Expecting the Unexpected: Towards Robust Credential Infrastructure

Shouhuai Xu[1] and Moti Yung[2]

[1] Department of Computer Science, University of Texas at San Antonio
shxu@cs.utsa.edu
[2] Google Inc. and Department of Computer Science, Columbia University
moti@cs.columbia.edu

Abstract. Cryptographic credential infrastructures, such as Public key infrastructure (PKI), allow the building of trust relationships in electronic society and electronic commerce. At the center of credential infrastructures is the methodology of digital signatures. However, methods that assure that credentials and signed messages possess *trustworthiness* and *longevity* are not well understood, nor are they adequately addressed in both literature and practice. We believe that, as a basic engineering principle, these properties have to be built into the credential infrastructure rather than be treated as an after-thought since they are crucial to the long term success of this notion. In this paper we present a step in the direction of dealing with these issues. Specifically, we present the basic engineering reasoning as well as a model that helps understand (somewhat formally) the trustworthiness and longevity of digital signatures, and then we give basic mechanisms that help improve these notions.

Keywords. Credential infrastructures, PKI, digital signatures, key compromise, hit-and-run attack, hit-and-stick attack, insider attack.

1 Introduction

The celebrated notion of digital signing was put forth as modern cryptography started. Its security definition [10], and the security of many of its derived notions (like that of group signature [5]), does not capture the fact that the signature lives in a system and does not assure the *trustworthiness* and *longevity* of digital signatures over time within a system context, due to the following reasons.

First, trustworthiness of digital signatures is questionable when a verifier does not have other means to determine that a digital signature was indeed issued or activated by the alleged signer (as was mentioned in [8]). To see this, observe that private signing keys can be compromised in real computer systems (cf., for example, [24, 13] for practical attacks). Such attacks can, in fact, defeat even advanced digital signature techniques (e.g., forward-secure signing [1, 3], key-insulated signing [7], intrusion-resilient signing [17], threshold signing [6], proactive signing [23]), although the damage may be mitigated. Moreover, even if a private signing key is stored in a tamper-resistant hardware (e.g., cryptographic

co-processor [29] or Trusted Platform Module [26]) that may also frustrate side-channel attacks [19], the private signing function could still be compromised because an attacker, who has compromised the computer of a signer, can request the hardware to sign messages [21]. Note that compromises like the above (i.e., access to the signing function) were assumed also in the digital signature security definition of [10] as well as in the relevant variants (e.g., threshold and proactive signatures [14], group signing [2]).

Second, longevity of digital signatures becomes questionable because a dishonest signer can later "plausibly" repudiate some (past) signatures. To see this, it has been observed early on that a dishonest signer could abuse her private signing key or function to commit unlawful activities, while blaming them to the attacker who has compromised the private signing key. As an extreme example, a dishonest signer can launch attacks against her own computer so as to fool the machine forensics mechanisms and commit fraud without being held accountable. Note that such threats were not accommodated in the cryptographic model of digital signatures [10] as well as its variants (e.g., [14, 2]), because it always treats the signers as the target of attacks.

The above two threats to trustworthiness and longevity of (non-anonymous and anonymous) digital signatures are **inevitable** due to the imperfection of forensics analysis mechanisms. Moreover, there is an **unexpected** threat to the trustworthiness and longevity of digital signatures — many or all of the cryptographic keys in use may be compromised — either due to a fundamental progress in cryptanalysis (e.g., a polynomial-time factorization algorithm) or due to the more likely Trojan Horses in operating systems and/or hardware devices. A specific attack of this kind is the recent incident of rogue CA [25].

Our contributions. We propose a novel model (Section 2) for helping understand the trustworthiness and longevity of digital signatures based on various realistic threats. Our model has the following features. First, it accommodates a participant we call "liability-holder" (e.g., an employer, an insurance vendor or the signer herself), which is responsible for the consequences of digital signatures. This allows us to capture insider threats (i.e., malicious signers). Second, it reflects the strength of the relevant security mechanisms: (1) *compromise-resistant* mechanisms that may be deployed to prevent attacks from compromising the private signing keys or functions, or from compromising the signers; (2) *compromise-detection* mechanisms that may be deployed to detect the compromise of private signing keys or functions; (3) *history-preservation* mechanisms that may be deployed to ensure the integrity of the system history state information; and (4) *forensics-like analysis* mechanisms that may be deployed to determine when an attack actually occurred. Third, it brings a useful concept of "grey period," during which there may be some signatures for which we do not know for sure who should be held accountable: the private signing key owners, or the attacker that has compromised the private signing keys or functions.

Our model suggests as an ultimate goal to eliminate the grey periods (i.e., uncertainties) which capture the afore-mentioned **inevitable** threats against trustworthiness and longevity of digital signatures. Although we are unable to

accomplish this, we present a solution (Section 3) reducing the length of grey periods. Our solution is based on a digital signature anchoring service, whereby digital signatures can be deposited at some servers that are operated by agents we call anchors. The anchors and servers are only *semi-trusted* because they cannot frame the honest users, and any misbehaving anchors/servers can be immediately detected by any honest participant.

Finally, we extend (in Section 7) our solution to deal with the **unexpected** threats that many or all of the employed cryptosystems may be broken.

Related works. We are not aware of works similar to the model we put forth here. On the other hand, regarding our concrete solution to dealing with the *inevitable*, namely for reducing the length of grey periods, there are three related prior works. These three important works represent the state of uneasiness regarding the actual trust and robustness of credential mechanisms. First among them is digital timestamping, due to Haber and Stornetta [12], which aimed at improving the trustworthiness of digital signatures. However, the similarity between timestamping and our solution is limited to the fact that both of them use collision-resistant hash functions to build data structures that are variants of Merkle trees [22]. Whereas, the important differences between them are the following. (i) Digital timestamping only asserts when a signature was issued, and does not offer any extra assurance that a signature was indeed issued by the alleged signer. That is, timestamping cannot deal with what we will call hit-and-run and insider attacks, which are alleviated by our solution. (ii) Our data structure adopts a "signature verification keys"-oriented organization, which leads to convenient queries and signature verifications even if the past signatures are truncated (so as to avoid monotonic increase of the tree size). This has no counterpart in [12].

Second, Just and van Oorschot [18] investigated the problem of undetected key compromises, and proposed an architecture-centric solution by introducing a third party. Although their solution bears some similarity to ours, there are important differences. (i) They assumed that each user has two cryptographic keys — one private signing key and one symmetric message authentication key — such that compromise of one does not mean compromise of the other. In their suggested scenario, one key may be stored on a user's local computer whereas the other is stored on a hardware token. In contrast, we assume that a user may keep her keys on a single computer, which may be compromised. (ii) The third party in their model is assumed to be *fully-trusted*; otherwise, their constructions would allow the third party, who may be colluding with an attacker that may have compromised the private signing key of an honest user, to frame the honest user without being held accountable. In contrast, the third party in our solution is only *semi-trusted* because it has no power to frame any user and its misbehavior can be detected by any honest user.

Third, Itkis [15] investigated a primitive-centric method, which requires the use of absolutely random bits (i.e., even pseudorandom bits are not sufficient) due to a subtle technical reason. This is very restrictive, and our architecture-centric method does not suffer from this (i.e., pseudorandomness is sufficient

in our approach). Moreover, Itkis [15] did not consider the important issue of managing digital signatures, whereas we do.

Outline. Section 2 presents our model of digital signature trustworthiness and longevity. Section 3 presents a solution framework for reducing the length of grey periods. Before presenting an instantiation of the framework in Section 6, we present two building-blocks in Section 4 and Section 5, respectively. In Section 7 we discuss how to deal with the unexpected situation where cryptosystems in use are broken. Section 8 concludes this paper with some open problems. Due to space limitation, we leave the review of cryptographic primitives to Appendix A and analyses of the schemes to the full version of the present paper [28].

2 Modeling Signature Trustworthiness and Longevity

Participants. We consider a *liability-holder* in addition to the signer, verifier and attacker in the cryptographic model of digital signatures [10]. The signer or user u has a pair of public and private keys $(\mathsf{pk}_u, \mathsf{sk}_u)$ with respect to some secure signature scheme (in the sense of [10]). We assume that pk_u is published via some reliable means (e.g., certified by a certificate authority). Since the private key sk_u is often stored on u's computer, it can get compromised (e.g., when u's computer is compromised). Moreover, u can become dishonest or malicious at some point in time. We assume that all the participants are PPT algorithms.

Adversary. In addition to the traditional attacker based on pure cryptanalysis, we consider three attacks against the trustworthiness and longevity of digital signatures. Among the three attacks, which we call *hit-and-run*, *hit-and-stick* and *insider*, we are only able to deal with the hit-and-run attack and the insider attack (dealing with the hit-and-stick attack is a challenging open problem).

- Hit-and-run attack: Such an adversary compromises u's computer, steals the private key sk_u, and then leaves the computer (i.e., does not reside on the computer or tamper with it). The adversary may abuse the compromised sk_u to produce digital signatures that can be verified using pk_u.
- Hit-and-stick attack: Such an adversary resides on the victim computer after compromising it (e.g., by embedding Trojan Horses or tampering with the system). In this case, the victim computer is virtually controlled by the adversary until the compromise is detected. This is a very powerful attack that dismisses many countermeasures. For example, deploying a mechanism to tell a computer program and a human being apart (in hope of ensuring that every signing request is issued by a human being) does not necessarily defeat the attack, as long as the mechanism is implemented on the same victim computer. Defeating such an adversary is left open, and seems to require independent replication (in different machines and so on).
- Insider attack: Such an attack is launched by u herself. In the case that some third party (e.g., employer of u or some insurance provider) is the liability-holder for signatures generated using sk_u, the attack is clearly possible. Even if u is the liability-holder, the attack is still possible because u may have the

incentive to deny some (past) signatures by blaming them on the attacker who compromised sk_u. Moreover, u can launch an attack against sk_u so that she can attribute signatures to the compromise of sk_u.

Private signing key lifecycle. As depicted in Figure 1, a private signing key sk_u becomes effective (e.g., via the certification of pk_u) at time T_0, ceases functioning (e.g., via the immediate revocation of pk_u) at time T_4 because its compromise has become evident. At time T_4, forensics-like analysis may be invoked to help determine the time interval $[T_0, T_1]$ during which neither the user u nor the private key sk_u was compromised, and the time interval $[T_3, T_4]$ such that the user u or the private key sk_u was compromised at time no later than T_3. The interval $[T_1, T_3]$ can be seen as the approximation of T_2, which is the actual time at which u or sk_u gets compromised but may be known only to the attacker (i.e., the defender may never discover the time T_2 for certain).

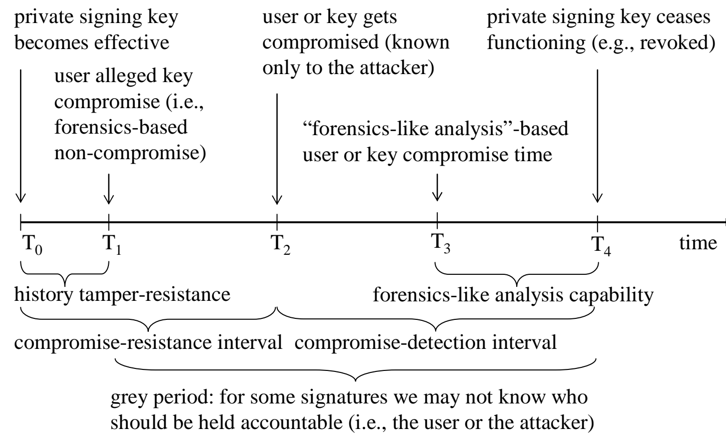


Fig. 1. A scenario of private signing key lifecycle

Applying the model to analyze the hit-and-run and hit-and-stick attacks. The following observations (see the lower-half of Figure 1) apply to both attacks, no matter if u is the liability-holder or not. First, the time interval $[T_3, T_4]$ captures the capability of the forensics-like mechanisms in after-the-fact investigation of attacks. A better capability means a longer $[T_3, T_4]$ or smaller T_3. Second, the time interval $[T_0, T_2]$ captures the security strength of u's computer system in tolerating attacks. A better security means a longer $[T_0, T_2]$. Third, the time interval $[T_2, T_4]$ captures the capability of the mechanisms for detecting compromises. Fourth, the time interval $[T_1, T_4]$ is called the "grey period" because there may exist some signatures that were issued during this time interval, but cannot be attributed to the actual producer (i.e., u or the adversary who has compromised sk_u) with certainty.

Applying the model to analyze the insider attacks. The following observations apply, again, regardless of whether u is the liability-holder or not. First, the time interval $[T_0, T_1]$ captures strength of u's computer in tolerating tampering attacks. This is important because u may be honest at time T_0, becomes dishonest at time T_2, and may have the incentive to tamper the system history information. A better history tamper-resistance means a longer $[T_0, T_1]$. Second, the time interval $[T_3, T_4]$ captures the capability of the forensics-like mechanisms for after-the-fact investigation of attacks. A better capability means a longer $[T_3, T_4]$. Third, the time interval $[T_0, T_2]$ captures the security strength of u's computer system in tolerating attacks (e.g., preventing or deterring u from being compromised). A better security means a longer $[T_0, T_2]$. Fourth, the time interval $[T_2, T_4]$ captures the capability of the mechanisms for detecting compromises. Fifth, the time interval $[T_1, T_4]$ is again the "grey period."

Properties of ideal solutions. The above analyses offer the following insights. First, it is an ideal case to maximize the interval $[T_0, T_1]$. Namely, to make u's system history state information tamper-resistant, which means that $T_2 - T_1 = 0$ and thus the signer cannot deny any past signatures it generated before u or sk_u is compromised. Second, it is the ideal case to maximize the interval $[T_3, T_4]$, namely to deploy perfect forensics-like mechanisms for after-the-fact investigation of compromise of either u or sk_u, which would imply $T_3 - T_2 = 0$. Third, it is ideal to maximize the interval $[T_0, T_2]$ by enhancing the security of a user's computer and/or private signing key sk_u. Fourth, it is ideal to minimize the interval $[T_2, T_4]$. Since it may not be possible to absolutely prevent the compromise of keys or computers, we do need mechanisms that can detect their compromise as soon as possible. We require that the compromise-detection mechanism have *no false positives*, although this may imply that it can return an answer like "I don't know." This is crucial because in many cases the disputes can lead to lawsuits that require reliable evidence. Fifth, it is ideal to minimize the "grey period" interval $[T_1, T_4]$. Given that private signing keys can eventually get compromised and that the attackers may not always get held accountable, insurance would become a very useful mechanism for enhancing the trustworthiness of digital signatures (e.g., a signature assured either by an employer or by a third party would be more trustworthy). It is thus important to shorten the "grey period" so as to protect the liability-holder.

3 DSAS: A Framework for Reducing Grey Periods

Our solution framework, as depicted in Figure 2, is called Digital Signature Anchoring Service (DSAS). In this framework, we consider a set of users or signers, verifiers, and anchors that can be the liability-holders or some economically-motivated third parties. Note that some participants may play the roles of both verifiers and signers. Suppose that each signer has a pair of public and private keys with respect to a digital signature scheme that is secure in the sense of Goldwasser et al. [10]. The anchors are assumed to be highly secure — their systems cannot be compromised by average attackers (nevertheless, in Section 7 we

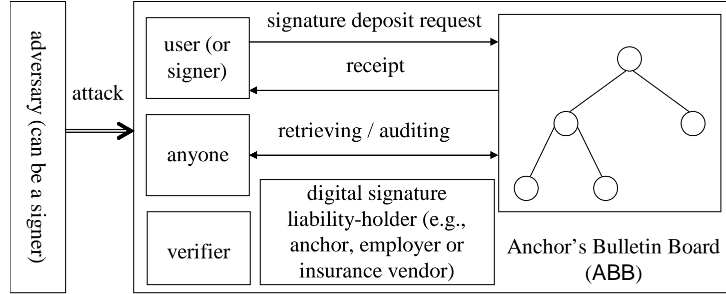


Fig. 2. DSAS framework

will discuss what if the anchors' cryptosystems may be compromised). However, the anchors are assumed to be only *semi-trusted*, meaning that they may launch attacks against some honest signers as long as such attacks cannot be traced back to them. An anchor maintains an Anchor's Bulletin Board (ABB), which is used to publish the digital signatures deposited at it.

Definition 1. (DSAS) *A* DSAS *scheme consists of the following protocols.*
DSAS.Initialization*: Given a primary security parameter κ, each participant generates the cryptographic keys. The signature verification keys are appropriately published. Moreover, a data structure called Anchor's Bulletin Board (*ABB*) is initialized as* ABB_0 *at time* t_0*.*
DSAS.Deposit*: Suppose the current content of* ABB *is* ABB_{i-1}*, which was updated at time* t_{i-1}*. When a signer, u, deposits a signature, sig, the anchor* authenticates *u as well as sig. If both authentications succeed (i.e.,* TRUE $\leftarrow$ $\mathsf{DSAS.Deposit}(1^\kappa, u, sig)$*), the anchor returns a receipt (e.g., the anchor signs the message that "sig will appear in* ABB_i *at time* t_i*"). Note, as mentioned above, that the upper-layer application of the* DSAS *service is orthogonal to the focus of the present paper.*
DSAS.Update*: Denote by Δ_i the signatures deposited after time t_{i-1}. At time t_i, the anchor updates* ABB_{i-1} *as* $\mathsf{ABB}_i \leftarrow \mathsf{DSAS.Update}(1^\kappa, \mathsf{ABB}_{i-1}, \Delta_i)$*. The anchor may send back to every user, who deposited a signature after t_{i-1}, an "attestation" indicating how the user may verify that her signature is appropriately published in* ABB_i*.*
DSAS.Retrieve*: Kinds of queries can be issued with respect to* ABB_i*, dependent upon the applications. The first example is for a signer to check that her deposited signature appropriately appeared in* ABB_i*. The second example is for anyone to check that, given* ABB_{i-1} *and* Δ_i*,* ABB_i *was appropriately updated. The third example is for anyone to retrieve the signatures deposited during a time period.*

To define security of DSAS, let adversary $\mathcal{A}$ have access to the following oracles: $Init(1^\kappa)$, which executes $\mathsf{DSAS.Initialization}(1^\kappa)$; $Deposit(1^\kappa, u, sig)$, which executes $\mathsf{DSAS.Deposit}(1^\kappa, u, sig)$ that returns TRUE; $Update(1^\kappa, \mathsf{ABB}_{i-1}, \Delta_i)$,

which executes $\mathsf{DSAS.Update}(1^\kappa, \mathsf{ABB}_{i-1}, \Delta_i)$; $Retrieve(1^\kappa, \ldots)$, which executes $\mathsf{DSAS.Retrieve}(1^\kappa, \ldots)$; $Corr$, which captures that the anchor becomes dishonest and returns all the secrets of the anchor; $HaR(u)$, which captures the hit-and-run attack and returns the cryptographic secrets of signer u; $Insider(u)$, which turns an honest signer u into an insider attacker.

Note that $HaR(\cdot)$, $Insider(\cdot)$, and $Corr$ may be queried immediately after querying $Init$ so as to accommodate the situations where some participants are compromised at system initialization. Note also that multiple $Init$ queries may be made, but we only need to consider one in which the attacker may succeed. The notations $\exists$ and $\nexists$ indicate whether a specific query was ever made. In order to capture the successful attack events, we allow $\mathcal{A}$ to invoke DSAS.Deposit and DSAS.Update. Such executions are different from the $Deposit$ and $Update$ oracle queries, which lead to executions on behalf of the signers or the anchor. Formally,

Definition 2. (properties of DSAS) *A* DSAS *scheme should possess:*
DSAS.correctness*: If the signers and the anchor are honest, the anchor's* ABB *is always appropriately updated with respect to the deposits and, for any i, anyone can verify* $\mathsf{ABB}_i = \mathsf{DSAS.Update}(1^\kappa, \mathsf{ABB}_{i-1}, \Delta_i)$.
DSAS.no-impersonation*: The probability that an attacker successfully impersonates an honest signer, whose cryptographic secrets are not compromised, to the anchor is negligible in κ. Formally,*

$$\Pr\left[\begin{array}{l}\sigma \leftarrow \mathcal{A}^{Init(1^\kappa), Deposit(1^\kappa,\cdot,\cdot), Update(1^\kappa,\cdot,\cdot), Retrieve(1^\kappa,\ldots), HaR(\cdot), Insider(\cdot)}(1^\kappa): \\ \nexists Insider(u) \wedge \nexists HaR(u) \wedge \text{TRUE} \leftarrow \mathsf{DSAS.Deposit}(1^\kappa, u, sig)\end{array}\right] = \epsilon(\kappa),$$

DSAS.uniqueness*: The probability for anyone to provide* $\mathsf{ABB}'_{i-1} \neq \mathsf{ABB}_{i-1}$ *or* $\Delta_i \neq \Delta'_i$ *but* $\mathsf{DSAS.Update}(1^\kappa, \mathsf{ABB}_{i-1}, \Delta_i) = \mathsf{DSAS.Update}(1^\kappa, \mathsf{ABB}'_{i-1}, \Delta'_i)$ *is negligible in κ.*

$$\Pr\left[\begin{array}{l}(i, \mathsf{ABB}_{i-1}, \mathsf{ABB}'_{i-1}, \Delta_i, \Delta'_i) \leftarrow \\ \mathcal{A}^{Init(1^\kappa), Deposit(1^\kappa,\cdot,\cdot), Update(1^\kappa,\cdot,\cdot), Retrieve(1^\kappa,\ldots), HaR(\cdot), Insider(\cdot), Corr}(1^\kappa): \\ \mathsf{DSAS.Update}(1^\kappa, \mathsf{ABB}_{i-1}, \Delta_i) = \mathsf{DSAS.Update}(1^\kappa, \mathsf{ABB}'_{i-1}, \Delta'_i) \\ \wedge(\mathsf{ABB}_{i-1} \neq \mathsf{ABB}'_{i-1} \vee \Delta_i \neq \Delta'_i)\end{array}\right] = \epsilon(\kappa),$$

DSAS.attack-evidence*: Suppose the private key of an honest signer was compromised at time t and the attacker deposited a signature using the compromised key at time $t' > t$. Then this compromise can be detected when the victim user deposits her first signature at time $t^* > t' > t$. Moreover, given two conflicting signatures, it is possible to infer when the signer's computer compromised by a hit-and-run attack or the signer became an insider.*

4 Building-block I: Anchor's Bulletin Board (ABB)

Definition 3. (ABB) *An* ABB *scheme consists of the following algorithms.*
ABB.Initialization*: Initialization of the data structure* ABB_0 *at time* t_0.
ABB.Update*: Denote by Δ_i the signatures deposited by the honest users after time t_{i-1}. At time t_i, the anchor updates* ABB_{i-1} *to* ABB_i*, where* $\mathsf{ABB}_i \leftarrow \mathsf{Update}(1^\kappa, \mathsf{ABB}_{i-1}, \Delta_i)$.
ABB.Retrieve*: Kinds of queries can be issued with respect to* ABB_i*, dependent on the applications.*

To define security of ABB, let adversary $\mathcal{A}$ have access to the following oracles: $Init(1^\kappa)$, which executes $\mathsf{ABB.Initialization}(1^\kappa)$; $Update(1^\kappa, \mathsf{ABB}_{i-1}, \Delta_i)$, which executes $\mathsf{ABB.Update}(1^\kappa, \mathsf{ABB}_{i-1}, \Delta_i)$; $Retrieve(1^\kappa, \ldots)$, which executes $\mathsf{ABB.Retrieve}(1^\kappa, \ldots)$; $Corr$, which captures that an anchor becomes dishonest and returns all the secrets of the anchor. Note also that multiple $Init(1^\kappa)$ queries may be made, but we only need to consider one in which the attacker may succeed. To capture the successful attack events, we allow the attacker to explicitly execute $\mathsf{ABB.Update}$, which is different from the oracle query of $Update(1^\kappa, \cdot, \cdot)$.

Definition 4. (properties of ABB) *An* ABB *scheme should have:*
ABB.correctness*:* ABB *is always appropriately updated, meaning that anyone can always verify that* $\mathsf{ABB}_i = \mathsf{ABB.Update}(1^\kappa, \mathsf{ABB}_{i-1}, \Delta_i)$ *for any* i.
ABB.uniqueness*: The probability for anyone to provide* $\mathsf{ABB}_{i-1} \neq \mathsf{ABB}'_{i-1}$ *or* $\Delta_i \neq \Delta'_i$ *but* $\mathsf{ABB.Update}(1^\kappa, \mathsf{ABB}_{i-1}, \Delta_i) = \mathsf{ABB.Update}(1^\kappa, \mathsf{ABB}'_{i-1}, \Delta'_i)$ *is negligible in* κ. *Formally,*

$$\Pr\left[\begin{array}{l}(i, \mathsf{ABB}_{i-1}, \mathsf{ABB}'_{i-1}, \Delta_i, \Delta'_i) \leftarrow \mathcal{A}^{Init(1^\kappa), Update(1^\kappa,\cdot,\cdot), Retrieve(1^\kappa,\ldots), Corr}(1^\kappa): \\ \mathsf{ABB.Update}(1^\kappa, \mathsf{ABB}_{i-1}, \Delta_i) = \mathsf{ABB.Update}(1^\kappa, \mathsf{ABB}'_{i-1}, \Delta'_i) \\ \quad \wedge(\mathsf{ABB}_{i-1} \neq \mathsf{ABB}'_{i-1} \vee \Delta_i \neq \Delta'_i)\end{array}\right] = \epsilon(\kappa),$$

Construction. We design ABB as a three-level, binary Merkle hash tree. At the bottom there are many Level 3 hash trees, each of which represents the signatures with respect to a signature verification key. At the middle there is a single Level 2 hash tree, each leaf of which corresponds to the root of a Level 3 hash tree. At the top there is a single Level 1 hash tree, the right-most child of which corresponds to the root of the Level 2 hash tree. The ABB is "signature verification keys"-oriented. This is to allow efficient queries about (some) digital signatures with respect to a public verification key deposited during a time interval. This is fulfilled by retrieving only one leave of the Level 2 hash tree.

Given ABB published at time t_i, we denote by $M_j(t_i)$ the root of the jth Level 3 hash tree in ABB_i, by $N(t_i)$ the root of the Level 2 hash tree in ABB_i, and by $R(t_i)$ the root of the Level 1 hash tree in ABB_i.

An ABB may be *signatures-preserved*, meaning that all the signatures that have been deposited so far appear in the ABB (i.e., the size of the ABB is monotonically increasing), or *signatures-compressed*, meaning that only the most recently deposited signatures explicitly appear in the ABB whereas previously deposited signatures are "compressed" in a certain way. We will mention their differences in our ABB construction, which is given below and analyzed in [28]. An illustrative example is given in Appendix B.

ABB.Initialization: ABB_0 is initiated at time t_0 as a Level 2 hash tree, whose root $N(t_0) = R(t_0)$ is the Level 1 hash tree (i.e., a single node tree at this point), and leaves $M_1(t_0), M_2(t_0), \ldots$ correspond to the individual-wise or group-wise signature verification keys.

ABB.Update: Denote by $\Delta_{j,i}$ an ordered set (or list) of the signatures that have been deposited with respect to the jth Level 3 hash tree since time t_{i-1}. At time t_i, the anchor executes the following to update ABB_{i-1} to ABB_i.

1. Update the Level 3 hash trees: There are two cases.

Signatures-preserved case: For each j, the jth Level 3 hash tree with root $M_j(t_{i-1})$ in ABB_{i-1} becomes the left-most leaf node in the new jth Level 3 hash tree, and the signatures in $\Delta_{j,i}$ appear as the other leaf nodes, whose left-to-right order corresponds to the order at which they were deposited.

Signatures-compressed case: For each j, the root $M_j(t_{i-1})$ of the jth Level 3 hash tree in ABB_{i-1} (i.e., every node other than the root is "pruned") becomes the left-most leaf node in the new Level 3 hash tree, and the signatures in $\Delta_{j,i}$ appear as the other leaf nodes, whose left-to-right order corresponds to the order at which they were deposited.

The values of the roots of the new Level 3 hash trees in ABB_i are computed as usual. In the ideal case, the new Level 3 hash trees are perfect binary trees.

2. Update the Level 2 hash tree: After updating the roots of the new Level 3 hash trees, the value of the root of the new Level 2 hash tree is also updated as $N(t_i)$.
3. Update the Level 1 hash tree: The root of the Level 1 hash tree in ABB_{i-1}, namely $R(t_{i-1})$, becomes the left child of the new Level 1 hash tree in ABB_i. The root of the new Level 2 hash tree, namely $N(t_i)$, becomes the right child of, the new Level 1 hash tree in ABB_i. The value of the root of the new Level 1 hash tree, namely $R(t_i)$, is computed as usual. The resulting signature as well as the new Level 1, 2, and 3 trees are published as ABB_i.

ABB.Retrieve: Kinds of queries can be issued with respect to ABB_i, dependent upon the applications. Examples are: *First*, given a signature verification key corresponding to the jth Level 3 hash tree and a time interval $[t, t']$, one can immediately find all the signatures deposited during that time period in the signatures-preserved case. This can be done by, for example, computing the difference between the corresponding two Level 3 trees updated at time t and t', respectively. In the case only one copy of the tree is preserved (although this is unlikely because storage is getting cheaper and cheaper), the same task can be done by extending the root of Level 3 trees to include the time at which the ABB, and thus the Level 3 trees, are updated. Similarly, given a time interval $[t, t']$, one can find all the signatures deposited during that time period in the signatures-preserved case (e.g., by combining the signatures deposited during that period of time). *Second*, given a signature deposit receipt, one can immediate check whether the signature in question does appear in the Level 3 tree corresponding to the signature verification key. *Third*, given the "attestation" of a deposited signature — the values of the siblings of the nodes on the path from the signature in question to the root, one can immediately check whether the attestation ends at a leaf or the root of the Level 1 hash tree. If there is anything wrong, a complaint is issued against the anchor. The validity of the complaint can be checked by any honest party (e.g., judge) or in a distributed fashion, and the dishonest participant may be appropriately punished.

5 Building-block II: Stateful Authentications (AUTH)

Definition 5. (stateful authentication method) *A stateful authentication method* AUTH *consists of the following (interactive) algorithms.*

AUTH.Initialization*: Given a primary security parameter κ, this (interactive) algorithm bootstraps some cryptographic contexts. Moreover, the user u maintains some state information $\mathsf{state}_{u,v}$ and the verifier v maintains some state information $\mathsf{state}_{v,u}$.*

AUTH.Authentication*: This is an interactive algorithm run by u and v.*

1. *The user u presents the verifier v a bitstring η, a function of $\mathsf{state}_{u,v}$.*
2. *Upon receiving from u a bitstring η, v executes an algorithm to decide whether to accept the bitstring. The decision is based on, among other things, the state information $\mathsf{state}_{v,u}$. If the authentication is successful, denote it by $\text{TRUE} \leftarrow \mathsf{AUTH.Authentication}(1^\kappa, \langle \eta, \mathsf{state}_{u,v}\rangle, \mathsf{state}_{v,u})$.*
3. *If v accepts, v updates $\mathsf{state}_{v,u}$ and u updates $\mathsf{state}_{u,v}$ appropriately.*

To define security of AUTH, let $\mathcal{A}$ have access to the following oracles: $Init(1^\kappa)$, which executes $\mathsf{AUTH.Initialization}(1^\kappa)$; $Authentication(1^\kappa, u, v)$, which executes $\mathsf{AUTH.Authentication}(1^\kappa, \langle \eta, \mathsf{state}_{u,v}\rangle, \mathsf{state}_{v,u})$ that returns TRUE; $HaR(u)$, which captures the hit-and-run attack and returns the cryptographic secrets of signer u; $Insider(u)$, which turns an honest signer u into an insider attacker. Note that $HaR(\cdot)$ and $Insider(\cdot)$ may be queried immediately after querying $Init(1^\kappa)$ so as to accommodate the situations where some participants are compromised at system initialization. Note also that multiple $Init(1^\kappa)$ queries may be made, but we only need to consider one in which the attacker may succeed. The notation $\exists$ and $\nexists$ indicate whether a specific query was ever made. To capture the successful attack events, we allow the attacker to explicitly execute AUTH.Authentication. Such executions are different from the *Authentication* oracle queries, which lead to executions on behalf of the authenticators.

Definition 6. (properties of stateful authentication methods) *A stateful authentication method should have the following properties:*

AUTH.`correctness`*: For any execution of* AUTH.Authentication *between an honest user u and an honest verifier v, v always accepts.*

AUTH.`no-impersonation`*: An adversary, who does not compromise the cryptographic key of an honest user u, can impersonate u with only a probability negligible in κ. Formally,*

$$\Pr\left[\begin{array}{l}(\sigma, \mathsf{state}_{\mathcal{A}}) \leftarrow \mathcal{A}^{Init(1^\kappa), Authentication(1^\kappa,\cdot,\cdot), HaR(\cdot), Insider(\cdot)}(1^\kappa): \\ \nexists Insider(u) \wedge \nexists HaR(u) \wedge \\ \text{TRUE} \leftarrow \mathsf{AUTH.Authentication}(1^\kappa, \langle \cdot, \mathsf{state}_{\mathcal{A}}\rangle, \mathsf{state}_{v,u})\end{array}\right] = \epsilon(\kappa),$$

AUTH.`attack-evidence`*: Suppose the cryptographic key of an honest user was compromised at time t and the attacker authenticated at least once using the compromised key to the verifier at time $t' > t$. Then this compromise can be detected when the victim user authenticates herself to the verifier the first time at time $t^* > t' > t$.*

Construction. The design rationale behind our construction is given in Appendix C. The construction is based on the afore-discussed "twisted" use of forward-secure signatures, where the signer plays the role of a user in the AUTH scheme. It can be based on any concrete forward-secure signature scheme (e.g., [1, 3, 16]), as long as it satisfies the properties reviewed in Section A. Let δ_T be the allowed maximal time interval before a forced key update, and θ is the allowed number of authentications before a forced key update. Denote by T the system time corresponding to the most recent execution of the *key update algorithm*, by T' the current system time, by α the index of the periods that have elapsed, by β the accumulated number of authentications since system initialization, by γ the the number of new authentications since time T. Selections of these parameters are dependent upon the system policies. The construction is presented below, analysis of which is given in [28].
<u>AUTH.Initialization:</u> A user u, who plays the role of the signer in a forward-secure signature scheme, generates its public and private key pair $(pk_u, sk_{u,0})$. The user u sends pk_u to the verifier v and sets $\mathsf{state}_{u,v} \leftarrow \langle T, \delta_T, \theta, \alpha = 0, \beta = 0, \gamma = 0, pk_u, sk_{u,\alpha}\rangle$, whereas v sets $\mathsf{state}_{v,u} \leftarrow \langle T, \delta_T, \theta, \alpha = 0, \beta = 0, \gamma = 0, pk_u, pk_{u,\alpha}\rangle$ where $pk_{u,\alpha}$ can be derived from pk_u.
<u>AUTH.Authentication:</u> Suppose u holds $\mathsf{state}_{u,v} = \langle T, \delta_T, \theta, \alpha, \beta, \gamma, pk_u, sk_{u,\alpha}\rangle$ and v holds $\mathsf{state}_{v,u} = \langle T, \delta_T, \theta, \alpha, \beta, \gamma, pk_u, pk_{u,\alpha}\rangle$.

- The *key update algorithm* is executed when one of the following three conditions is satisfied: (1) the system is just initialized; (2) the user has conducted θ authentications; (3) $T' - T \geq \delta_T$. In any case, u sets $\alpha \leftarrow \alpha + 1$ and $\gamma \leftarrow 0$, derives $sk_{u,\alpha}$ from $sk_{u,\alpha-1}$, and sets $\mathsf{state}_{u,v} \leftarrow \langle T, \delta_T, \theta, \alpha, \beta, \gamma, pk_u, sk_{u,\alpha}\rangle$; whereas v sets $\alpha \leftarrow \alpha + 1$ and $\gamma \leftarrow 0$, possibly derives $pk_{u,\alpha}$ from pk_u, and sets $\mathsf{state}_{v,u} \leftarrow \langle T, \delta_T, \theta, \alpha, \beta, \gamma, pk_u, pk_{u,\alpha}\rangle$.
- The following authentication protocol is executed whenever one of the following two conditions is satisfied: (1) the key has just been updated and thus a dummy authentication is executed; (2) the user needs to authenticate herself to the verifier. The protocol has the following steps:
  1. User u generates a forward-secure signature σ on the concatenation of T, α, β, γ as well as possibly a (dummy) message using private key $sk_{u,\alpha}$. Then, it sends σ as well as the relevant information to the verifier v.
  2. If σ is valid with respect to $pk_{u,\alpha}$, v accepts and sets $\mathsf{state}_{v,u} \leftarrow \langle T, \delta_T, \theta, \alpha, \beta + 1, \gamma + 1, pk_u, pk_{u,\alpha}\rangle$.
  3. If v accepts, u sets $\mathsf{state}_{u,v} \leftarrow \langle T', \delta_T, \theta, \alpha, \beta + 1, \gamma + 1, pk_u, sk_{u,\alpha}\rangle$.

6 Putting the Pieces Together to Instantiate DSAS

Having explored the building-blocks, now we present our DSAS main construction, which is an integration of the above **Constructions I** and **II**. Its security and extensions are described in [28].
<u>DSAS.Initialization</u>: Given a primary security parameter κ, the anchor SA generates a pair of public and private keys $(pk_{\mathsf{SA}}, sk_{\mathsf{SA}})$ for signing receipts and possibly the roots of the Level 1 hash trees. A user u initiates its own cryptosystem

$(\mathsf{pk}_u, \mathsf{sk}_u)$ for generating digital signatures that need be deposited. Moreover, the following two procedures are executed. (i) Execute AUTH.Initialization to initialize a stateful authentication method (as in **Construction II**). Especially, $(pk_u, sk_{u,0})$ is generated. (ii) Execute ABB.Initialization to initialize ABB_0.
DSAS.Deposit: A user u executes AUTH.Authentication to authenticate herself to the anchor using $sk_{u,i}$ (as in **Construction II**) on either a dummy message M' or a signature sig with respect to pk_u, where sig is to be deposited. The anchor SA verifies the validity of the request as in AUTH.Authentication using $pk_{u,i}$, and in the case of depositing a digital signature, the validity of sig using pk_u. The anchor may return a receipt signed with sk_{SA} (e.g., its signature on the message that "this signature, sig, will appear in ABB_i at time t_i"). The receipt may be forwarded by the signer to the signature verifier.
DSAS.Update: The anchor SA executes ABB.Update, and may send back to the users the "attestations" of their newly deposited signatures. An attestation includes (1) the time t_i at which ABB_i is published, and (2) the siblings of all the nodes on the path from the node that is being attested to the root $R(t_i)$.
DSAS.Retrieve: This is the same as ABB.Retrieve.

7 Dealing with the Unexpected

Recall that we assumed that the hash functions are collision resistant, the digital signature schemes and the forward-secure digital signature schemes are secure with respect to the respective well-accepted definitions. What if some or even all of these assumptions are broken by a powerful attacker? Note that our model already accommodated that the private signing keys may be compromised by whatever means, which subsumes that the private keys are cryptanalyzed, which in turn breaks the security of the digital signature schemes. Moreover, if the private signing keys are compromised by whatever means, it would be possible that the forward-secure signing keys are compromised. Since the forward-secure signing scheme is employed to provide another layer of protection, it would be without loss of generality to focus on the situation where the hash functions may be broken [27] and the private signing keys may be compromised. For example, the very recent incident — digital signatures based on MD5 hash function allow the attacker to obtain a rogue CA certificate [25] — can be adequately dealt with using our solution by depositing the certificate signatures.

Given such a powerful attacker, it is possible that the attacker can present faked signatures that can be verified with respect to the ABB. As we now discuss, there are a range of methods for alleviating the damage of such an attacker.

We start with the scenario that the hash function h may be broken (i.e., it turns out not to be collision-resistant). To deal with this, we can append each node (both leaf and internal) of the ABB tree with a value computed using a "newly-available", supposed-to-be-more-secure hash function, denoted by h. For example, in the case of Figure 3(c), the leaf node annotated with $h(pk_4)$ now becomes a pair $(h(pk_4), \mathsf{h}(h(pk_4)))$, and the leaf node annotated with $h(sig_1)$ now becomes a pair $(h(sig_1), \mathsf{h}(h(sig_1)))$. Then, the internal node annotated

as $a = h(h(pk_4), h(sig_1))$ now becomes a pair $(a, \mathsf{h}(\mathsf{h}(h(pk_4)), \mathsf{h}(h(sig_1)))))$. The same procedure is applied throughout the tree in a bottom-up fashion. Note that it should be clear that we cannot simply replace, for example, the root $R(t_2)$ with $(R(t_2), \mathsf{h}(R(t_2)))$. Note also that the above method was inspired by Haber [11], who deals a similar problem but in a simpler situation. This way, compromising hash function h and all the digital signing keys — except the anchor's private key sk_{SA} for signing the root of ABB— *after* the employment of h does not allow the attacker to breach security of DSAS. In what follows we deal with the two exceptions: (1) the anchor's private signing key sk_{SA} may be compromised; (2) no such h is available.

Q1: What if the signing key sk_{SA} of the anchor is compromised? Recall that the anchor may use sk_{SA} to issue deposit receipts and/or sign the root of the ABB trees. In the case sk_{SA} is compromised, the attacker could abuse it to impersonate the anchor to issue cryptographically-legitimate receipts. However, such an attack can be detected when the anchor updates the ABB because the signature verifiers cannot validate the receipts, which are forwarded by the signer to the verifiers. In response to such an emergence, the anchor needs to identify which signatures are truly deposited at its ABB via the DSAS service, and which signatures are not. For this purpose, we can let the anchor commit another pair of public and private keys, say $(pk'_{\mathsf{SA}}, sk'_{\mathsf{SA}})$, when pk_{SA} is first published or certified (by a higher-level CA). To further enhance security, the commitment scheme could be "information-theoretically hiding and computationally binding" such that even a computationally unbounded attacker cannot figure out pk'_{SA} before the anchor decommits it, except for a negligible probability. Moreover, the cryptosystem corresponds to pk'_{SA} may be different from the cryptosystem corresponds to pk_{SA} (e.g., "discrete logarithm"-based vs. factorization-based) and may use a larger security parameter. Then, the anchor could use sk'_{SA} to sign the receipts of the signatures that were deposited at the anchor itself, where the receipts were previously signed using sk_{SA}. Of course, if sk_{SA} is compromised by the attacker who breaks into the anchor's computer or device, it is natural that sk'_{SA} is stored on a device different from the one that stores sk_{SA}, which is always a prudent practice anyway.

Note that the above method of introducing a new pair of public and private keys $(pk'_{\mathsf{SA}}, sk'_{\mathsf{SA}})$ can be extended to introduce a set of such cryptosystems, which exhibit increasing strength of security (e.g., using increasingly larger security parameters). Note also that the above method has the consequence that we must put a stronger trust, than in the basic scheme, in the anchor because the anchor has the potential to dispute signatures it endorsed before (e.g., when the anchor realized the risk of endorsing certain signatures may be too high at a later point in time). Fortunately, this may be tolerable because the anchor has a short period of time (i.e., between two updates of ABB) to decide whether to cheat or not. Thus, there is still a "grey period" as indicated in the model discussed in Section 2, which is however short.

Q2: What if there is no hash function such as h that is available, or such h is available only after the attacker compromises the cryptographic signing

keys (either by cryptanalysis of by breaking into the anchor's computer or device) as well as h? This scenario is similar to the case that the private key of the anchor, namely sk_{SA}, may be compromised. Thus, we can adopt a similar countermeasure, namely by including an "information-theoretically hiding and computationally-binding" commitment of public key pk'_u in the certificate of the public key pk_u. This way, when sk_u is compromised, which can be detected after at most a single period of time (i.e., between two updates of ABB), user u can use sk'_u to certify the signatures generated using sk_u in the past periods of time. Of course, we must assume that sk'_u is stored at a secure place different from the place where sk_u was stored, at least in the case that sk_u could be compromised by breaking into u's computer or device (rather than cryptanalysis). Note that this mechanism can alleviate the problem when the signing algorithms of the users used some hash functions that may be later broken — a scenario not accommodate in the afore-discussed h being broken later. Note also that user u could abuse this method to dispute some signatures she issued before, but arguably within the last period of time (i.e., between two updates of ABB). That is, there is still a "grey period" as indicated in the model discussed in Section 2, but it is short.

8 Conclusion and Future Work

We presented a model for understanding trustworthiness and longevity of digital signatures in the presence of compromised private signing keys/functions, or malicious signers. The model offers hints for designing solutions to alleviate the problem of grey periods, during which there are signatures for which we are not certain who should be held accountable. The hints guided us to design a solution to deal with the inevitable threats. We also showed how to extend our solution to deal with the unexpected threats that all of the deployed cryptosystems are broken.

Our investigation inspires several interesting open problems. First, how can we defeat the hit-and-stick attack? Second, is it possible to eliminate grey periods? Third, the compromise-detection mechanism we investigated is passive. How can we design a proactive one (e.g., is it possible to exploit some HoneyKeys — the cryptographic analogy of techniques known as Honeynet — to help detect compromises of computers)? Fourth, how should we deal with the case of some "non-traditional" use of digital signatures. For example, abuse-free contract signing [9] is a kind of signatures useful in contract signing. It is not clear how can we adapt the present solution to accommodate them *without* jeopardizing the abuse-freeness property to some extent.

Acknowledgement. We thank the anonymous reviewers for their useful comments.

Shouhuai Xu was supported in part by AFOSR, NSF, and UTSA.

References

1. R. Anderson. On the forward security of digital signatures. Technical report, 1997.
2. M. Bellare, D. Micciancio, and B. Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In *Eurocrypt'03*.
3. M. Bellare and S. Miner. A forward-secure digital signature scheme. In *Crypto'99*.
4. M. Bellare and B. Yee. Forward-security in private-key cryptography. In *CT-RSA'03*.
5. D. Chaum and E. Van Heyst. Group signatures. In *Eurocrypt'91*.
6. Y. Desmedt and Y. Frankel. Threshold cryptosystems. In *Crypto'89*.
7. Y. Dodis, J. Katz, S. Xu, and M. Yung. Strong key-insulated signature schemes. In *PKC'03*.
8. C. Ellison and B. Schneier. Ten risks of pki: What youre not being told about public key infrastructure. *Computer Security Journal*, XVI(1), 2000.
9. J. Garay, M. Jakobsson, and P. MacKenzie. Abuse-free optimistic contract signing. In *Crypto'99*.
10. S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Computing*, 17(2):281–308, April 1988.
11. S. Haber. Long-lived digital integrity using short-lived hash functions. Cryptology ePrint Archive, Report 2007/238, 2007. `http://eprint.iacr.org/`.
12. S. Haber and W. Stornetta. How to time-stamp a digital document. In *Crypto'90*.
13. K. Harrison and S. Xu. Protecting cryptographic keys from memory disclosures. In *IEEE DSN'07*.
14. A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk, and M. Yung. Proactive public key and signature schemes. In *ACM CCS'97*.
15. G. Itkis. Cryptographic tamper evidence. In *ACM CCS03*.
16. G. Itkis and L. Reyzin. Forward-secure signatures with optimal signing and verifying. In *Crypto'01*.
17. G. Itkis and L. Reyzin. Sibir: Signer-base intrusion-resilient signatures. In *Crypto'02*.
18. M. Just and P. van Oorschot. Addressing the problem of undetected signature key compromise. In *NDSS'99*.
19. P. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Crypto'96*.
20. L. Lamport. Password authentication with insecure communication. *Comm. of ACM*, 24(11):770–771, November 1981.
21. P. Loscocco, S. Smalley, P. Muckelbauer, R. Taylor, S. Turner, and J. Farrell. The inevitability of failure: The flawed assumption of security in modern computing environments. In *NISSC'98*.
22. R. Merkle. *Secrecy, Authentication, and Public Key Systems*. PhD thesis, June 1979.
23. R. Ostrovsky and M. Yung. How to withstand mobile virus attacks (extended abstract). In *PODC'91*.
24. A. Shamir and N. van Someren. Playing 'hide and seek' with stored keys. In *FC'99*.
25. A. Sotirov, M. Stevens, J. Appelbaum, A. Lenstra, D. Molnar, D. Osvik, and B. de Weger. Md5 considered harmful today: Creating a rogue ca certificate. `http://www.win.tue.nl/hashclash/rogue-ca/`, 2009.
26. TCG. `https://www.trustedcomputinggroup.org/`.
27. X. Wang, Y. Yin, and H. Yu. Finding collisions in the full sha-1. In *Crypto'05*.

28. S. Xu and M. Yung. Expecting the Unexpected: Towards Robust Credential Infrastructure. Full version of the present paper available at `www.cs.utsa.edu/~shxu`.
29. B. Yee. *Using secure coprocessors.* PhD thesis, Carnegie Mellon University.

A Cryptographic Preliminaries

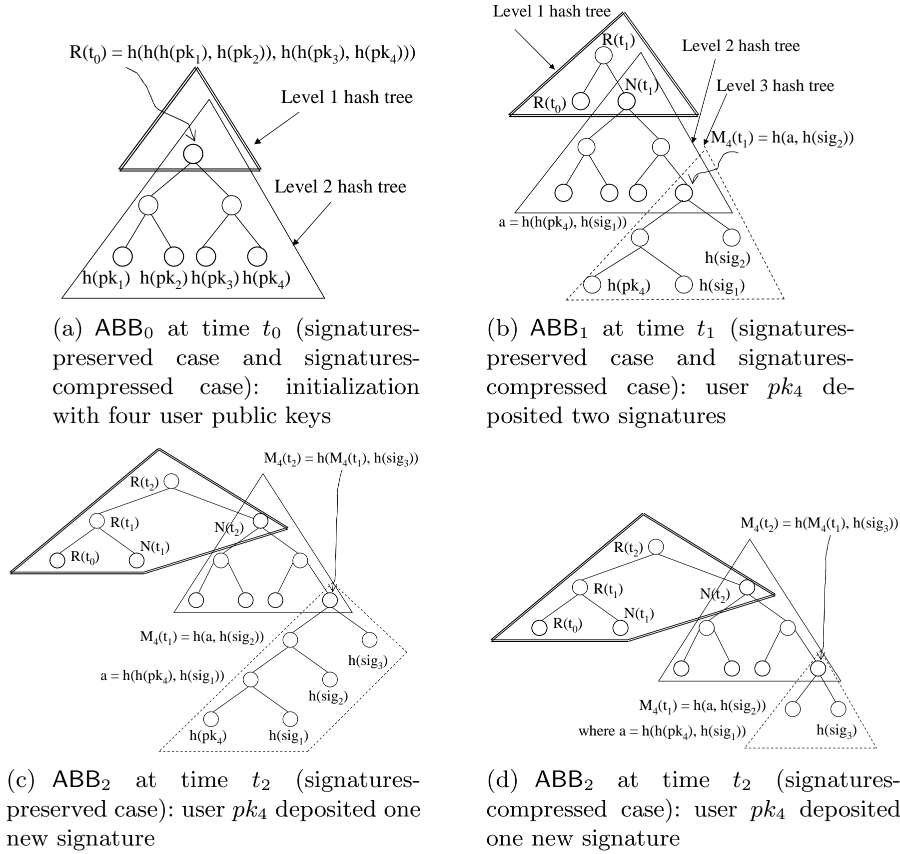
Let κ be a security parameter. We often prove the security of a cryptographic scheme by showing that the probability an adversary breaks the scheme, $\epsilon(\kappa)$, is negligible. A function $\epsilon : \mathbb{N} \rightarrow \mathbb{R}^+$ is negligible if for any c there exists κ_c such that $\forall \kappa > \kappa_c$ we have $\epsilon(\kappa) < 1/\kappa^c$. We say a family of hash functions $h : \{0,1\}^k \times \{0,1\}^* \rightarrow \{0,1\}^\kappa$ is collision-resistant if, for any $K \in \{0,1\}^k$, the probability for any probabilistic polynomial-time (PPT) algorithm to find x_1 and x_2, such that $x_1 \neq x_2$ but $h_K(x_1) = h_K(x_2)$, is negligible in κ. Given that $h_K(\cdot)$ is determined once K is chosen, we will write it as $h(\cdot)$ for short. We will use Merkle hash trees [22], in which the value of an internal node is the hash of its children's values.

Digital signatures. A signature scheme consists of: a *key generation algorithm* that takes as input a security parameter κ and outputs a pair of public and private keys (pk, sk); a *signing algorithm* that takes as input a message m and a private key sk, and outputs a signature σ; a *verification algorithm* that takes as input a message m, a public key pk and a candidate signature σ, and decides whether to accept the signature. Security of digital signatures is traditionally captured by the existential `unforgeability` under adaptive chosen-message attack, meaning that the probability for an attacker, who may have access to many message-signature pairs, to generate a new signature is negligible in κ [10].

Forward-secure signatures. In a forward-secure signature scheme [1, 3], the system time is divided into periods (e.g., days) such that the private key of a signer is changed periodically (i.e., daily), but the public key remains unchanged. Such a scheme consists of: a *key generation algorithm* for generating a pair of public and private keys (pk, sk_0); a *key update algorithm* for the signer to periodically update its period private key as sk_i at the beginning of the ith period, and perhaps also for a verifier to derive the corresponding period public key pk_i (from pk) at the beginning of the ith period; a *signing algorithm* for the signer to sign messages using the period private key sk_i; and a *verification algorithm* for a verifier to check the validity of a signature using the period public key pk_i. Basically, the `forward-security` property means that an adversary, who may have compromised period private key sk_i (possibly $i = \infty$), can generate a valid signature with respect to any pk_j with only a negligible probability, where $j < i$. The intuition is that compromise of a current private key does not allow the adversary to compromise any past private key.

B An Illustrative Example

Figure 3 shows some illustrative snapshots of an ABB. The trees framed by



(a) ABB_0 at time t_0 (signatures-preserved case and signatures-compressed case): initialization with four user public keys

(b) ABB_1 at time t_1 (signatures-preserved case and signatures-compressed case): user pk_4 deposited two signatures

(c) ABB_2 at time t_2 (signatures-preserved case): user pk_4 deposited one new signature

(d) ABB_2 at time t_2 (signatures-compressed case): user pk_4 deposited one new signature

Fig. 3. An illustration of an example ABB snapshots

double solid lines are the Level 1 hash trees, the trees framed by single solid lines are the Level 2 hash trees, and the trees framed by single dashed lines are the Level 3 hash trees.

Suppose there are four users (or groups of users in the case of depositing anonymous signatures). Figure 3(a) depicts ABB_0, which was initialized at time t_0. Specifically, the leaves of the Level 2 hash tree correspond to the four public keys. The root value is $R(t_0) = N(t_0) = h(h(h(pk_1), h(pk_2)), h(h(pk_3), h(pk_4)))$, where h is a collision-resistant hash function. Note that the Level 1 hash tree consists of a single node, namely the root of the Level 2 hash tree. Moreover, each leaf of the Level 2 hash tree can be seen as the root of the corresponding Level 3 hash tree, which consists of a single node though.

Suppose during the time interval between t_0 and t_1 the owner of pk_4 deposited two signatures, sig_1 and sig_2. ABB_1 is depicted in Figure 3(b). Note that the signatures-preserved case and the signatures-compressed case are the

same, because there are no signatures to compress at this point. Note that the root of the updated Level 3 hash tree corresponding to pk_4 is $M_4(t_1) = h(h(h(pk_4), h(sig_1)), h(sig_2))$, the root of the updated Level 2 hash tree is $N(t_1)$, and the root of the updated Level 1 hash tree is $R(t_1) = h(R(t_0), N(t_1))$. The attestation for signature sig_1 is $(t_1; h(pk_4), h(sig_2), \ldots, R(t_0), R(t_1))$, and so on.

Suppose during the time interval between t_1 and t_2, the owner of pk_4 deposited one signature, sig_3. There are two cases:

- Figure 3(c) depicts ABB_2 in the signatures-preserved case. The root of the updated Level 3 hash tree becomes $M_4(t_2) = h(M_4(t_1), sig_3)$, the root of the updated Level 2 hash tree becomes $N(t_2)$, and the root of the updated Level 1 hash tree becomes $R(t_2) = h(R(t_1), N(t_2))$. Moreover, the updated Level 3 hash tree has all previously deposited signatures, as well as $h(pk_4)$, as its leaves.
- Figure 3(d) depicts ABB_2 in the signatures-compressed case. The updated Level 1 and 2 hash trees are the same as in the signatures-preserved case, but the updated Level 3 hash tree does not have all previously deposited signatures as its leaves. Indeed, the new root $M_4(t_2)$ only has the "compression" of the signatures, denoted by $M_4(t_1)$, as its left child.

C Design Rationale

As mentioned before, our framework aims at detecting the compromise of a private signing key as soon as possible. This can be fulfilled via a stateful authentication method coupled with a "twisted" use of forward-secure signatures; this is done in a fashion independent of the digital signatures that are being deposited. The twist is due to the following. First, a user updates her private key with respect to the adopted forward-secure signature scheme either after signing a pre-determined number of messages, or after a pre-determined period of time (this is particularly relevant when a user does not issue digital signatures often). Second, whenever a user updates her private key in the adopted forward-secure signature scheme, the user should use the updated private key to authenticate herself to the anchor for a dummy message (this can be automatically done by the user's software for a better deployment convenience). The design can be justified by answering the following two questions.

Q1: Why forward-secure signatures, but not others? We examined other seemingly plausible designs, which however do not fulfill the desired assurance. First, we notice that a *symmetric* key authentication system, traditional message authentication scheme and forward-secure message authentication scheme [4] alike, does not fulfill the desired assurance. This is because the anchor is only semi-trusted, and thus can leak an honest signer's symmetric authentication key to an attacker without being held accountable. If the attacker compromises an honest signer's private key, which is used to generate digital signatures that need be deposited, the attacker can generate valid signatures with respect to some past time. These signatures can make (some of) the honest signer's past signatures questionable, because the signed messages may be contradictory to each other.

Perhaps more importantly, a dishonest signer can plausibly repudiate some previously deposited signatures by claiming that they were generated by an attacker, which causes a longer grey period $[T_1, T_4]$ because the virtual interval $[T_1, T_2]$ becomes longer. For a similar reason, it does not work to let a signer and the anchor maintain a common state information such as an incremental counter, or the time at which the last signature was deposited. Second, the above vulnerability suggests to adopt an *asymmetric* design. A concrete example is to let a user set up a one-way hash chain (cf. Lamport [20]) such that the user selects s_0 and sends $s_\ell = H^\ell(s_0)$ to the anchor. Then the ith deposit request is associated with $s_{\ell-i} = H^{\ell-i}(s_0)$, where H is a member of a one-way hash function family. However, this design still has the afore-mentioned vulnerability that can cause a longer grey period $[T_1, T_4]$. This is because when the user is compromised, s_0 is compromised and thus the attacker can derive any s_i (even without colluding with the anchor).

By utilizing forward-secure signatures, the above vulnerabilities are dismissed and $[T_1, T_2]$ is reduced, even if T_2 is not known to the defender. It is possible to replace forward-secure signatures with, for example, signatures corresponding to independent period public keys. However, this would require the users to frequently generate fresh public and private key pairs.

Q2: Why the twisted, but not the standard, use of forward-secure signatures for authentication? A standard use of forward-secure signatures, while providing the desired "asymmetry," has the following vulnerability. Suppose a signer is honest at system initialization time T_0, but becomes dishonest at time T_2. Suppose $T^* > T_2$ is the time at which the private key with respect to the adopted forward-secure signature scheme should be updated, but the now dishonest user does not follow the protocol. When the compromise becomes evident at time T_4, the dishonest signer can blame all the signatures generated during the time interval $[T_2, T_4]$ to the attacker who has compromised its private key. That is, the standard use of forward-secure signatures does not provide any means to deal with such a malicious behavior. The problem is caused by the fact that the periodical key update operations are done at the signer end and at the anchor end in an independent fashion. Our "twist" alleviates this problem, by forcing a signer to authenticate herself to the anchor whenever there is a private key update. Moreover, the signer is forced to update her period private key whenever (1) she has authenticated a pre-determined number of times since the last key update, or (2) a pre-determined length of time has elapsed since the last key update. This means that the resulting periods are not necessarily of the same length, but the longest time interval between two authentications conducted by a signer is upper bounded by a pre-determined parameter. Putting this into the context of the above example, the signer is thus forced to authenticate herself to the verifier at time T^*, where $T_2 < T^* < T_4$. This leads to a shorter grey period $[T^*, T_4]$.