469	0.482	0.493	0.499

3.2 Authenticating Requests to the Mediator

Furthermore, to prevent offline attacks on encrypted key fragments, requests to the mediator must be authenticated. To see why this is the case, consider an unauthenticated mediator that accepts any message m and returns $m^{d_b} \mod n$ as in (4). An attacker could use this response to test whether a password guess $pass'$ is correct, by using (7) to compute d'_a and checking whether $m^{d'_a} m^{d_b} \mod n$ is a valid RSA signature.

To prevent this, a client must prove to the mediator that it knows the correct password *pass* without revealing the password or the decrypted key fragment d'_a. This is accomplished by the following protocol:

1. When the client requests the mediator to compute a partial signature on a message m, it must also include a partial signature s_m using d'_a:

$$m = H(c \parallel cb \parallel u \parallel r) \tag{8}$$

$$s_m = H(m \parallel cb_m)^{d'_a} \mod n \tag{9}$$

   where cb_m is a channel binding [1] of the TLS connection between the client and the mediator. Note that cb is between client and server, whereas cb_m is between client and mediator.
2. The mediator uses its own channel binding cb'_m of the connection from the client to compute:

$$s_m \cdot H(m \parallel cb'_m)^{d_b} = H(m \parallel cb_m)^{d'_a} \cdot H(m \parallel cb'_m)^{d_b} \mod n \tag{10}$$

and checks whether the result is a valid signature of $m \parallel cb'_m$ for the user's public key (n, e). This check succeeds if $d'_a = d_a$ (i.e. the user's password was correct), and if $cb'_m = cb_m$ (preventing MITM and replay attacks).

3. If the signature is valid, the mediator computes

$$resp = m^{d_b} = H(c \parallel cb \parallel u \parallel r)^{d_b} \mod n \tag{11}$$

as before, and returns it to the client. If the signature is not valid, the mediator returns "bad signature".

When a password-guessing attacker receives a "bad signature" response, it learns that the password guess *pass′* was incorrect, but it does not gain any additional information that would help it determine whether any other password guess *pass″* is correct or not. Thus, the attacker must make a request to the mediator for every guess. If the mediator receives too many requests for a signature with a particular fragment within a short time, it returns an error.

The same mechanism can be used to authenticate key revocation: the mediator only processes a revocation request for a device if it is authenticated by another device of the same user. This avoids relying on a central authority.

4 Conclusion

The security of key-based authentication is only as good as the protection of the private key material. In this paper we extend mRSA, an existing method for revocation of private keys, by authenticating requests to the mediator.

Our algorithm ensures that an attacker who has stolen a password-encrypted key, and wants to guess the password, must make a request to a mediator for every attempt. This gives the mediator the opportunity to limit the rate at which passwords can be tested, giving the user more time to revoke the lost device's key. No special hardware is required, and the server just performs standard RSA signature verification, making our approach compatible with existing systems.

Acknowledgements. We thank Alastair R. Beresford and the reviewers for their helpful feedback.

References

1. Altman, J., Williams, N., Zhu, L.: Channel bindings for TLS. IETF RFC 5929, July 2010
2. Boneh, D., Ding, X., Tsudik, G., Wong, C.M.: A method for fast revocation of public key certificates and security capabilities. In: Proceedings of the 10th USENIX Security Symposium, pp. 297–308, August 2001
3. Dierks, T., Rescorla, E.: The Transport Layer Security (TLS) protocol version 1.2. Network Working Group RFC 5246, August 2008
4. Dietz, M., Czeskis, A., Balfanz, D., Wallach, D.S.: Origin-bound certificates: a fresh approach to strong client authentication for the web. In: 21st USENIX Security Symposium, pp. 317–332, August 2012

5. Jonsson, J., Kaliski, B.: Public-key cryptography standards (PKCS) #1: RSA cryptography specifications version 2.1. Network Working Group RFC 3447, February 2003
6. Kutyłowski, M., Kubiak, P., Tabor, M., Wachnik, D.: Mediated RSA cryptography specification for additive private key splitting (mRSAA). IETF Internet Draft, November 2012
7. Lipmaa, H., Rogaway, P., Wagner, D.: Comments to NIST concerning AES modes of operations: CTR-mode encryption, September 2000
8. Parsovs, A.: Practical issues with TLS client certificate authentication. In: Network and Distributed System Security Symposium (NDSS), February 2014
9. Percival, C.: Stronger key derivation via sequential memory-hard functions. BSDCan 2009, May 2009
10. Srinivas, S., Balfanz, D., Tiffany, E., Czeskis, A.: Universal 2nd factor (U2F) overview. FIDO Alliance Proposed Standard, May 2015
11. Ylonen, T., Lonvick, C.: The Secure Shell (SSH) authentication protocol. Network Working Group RFC 4252, January 2006