Tamper-Evident Digital Signatures: Protecting Certification Authorities Against Malware

Jong Youl Choi[1], Philippe Golle[2], and Markus Jakobsson[3]

[1] Computer Science Department, Indiana University at Bloomington, IN 47405
[2] Palo Alto Research Center, 3333 Coyote Hill Rd, Palo Alto, CA 94304
[3] School of Informatics, Indiana University at Bloomington, IN 47406

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 network 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 RSA-PSS and DSA signature schemes and how the same technique can be applied to Feige-Fiat-Shamir (FFS) and Schnorr 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.

Keywords. covert channel, malware, observer, subliminal channel, tamper-evident, undercover

1 Introduction

Malware and insider attacks pose an increasing threat to our infrastructure. Remedies are complicated by the intrinsic difficulties to quickly detect such attacks after they have been mounted. Since current protection mechanisms against malware rely on the collection of detected code in the wild, these measures are not meaningful against carefully targeted attacks, e.g., on crucial nodes in the infrastructure. An example of such a target is a certification authority. The consequences would be severe if an attacker could corrupt a certification authority and 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 infrastructure. The severity of the situation is aggravated by the fact that the leak may be performed using a *covert channel*.

The threat of covert channels (also called subliminal channels) was first studied by Simmons [16, 17] and later by, among others, Young and Yung [21, 22]. These authors expose the risk of an attacker causing the leak of bits of the private key in covert channels present in the randomized key generation or signing algorithm. For sake of concreteness, let us consider the example of how to leak secret information in the RSA-PSS signature scheme [7]. The message encoding scheme of RSA-PSS specifies that the hash of the message 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 type of leakage is detectable: a third party could verify (for each signature) whether the randomness equals the secret key corresponding to the known public key of the signer. Upon the detection of such an event, the signer would be isolated and decommissioned. However, an attacker might just as well leak an *encryption of* the secret key of the signer, where the attacker's public key is used to generate the ciphertext. For a semantically secure encryption scheme, and appropriate parameter choices, such a leakage could be made in a truly covert manner.

It is not only RSA-PSS that is vulnerable to attacks employing covert channels. Discrete-log based signatures schemes, such as Schnorr signatures [15] and DSA [18], can be seen to be just as vulnerable as RSA-PSS, given their use of randomness in the generation of each and every signature. To various extents, however, *all* signature schemes are vulnerable to these attacks, since they all make use of randomness for key generation. Still, the ongoing use of randomness to generate signatures is arguably a much greater security concern than the one-time use of randomness in the initial key-generation step.

In this paper, we raise and address the issue of how to detect (potentially covert) actions caused by the corruption of certification authorities and other generators of digital signatures. In particular, we describe how to implement *tamper-evident* digital signatures; while we do not propose any universal technique to *prevent* corruption, we do show how any corruption can be detected as soon as it causes *any* modification of transcripts. We note that our technique is meaningful not only in the context of malware, but also in the context of insider attacks. One such attack (as described in [14]) could be performed by the implementors of a cryptographic application, and could involve code that *detects* when it is being tested versus when it is being deployed, and switches from an honest to a corrupted mode of operation once the latter situation occurs. The complexity of even the smallest piece of software (and hardware) makes it difficult to detect the presence of such malicious code components, in turn making software (and hardware) audits relatively meaningless. Such attacks, obviously, are going to be very difficult to defend against.

Thus, while most 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 corruption takes place, the victim node replaces its program by one provided by the attacker. Given that the attacker cannot *create* any new communication channels (whether wired or wireless), the only meaningful thing for him to do is to *either* modify the operation of the victim in order to (covertly) leak secret information, or to modify its operation purely for the sake of causing the victim node to generate incorrectly derived transcripts. We assume the existence of an *observer*; this is an external node whose task is to detect any modification of how transcripts are generated. This is not a trivial task given that we do not wish to place any trust with observers (as we would if we provided them with a complete copy of the state of the network nodes they are to observe.) 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.

Our focus is on eliminating the possible existence of covert channels from the signature algorithms of discrete-log based schemes (DSA and Schnorr) and the Feige-Fiat-Shamir (FFS) signature scheme. We believe that our techniques can be extended to any type of signature schemes, and that they can be combined with previously proposed techniques in [9] that instead consider only the key generation phase.

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 a 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, then the observer raises an alarm.

It is meaningful to distinguish between observers that require interaction to verify the absence of covert channels, and those that do not. The latter type – which we name *undercover* observers – provide a greater degree of protection, as these do not need to expose their existence until they raise an alarm. This is better, given that "visible" observers constitute desirable targets of coercion attacks. In our RSA-PSS example, it is easy to see that the observer is undercover. Onwards, we only consider undercover observers, but note that there may exist computational problems that can be made tamper-evident, but for which an undercover observer is impractical or even unattainable.

An important distinction to make is whether the introduction of our techniques modifies the distribution of transcripts (given an initial state of a pseudo-random generator), and if so, whether the modified transcripts are polynomial-time distinguishable from unmodified transcripts. For these distinctions to be meaningful, we must first remove the proofs constituting evidence of tamper-freeness, and only consider the resulting "raw" signatures. Whereas the distribution of the transcripts generated by our modified version of RSA-PSS are distinguishable from that of the standard signatures (given the relation of salts), this can easily be avoided. We show how to make signature schemes tamper-proof using undercover observers, and without altering the distribution of signature transcripts.

It is worth mentioning that our techniques 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 a description of how to make some other signature schemes tamper-evident.

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 [16, 17] 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 secrete message could be hidden inside the authenticator.

A few years later, Desmedt [2] 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 [21, 22] 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 for a user to prove secure generation of his private keys 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. However, there is work in which covert channels are used to achieve a desirable security goal. For example, so-called funkspiel schemes [5] 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 to detect that this is taking place. In contrast, we develop methods to *eradicate* covert channels.

We use the word tamper-evidence to describe a property of an *algorithm*. Traditionally, it is instead used to describe a property of hardware. Smart cards, SIM cards, and satellite decoders all implement varying degrees of physical tamper-evidence, as do funkspiel schemes by means of signaling alerts via the covert channel. 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, 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 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 $\mathsf{Verify}^*(M, K^*_{pub}, (\sigma, \tau), T) \in \{\mathsf{valid}^*, \mathsf{invalid}^*\}$[4].

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 $T_i = T_{i-1} \cup (\sigma_i, \tau_i)$.
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 $(\sigma, \tau) \neq \mathsf{Sign}^*(M, K^*_{priv}, T)$.

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. Finally, consider a sequence of witnesses w_i, $0 \leq i \leq n$, 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.

[4] 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.

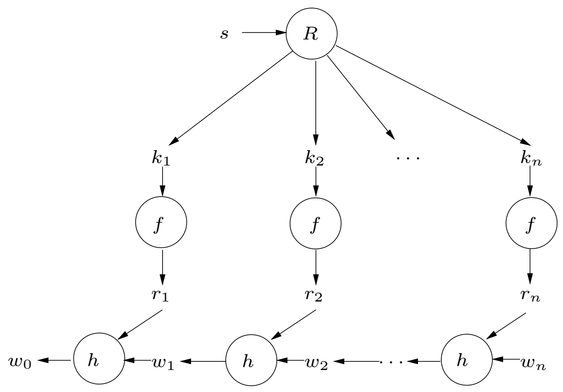

Fig. 1. A model of consistent system: Let the value k_i be the i^{th} output string generated by a pseudo-random generator R that given a seed s and let $r_i = f(k_i)$ for some one-way function f, where $1 \le i \le n$ for some system parameter n. Two witnesses w_{i-1} and w_i imply that the value r_i 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.

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 denotation 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-free.*

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. In the above, we have ignored the threat of timing channels, and only considered 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[5] constitute an open research problem.

[5] 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

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.

In order to avoid having to store all the witnesses until they are output one by one, one may employ a simple extension on the fractal hash traversal methods of Coppersmith and Jakobsson [1] to limit the computational costs per round and the storage requirements to maintain the set of witnesses w_i to $O(\log n)$, where n is the number of signatures that can be generated. This can be done if the values k_i can be computed from the seed given the index i alone, and without having to compute other such values first.

4 Tamper-Evident DSA Signatures

We start with a review of the DSA signature scheme [12]. 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 $h : \{0,1\}^* \to \mathbb{Z}_q$ 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 $w = s^{-1} \mod q$, $u_1 = h(m)w \mod q$, and $u_2 = rw \mod q$. 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.

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.

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.*

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 generates 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[6] witness w_i.

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 $w_i = s_i^{-1} \mod q$, $u_1 = h(m_i) w_i \mod q$, and $u_2 = r_i w_i \mod q$. The output is valid if $r_i = (g^{u_1} y^{u_2} \mod p) \mod q$; otherwise the output is $\mathsf{invalid}$.

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 $\mathsf{Verify}(m_i, r_i, s_i)$; if this output is $\mathsf{invalid}$ then output $\mathsf{invalid}^*$ 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 $\mathsf{invalid}^*$ 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 $\mathsf{invalid}^*$.

[6] 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.

Soundness. An honest signer always succeeds in generating a valid witness as specified by relation (3). Given the assumption of collision-freeness, 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.*

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 $(m_i, r_i, s_i) = \mathsf{Sign}(m_i, K_{priv})$. $\mathcal{B}$ 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. $\square$

5 Making Other Signature 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 [15]. 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 $\mathsf{invalid}$.

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 $\{k_i\}$ $(1 \leq i \leq n)$ and the corresponding witnesses $\{w_i\}$ $(0 \leq i \leq n)$ almost as in tamper-evident DSA:

$$r_i = g^{k_i} \mod p \tag{4}$$

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

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 both [10] and [6]– which is based on the earlier signature scheme of Fiat and Shamir [3]. 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 $(\{u_j\}, \{y_j\})$, $1 \leq j \leq l$, 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 $(1 \leq i \leq n)$ and then computes the corresponding public values $\{z_i\}$ and a hash chain of witnesses $\{w_i\}$ as follows:

$$z_i = {r_i}^2 \mod pq \tag{6}$$

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

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 [10] and [6] 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 *under-cover* observers. Under-cover observers can operate stealthily and are thus less vulnerable to attacks aimed at suppressing their activity.

References

1. D. Coppersmith and M. Jakobsson, "Almost Optimal Hash Sequence Traversal," Financial Cryptography '02.
2. Y. Desmedt. Subliminal-free authentication and signature. In *Advances in Cryptology – Eurocrypt '88*, LNCS 330. Springer-verlag, 1988, pp. 23–33.
3. 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
4. S. Goldwasser, S. Micali and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. In *SIAM Journal of Computing*, Vol. 17, No. 2, pp. 281–308. 1988.
5. J. Hȯastad, J. Jonsson, A. Juels, and M. Yung J. Hastad. 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.
6. M. Jakobsson. Reducing costs in identification protocols. In *Rump session on Crypto '92*. 1992.
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. A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, Handbook Applied Cryptography, CRC Press, 1997, pp. 447–449
11. R. Merkle, Secrecy, authentication, and public key systems, Ph.D. dissertation, Dept. of Electrical Engineering, Stanford Univ., 1979.
12. National Institute of Standards and Technology (NIST). FIPS Publication 186-2: Digital Signature Standard (DSS), 2000.
13. D. Pointcheval and J. Stern. Security proofs for signature schemes. In *Advances in Cryptology – Eurocrypt '96*, LNCS 1070. Springer-Verlag, 1996, pp. 387–398.
14. Pedro A.D. Rezende. Electronic Voting Systems – Is Brazil Ahead of its Time? RSA CryptoBytes, Volume 7, No. 2, 2004.
15. C. P. Schnorr. Efficient Signature Generation for Smart Cards. In *Proc. of Crypto '89*, pp. 239–252.
16. G. J. Simmons. The prisoners' problem and the subliminal channel. In *Proc. of Crypto '83*, pp. 51–67.
17. G. J. Simmons. The subliminal channel and digital signature. In *Proc. of Eurocrypt '84*, LNCS 209, Springer-Verlag, 1996, pp. 364–378.
18. 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.
19. M. Stadler, Publicly Verifiable Secret Sharing, In *Advances in Cryptology – Eurocrypt '96*, LNCS 1070, Springer-Verlag, 1996, pp. 190–199.
20. A. Young. Mitigating insider threats to RSA key generation. In *Cryptobytes*, Vol. 7 No 1, Spring 2004.
21. A. Young and M. Yung. The prevalence of Kleptographic attacks on discrete-log based cryptosystems. In *Proc. of Crypto '97*, pp. 264–276.
22. A. Young and M. Yung. Kleptography: using cryptography against cryptography. In *Proc. of Eurocrypt '97*, pp. 62–74.
23. A. Young, M. Yung. Auto-Recoverable and Auto-Certifiable Cryptosystems. In *Advances in Cryptology – Eurocrypt '98*, LNCS 1403, Springer-Verlag, 1998, pp. 119–133.