Addressing the Problem of Undetected Signature Key Compromise

Mike Just*† Paul C. van Oorschot*

Abstract

Suppose that messages have been signed using a user's signature private key during the period of time after *a key compromise but* before *the compromise is detected. This is a period of* undetected key compromise. *Various techniques for detecting a compromise and preventing forged signature acceptance are presented.*

Attack protection is achieved by requiring a second level of authentication for the acceptance of signatures, based on information shared with a trusted authority, independent of the signature private key and signing algorithm. Alternatively, attack detection is achieved with an independent sychronization with the authority, using a second factor/adaptive (non-secret) parameter. Preventing forged signature acceptance subsequent to the detection is achieved by the use of a cooling-off or latency period, combined with periodic resynchronization.

Keywords: *digital signatures, key compromise detection, second level authentication, timestamping.*

1 Introduction

The digital signature is the digital counterpart to the physical, handwritten signature. Each permits authorization in the name of the corresponding identity. A handwritten signature permits authorizations corresponding to the particular name that is being signed. A digital signature private key may be used, together with additional controls, to allow authorizations in the name of the identifier associated with the corresponding public key certificate. The compromise of the private key results in a loss of the exclusive control over associated privileges, and allows impersonation.

Once it is known that a key has been compromised, suitable recovery actions may be taken to prevent further damage. For example, various means for key revocation may provide some recourse. However, even if all practical systems in use today contained effective revocation means (which, by the way, is not the case), such revocation requires knowledge that a compromise has occurred. Therefore, it is not surprising that to date, the problem of protecting against the forgery of signatures resulting from an *undetected* compromise of a user's signature private key has not, to our knowledge, even been considered in the open literature, let alone solved in any way.

Consequently, one of the major contributions of this paper is to introduce, and present a first study of the problem of *undetected key compromise.* Moreover, and perhaps counter-intuitively, we are able to provide solutions which prevent even an attacker who has obtained or deduced (by any means, for any signature algorithm) a user's signature private key, from being able to have fraudulently produced signatures accepted by an unsuspecting recipient. It is our hope that this paper will cause others to consider this problem as a new challenge, leading to further proposals of practical interest.

Overview and Summary of Results

In Section 2, we more precisely define the events relevant to a signature key compromise and elaborate on various types of attacks that result in such a compromise. Section 3 first reviews some previous solutions and techniques useful for reducing the damage resulting from a signature key compromise. Subsequently, we proceed to overview new solutions in which independent means are used to authenticate the signing user. The signing private key remains necessary for the production of digital signatures, but is no longer sufficient for signature acceptance.

In Section 5, we elaborate on a first solution in which a secondary (independent) authentication mechanism is used for enhanced protection against an undetected key compromise. This method requires a second factor secret key. In Section 6 stronger techniques are presented which use a secondary (independent) synchronization method to allow the legitimate signer to detect when forged signatures have been produced. This method uses a second factor adaptive parameter, of which only the authenticity need be maintained (vs. the privacy). Combining a periodic check-in by the legitimate user with a cooling-off period for

*Entrust Technologies, 750 Heron Road, Suite E08, Ottawa, ON, K1V 1A7, Canada.

†This work was completed while the first author was with the School of Computer Science, Carleton University, Ottawa, ON, K1S 5B6, Canada, and partially supported by a NSERC grant. Contact author: mike.just@entrust.com

the acceptance of signatures allows for the detection to be enhanced so that forged signatures will not be accepted by a recipient. In Section 8, we conclude with some discussion clarifying the usefulness of our solutions.

2 Definitions, Assumptions and Motivation

Consider a user u who owns a private key s_u used only for signing messages, and public key p_u used for signature verification, under a suitably secure digital signature algorithm such as RSA [21] or DSA [7]. p_u is certified by a Certification Authority (CA), resulting in a CA-signed public-key certificate containing the user's name (subject name), p_u, and a validity period.

Key compromise has occurred when knowledge of s_u is possessed by an individual other than the legitimate user u, and there exists a potential for the misuse of s_u by this other entity (i.e., the forging of signatures). If a private key compromise is reported by a user, the corresponding public key certificate can be revoked, and a new key pair can then be introduced.

Let us consider more closely the timeline of actions related to a signature key compromise, as given in Figure 1. The compromise of u's key takes place at time t_0. The compromise may be suspected at time t_1 (the time of detection of the key compromise; u may or may not be aware of the precise time of its occurrence).[1] u reports the compromise at time t_2 and this information is received by the CA at time t_3. Knowledge of the information is made available to users at time t_4 (e.g., by the creation of a certificate revocation list CRL [11] or an on-line check [17]). Note that some time may elapse between t_3 and t_4, e.g., if protocol dictates that compromises are published within 12 hours. Knowledge of the key compromise is obtained by users as early as time t_5; different users may obtain this information at different times. We have $t_0 \leq t_1 < t_2 \leq t_3 < t_4 \leq t_5$.

Even if existing certificate revocation techniques were used in response to a key compromise, they were not intended to handle the situation of an undetected compromise (since they rely on the compromise being reported, and hence detected). Consider that during the period $[t_0, t_1)$ of *undetected key compromise*, a number of messages may be signed. In the worst case, the compromise might not be detected at all, thereby allowing signatures to be forged at least until the date

[1] A realization of the compromise may come to u based on local access control records. It may even be the case though that u is not made aware of a compromise until some party v presents an inquiry with regard to a signed message.

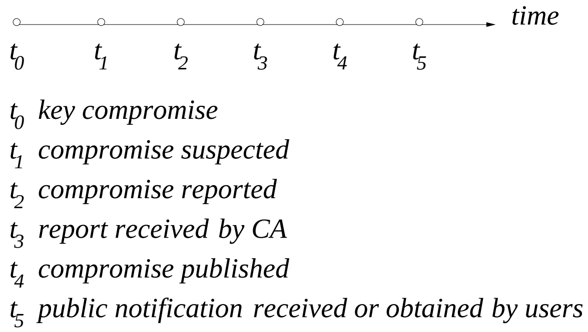

Figure 1: Timeline of events related to a key compromise. From time t_0 to t_1 is a period of *undetected key compromise*.

of expiry of the corresponding public key certificate. Using current techniques, it is difficult to distinguish whether, for the case of disputed signatures,

1. u did not actually sign the messages (i.e., an attacker did), or
2. u is attempting to repudiate signed messages, by either claiming a signature private key compromise at a time before the actual compromise or claiming a compromise when in fact there was no actual key compromise.

Note that the revocation information may be the only evidence available to an adjudicator asked to resolve if and when a key compromise may have occurred.[2] However, this may place an unexpected burden or unfair penalty on the user in cases where a user's private signature key is indeed compromised without his/her knowledge. Indeed, u may not even be able to pinpoint the exact time of the compromise. However, allowing u to repudiate signatures that may have already been accepted is equally unfair to the recipients of the signatures.

2.1 Timestamping is Necessary but not Sufficient

To distinguish when messages were signed relative to various events, signed messages may be timestamped [9]. The determination of whether a signed message is valid depends on the current condition of both the private and public keys of the signing user. The time of a key revocation (or expiry) of the public key certificate can be compared to the time of signing of messages to allow determination of whether a message was signed before or after a revocation. Such a

[2] In some cases, additional information may be available, for example *physical* evidence. However, we would like to focus on solutions that do not rely on such evidence.

procedure may not be sufficient in the case of undetected key compromise. Given that the signed message was timestamped at time t_s, a discrepancy (honest or otherwise) between the detected or reported times of compromise and the time of actual compromise leaves room for potential abuse by the legitimate signer as well as unfair treatment in the case the legitimate signer was honestly not aware of a compromise for some period of time. (Keep in mind that even the legitimate signer may not be aware of the exact time t_0.) These arguments motivate the need for a method for dealing with the problems of undetected key compromise.

2.2 Compromising Signature Private Keys

We identify here potential attacks whose outcome is the compromise of a signature private key. Although access controls are necessary in many cases, they might not be sufficient. It is important to recognize that despite various controls and protections, some keying material may eventually be compromised.

One can identify the following attacks which make the private signature key vulnerable.

1. *Algorithmic Attack.* The signature algorithm itself has succumbed to mathematical or cryptanalytic attack, e.g., the Ong-Schnorr-Shamir signature scheme [18] as broken by Pollard [19].

2. *Implementation Failure.* A particular signature algorithm has been poorly implemented. We include here the possibility of weak keys being chosen, a poor random number generator being used, or the private key not being adequately protected. As a specific example, note the attack on ElGamal signatures [2].

3. *Insider Attack.* This includes attacks whereby the private key is read from temporary memory (in which it is stored while being used). As well, an attacker might read as a user enters a password that is used by the user to compute their private key or to decrypt keying information. This can also include a *social engineering* attack whereby a user may be fooled into giving up a password or key, or a system administrator may be bribed into revealing it.

4. *Brute-force attack.* An attack whereby the password (used to encrypt your keying material) or private key itself is guessed. Schemes with low-entropy passwords are most susceptible to such an attack.

Implemented correctly, a hardware token allows compromise to be easily detected, i.e., the user would recognize the missing token. However, it does not necessarily protect against an algorithmic attack or implementation failure (e.g., if a weak random number generator were used), and care must be taken in their use [3, 14]. The techniques presented herein provide protection even in the case of signature key compromise due to these failures.

3 Dealing with Signature Key Compromise

In this section, we review and discuss techniques that can be used to deal with a signature key compromise by using either of the following general methods:

1. *Providing redundancy.* A single key compromise is rendered insufficient to allow the forgery of signatures by requiring multiple keys for signature production. For example, requiring a private key to be compromised from each of a group of users, thereby requiring multiple, subsequent attacks against different users in order to successfully forge a signature;

2. *Limiting exposure.* Limiting the number or type of signatures that may be forged or the amount of time that undetected forgery can persist may limit the quantity of forged signatures resulting from a key compromise.

Threshold signature schemes. Threshold signature schemes (e.g., [5]) are protocols in which n shares or pieces of a secret signing key are distributed amongst n users (one share per user). To produce a signature (verifiable with a single verification key), at least $t \leq n$ users must cooperate, each producing partial signatures that are thereafter combined to produce a resultant signature.

Redundancy (against one class of attacks) is provided since compromise of a single user's share does not allow one to forge a signature (unless cooperation is obtained from $t-1$ other users). Exposure is limited so long as compromises are detected and subsequent regeneration of signature keys is performed. However, there exists the possibility that over a period of time, t signature shares may be compromised.

Proactive signatures. In anticipation of the possibility of a long-term attack in which multiple shares of a signature key are eventually compromised (without detection), a proactive approach has been proposed

[10] whereby the shares corresponding to a single signature key (where as above, a threshold of signature key pieces are required to produce a signature verifiable by the single verification key) are periodically renewed so that an attacker would be required to compromise a threshold of the shares all within a given time period in order to successfully forge a signature. One advantage is that despite the refreshment of the shares, the underlying private/public key pair can remain fixed for a long time, e.g. several years. This renewal of shares can be performed periodically or can be triggered by the detection of a share compromise. A second advantage is that if one of n parties holding a key share leaves an organization or is dismissed, even without explicit revocation of his key share, the periodic update will cause his key share to be invalidated.

Though suitable for some applications, for protecting individual users against key compromise a disadvantage of using threshold schemes (proactive or otherwise) is the requirement of involving a number of users to produce a single, verifiable signature. Furthermore, it is important to note that such threshold and proactive schemes do not preclude an algorithmic or brute force attack that would discover the single signing equivalent key.

Proactive certification. To remove the requirement of multiple users for the production of a verifiable signature, Canetti, Halevi and Herzberg [4] use the same proactive, distributed concept (as described above for 'Proactive signatures') to allow for a proactive distributed certification of an individual user's signature key, whereby a single signature key is sufficient for the production of a signature, as opposed to a distributed signature construction. Their proactive solution requires periodic refreshment phases in which new signing key pairs are generated by each user. Users additionally store shares of a global, private signature (certification) key, corresponding to a global, public verification key. These shares are used in process (similar to the proactive signature scheme described above) to certify the new signature keys (just as would be done in a centralized scheme by a certification authority). The shares are also periodically refreshed (in addition to the signature key pairs).

A weakness of this approach is that, although signing key pairs are refreshed at regular intervals, there is no protection in the case that a single user's signing key is compromised (without detection by the private key owner) and used to produce a signature within a given time unit. This technique therefore provides some protection against malicious certification of public keys (by providing for a decentralized certification process in which the shares corresponding to the private certification key are periodically renewed) and simultaneously limits the number of (as opposed to preventing) forged signatures that can be produced for a user by imposing periodic renewal of the user's personal signature keys.

Restricted signature privileges. An alternative technique for limiting the effects of key compromise (e.g., forgery of signatures) is related to the idea of attribute certificates. These are certificates that allow for additional information, other than a public-key, to be conveyed in an authentic manner [11]. For example, the additional information may be privileges which can be certified by an attribute authority in separate certificates, or included as an optional field directly in a user's certificate. Suppose, for example, that different privileges were assigned to different users so that only certain classes of messages can be signed by particular users. For example, only users with "signing officer" privileges might be able to sign cheques in the name of their company. An attacker with such a goal in mind, now has a smaller number of users that can be attacked since the compromise of a particular signature key may not allow for the production of forged cheques. This technique can be combined with threshold signatures or proactive signatures (see above) whereby combinations of users with different attributes are required to produce a signature.

Limiting the number of signatures. While such a solution above limits the *types* of signatures that can be produced (and hence forged), one might also try to bound the *number* of signatures that can be produced for a given certificate. This idea can be implemented by using an intermediate trusted third party to decrement the remaining signature count after the production of each signature. Such a technique is used in the Lamport variation of Section 5.1.

Signature insurance. Related to the reduction of risk for a particular user or group of users is the protection against liability in the case of undetected key compromise. Paralleling the paper world, insurance might be useful for protection in such situations, i.e., each user pays insurance premiums for each certificate, protecting against the results of a key compromise. For example, comparing a system where single user signatures are required with one where threshold signatures are required, the former might require for higher insurance premiums.

4 Overview of New Approach

The new solution makes use of a second level of authentication, the result of which allows the recipient v of a signed message to confirm (with a higher degree of assureness than with the original (first level) digital signature protocol) that u did indeed sign m despite a malicious attacker's possible possession of s_u (or equivalent key). It differs from the schemes discussed in Section 3 in that two 'independent' factors (necessary for producing an acceptable signature) are maintained by the signing user as opposed to, say, distributed among a number of users. More detailed descriptions of our particular solutions are given in Section 5 and Section 6.

4.1 A Second Level Authentication for Signature Production

The secondary method can be thought of as a multi-factor method (in this case, two-factor; one for the original signature scheme and one for the secondary authentication). It is an authentication made between the signing user and a *trusted register* (TR). The function of the TR will be to validate an authentic protocol between itself and the originator of the signature, and subsequently produce some information (the purpose of which is to enhance the acceptability of a message signed with s_u) that is bound to the signature in question. Throughout this paper, we consider the technique whereby the TR produces a signature over (at least) the signed message from the user, should the second level authentication be successful. Optionally, one might record user signatures in an integrity-protected database at the TR.

The secondary authentication mechanism has the following properties.

1. Any secret information or algorithms upon which the secondary authentication mechanism relies (or more generally, things that may be vulnerable to the attacks of Section 2.2) should be independent of the signing private key or algorithm used for the signature production itself, i.e., compromise of one doesn't reveal information about the other.

2. The secondary authentication is cryptographically bound or associated with the current signature in question, i.e., is computed as a cryptographic function of the signature.

3. The method permits a suitable authentication to the TR, i.e., allows the TR to verify that only u could have produced a particular signature.

Multiple factors are advantageous in that compromises subsequent to a compromise of the signing private key (or equivalent attack), increase the likelihood of detection (of an attack). This property relies on the independence of the secondary authentication from the first level signing private key and algorithm. This independence increases the likelihood that a second attack would be required subsequent to compromise of the first level signing key. In this way, the independence from the first level allows one to better survive attacks that may only succeed against the first level.

Specifying the general attacks described in Section 2.2, this includes attacks such as factoring [20], timing analysis [13], quantum computing [8, 22], and differential power analysis [14]. The second level can be a signature algorithm but not necessarily. Though if it were, and the second level used DSA signatures while the first used RSA, the second level would be resistant to attacks that existed only against RSA. Even further, the second level need not be cryptographic, e.g., it may involve a phone call from the signature originator to the TR, acknowledging the production of a signature.

The authentication to the TR may be viewed as a form of identification. Identification of a user can be based on, something known (e.g. a password), something possessed (e.g. a smartcard), something inherent (e.g. a fingerprint). Isolating on 'something known', we observe that the known token can be either *static* (e.g., mother's maiden name, birthdate) or *dynamic* (e.g., a periodically changed password). We can also identify non-secret parameters (adaptive tokens) which we specifically use for a synchronization scheme with the TR (see paragraph on 'Attack Detection' below). A lack of synchronization allows for the detection of forged signatures. Only the authenticity of this parameter need be maintained.

4.2 Overview of Secondary Authentication Schemes

Below, we overview two general solutions allowing the originator of a signed message to obtain a second level authentication from the TR. The first makes use of a secret key while the second uses an adaptively changing (non-secret) parameter. A successful second-level authentication results in a signature (produced by the TR) over the signed message to be returned to the originating user. After successful verification by the originating user, the original signature and message can be sent to other users, accompanied by the

TR-signature over the original signature.

Attack Protection A more complete description is given in Section 5. The basic idea is that u maintains a secondary secret key K. u might have several signature keys (corresponding to several public keys) but need only keep a single secondary key. The location and algorithm used with K are independent of the location of the signing private key and signature algorithm. After signing a message m, u sends a message authentication code (MAC) over the signature to the TR. Given possession of K (or some function of it) allows the TR to verify that K was used to produce the MAC in question. A second authentication factor is returned to the originator of the signed message. The independence of K and the MAC, from the original signature algorithm and key, ensure that possession of s_u is not sufficient to produce a signed message that would be acceptable by a recipient. As well, their independence increases the likelihood that an alternative attack would be required to compromise K. Beyond offering a second level of protection, a second attack increases the likelihood of compromise detection by the legitimate owner of the keys.

Attack Detection A more complete description is given in Section 6. Detection involves the discovery of a lack of synchronization between the signing user and the TR. This involves the use of a time-variant or signature-dependent parameter, e.g., a counter (adaptive token). Only the authenticity of this counter need be maintained, not its privacy. For every message signed by a user (even if a message is signed by an attacker in possession of the legitimate user's signature private key), the counter is updated. The key to detection is that the legitimate signer will not be synchronized with the TR at a given message signing, if an attacker has forged signatures since the last message signed by the legitimate signing user. Preventing acceptance of signatures that were not signed by the legitimate user is achieved by imposing a cooling-off period (COPE) (of, say, t time units), for which signed messages are not accepted until t time units have expired and by requiring the legitimate signer to check-in (e.g., by signing a message) every t time units. Since a user is required to check-in over a time interval equal in length to the period of latency before a recipient accepts a signed message, the burden of detecting forged signatures is placed on the legitimate signing user. Besides detecting and preventing forged signature acceptance, this technique also prevents a user from later repudiating a signature that has been accepted by a recipient, since if the message were actually forged, the legitimate user would have noted (and be required to note this because of the check-in) a lack of synchronization with the TR.

4.3 Temporal Functions of the Trusted Register

While the solutions presented here are positioned primarily for the purpose of providing a second level authentication to protect against the case when an undetected signature key compromise (or equivalent attack) has occurred, they are intimately related to the temporal requirements associated with a digital signature. Although time stamping is not sufficient in the case of an undetected key compromise (see Section 2.1) it is necessary. One way to accomplish this is for the second level authentication requester to obtain a time stamp for a signature, independent of the request made to the TR. The request for a time stamp and a second level authentication can be made in parallel. Alternatively, the TR may simultaneously provide the functionality of a time stamp authority and time stamp the signature as sent as part of the secondary authentication request. Such a time stamp is incidently provided by the scheme of Figure 3. The latter appears to be more sufficient since the participation of only a single trusted third party would be maintained for the implementation of our second level solutions.

5 Preventing Successful Attacks

In this section, we elaborate on a particular scheme useful for the prevention of a successful attack in the event of a signature private key compromise. A successful attack would involve the attainment of a forged signature and second level authentication that would be accepted by a recipient.

User u privately shares a symmetric key K with a Trusted Register (TR). The scheme prevents an attacker from succeeding at having forged signatures accepted so long as he/she is not able to recover K, in addition to the private signature key.[3] The particular scheme is shown in Figure 2. Suppose that u wants to send a signed message to user v. To provide a second level of authentication, u sends c and $E_K(c)$ to the TR where $c = sig_A(m)$ and E is a MAC algorithm. Given possession of K, the TR verifies z by recomputing $z = E_K(c)$, and returns a signature r over c to u. Upon receipt of r, u verifies the TR signature over

[3] As indicated in Section 4.2, the independence of K from the private signature key increases the likelihood of a secondary attack being required to compromise K, subsequent to the possible compromise of the signing private key.

c (given possession of the TR's public key certificate). u can now forward $\{m, c, r\}$ to another user v who would verify the signature c over m (given u's public key certificate) and the signature r over c (given TR's public key certificate). If the verification is successful, v would accept the signature c. Note that an attacker, in possession of only the signature private key s_u, would not be able to obtain, nor compute r for a forged signature.

5.1 A Variation Using Lamport Keys

A variation in which the TR need not maintain the secrecy of any information for u uses Lamport keys [15]. u can first share $f^{100}(s)$ with the TR where s is a random, secret seed and f is a one-way function (100 is used only for illustrative purposes here; any positive integer would suffice). For the signature c_i, u can send $z_i = E_{K_i}(c_i, K_{i+1})$[4] (where ',' denotes concatenation) to the TR where $K_i = f^{100-i}(s)$. Given possession of $f^{100-i}(s)$, the TR verifies z_i by recovering K_{i+1} and computing $f(K_{i+1})$ and ensuring that it equals K_i. The TR subsequently stores K_{i+1} in place of K_i. Note that E must be invertible in this case to allow recovery of K_{i+1}. An advantage of this scheme is that a different, pseudo-independent key is used to produce z_i, for each i. As well, compromise of s limits an attacker to a fixed number of signatures. (A variation of this scheme whereby the secrecy of s is not required, is given in Section 6.)

Notice here that the TR can also act as a timestamp authority and provide absolute temporal authentication (see [12]) for c by returning $r = sig_{TR}(c, current\_time)$.

5.2 Prevention of Forged Signature Acceptance

The use of two keys and algorithms that are independent in the sense that the compromise (allowing the production of forged signatures) of one does not imply the compromise of the other, provides protection in the case of *cryptanalytic attacks*. This offers increased protection against such attacks, versus a scheme in which the user might apply secret sharing or proactive techniques independently to obtain multiple pieces of a single signature key. If the first and second level keys happen to be stored in the same location, then no protection is offered against *physical attacks* that compromise a key.

However, if the keys are independently stored as well, then additional protection can be achieved against such physical attacks. For example, if the first level signature key is stored on the legitimate user's local computer and the secondary key is stored on a hardware token, additional protection can be achieved versus a single physical attack; two physical attacks would typically be required to compromise both keys. Should a first attack (key compromise) be detected prior to a successful second key compromise, then the verification certificate of the legitimate user can be revoked and both keys can be renewed.

[4] Some form of integrity (e.g., a MDC) for (c_i, K_{i+1}) should be appended prior to encryption and verified by TR upon receipt of z_i.

6 Detecting Attacks

Attack detection involves the detection of the production of a forged signature. Detection alone does not prevent an attacker (in possession of a user's signing private key) from forging signatures which would normally be accepted by a recipient. However, it does allow detection, and action can be taken to prevent continued forgeries. Section 6.3 uses the concept of a cooling-off period to extend the detection methods described in Section 6.1 and Section 6.2, to allow for the prevention of forged signature acceptance.

6.1 Attack Detection Using Synchronization

The detection of an attack, using the schemes proposed below and in Section 6.2, involves the detection of a lack of synchronization between the signing user u and the TR, and occurs at points when a legitimate signer produces signatures. This synchronization is implemented using time-variant parameters (see [16, Chapter 10], [6, Chapter 5]) as an *adaptive token* that is shared between u and the TR. The key distinguishing feature with these values is that they need not be kept private; only the authenticity of the values need be maintained. Protection is provided by using synchronization to detect forgeries. Each signature for which a second factor authentication is obtained results in an update to this parameter by both the signature originator and the TR. If the signature originator is not the legitimate private key owner, then during a subsequent attempt by the legitimate owner to receive a secondary authentication of a signature, the legitimate owner will not be synchronized with the TR.

We identify two types of values that can be used for this synchronization parameter:

1. the output of a function that is monotonically increasing with time or
2. the output of a collision-resistant function that is dependent upon all previously constructed signatures.

v		u		TR
		msg m	$\xrightarrow{c = sig_u(m), z = E_K(c)}$	verify z
		verify r	$\xleftarrow{r = sig_{TR}(c)}$	
verify c, r	$\xleftarrow{\{m, c, r\}}$			

Figure 2: Prevention Using a Shared Key. K is shared between the signature originator u and the TR. v is the signature recipient.

Using an explicit time itself satisfies Item 1, and is described in Section 6.2. An alternative technique involves using a counter or sequence number. Item 2 can consist of the concatenation or recursive hash of all previously produced signatures. Introducing cooling-off and check-in periods in Section 6.3 weakens this condition so that not *all* previously constructed signatures are necessary.

Using Lamport Keys for Detection

A modification of the variation using Lamport keys given in Section 5 is helpful for illustrative purposes to demonstrate the basic idea of detection. User u initially choses a random seed s and gives $K_i = f^{100}(s)$ to the TR (100 is used for illustrative purposes here only; other positive integers can be used), where f is a one-way function. Rather than using a private key K, various functions of s are used here as the second level parameter (the privacy of s need not be maintained though it can be, for enhanced protection). During round i, for message m_i, u sends $c_i = sig_u(m_i)$ and $z_i = K_{i+1}$ to the TR who computes $f(K_{i+1})$ to ensure that it is equal to the stored value K_i.[5] If it is, the TR stores K_{i+1} in anticipation of the next request, and returns $r_i = sig_{TR}(c_i, K_{i+1})$ to the signature originator. If it is not, then the TR is alerted of a problem, and possibly contacts u or initiates a revocation of u's public key, e.g., by contacting the issuing CA.

One potential problem is that an attacker (possessing the signature private key) can simply intercept and recover K_{i+1} and obtain a second-level authentication on a different message m'_i by computing $c'_i = sig_u(m'_i)$. However, either u will detect that $r_i = sig_{TR}(c'_i, K_{i+1})$ has been incorrectly computed for c'_i as opposed to c_i, or that no response has been received from the TR at all (if the attacker were to "block" the return from the TR). This *detection* signals u that his signing private key may have been compromised. If attacked independent of a signature sent by u (given that privacy of s need not necessarily be maintained), the attacker can obtain a second level authentication for a forged signature. However, a lack of synchronization will be detected by u when attempting to obtain a secondary authentication for a subsequent signature. In Section 6.3 we see how a recipient is prevented from accepting such forged signatures.

[5] One might prefer sending $z_i = sig_u(c_i, K_{i+1})$ instead of simply K_{i+1} to prevent possible denial of service attacks.

6.2 Detection Using the Time of Last Signature for Synchronization

In this section, we present a scheme in which we make use of the time at which signatures are produced, as an adaptive parameter used to synchronize u with the TR. The use of the time here is advantageous in that beyond the usefulness of allowing a synchronization, it can provide a timestamp for the submitted signature as well as possessing sufficient information for the implementation of the cooling-off period described in Section 6.3.

Referring to Figure 3, u initializes ("synchronizes") with the TR by the exchange of c_1 and z_1. Subsequently, the TR stores the time of the last signature associated with u (in this case, t_1). To obtain a second level authentication for a message m_i, u sends the signature c_i as well as t_{i-1}, which represents the time returned to u during the second level authentication of c_{i-1} (i.e., u, or more correctly, the originator of c_{i-1}, would have had $z_{i-1} = sig_{TR}(c_{i-1}, t_{i-2}, t_{i-1})$ returned) to the TR. The TR verifies that t_{i-1} matches the stored value corresponding to u. If the verification of t_{i-1} is successful, then the TR stores the current time t_i for u and returns $z_i = sig_{TR}(c_i, t_{i-1}, t_i)$. u_i verifies that z_i is the correct signature of the TR (given *a priori* possession of the TR's signature verification public key) over c_i, that t_{i-1} is the time of last signature and that t_i is the current time.

Detecting a Lack of Synchronization

If there is a discrepancy between the time sent and the time stored at the TR for u, then the TR has noticed

v		u		TR
		msg m_1	$\xrightarrow{c_1 = sig_u(m_1)}$	stores $\{u, t_1\}$
		verifies z_1 stores t_1	$\xleftarrow{z_1 = sig_{TR}(c_1, t_1)}$	
verify c_1, z_1	$\xleftarrow{\{m_1, c_1, z_1\}}$			
	$\vdots$		$\vdots$	
		msg m_i	$\xrightarrow{c_i = sig_u(m_i), t_{i-1}}$	verify t_{i-1}
		verify z_i stores t_i	$\xleftarrow{z_i = sig_{TR}(c_i, t_{i-1}, t_i)}$	store $\{u, t_i\}$
verify c_i, z_i	$\xleftarrow{\{m_i, c_i, z_i\}}$			

Figure 3: Detection using the time of the last signature. u is the signature originator and v is the signature recipient. One might prefer to send $sig_u(c_i, t_{i-1})$ instead of simply t_{i-1} to prevent possible denial of service attacks.

a lack of synchronization. Now, either the TR has detected that messages have recently been forged in u's name, or u is attempting to repudiate some signatures by purposely causing a lack of synchronization. For example, if u were to send t_{i-2} instead of t_{i-1}, then this would indicate to the TR that the second level authentication obtained for c_{i-1}, might have been obtained by an attacker (and may have already been sent to and accepted by a recipient). Though it may also be that u is attempting to falsely repudiate a message (m_{i-1}) that he may have indeed signed.

It is possible as well, that in the case that there is a lack of synchronization because of a forged message, and u sends the time t_{i-1} when the TR expects the time t_i, that an attacker can simply replace t_{i-1} by t_i in the message from u to the TR. However, in the returned response from the TR, z_i, u will notice the modification and be alerted to the problem.

In the following section, we make novel use of a cooling-off period, so that signatures with a second level authentication, are not accepted by recipients until the legitimate signer u has verified that they were indeed legitimately constructed.

6.3 Prevention of Forged Signature Acceptance Using a Cooling-Off Period

Notice that for the detection schemes described in Section 6.1 and Section 6.2, the legitimate signer or the TR is able to detect when a signature has been (potentially) forged in u's name. However, this still does not prevent the possibility that u may repudiate a legitimately signed message which a recipient has accepted as valid. As well, it does not prevent a recipient of a signed message from potentially accepting a signed message that has been forged by an attacker.

However, suppose that signed messages are not accepted as being valid until some period of time has elapsed, i.e., a cooling-off period (COPE). The purpose of this COPE is to allow for "late" forgery detections or revocations, possibly resulting from a compromise, i.e., in the case a forged message has been detected. For example, if a message is signed on Friday, it may be part of policy to not accept the signature until Saturday. (Finer or coarser granularities may also be used.) This allows a day of grace for the owner of the private signature key to claim the possible compromise of his key.

However, on its own, this COPE does not preclude the possibility that a compromise is not detected until after the COPE has expired (and hence some forged messages may have been accepted). As well, even if the compromise is detected on time, there is typically a delay before the corresponding certificate is revoked (see Figure 1). To facilitate the detection of forged signatures, we incorporate into the COPE with a so-called *Check-In Period* (CHIP) giving CHIP/COPE.

Definition 1 *A CHIP/COPE refers to a check-in period (CHIP) during which time the legitimate owner of the signing private key is required to (at least once during the period) ensure synchronization with the trusted register (TR) (e.g., by obtaining a second level authentication for a signature),*[6] *and a cooling-off pe-*

[6] Certain scalability and denial of service issues would have

riod (COPE) during which time, received signatures are still considered to be potentially forged.

For example, if the length of the COPE is a single day, then the legitimate user can wait no longer than 24 hours after a legitimate signing, before performing a check-in. (To allow for other tasks to be performed subsequent to the detection of a compromise (cf. Figure 1), in practice the length of the COPE will be buffered slightly so that it exceeds the length of the CHIP.)

Notice that since the legitimate owner of the signing private key is responsible for checking-in (i.e., synchronizing) during a given time period, he is not able to repudiate a message that was legitimately signed. This is because for signatures that have been accepted by the recipient (i.e., signature has been received and the COPE has expired), the latency period must have passed and the loss of synchronization would have been detected for the time period in which the signature was sent. This idea is captured by the following proposition.

Proposition 1 *Let the length of the CHIP and COPE be t time units. Given that u must check-in (i.e., verify synchronization) every t time units and that signatures (accompanied by second level authentication) are not accepted until t time units after receipt, u detects when any forged signatures have been produced and cannot successfully repudiate signatures accepted by recipients (given that the COPE has expired).*

Proof (Outline) Assume that u has initially synchronized with the TR. Consider a message m, signed at time s. Now, the legitimate signer must check-in every t time units. Therefore, a legitimate check-in occurred during the time interval $[s-t, s)$ and another will occur during the interval $(s, s+t]$. Given the COPE, the signature on m will not be accepted until time $s+t$. However, a check-in, and hence a lack of synchronization will have been detected by u as of this time. Therefore, forgery is detected as well (and hence, appropriate action has been performed (e.g., revocation of the public key certificate), so that a recipient of the signature will be aware that it is a forgery. For the same reason, u could not successfully repudiate the signature on m if it were legitimately signed since a legitimate check-in is performed (verifying the maintenance of synchronization and hence, verifying that no signatures have been forged) prior to signature acceptance. ■

to be considered in practice, related to the potential inability of a user to check in because of an overwhelmed TR.

In this way, once a recipient of a message has waited a length of time equal to the length of the COPE (plus additional time allowing for revocation etc.), and subsequent to a check of the revocation status of u's public key, she can be sure that the signature was legitimately constructed. The signatures are *committed* at this time, in the sense that the CHIP/COPE is similar to an *atomic* transaction or protocol. The legitimate signer must have legitimately signed a message subsequent to the signing of the message for the aforementioned user, yet before the CHIP expiry for the recipient. By designing a protocol in such a way that the legitimate user confirms that the messages signed during the last CHIP were indeed signed by him, the signing user is limited in his ability to later deny having signed any of the messages in question.

6.3.1 Combining a Cooling-Off Period with Detection

How does this alter the scheme shown in Figure 3? Let the length of the CHIP/COPE (see Definition 1) be t time units. Beyond requiring u to check-in with the TR at least every t units, the TR would also perform a check that $t_i - t_{i-1} < t$. So long as a recipient waits t time units before accepting a signature, forged messages can be detected by the TR. As well, t may be different for each user. Allowing the recipient of a signed message to determine the length of the COPE for a particular message can be achieved by having the TR return $z_i = sig_{TR}(c_i, t_{i-1}, t_i, t_i + t)$. If t is not so dynamic for users though, it may well be that it be included as a parameter in the user's public key certificate.

6.3.2 Implementation and Practicality

Coordinating the CHIP with an actual user may require, for example, that temporary revocations [1] are allowed in the case of long-term absences by a user. Also related to the practical implementation of such a scheme is that once a lack of synchronization is detected by the TR, additional time will be required before revocation information can be obtained by signature recipients. Therefore, in practice, the length of the COPE should be $t + \epsilon$ for a suitable ϵ, where the CHIP is t time units.

With regard to the practicality of using a CHIP/COPE, imposing such restrictions on both the signer and recipient may appear unreasonable. Though there already exists examples of its use in current society (e.g., depositing a cheque normally requires a waiting period before the amount is included in the account), it is certainly not practical for all situations. Yet there are situations in which it can be very helpful, i.e., schemes for which undetected key compromise is intolerable yet can tolerate a time delay before the acceptance of a signature. Such high valued transactions include major business deals, mergers and acquisitions, and real estate deeds; transactions that want to use digital signatures for their convenience, but are so costly that they require an extra level of assurance.

Compared to alternative solutions to the key compromise problem (see Section 3), note that threshold or proactive schemes do not provide protection against cryptanalytic attacks that would recover the single signature key from which shares of the key are created. Also note that the solution presented here allows signatures to be constructed by a single individual as compared to the cooperation between more than one user required for signatures produced with threshold or proactive schemes.

7 Summary and Future Work

This paper has proposed the use of a second level authentication for dealing with the problem of undetected key compromise. Techniques have been described that use two alternative approaches for implementing this second level.

1. In Section 5, schemes were presented in which a secret key is shared between a user and the trusted register. A function of this key and the signature for which a second level authentication was being requested was used as an identification to the TR.

2. In Section 6, schemes were presented in which a non-secret key is used to synchronize a user with the TR so that forged signatures can be detected at the next legitimate request by a user, through the lack of synchronization with the TR. The combination of these synchronization schemes with a check-in-period (CHIP) and cooling-off period (COPE) allows for the detection and *rollback* of forged signatures.

For Item 1, the independence of the two keys (and algorithms) is important. For example, algorithms must be chosen such that the compromise of one does not necessarily imply the compromise of the other. With regard to physical attacks, it is therefore important that the storage of these two keys also be independent. One option is for one key to be stored on the user's local computer with the other on a hardware token. It would be of interest to provide some form of key management for the user's multiple keys that maintains their independence, yet limits the inconvenience to users. Related to the independence, each system should weigh the likelihood of compromise possibilities when choosing to implement a second level authentication. For example, if a system has suitable access controls, effectively minimizing the possibility of a successful physical attack, it may be unnecessary to select a second level authentication scheme in which the key is stored independently of the primary signature key.

For Item 2, although secrecy of the synchronization parameter is not necessary, in cases where its value is not computationally feasible to predict (e.g., the variation using Lamport keys in Section 5), maintaining the secrecy of the parameter can only improve the security of the scheme. It would be helpful for the practical implementation of the CHIP/COPE, to minimize delays between and during the events identified in Figure 1 so as to improve the efficiency of this CHIP/COPE. Differing lengths for the CHIP/COPE are likely desireable from one system to the next. It would be beneficial to study what period lengths would be optimal for varying schemes, balancing the requirements of the signing users as well as signature verifiers.

8 Concluding Remarks

Protecting against signatures forged during a period of undetected key compromise is not a trivial task. In this paper, we have provided several solutions which, beyond allowing one to protect or detect such forgeries, require the use of an on-line trusted third party (TTP). In this final section, we address some concerns regarding the use of a TTP.

Consider the concern that the TTP adds unnecessary complexity to the process of sending a digital signature. However, as observed in Section 2.1, the time relationship between the signing of a message and the revocation of keying material necessitates a timestamped, signed message. Although there exists distributed methods for achieving a timestamp (e.g. [9]), they are not efficient in practice. Therefore, the use of a third party (i.e., timestamp authority) is necessary in any case (for schemes requiring non-repudiation), even without the use of our schemes.

Also notice that while the solutions described here were positioned primarily for the purpose of providing a second factor authentication to protect against the case when an undetected private key compromise has occurred, the TR may simultaneously act as a timestamp authority and timestamp the submitted signature (this was an incidental result for the scheme of

Section 6.2 for example).

How do the new solutions differ from those that would simply use a TTP without digital signatures, i.e., symmetric key signatures? The proposed schemes still maintain the provision of non-repudiation while symmetric key signatures with a TTP do not. The original first level authentication provides for a digital signature while the second level authentication aids in the prevention and detection of forged signature acceptance, forged as the result of a key compromise (or equivalent attack). Unlike symmetric key signatures though, the trusted register (TR) used for our second level authentication, does not necessarily maintain the privacy of any information related to users (see the variation described in Section 5 and both techniques described in Section 6).

Acknowledgements

Thanks to Michael Wiener for suggesting that a solution to the problem of undetected signature key compromise might exist. Thanks also to Pat Morin and Evangelos Kranakis for several helpful comments and to the anonymous referees for helping to improve the clarity of several of our results.

References

1. ANSI X9.57, "Public Key Cryptography for the Financial Services Industry: Certificate Management", American National Standard for Financial Services, February, 1997.
2. Daniel Bleichenbacher, "Generating ElGamal Signatures Without Knowing the Secret Key", *Advances in Cryptology: Eurocrypt '96*, Springer-Verlag, pp. 10-18, 1996.
3. Dan Boneh, Richard A. Demillo, Richard J. Lipton, "On the Importance of Checking Cryptographic Protocols for Faults", *Advances in Cryptology: Eurocrypt '97*, pp. 37–51, 1997.
4. R. Canetti, S. Halevi, A. Herzberg, "Maintaining Authenticated Communication in the Presence of Break-ins", in *Proceedings of the 16th Annual ACM Symposium on Principles of Distributed Computing*, 1997.
5. Y. Desmedt, "Threshold Cryptography", *European Transactions on Telecommunications*, Vol. 5, No. 4, pp. 449-457, July, 1994.
6. D. Davies, W. Price, *Security for Computer Networks*, John Wiley & Sons, 2nd edition, 1989.
7. FIPS 186, "Digital Signature Standard", Federal Information Processing Standards Publication 185, U.S. Department of Commerce/N.I.S.T., National Technical Information Service, Springfield, Virginia, 1994.
8. N. Gershenfeld, I. Chuang, "Quantum Computing with Molecules", *Scientific American*, June, 1998.
9. S. Haber, W.S. Stornetta, "How to Time-Stamp a Digital Document", *Journal of Cryptology*, Vol. 3, No. 2, pp. 99-111, 1991.
10. A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk, "Proactive Public Key and Signature Systems", in *Proceedings of the 4th ACM Conference on Computer and Communications Security*, 1997.
11. ITU-T Recommendation X.509, "The Directory - Authentication Framework" International Telecommunication Union, Geneva, Switzerland, November, 1993. (equivalent to ISO/IEC 9594-8:1990&1995)
12. Mike Just, "Some Timestamping Protocol Failures", *Proceedings of the 1998 Symposium on Network and Distributed System Security*, pp. 89-96, March 1998.
13. Paul Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems", *Advances in Cryptology: Crypto '96*, pp. 104–113, Springer-Verlag.
14. Paul Kocher, J. Jaffe, B. Jun, "Differential Power Analysis", available from `http://www.cryptography.com/dpa/`.
15. L. Lamport, "Password Authentication with Insecure Communication", *Communications of the ACM*, **24**, pp. 770-772, 1981.
16. A. Menezes, P. van Oorschot, S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997.
17. M. Myers, R. Ankney, "Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP", IETF PKIX Working Group Internet Draft, February 1998 (work in progress).
18. H. Ong, C.P. Schnorr, A. Shamir, "An Efficient Signature Scheme Based on Quadratic Equations", *Proceedings of the 16th Annual ACM Symposium on Theory of Computing*, pp. 208-216, 1984.
19. J.M. Pollard, C.P. Schnorr, "An Efficient Solution of the Congruence $x^2 - ky^2 = m \pmod{n}$", *IEEE Transactions on Information Theory*, Vol. 33, pp. 702-709, 1987.
20. Carl Pomerance, "Factoring", in *Cryptology and Computational Number Theory*, American Mathematical Society, ed. Carl Pomerance, pp. 27–47, 1990.
21. R. Rivest, A. Shamir, L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems", *Communications of the ACM*, **21**, pp. 120-126, 1978.
22. Peter W. Shor, "Algorithms for Quantum Computation: Discrete Logarithms and Factoring", *Proceedings of the 26th Symposium on Theory of Computing (STOC)*, Montreal, Canada. pp. 124–134, 1994.