Tamper-Evident Digital Signatures: Protecting Certification Authorities Against Malware

Jong Youl Choi
Dept. of Computer Science
Indiana Univ. at Bloomington
Bloomington, IN 47405

Philippe Golle
Palo Alto Research Center
3333 Coyote Hill Rd
Palo Alto, CA 94304

Markus Jakobsson
School of Informatics
Indiana Univ. at Bloomington
Bloomington, IN 47408

Abstract

We introduce the notion of tamper-evidence for digital signature generation in order to defend against attacks aimed at covertly leaking secret information held by corrupted signing nodes. This is achieved by letting observers (which need not be trusted) verify the absence of covert channels by means of techniques we introduce herein. We call our signature schemes tamper-evident *since any deviation from the protocol is immediately detectable. We demonstrate our technique for the RSA-PSS (known as RSA's Probabilistic Signature Scheme) and DSA signature schemes and show how the same technique can be applied to the Schnorr and Feige-Fiat-Shamir (FFS) signature schemes. Our technique does not modify the distribution of the generated signature transcripts, and has only a minimal overhead in terms of computation, communication, and storage.*

1 Introduction

Malware and insider attacks pose an increasing threat to our cyber infrastructure. Current protection mechanisms against malware rely on the collection of malicious code found in the wild. For this protection to be effective, malicious code must be detected as quickly as possible after an attack is mounted. Unfortunately detecting malicious code quickly is difficult, particularly in the case of carefully targeted attacks, such as for example an attack narrowly focussed on critical nodes in the infrastructure.

A perfect example of such a target is a certification authority. The consequences would be severe if an attacker succeeded in corrupting a certification authority and managed to cause it to leak its secret key material. The mere *possibility* of such an attack (as opposed to its actual occurrence) may in fact weaken the trustworthiness of our public key infrastructure. This threat is all the more severe as the leak may be performed using a *covert channel*.

The threat of covert channels (also called subliminal channels) was first studied by Simmons [18, 19] and later by, among others, Young and Yung [23, 24]. These authors show how an attacker may leak bits of the private signing key in covert channels present in the randomized key generation or signing algorithm. For the sake of concreteness, let us consider the RSA-PSS signature scheme (known as RSA's Probabilistic Signature Scheme [7]) as an example. The message encoding scheme of RSA-PSS specifies that the hash of the message to be signed be concatenated with a random octet string known as a "salt". A malicious implementation of RSA-PSS may choose bits of the private key for the salt, instead of a value produced by the pseudo-random number generator. One may argue that this simple type of leakage is detectable: a third party could test (for each signature) whether the salt is a substring of the secret key corresponding to the known public key of the signer (the third party would need to be trusted with the secret key). Upon the detection of such an event, the signer would be isolated and decommissioned. However, an attacker could just as easily leak an *encryption of* some of the bits of the secret signing key, under a symmetric or public-key encryption scheme for which the attacker knows the decryption key. For a semantically secure encryption scheme, and appropriate parameter choices, such a leakage could be made in a truly covert manner.

RSA-PSS is not the only signature scheme that is vulnerable to covert channel attacks. To some extent, *all* signature schemes are vulnerable to these attacks, since they all make use of randomness for key generation. Still, covert channel attacks pose the greatest threat to signature schemes that use randomness not just once in the initial key generation step, but for the generation of every signature. Thus discrete-log based signatures schemes, such as Schnorr signatures [17] and DSA [20], are particularly vulnerable to covert channels given their ongoing use of randomness in the generation of each and every signature. Our techniques focus on eliminating covert channels from the signature algorithm of discrete-log based signature schemes, but they can be com-

bined with previously proposed techniques [9] that consider only covert channels in the key generation phase.

In this paper, we consider the problem of detecting (potentially covert) malicious behavior by corrupt certification authorities and other generators of digital signatures. We propose to solve this problem with *tamper-evident* digital signatures. Tamper-evident signatures do not *prevent* corruption, but they ensure the immediate detection of *any* corruption that causes a signer to output signatures that are in *any* way different from those of an honest signer. Tamper-evident signatures are useful to defend not only against malware, but also against insider attacks. A typical insider attack [16] is performed by the implementor of a cryptographic application, who designs code that operates correctly while under test, but switches to a corrupt mode of operation after deployment. Insider attacks are notoriously hard to prevent, given the difficulty of auditing even small pieces of software or hardware. Tamper-evident signatures do not prevent insider attacks, but ensure that such attacks are immediately detected if they cause a signer to output signatures that differ in *any* way from the signatures of an honest signer.

While current security models for digital signatures consider the signer as an oracle, we take a large step towards a more realistic threat model by allowing – in addition – attackers to corrupt signers and attempt to make them leak their secret information to the attacker. When an attacker corrupts a node, it replaces the code run by that node with code chosen by the attacker. We assume however that the attacker cannot *create* any new communication channels for the nodes under its control. The goal of our attacker, as in the standard definition of security for signature schemes, is to generate a valid signature on a new message.

We assume the existence of one or several *observers*. An observer is an external node whose task is to inspect all signature transcripts produced by a signer and detect any deviation from honest signature generation. This task would be trivial if we gave the observer knowledge of all the secrets known to the signer. However, we do not wish to place any trust with observers. Instead, the observers need to detect the presence of a covert channel given only publicly available information. Once a covert channel is detected, the observer alerts the public of this fact, and proves the existence of the covert channel to avoid false alarms. This allows corrupted nodes to be manually disconnected and reconfigured, and minimizes the effects of the attack.

Example. To illustrate our techniques, we return to the example of RSA-PSS. A small change to the signing algorithm makes RSA-PSS tamper-evident. Let us replace the sequence of random salt used in the message encoding scheme with successive pre-images of a fixed value by a one-way hash function. More precisely, let $(s_0, s_1, \ldots, s_n)$ be the values of a hash chain such that $s_i = h(s_{i+1})$ $(0 \leq i \leq n-1)$, for some secret seed s_n which is known only to the signer. The value s_0 is made public by the signer during a setup phase. Afterward, the signer uses the value s_i as a salt for signature $i = 1, \ldots, n$. An observer can check the correctness of the salt used in the i^{th} signature with the equation $s_{i-1} = h(s_i)$. If this verification fails, the observer raises an alarm.

Properties. We describe next two important properties of the tamper-evident signature schemes proposed in this paper.

1. *Undercover observers.* In our RSA-PSS example, the observer can verify the absence of covert channels without interacting with the signer. We say that our observer is *undercover*. Undercover observers quietly "eavesdrop" on available network traffic to detect irregularities and need not expose their existence until they raise an alarm. Undercover observers, by virtue of being hidden, cannot be targets of coercion attacks (unlike the signer whose existence is of necessity public). Onwards, we only consider undercover observers, but note that interactive observers may be useful in other contexts to design tamper-evident protocols.
2. *Distribution of signatures.* In the simple example given above, the distribution of the transcripts generated by tamper-evident RSA-PSS is easily distinguishable from that of standard RSA-PSS, given the relation of salts. This is undesirable and can easily be avoided. The tamper-evident signature schemes that we propose in this paper produce a distribution of signature transcripts that is polynomial-time indistinguishable from regular transcripts (after removing the proofs constituting evidence of tamper-freeness).

Assumptions. The tamper-evident signature schemes proposed in this paper make the following assumptions:

1. *Unencrypted traffic.* We assume that traffic to and from the signer is not encrypted. The adaptation of our methods to a model in which traffic is sent over an encrypted communication channel to which the observer does not have decryption abilities is *theoretically possible* – given well-known results relating to general multi-party computation. However, an efficient solution that remains compatible with existing VPN techniques is a challenging open problem.
2. *Timing attacks.* Tamper-evident signatures do not protect against timing attacks. However, by imposing strict requirements on synchronization, one can protect against these as well, at the cost of a reduced (but predictable) throughput.

Organization of the paper. We describe related work in section 2. Our definition of tamper-evident signatures follows in section 3. In section 4 we present a tamper-evident version of the DSA signature scheme, followed in section 5 by tamper-evident variants of the Schnorr and Feige-Fiat-Shamir (FFS) signature scheme.

2 Related Work

The study of covert channels is rooted in military history. In the 1970's, the problem of "message authentication without secrecy" arose in the context of the comprehensive nuclear test ban treaty. The goal was to allow the US and Russia to monitor each other's compliance with the treaty, while ensuring that the monitoring equipment was not also used for spying. In 1983, Simmons [18, 19] introduced the concept of covert channels in cryptographic protocols and specifically demonstrated the use of the Digital Signature Standard (DSS) signature scheme for covert communication. This showed that a secret message could be hidden inside the authenticator.

A few years later, Desmedt [3] presented a practical subliminal-free authentication scheme, in which an observer (named "active warden") handles all messages sent between two prisoners, and verifies that these are free from covert information before passing them on. The observer in this scheme is not undercover. Undercover observers are more desirable since they operate stealthily and are thus less vulnerable to attacks aimed at suppressing their activity. Consider that an interactive observer (whose interaction with the signer is evident to all), could well be the first target of an attacker, a virus or a Trojan horse. Once the observer is eliminated or compromised, the adversary can take over the signer without triggering an alarm. In contrast, the activity of undercover observers is undetectable to the adversary, at least until the point when an undercover observer raises an alarm. It is also possible to set up several undercover observers, further complicating the task of an adversary intent on finding and compromising them.

Young and Yung [23, 24] showed the existence of covert channels in the key establishment algorithms of signature schemes (an attack not considered in the previous work by Desmedt). Juels and Guajardo [9] proposed a zero-knowledge key validation scheme to avoid such attacks. In this paper, we focus our attention on corruption that occurs after the key generation phase has concluded, and thus, like Desmedt, only consider how to detect covert channels during the signature generation phase.

In the context of our paper, we consider covert channels harmful. For example, as shown in [12], covert channels in electronic voting system will undermine voters' privacy. However, there is work in which covert channels are used to achieve a desirable security goal. For example, so-called funkspiel schemes [6] use a covert channel to signal alerts by devices that have been corrupted by an attacker. Namely, when such a scheme detects an intrusion attempt, it changes its state, causing future transcripts to signal an alarm to an authority via a covert channel, but preventing the attacker from detecting that this is taking place. In contrast, we develop methods to *eradicate* covert channels.

Lepinksi, Micali, and Shelat [10] recently proposed a generic collusion-free protocol that detects any collusion between malicious participants and thus prevents the existence of covert channels. However, this generic protocol does not give an efficient construction of a tamper-free signing protocol for a certification authority. The problem of designing efficient tamper-free protocols for important real-world applications remains a promising area of research. For example, recent work [1] shows how to design an efficient tamper-evident mix network. Here, we propose an efficient tamper-evident signing protocol for certification authorities.

We use the word tamper-evidence to describe a property of an *algorithm*. Traditionally, it has been used to describe a property of hardware. Smart cards, SIM cards, and satellite decoders all implement varying degrees of physical tamper-evidence. While we use the same term – tamper-evidence – to describe the defense mechanism we introduce, we emphasize that any comparison beyond the truly superficial makes it clear that these two types of tamper-evidence are not closely related in any technical sense.

3 Definition of Tamper-Evident Signature Schemes

Recall that a signature scheme is a triplet of algorithms $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$, where:

- The key generation algorithm Gen, on input 1^k outputs a public/private key pair K_{pub}, K_{priv}.
- The signing algorithm, on input M and K_{priv}, outputs a signature $\sigma = \mathsf{Sign}(M, K_{priv})$.
- The verification algorithm outputs $\mathsf{Verify}(M, \sigma, K_{pub}) \in \{\mathsf{valid}, \mathsf{invalid}\}$.

Intuitively, a signature scheme is tamper-evident if a signer cannot leak K_{priv} without generating at least one "bad" signature that triggers an alarm. We call such "bad" signatures covert-invalid (denoted $\mathsf{invalid}^*$) and their complement covert-valid (valid^*). Note that covert-validity is different from validity as defined by the verification algorithm Verify. Indeed, randomized signature algorithms (such as Schnorr or DSA) permit leakage of the private key via *valid* signatures.

To define a tamper-evident signature scheme, we augment a regular signature scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ with a

new key-generation algorithm (denoted Gen^*), a new signing algorithm (denoted Sign^*) and a new algorithm to verify the covert-validity of signatures (denoted Verify^*). In practice, the tamper-evident signature schemes we propose require only a minuscule augmentation, and incur very small overhead. Only observers would have to take note of the augmentation, and other nodes would simply truncate the transcript to obtain the expected signature.

In what follows, we let T denote a transcript that consists of the ordered list of all the signatures that have ever been output by the signer. The transcript T is one of the inputs to the algorithms Sign^* and Verify^*. (This is for reasons of generality alone, and only a tiny fraction of this information needs to be carried as state.) The augmented signature scheme is defined as follows:

- Gen^*, on input 1^k computes $\mathsf{Gen}(1^k) = (K_{pub}, K_{priv})$ then outputs (K^*_{pub}, K^*_{priv}), where $K^*_{pub} = (K_{pub}, \beta)$ and $K^*_{priv} = (K_{priv}, \alpha)$. As we shall see, the strings α and β are used to ensure tamper-evidence.
- Sign^*, on input a message M, the private key $K^*_{priv} = (K_{priv}, \beta)$ and the transcript T, computes $\sigma = \mathsf{Sign}(M, K_{priv})$ then outputs $\mathsf{Sign}^*(M, K^*_{priv}, T) = (\sigma, \tau)$. As we shall see, the additional string τ allows an observer to verify that the signature is covert-valid.
- Verify^*, on input a message M, the public key K^*_{pub}, a signature (σ, τ) and the transcript T, outputs $\mathsf{Verify}^*(M, K^*_{pub}, (\sigma, \tau), T) \in \{\mathsf{valid}^*, \mathsf{invalid}^*\}$[1].

We can now give a formal definition of a tamper-evident signature scheme. Let $(\mathsf{Gen}^*, \mathsf{Sign}^*, \mathsf{Verify}^*)$ be an augmented signature scheme based on a regular signature scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$. We consider the following game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

Game TE:

1. $\mathcal{C}$ computes $\mathsf{Gen}^*(1^k) = (K^*_{pub}, K^*_{priv})$ and outputs K^*_{pub}.

2. $\mathcal{A}$ requests tamper-evident signatures on adaptively chosen messages. When $\mathcal{A}$ requests a signature on a message m_i, $\mathcal{C}$ outputs $(\sigma_i, \tau_i) = \mathsf{Sign}^*(m_i, K^*_{priv}, T_{i-1})$ and defines $T_i = T_{i-1} \cup \{(i, \sigma_i, \tau_i)\}$.

3. $\mathcal{A}$ outputs a transcript T, a message M and a tamper-evident signature (σ, τ), and wins if $\mathsf{Verify}^*(M, K^*_{pub}, (\sigma, \tau), T) = \mathsf{valid}^*$ and $(\sigma, \tau) \neq \mathsf{Sign}^*(M, K^*_{priv}, T)$.

[1] As described here, Verify^* is a non-interactive protocol, as we focus our attention on undercover observers. A more general definition that allows for interaction can easily be formulated.

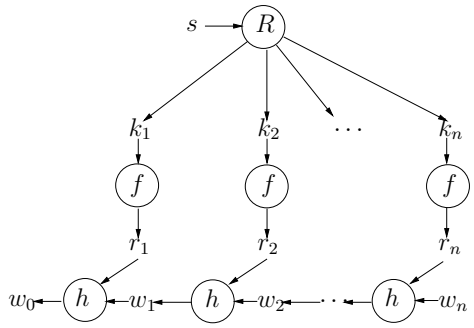

Figure 1. A model of a consistent system. Let the value k_i be the i^{th} output string generated by a pseudo-random generator R that is given a seed s, and let $r_i = f(k_i)$ for some one-way function f, where $1 \leq i \leq n$ for some system parameter n. Two witnesses w_{i-1} and w_i imply that the value r_i is *consistent* if and only if $w_{i-1} = h(w_i, r_i)$, where h is a publicly available hash function and w_n is a κ-bit random value selected uniformly at random.

Definition 1 (Tamper-evidence) *A signature scheme is* tamper-evident *if no polynomial-time algorithm $\mathcal{A}$ wins Game TE with non-negligible advantage.*

Intuitively, a signature scheme is tamper-evident if there is only a single valid signature for any given message and any given transcript. Note that in practice a verifier (whom we call observer) must check the validity and covert-validity of all the signatures output by the signer. If the signer refuses to engage in the Verify^* protocol with the observer, or (in the non-interactive case which we focus on), if it does not output a proof of covert-validity, the observer announces that the signer failed the test of covert-validity and is thus untrustworthy.

Designing tamper-evident signature schemes In what follows, we give an overview of our technique for designing tamper-evident signature schemes. We start by defining consistency with respect to a pair of deterministic functions.

Definition 2 (Consistent system) *We consider a pseudo-random generator R that given a seed s produces a sequence of outputs, each of some uniform size κ corresponding to an external security parameter. Let the value k_i be the i^{th} output string generated by R, and let $r_i = f(k_i)$ for some one-way function f, where $1 \leq i \leq n$ for some system parameter n (Figure. 1). Finally, consider a sequence of witnesses, w_i for $0 \leq i \leq n$, where the (committed) witness w_0 is made public at setup time. We say that two*

witnesses w_{i-1} and w_i imply that the value r_i is consistent *if and only if $w_{i-1} = h(w_i, r_i)$, where h is a publicly available hash function, and where w_n is a κ-bit random value selected uniformly at random. A value r_i is* consistent with the seed s *if and only if there is a set of witnesses that imply that all values r_j, $1 \leq j \leq i$, are consistent.*

We note that the above definition only considers the case where the observer is undercover; in the more general case we have to replace witnesses by executions of interactive proof protocols.

Tamper-evident signatures We base our constructions of tamper-evident signature schemes on a consistent system. Here, we give only the intuition of our general approach. Precise definitions are found in Sections 4 and 5. Let (k_i, r_i, w_i) be values produced by a consistent system. We define an augmented signature scheme $(\mathsf{Gen}^*, \mathsf{Sign}^*, \mathsf{Verify}^*)$ for which, using the notation of Definition 2:

1. The generation of the i^{th} signature relies on no random number other than k_i;
2. The values r_i, w_i are part of the corresponding augmented signature transcript;
3. The seed to the pseudo-random generator R employed for signature generation is s and the committed witness is w_0.

Proposition 1 *The augmented signature scheme $(\mathsf{Gen}^*, \mathsf{Sign}^*, \mathsf{Verify}^*)$ is tamper-evident according to Definition 1 if the function h is collision-resistant.*

The proof is immediate: in order to win Game TE, the adversary must output a covert-valid signature which is different from that produced by Sign^*, thus producing a collision for the function h.

A Remark on Timing Channels. We have ignored the threat of timing channels and focussed only on covert channels in the data that is being transmitted. A party can use a timing channel to communicate information by encoding the covert message in the delay before a response to a request is produced. This type of threat can be addressed by (a) partitioning the time into intervals of a length sufficient to always generate the response to one request in one time interval, and (b) prescribing that the signer would output a signature on a message at the very end of the time interval after the interval during which the request to sign the message was received. In a system in which signers are observed and always remain *perfectly* synchronized, this approach eliminates the timing channel. Realistically speaking, though, such a measure does not entirely eliminate the channel, but drastically reduces its bandwidth. Good practical measures to further reduce the bandwidth[2] constitute an open research problem.

A Remark on Generation Costs. The random values k_i are used for signatures in order of consecutive increments of the index i, starting at $i = 1$. They are generated from the pseudo-random generator R, as they would have been for regular DSA signatures. In our augmented scheme, the same sequence of values k_i is generated during the setup phase, in order to allow the correct generation of the sequence of witnesses w_i, where w_0 will be made part of the public key of the signer. To not require the signer to store all the witnesses, while avoiding severe computational loads, one may employ fractal hash traversal methods [2] for the generation of the sequence of random values k_i.

4 Tamper-Evident DSA Signatures

We start with a review of the DSA signature scheme [14]. Let p, q be large primes such that $q|(p-1)$, and let $g \in \mathbb{Z}_p^*$ be an element of order q. Let $h : \{0,1\}^* \rightarrow \mathbb{Z}_q$ denote a hash function.

Gen **algorithm.** The secret key is an element $x \in \mathbb{Z}_q^*$ and the corresponding public key is $y = g^x \mod p$.

Sign **algorithm.** To sign a message m, the signer chooses uniformly at random $k \in \mathbb{Z}_q^*$ and computes $r = (g^k \mod p) \mod q$ and $s = k^{-1}(h(m) + xr) \mod q$. The pair $(s, r) \in \mathbb{Z}_q^2$ is a DSA signature on m.

Verify **algorithm.** Given a signature (s, r) on a message m, compute $\omega = s^{-1} \mod q$, $u_1 = h(m)\omega \mod q$, and $u_2 = r\omega \mod q$. Output valid if $r = (g^{u_1} y^{u_2} \mod p) \mod q$; otherwise output invalid.

The DSA signature scheme described above contains an obvious covert channel. Indeed, every DSA signature reveals a value $r = (g^k \mod p) \mod q$, where $k \in \mathbb{Z}_q^*$ is a random value chosen by the signer. At a cost of $2^{\lambda-1}$ modular exponentiations on average, the signer can find a value k such that λ bits of $r = g^k$ are as chosen by the signer. The signer can thus leak at least a few bits of information in every DSA signature it generates. Furthermore, the existence of this covert channel is undetectable to an observer.

[2] For example, one could offset the output time by a small delay of pseudo-randomly determined length, where the seed to this pseudo-random generator is known to the observers. Note that while such observers do know some secret information, they do not have to be trusted with any secret information necessary to generate the signature. We may call such an observer *semi-trusted*.

Tamper-evident variant with undercover observer. We propose a tamper-evident DSA signature scheme in which the signer pre-generates the sequence of random numbers later to be used, and computes witnesses to the elements of this sequence. If any member of the sequence is modified, then the corresponding witness is invalidated. Witnesses are generated in a manner that ensures that it is infeasible to modify these without invalidating the same. Note that the use of pre-generated random numbers will not decrease the level of security of our variant DSA signature scheme. The proof is given below in Proposition. 3.

A formal description of our tamper-evident DSA signature scheme follows:

Gen* **algorithm.** Let $(x, y = g^x)$ be a private/public key pair for DSA output by Gen. After executing Gen, the algorithm Gen* pre-generates the sequence of random values $\{k_i\}$ $(1 \leq i \leq n)$ that are later to be used in the generation of signatures. This is possible given access to the seed to the pseudo-random generator employed. Then, the witnesses $\{w_i\}$ $(0 \leq i \leq n)$ are generated as follows: First, w_n is chosen uniformly at random from $\{0,1\}^\kappa$, for some security parameter κ associated with the choice of hash function h used for witness generation. Consecutive witnesses are generated as follows:

$$w_{i-1} = h(r_i \,||\, w_i) \; (1 \leq i \leq n) \tag{1}$$

where $r_i = (g^{k_i} \mod p) \mod q$.

Finally, the algorithm outputs $K^*_{priv} = (x, \{k_i\}, \{w_i\})$ and $K^*_{pub} = (y, w_0)$.

Sign* **algorithm.** To sign the i^{th} message m_i for $i \geq 1$, the signer computes

$$s_i = {k_i}^{-1}(h(m_i) + x r_i) \mod q \tag{2}$$

The signer outputs the standard DSA signature (m_i, r_i, s_i) along with the previously computed witness w_i. [3] [4]

Verify **algorithm.** Let (m_i, r_i, s_i, w_i) be the i^{th} transcript output by the signer. Any verifier can check that (m_i, r_i, s_i) is a valid DSA signature with respect to the signer's public key y. This is done exactly as for regular DSA signatures by computing $\omega_i = s_i^{-1} \mod q$, $u_1 = h(m_i)\omega_i \mod q$, and $u_2 = r_i\omega_i \mod q$. The output is valid if $r_i = (g^{u_1} y^{u_2} \mod p) \mod q$; otherwise the output is invalid.

Verify* **algorithm.** To verify the covert-validity for the i^{th} transcript (m_i, r_i, s_i, w_i), and given the previous witness w_{i-1}, the observer performs the following computation:

1. The observer runs Verify(m_i, r_i, s_i); if this output is invalid then output invalid* and halt.
2. The observer checks whether the following equation holds:

$$w_{i-1} = h(r_i \,||\, w_i) \tag{3}$$

for the publicly known hash function h. If the equivalence holds, then Verify* outputs valid* and halts; otherwise, it outputs invalid* and halts.

Note that the observer does not need to communicate with the signer in order to verify the consistency of the random numbers employed. Thus, it can avoid revealing its whereabouts until it detects an inconsistency, at which time it draws attention to the corruption by outputting invalid*.

Soundness. An honest signer always succeeds in generating a valid witness as specified by relation (3). Given the assumption of collision-resistance, we see that an adversary cannot have the signer generate and output a pair $\overline{r_i}$, $\overline{w_i}$ satisfying $w_{i-1} = h(r_i \,||\, w_i) = h(\overline{r_i} \,||\, \overline{w_i})$. Thus, it is not feasible to modify any signature transcript without invalidating at least one witness.

Proposition 2 *If the hash function h is collision-free then our variant DSA scheme is tamper-evident.*

This proposition is an immediate corollary of Proposition 1.

Proposition 3 *If the DSA signature scheme is secure against an adaptive chosen message attack, then our variant DSA scheme is also secure against an adaptive chosen message attack in the random oracle model.*

Proof Let $\mathcal{A}$ be an adversary who mounts an existential forgery attack against our tamper-evident variant of DSA. We define an algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to mount an existential forgery attack against the regular DSA signature scheme. We model the hash function h as a random oracle and let algorithm $\mathcal{B}$ answer A's queries to the hash function h.

The algorithm $\mathcal{B}$ receives a public key for DSA and passes it on to $\mathcal{A}$. When $\mathcal{A}$ requests a tamper-evident DSA signature on a message m_i, $\mathcal{B}$ requests a normal DSA signature on m_i and obtains $(m_i, r_i, s_i) = \mathsf{Sign}(m_i, K_{priv})$.

[3] For simplicity, we may assume that all witnesses are stored by the signer after being generated; however, and as previously noted, one can employ fractal traversal techniques in order to reduce the required amount of storage, while maintaining low computational requirements on the scheme.

[4] The lack of integrity protection in the delivery of witness values may be exploited by an attacker to launch an attack to mislead the verification process using Verify* algorithm by simply tampering with the witness values. To defend against such attacks, the signer can append the witness value w_i to the message m_i before proceeding with equation (2).

$\mathcal{B}$ then chooses a random witness w_i and outputs the tamper-evident signature (m_i, r_i, s_i, w_i) for $\mathcal{A}$. When $\mathcal{A}$ queries the hash function h on $r_i \,||\, w_i$, $\mathcal{A}$ answers with $w_{i-1} = h(r_i \,||\, w_i)$. On all other values, $\mathcal{A}$ answers $\mathcal{B}$'s queries with consistent random values, as is standard. □

5 Making Other Schemes Tamper-Evident

Schnorr signatures. The techniques we have used to design a tamper-evident variant of the DSA signature scheme can also be used to design a tamper-evident variant of the Schnorr signature scheme [17]. The Schnorr and DSA signatures schemes are both based on the discrete logarithm problem and share a lot of common features. Let $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ denote the Schnorr signature scheme:

- As in DSA, the algorithm Gen outputs a private/public key pair $(x, y = g^x)$, where the secret key is an element $x \in \mathbb{Z}_q^*$ and the corresponding public key is $y = g^x \mod p$.
- To sign a message m, the signer chooses uniformly at random $k \in \mathbb{Z}_q^*$ and computes $r = g^k \mod p$. Let $c = h(m||r)$ and $s = xc + k \mod q$. The pair $(s, c) \in \mathbb{Z}_q^2$ is the Schnorr signature on m.
- Given a Schnorr signature (s, c) on a message m, Verify computes $v = g^s y^{-c}$ and outputs valid if $c = h(m||v)$; otherwise outputs invalid.

We design a tamper-evident variant $(\mathsf{Gen}^*, \mathsf{Sign}^*, \mathsf{Verify}^*)$ of Schnorr signatures as follows. After executing Gen, the algorithm Gen^* pre-generates the sequence of random values $\{k_i\}$ $(1 \le i \le n)$ and the corresponding witnesses $\{w_i\}$ $(0 \le i \le n)$ almost as in tamper-evident DSA:

$$r_i = g^{k_i} \mod p \quad (4)$$

$$w_{i-1} = h(r_i \,||\, w_i) \quad (5)$$

As in tamper-evident DSA, we define $K^*_{priv} = (x, \{k_i\}, \{w_i\})$ and $K^*_{pub} = (y, w_0)$. The i^{th} tamper-evident Schnorr signature on message m_i is a triplet (s_i, c_i, w_i) where $c_i = h(m_i||r_i)$ and $s_i = xc_i + k_i \mod q$. The tamper-evident verification algorithm Verify^* is defined in exactly the same way as for DSA in section 4.

Feige-Fiat-Shamir signatures. Another application of our techniques it to build a tamper-evident variant of the Feige-Fiat-Shamir (FFS) signature scheme – described in, e.g., [11] – which is based on the earlier signature scheme of Fiat and Shamir [4]. The algorithm Gen in FFS is defined as follows. For two large prime p,q and some system parameter l, the private/public key pair is $(\{u_j\}, \{y_j\})$, $1 \le j \le l$, for $u_j \in \mathbb{Z}^*_{pq}$ and $y_i = {u_j}^{-2}$.

In the tamper-evident variant, the algorithm Gen^* generates a set of random numbers r_i $(1 \le i \le n)$ and then computes the corresponding public values $\{z_i\}$ and a hash chain of witnesses $\{w_i\}$ as follows:

$$z_i = {r_i}^2 \mod pq \quad (6)$$

$$w_{i-1} = h(z_i \,||\, w_i)\ (1 \le i \le n) \quad (7)$$

by choosing hash chain root w_n randomly in $\mathbb{Z}^*_{pq}$.

The i^{th} signature on a message m_i is (e_i, s_i), where e_i denotes the bits of $h(m_i \,||\, z_i)$, $s_i = r_i \prod_{j=1}^{l} {u_j}^{e_j}$ and e_j is the j^{th} bit of e_i. The Verify^* algorithm combined with the verification algorithm of [11] is the same as in the tamper-evident variants of DSA and Schnorr.

6 Conclusion

We have presented attacks on Certificate Authorities based on covert channels that exploit the randomness used to generate signatures in the RSA-PSS, DSA, Schnorr and FFS signature schemes. To detect such attacks, we defined tamper-evident variants of these signature schemes, in which every signature is accompanied by a proof of validity. These proofs are verified by non-interactive observers, whom we call *undercover* observers. Undercover observers can operate stealthily and are thus less vulnerable to attacks aimed at suppressing their activity.

References

[1] J. Choi, P. Golle, and M. Jakobsson. Auditable Privacy: On Tamper-evident Mix Networks. In *Financial Cryptography '02*, 2006.

[2] D. Coppersmith and M. Jakobsson. Almost Optimal Hash Sequence Traversal. In *Financial Cryptography '02*, 2002.

[3] Y. Desmedt. Subliminal-free authentication and signature. In *Advances in Cryptology – Eurocrypt '88*, LNCS 330. Springer-verlag, 1988. pp. 23–33.

[4] A. Fiat and A. Shamir. How to prove yourself: Practical Solution to Identification and Signature Problems. In *Advances in Cryptology – Crypto'86*, LNCS 26. Springer-Verlag, 1987. pp. 186–194

[5] S. Goldwasser, S. Micali and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. SIAM Journal of Computing, Vol. 17, No. 2, 1988. pp. 281–308.

[6] J. Håstad, J. Jonsson, A. Juels, and M. Yung. Funkspiel Schemes: An Alternative to Conventional Tamper Resistance. In *S. Jajodia*, ed., Seventh ACM Conference on Computer and Communications Security, ACM Press, 2000. pp 125–133.

[7] J. Jonsson and B. Kaliski. Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1. Internet RFC 3447. pp. 26–42, February 2003.

[8] M. Jakobsson and M. Yung. Distributed "Magic Ink" Signatures. In *Advances in Cryptology – Eurocrypt '97*, LNCS 1233, Springer-Verlag, 1997. pp. 450–464.

[9] A. Juels and J. Guajardo. RSA Key Generation with Verifiable Randomness. In *Public Key Cryptography 2002*, LNCS 2274, Springer-Verlag, 2002. pp. 357–374.

[10] M. Lepinksi, S. Micali, and A. Shelat. Collusion-Free Protocols. In *STOC '05*, ACM Press, 2005.

[11] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, Handbook Applied Cryptography. CRC Press, 1997. pp. 447–449

[12] C. Karlof, N. Sastry, and D. Wagner, Cryptographic Voting Protocols: A Systems Perspective. In *USENIX Security '05*, August 2005. pp. 33–50.

[13] R. Merkle. Secrecy, authentication, and public key systems. Ph.D. dissertation, Dept. of Electrical Engineering, Stanford Univ., 1979.

[14] National Institute of Standards and Technology (NIST). FIPS Publication 186-2: Digital Signature Standard (DSS). 2000.

[15] D. Pointcheval and J. Stern. Security proofs for signature schemes. In *Advances in Cryptology – Eurocrypt '96*, LNCS 1070, Springer-Verlag, 1996. pp. 387–398.

[16] Pedro A.D. Rezende. Electronic Voting Systems – Is Brazil Ahead of its Time? RSA CryptoBytes, Volume 7, No. 2, 2004.

[17] C. P. Schnorr. Efficient Signature Generation for Smart Cards. In *Proc. of Crypto '89*, 1989. pp. 239–252.

[18] G. J. Simmons. The prisoners' problem and the subliminal channel. In *Proc. of Crypto '83*, 1983. pp. 51–67.

[19] G. J. Simmons. The subliminal channel and digital signature. In *Proc. of Eurocrypt '84*, LNCS 209, Springer-Verlag, 1996. pp. 364–378.

[20] M. E. Smid and D. K. Branstad. Response to comments on the NIST proposed Digital Signature Standard. In *Proc. of Crypto '92*, LNCS 740, Springer-verlag, 1992. pp. 76–87.

[21] M. Stadler. Publicly Verifiable Secret Sharing. In *Advances in Cryptology – Eurocrypt '96*, LNCS 1070, Springer-Verlag, 1996. pp. 190–199.

[22] A. Young. Mitigating insider threats to RSA key generation. In *Cryptobytes*, Vol. 7, No 1, Spring 2004.

[23] A. Young and M. Yung. The prevalence of Kleptographic attacks on discrete-log based cryptosystems. In *Proc. of Crypto '97*, 1997. pp. 264–276.

[24] A. Young and M. Yung. Kleptography: using cryptography against cryptography. In *Proc. of Eurocrypt '97*, 1997. pp. 62–74.

[25] A. Young, M. Yung. Auto-Recoverable and Auto-Certifiable Cryptosystems. In *Advances in Cryptology – Eurocrypt '98*, LNCS 1403, Springer-Verlag, 1998. pp. 119–133.