Generalized Key-Evolving Signature Schemes or How to Foil an Armed Adversary*

Gene Itkis and Peng Xie

Boston University Computer Science Dept.
111 Cummington St.
Boston, MA 02215, USA
{itkis, xp}@cs.bu.edu

Abstract. Key exposures, known or inconspicuous, are a real security threat. Recovery mechanisms from such exposures are required. For digital signatures such a recovery should ideally —and when possible— include invalidation of the signatures issued with the compromised keys. We present new signature schemes with such recovery capabilities.
We consider two models for key exposures: full and partial reveal. In the first, a key exposure reveals *all* the secrets currently existing in the system. This model is suitable for the pessimistic inconspicuous exposures scenario. The partial reveal model permits the signer to conceal some information under exposure: e.g., under coercive exposures the signer is able to reveal a "fake" secret key.
We propose a definition of *generalized key-evolving signature scheme*, which unifies forward-security and security against the coercive and inconspicuous key exposures (previously considered separately [5, 18, 11]). The new models help us address repudiation problems inherent in the monotone signatures [18], and achieve performance improvements.

Keywords: *digital signatures, forward-security, monotone signatures, key-evolving signature schemes, key exposures, coercion, recovery.*

1 Introduction

Secret key exposures are a well-known threat for cryptographic tools. Such exposures may be inconspicuous (e.g., undetected theft) or obvious (e.g., extortion). In either case, recovery from such compromises presents an challenge which must be addressed. Typically, such recovery is achieved by revocation of the compromised keys and re-issue of the new version of the keys to the attacked user. Both of these operations are very expensive and complicated. Alternative recovery methods are thus highly desirable.

Coercive key exposures have been considered previously; in particular, [18] proposed a mechanism for invalidating all the signatures generated with the extorted keys, but not any of the signatures issued by the legitimate signer both

* Published in *The First MiAn International Conference on Applied Cryptography and Network Security*, 2003. This version adds explanations and fixes some typos. This research is supported in part by NSF grant CCR-0311485.

before or after the coercion attack (thus eliminating the need for revocation and re-issue).[1] The detection of inconspicuous key exposures was addressed previously in [11], which also suggested an alternative to revocation. A potential connection to the monotone signatures was noted in that paper as well; here we develop this connection further.

In this paper we consider both the coercive and inconspicuous key exposures. First, we focus on the coercive exposures: we discuss the original definition of the monotone signatures from [18], and highlight some of its shortcomings and potential pitfalls. Then we propose a new definition, which avoids these pitfalls. Our definition generalizes the monotone signatures of [18] on the one hand, and the key-evolving signatures [5][2] on the other, and as such, it may be of independent value. Our definition also allows us to address both the coercive/overt and inconspicuous exposures within one model. We then propose some schemes, which not only provide an improved functionality, but are also more efficient than the original monotone signatures. Finally, we prove some (tight) lower-bounds.

1.1 Monotone signatures — Evolution of the Public Keys

Let us briefly review the monotone signatures as they were originally defined in [18]. The reader is encouraged to consult the original paper for the formal definitions.

The monotone signatures are motivated by a scenario when the signer is forced to reveal a key which would enable the attacker to generate valid signatures. How can the system security be recovered after the extortion attack is over?

PUBLIC KEY EVOLUTION. It appears that whatever recovery mechanism is used, the public key must change. It therefore makes sense to consider explicitly a model with the evolving public keys. This is exactly what the monotone signatures attempt (without presenting it in exactly these terms).

MONOTONE SIGNATURES. The original definition of the monotone signatures allowed the verification algorithm (i.e., the public key) to be updated, e.g., after an attack. Namely, the signer under duress can reveal some (but not all!) of his secrets to the attacker. These extorted secrets enable the attacker to generate signatures valid under the *current* public key. However, after the signer is no longer under duress, the public key can be updated, so that all the signatures generated by the legitimate signer (both, before and after the update or the

[1] Alternative methods of creation of a "subliminal channel" allowing the coerced signer to covertly inform authorities of the coercion were also proposed (see, e.g., Funkspiel schemes of [10]). In this case, the key may not be actually exposed and one may hope that the notification of the authorities for the extorted signatures eliminates the need of the expensive global recovery.

[2] Key-evolving signature schemes were introduced as a functional definition for forward-secure signatures [3, 5, 16, 2, 13, 17, 12, 15]. They also served as a basis for new notions such as key-insulated [6], intrusion-resilient [14, 12], and tamper-evident [11] signatures.

attack) remain valid, but all the signatures generated by the attacker (using the extorted keys) are no longer valid under the *updated* verification algorithm. Thus, monotone signatures can be viewed as a generalization of the signature schemes which includes the revocation and re-issue.

SHORTCOMINGS OF THE ORIGINAL DEFINITION. Unfortunately, the definition of [18] gives the signer too much power: it undermines the non-repudiation property of the signatures — often their main functionality. Indeed, a signer can choose to sign any message "as an attacker" in the above scenario. Then, at any time later on, he may choose to update his public key so as to invalidate that signature.

1.2 Generalized key-evolving signatures schemes and extortion-resilience

In this paper, we modify the definition of the monotone signature schemes to limit this repudiation power of the signer. We also generalize the definition to include all the above mentioned schemes as its special cases. In particular we refine the secret key exposure model to include both undetected inconspicuous exposures as well as the extortions.

CLEARING PERIOD: RESTRICTING REPUDIATION. In particular, in our scenarios, similarly to the real life check payments, each signature — even if verified as valid — is not "cleared" for a pre-defined period of time. We limit the repudiation ability of the signers to that clearing period.

Similarly to the forward-secure signatures, in our system the legitimate signers obeying the protocol do not have the information that could allow them to back-date their signatures. Thus, an attacker can use the extorted secrets only to generate signatures dated after the extortion.

We use techniques similar to tamper-evidence [11] to invalidate the signatures generated by the extorted keys, with respect to the public keys updated by the signer after he is released.

FORCING SIGNER COMMUNICATION AND SUBLIMINAL CHANNELS. Given the above, the attacker may try to prevent a public key update altogether or at least to make sure that the updated public key does not invalidate the signatures based on the extorted secrets. To achieve this the attacker may resort to holding the signer hostage until the extorted signatures are cleared, or even killing the signer after the extortion. We need to eliminate the motivation for the attacker to resort to such measures.

To achieve this we require the signer to contact the authority at least once during the clearing period and perform the public key update. Thus, even if the attacker holds the signer captive during the clearing period, the attacker still needs to allow the signer to contact the authority, in order to have the signatures with the extorted keys cleared. However, this communication between the signer and authority, unlike the communication with the untrusted verifiers, can utilize subliminal channels (e.g., similar to the Funkspiel [10]). Thus at the very least the authority would know that the particular keys had been extorted. And even if these keys must be treated as valid in order to protect the signer, the authority can take whatever steps are available. And in particular, it may be able to reset the system afterwards so that the extorted keys become invalid even after the clearing period has passed. They would also, of course, be able to identify the extorted signatures potentially helping with arresting the attacker.

Thus, the attacker is faced with the choice of having her extorted signatures invalidated before they are cleared, or risking authority notification and subsequent tracing. It is our hope that since both of the options involve significant risks for the attacker, our scheme provides a significant deterrent for an attack.

2 Definitions

2.1 Functional definitions

GENERAL KEY-EVOLVING SIGNATURE SCHEMES (*GKS*). Intuitively, in *GKS* scheme, both the secret key and public key can evolve. The signer can invalidate extorted signature by updating the public key. Secret key evolution enable forward-security, which in turn enables the clearing period functionality.

We start with the functional definition of *GKS*: A *general key-evolving signature scheme GKS*=(**KeyGen**,**SUpd**,**PUpd**,**Sign**,**Ver**) is a collection of five (randomized) polynomial algorithms, where:

KeyGen is the *key generation* algorithm,
  Input: a security parameter $k \in \mathsf{N}$ (given in unary), and the total number of periods, T,
  Output: a pair $(\mathtt{SK}_0, \mathtt{PK}_0)$, the initial secret key and public key;
SUpd is the *secret key update* algorithm,
  Input: the secret key $\mathtt{SK}_t$ for the current period $t < T$, and control message $\mathtt{c}$, which determines the update type: $\mathtt{c} \in \{\mathtt{sk\_only}, \mathtt{sk\&pk}, \mathtt{pk\_only}\}$, corresponding to updating only the secret key, both secret and public, and public key only[3], respectively,
  Output: the new secret key $\mathtt{SK}_{t+1}$ and update message μ_t;
PUpd is the *public key update* algorithm,
  Input: the current public key $\mathtt{PK}_t$ and the update message μ_t,[4]
  Output: The new public key PK_{t+1};
Sign is the *signing* algorithm,
  Input: the secret key $\mathtt{SK}_t$ for the current period t and the message M to be signed,
  Output: signature sig_t of M (the time period t of the signature generation is included in sig_t);

[3] The `pk_only` option is included here for generality. In this paper we will not support this option: it is convenient to record the number of public key updates in the secret key; moreover, an update message for the public key update must be generated, and this typically requires information from the secret key. E.g., for the monotone signature schemes of [18] our **SUpd** algorithm's main function is to generate the update message.

[4] The update type `c` here can be inferred from the update message μ, if desired.

Ver is the *verification* algorithm,
Input: the public key $\mathtt{PK}_t$, a message M, and an alleged signature sig,
Output: **valid** or **fail**.

Intuitively, we count time as the number of key updates, see Experiment Forge in subsection 2.2 for the more precise definition. The key update frequency is selected by the user: it could correspond to physical time intervals (e.g., one update per day), or performed arbitrarily (e.g., more frequently when the signer feels vulnerable), or activity related (e.g., one update per each signature).

For simplicity we assume full synchronization of the system: namely, we assume that there are no outdated public keys. In particular, a signature generated at time t should never be verified using a public key $\mathtt{PK}_i$ from an earlier period $i < t$.

Completeness, Monotonicity, Non-Repudiation, Clearing Period. We require the *completeness* property (as in [18]). Namely, all signatures generated by a legitimate signer must be valid for *all* the subsequent legitimate public keys: $\mathbf{Ver}(\mathtt{PK}_i, M, \mathbf{Sign}(\mathtt{SK}_t, M)) = \mathbf{valid}$ for any message M and any time periods $i \geq t$. In particular, validity of a legitimately generated signature should not be changed by updates.

We also require *monotonicity* of the signatures: an invalid signature cannot become valid at a later time. Formally: $\mathbf{Ver}(\mathtt{PK}_{t'}, M, sig_t) = \mathbf{valid} \Rightarrow \mathbf{Ver}(\mathtt{PK}_j, M, sig_t) = \mathbf{valid}$ for all $j : t \leq j \leq t'$. [5]

The monotone signatures, by their very design, are intended to allow the signer to output alleged secret keys (and thus signatures) which look perfectly legitimate at the time they are output, but can be invalidated at a later time by an appropriate public key update. If unrestricted, this power builds in a repudiation mechanism contradicting the very design of the signatures.

As mentioned in the Introduction, we limit this repudiation power to the *clearing period* δ: if the signature remains valid after $\geq \delta$ public key updates, then it can never be repudiated by any subsequent updates. Namely, suppose that signature sig_i was generated at time period i, and let j be a time period at least δ public key updates after i; then we require that $\mathbf{Ver}(\mathtt{PK}_j, M, sig_i) = \mathbf{valid} \Rightarrow \mathbf{Ver}(\mathtt{PK}_{j'}, M, sig_i) = \mathbf{valid}$ for all $j' > j$, and thus by monotonicity for all $j' \geq i$.

Forward-Secure and Monotone Signatures. The above general key-evolving scheme definition includes as special cases the functional definitions for forward-secure and monotone signatures: forward-secure signatures [5] update only the secret keys, while the monotone signatures [18] update only the public keys (**SUpd** is used only to update the time period and generate the update message).

Key Exposures. In order to formalize key exposures we introduce a special function:

---

[5] This notion of monotonicity is slightly different from that of [18]: we do not require a signature to be valid for the "outdated" public keys preceding the signature — in fact, we rule out such a verification altogether.

Reveal the (possibly randomized) *secret key revealing* algorithm;
Input: the current secret key $\mathtt{SK}_t$, and number r of the previous known attacks (i.e., the number of times **Reveal** was used previously with the signer's knowledge);[6]
Output: alleged secret key $\mathtt{SK}'_t$: $\mathbf{Ver}(\mathtt{PK}_t, M, \mathbf{Sign}(\mathtt{SK}'_t, M)) = \mathbf{valid}$ for all messages M.

Intuitively, **Reveal** outputs key $\mathtt{SK}'$ given to the attacker when she tries to expose the signer's key $\mathtt{SK}$. For all exposure models below, $\mathtt{SK}$ and $\mathtt{SK}'$ should be indistinguishable at the exposure time. In particular, $\mathtt{SK}'$ should generate signatures valid for the current public key. But after the subsequent public key updates, $\mathtt{SK}$ and $\mathtt{SK}'$ are easily distinguished: $\mathtt{SK}'$ will now generate invalid signatures.

Three models of exposures could be considered: *full, partial* and *creative*. The first model corresponds to the attacker learning *all* the secrets of the signer, $\mathbf{Reveal}(\mathtt{SK}, r) = \mathtt{SK}$.

Partial reveal allows the signer to conceal some of his secrets from the attacker: i.e., the attacker obtains a subset of all the secrets of the signer. It is important to note that the set of the exposed secrets in this model is determined by the signer, and not by the attacker. This is the model used in [18].

Finally, the creative reveal model appears to give the signer the greatest defensive powers: the signer is allowed to "lie creatively" and generate the alleged secret key $\mathtt{SK}'$ in an arbitrary way. However, all the alleged secret keys $\mathtt{SK}'$ can be pre-computed and stored in the $\mathtt{SK}$ to be revealed at the appropriate times. Thus, the creative reveal is actually equivalent to the partial reveal model. So, in the rest of the paper we consider only the full and partial reveal models.

For the sake of simplicity, in this version of the paper, we consider partial and full models separately. However, a hybrid model allowing a combination of the reveal attacks is of interest as well. In particular, the full reveal is well-suited for the inconspicuous attacks, while the partial model can address the extortion attacks. Since in real life both types of attacks are possible, it would be useful to have schemes (and model) to handle both at optimal costs.

2.2 Security definitions

GKS SECURITY. In *GKS*, time is divided into time periods. The time is incremented by each key update. A key update can either update only the secret key (`sk_only`), or both secret and public keys (`sk&pk`). As noted in the footnote 3, while theoretically it is possible to allow a "public key only" update, we do not support it in this paper. We use $P(t)$ to denote the number of the public key updates (i.e., `sk&pk`'s) that occurred since the key generation and up to the current time period t.

[6] The number of r of previous attacks is needed only for partial and creative reveal models discussed below. For the full reveal model, this number can be ignored or even unknown.

We allow the adversary F to adoptively obtain signatures and secret keys (the later using **Reveal** function). We model these two types of information access with two oracles: *Osig* and *Orvl*. The most natural assumption is to allow only the queries relating to the current time t. We expand the adversary's powers to allow her to query signatures for the past (but not the future, since the future signatures may depend on the specific future evolution path the system takes).

Specifically, given message M and time period $i \leq t$, oracle $Osig_t(M, i)$ queried at time t returns the signature that the signer would have generated for the message M during the period i.

Similarly, the adversary can reveal the secret keys. In the *full-reveal* model, she can reveal both present and past, but not the future keys. The ability to fully-reveal past keys — which are supposed to have been erased by the time of the query/exposure — can model inability to securely erase keys.

In the case of the *partial-reveal* model, the number of past partial reveals may affect each future partial reveal (as, for example is the case for [18]). Therefore, adversary can partial-reveal only the current key. The number of partial reveals up to the present period is maintained by the signer. Thus, the signer is assumed to be aware of each partial reveal (again, in contrast to full reveals, which can be inconspicuous) — a natural assumption, if a partial reveal models, e.g., an extortion attack.

Namely, for $i \leq t$ the oracle $Orvl_t^{(\rho)}(i)$ returns $\mathbf{Reveal}^{(\rho)}(\mathtt{SK}_i, r)$, where r is the number of the previous partial reveal attacks, and $\rho \in \{\mathtt{pr}, \mathtt{fr}\}$ is the reveal type: partial or full. If $\rho = \mathtt{fr}$ then r can be omitted. However, if $\rho = \mathtt{pr}$ then the oracle enforces an additional restriction: $i = t$; in this case we can omit i in the oracle query, since it is redundant.

We use the following experiments to define the security of *GKS*.

$\mathtt{Experiment}$ $Forge_{GKS}(F, k, T, \delta)$
$t \leftarrow 0$; $\mu_t \leftarrow$ empty string;
$(\mathtt{SK}_t, \mathtt{PK}_t) \leftarrow \mathbf{KeyGen}(1^k, T)$
$\mathtt{repeat}$
  $q \leftarrow F^{Osig_t, Orvl_t}(\mathtt{PK}_t)$
  $\mathtt{if}$ $(q = \mathtt{sk\_only})$ $\mathtt{then}$ $\mathtt{SK}_{t+1} \leftarrow \mathbf{SUpd}(\mathtt{SK}_t, \mathtt{sk\_only})$
  $\mathtt{if}$ $(q = \mathtt{sk\&pk})$ $\mathtt{then}$
    $(\mathtt{SK}_{t+1}, \mu_t) \leftarrow \mathbf{SUpd}(\mathtt{SK}_t, \mathtt{sk\&pk})$
    $\mathtt{PK}_{t+1} \leftarrow \mathbf{PUpd}(\mathtt{PK}_t, \mu_t)$
  $t \leftarrow t + 1$
$\mathtt{until}$ $(q = forge)$
$(M, \sigma_i, j \geq i) \leftarrow F$
$\mathtt{if}$ $\mathbf{Ver}(M, \sigma_i, \mathtt{PK}_j) = \mathbf{valid}$, $i \leq j \leq T$, $\mathtt{and}$ neither $Osig(M, i)$ nor $Orvl^{(\mathtt{fr})}(i)$ were queried, $\mathtt{and}$ either $P(j) \geq P(i) + \delta$ or $Orvl^{(*)}(i')$ was not queried for any $i' \leq i : P(j) < P(i') + \delta$
  $\mathtt{then}$ $\mathtt{return}$ 1.

Intuitively, the last `if` statements disqualifies some of the forgeries from being considered as adversary's success. Let us trace these cases when the forger F's output is not considered successful.

First, most obvious case is when the signature is not valid. Also, we do not consider her successful if F outputs a signature she did not generate (i.e., obtained from the signer, via the *Osig* oracle).

Next, clearly, F can sign any message at time i, if F knows the actual secret key $\mathtt{SK}_i$ (obtained by the full exposure at i, modeled by $Orvl^{(\mathtt{fr})}(i)$ query). Since such a signature is equal to the signature the signer would have generated for the same message at the same time, such signatures cannot possibly be invalidated.

Finally, we consider how much time elapsed from the signature generation (i) to its verification (j). If this time is more than the clearing period, then the forged signature should be invalidated even if the forger *partially* exposed $\mathtt{SK}_i$. However, if the clearing period has not elapsed, then any — even partial — revealing of any secret key at any time $i' \leq i$ and within the same clearing period, could enable the forger to generate signatures that would appear valid. Indeed, this is directly required for the extortion scenario: Suppose that the signer is captured by the adversary at time i' and is forced to (partially) reveal the secret at that time. Then the adversary can test the validity of the revealed key by generating and verifying a signature at time $i \geq i'$. Thus, it is desirable that these signatures appear valid, and only as the clearing period elapses the signatures generated with the partially revealed keys should become invalidated. (Recall, that the clearing period is defined in terms of public key updates. Holding the signer hostage for longer than clearing period can be complicated for the adversary by requiring the public key update to involve procedural steps that facilitate tracing of the adversary by non-cryptographic means and/or create a subliminal channel between the signer and the central authority.)

Now, we can define the *GKS*-scheme security in terms of the above experiment: *GKS*-scheme is secure if any ppt adversary can succeed with at most negligible probability. More formally, let $\epsilon(k)$ be a positive function, e.g., $\epsilon(k) = 1/2^k$. We define $\epsilon(k)$-security as follows:

Definition 1 **(*GKS* security).** *Let* $\mathcal{S} = (\mathbf{KeyGen}, \mathbf{Sign}, \mathbf{SUpd}, \mathbf{PUpd}, \mathbf{Ver})$ *be a GKS-scheme with parameters* k, T, δ *as above. Say that* $\mathcal{S}$ *is* $\epsilon(k)$-secure *if* $\mathsf{Prob}[Forge_{\mathcal{S}}(F, k, T, \delta) = 1] < \epsilon(k)$ *for any PPT adversary* F.

Consider an attacker who broke in during time period t. Obviously, she can now generate valid signatures for that period at least until the public key update. Moreover, in the case of the full reveal, she can generate signatures for the time period t which will remain valid even after the public key updates. So, intuitively what *GKS* security requires is that these be the only forgeries that the attacker can generate successfully. In other words, *GKS* security confines the damage to the period of the exposure. In particular, the attacker still cannot forge valid signatures for any of the time periods before or after t. In the case of partial reveal, the attacker cannot even generate signatures for period t which will remain valid after the subsequent public key updates.

CMA, Forward-Security and Monotone Signatures Security. If all key updates and *Orvl* oracle are excluded in the experiment *Forge* (i.e., $q = forge$ is immediately selected by adversary using only the *Osig* oracle), then the above definition is equivalent to the standard definition of security against *adaptive chosen message attacks* [8].

If no public key updates are performed (i.e., only `sk_only` updates are allowed), then the above definition converges to that of *forward-security* [5].

The above definition also captures the *monotone signatures* of [18] as mentioned above: Allow only `sk&pk` updates and restrict the secret key update to change only the time period — thus, main purpose of **SUpd** is to produce the update message μ (assume $\delta = 1$ for the above definition, but the non-repudiation after the clearing period is not provided by [18]). Moreover, for monotone signatures, $\mathtt{SK} = \langle s_1, \ldots, s_T \rangle$ and $\mathbf{Reveal}(\mathtt{SK}, i) = \langle s_1, \ldots, s_i \rangle$.

SOUNDNESS. For monotone signatures described as above, the soundness defined by [18] means that without $Orvl(\geq i)$ any adversary has only a negligible probability of successfully forging a signature for a public key $\mathtt{PK}_i$. Our definition includes this notion of soundness and further refines it to allow `sk_only` updates and full reveal.

In principle, it may be interesting to consider *GKS* schemes with $\delta > 1$ or even variable (e.g. depending on time). Specifically, such schemes might offer better performance. But for the sake of simplicity, in the rest of this paper we assume $\delta = 1$.

3 Using *GKS* signatures

This section contains informal discussion about how the signer may wish to use the *GKS* signatures, a sort of short "user's manual" draft.

As discussed in the introduction, we consider two types of exposures: known (e.g., extortion) and inconspicuous (undetected theft). Clearly, a full reveal exposure, whether known or undetected, leads to a compromise of all the signatures issued during the period of the exposure — it is impossible to distinguish signatures generated with the same secret key. Dealing with these compromised signatures is one obvious challenge. Another challenge is presented by the requirement that if a signature remains valid for a certain amount of time (the clearing period), then it can never become invalid. This later requirement, in particular, implies that it must be impossible to "back-date" signatures, even after the exposures. We address this clearing period requirement by including the forward-security in our definitions and constructions. Using the forward-security, we can approach the first challenge also (though, forward-security by itself may not be enough to address it completely): in the extreme case, the signer may wish to perform an update (which includes erasing of the previous secret key) immediately before and immediately after each signature. Then the exposed secrets are the least likely to compromise any legitimate signatures.

While it may be too expensive to follow the above tactics all the time, it is recommended to increase frequency of updates whenever the signer feels there may be an increased chance of a key exposure. In particular, it is recommended to perform an update whenever the user enters a more risk-prone situation, when an attack is more likely. Namely, when facing danger, the signer would trigger the **SUpd** algorithm to erase the secret key and generate the new SK, which may be revealed to the attacker. The attacker can use the revealed secrets to generate forged signatures. Such signatures must be valid under the current public key. However, as a matter of policy, they might not be honored until the clearing period has passed. The same policy should require that passing of the clearing period requires public key update to be performed subsequently to the signature in question. However, all the signatures generated using extorted secrets would have to be invalidated with the subsequent public key updates.

In general, it is important to consider explicitly all the parties involved: *signer*, *verifiers*, *attacker*, and *authority*. Intuitively, the last party—authority—is responsible for communicating with the signer and performing the public key updates in the verifiers. It is the responsibility of the authority to ensure that there are no spurious public key updates even when the signer's keys are exposed. As expected, the signer generates the signatures and the verifiers verify the signatures. The signer also maintains the secret key by performing secret key update **SUpd** at the end of each time period, which generates update message μ. The signer then communicates μ to the authority, which performs the public key update **PUpd** and distributes the new public keys to the verifiers. We assume that the verifiers leak the public keys to the attacker.

The introduction of the authority in order to receive the update message from the signer allows us to "cheat" by going outside the scheme. In particular, it allows the use additional (unexposed) secrets, used only for communication between the signer and the authority. This secret cannot be "tested": for example, the signer may have two passwords, one of which would be used only in the case of an attack to communicate to the authority that the signer is being held by an attack and his secret is extorted — in this case, the authority may emulate the normal update behavior, but would notify the police or take other appropriate measures. More sophisticated subliminal channels (e.g. [10]) could be deployed for this purpose to allow the signer to communicate more information to the authority.

4 Constructions

4.1 Construction for Full Reveal Model

GKS AND TAMPER-EVIDENT SIGNATURES. One approach to construct a *GKS* scheme could be based on tamper-evident signatures [11]. Tamper-evident signatures are key-evolving signature schemes, using only secret key evolution. Their main feature is inclusion of a special predicate Div: given two signatures, Div determines whether only one of them is generated by the legitimate user (and the other by the forger), provided that both signatures were generated after the latest (inconspicuous) key exposure. Periodically including a recent legitimately generated tamper-evident signature into the public key can achieve most of the *GKS* security. The verification procedure would then include the Div test between the latest such signature and the signature being verified. The scheme below can be viewed as an optimization of the above approach. Perhaps the most surprising aspect of this scheme is that, despite its simple approach, it turns out to be optimal (as shown in Sec. 5). In the extreme case (for maximum security), the above approaches essentially boils down to the equivalent of keeping the log of signatures in the public key. The fact that this approach yields an optimal scheme (after some optimizations described below) can be seen as another interpretation of our results.

GKS CONCATENATION SCHEME. We propose a *GKS* scheme based on an arbitrary ordinary signature scheme and a forward-secure signature scheme. The idea behind this construction is quite simple: For each time period, a different ordinary secret-public key pair is used; these public keys are published in the *GKS* public key. These ordinary public keys enable verification of the *GKS* signatures generated before the latest public key update. In order to validate the signatures generated after the latest *GKS* public key update, we certify the ordinary public keys not yet included in the *GKS* public key. We use the forward-secure signatures for this certification. For each *GKS* public key update, the forward-secure signature scheme generates a new public key and corresponding initial secret key. The forward-secure public key is included in the published *GKS* public key. For each subsequent *GKS* secret key update, the forward-secure signature scheme updates its secret key. After the next *GKS* public key update, this certification key can be discarded, since the public keys it certified are now included directly in the *GKS* public key. The forward-security of certification is needed to assure that the forgeries predating the key exposures would not verify as valid at any times, even before the clearing period elapses. Next we give the formal description of the scheme.

Let Σ be any ordinary signature scheme and Ψ any forward-secure signature scheme. We use instances S_t of Σ for message signing and instances $\mathtt{C}$ of Ψ for certification: S_t are used during time periods t, and $\mathtt{C}$ is used to certify $S_t.\mathtt{PK}$ for the current time periods. $S_t.\mathtt{SK}$ and $S_t.\mathtt{PK}$ denote the secret and public keys of S_t, generated by the $\Sigma.\mathbf{KeyGen}$ algorithm. We write $S_t.\mathbf{Sign}(m)$ to denote $\Sigma.\mathbf{Sign}(S_t.\mathtt{SK}, m)$ (similarly, $S_t.\mathbf{Ver}(m, sig) \stackrel{def}{=} \Sigma.\mathbf{Ver}(S_i.\mathtt{PK}, m, sig)$). Let $\vec{\mathtt{PK}}_{[i,j]} \stackrel{def}{=} S_i.\mathtt{PK}||\ldots||S_j.\mathtt{PK}$; $\vec{\mathtt{PK}}_j \stackrel{def}{=} \vec{\mathtt{PK}}_{[1,j]}$; $\vec{\mathtt{PK}}_0 \stackrel{def}{=} \oslash$ (i.e., the *empty string*). Intuitively, μ' below is a "current draft" update message, stored in $\mathcal{S}.\mathtt{SK}$.

Since the previous certification keys are discarded at each public key update, we do not index the certification scheme instances — only the current one has any relevance. However, for each certification public key, the corresponding secret key evolves as for all the forward-secure signatures. For convenience of notation, we denote the key pair generated by $\Psi.\mathbf{KeyGen}$ for certifying the signing keys starting with $S_t.\mathtt{PK}$ as $\mathtt{C.SK}_t, \mathtt{C.PK}$ (the normal notation would be $\mathtt{C.SK}_0, \mathtt{C.PK}$). We write $\mathtt{C}_t(m)$ to denote $\Psi.\mathbf{Sign}(\mathtt{C.SK}_t, m)$.

Now, we specify a *GKS* scheme $\mathcal{S} = (\mathbf{KeyGen}, \mathbf{Sign}, \mathbf{SUpd}, \mathbf{PUpd}, \mathbf{Ver})$.

$\mathcal{S}.\mathbf{KeyGen}$:
$(\mathtt{C.SK}_0, \mathtt{C.PK}) \leftarrow \Psi.\mathbf{KeyGen}(1^k)$;

$\texttt{return}\ (\mathcal{S}.\texttt{PK}_0 \stackrel{def}{=} \langle \vec{\texttt{PK}}_0, \texttt{C.PK}\rangle,$

$\qquad\qquad \mathcal{S}.\texttt{SK}_0 \stackrel{def}{=} \langle -1, \texttt{C.SK}_0, \mu'_{-1} = \emptyset, S_{-1}.\texttt{SK} = \emptyset, \texttt{CPK} = \emptyset\rangle)$

$\mathcal{S}.\mathbf{SUpd}(\mathcal{S}.\texttt{SK}_t, \texttt{c})$:

Let $\mathcal{S}.\texttt{SK}_t = \langle t, \texttt{C.SK}_{t+1}, \mu'_t, S_t.\texttt{SK}, \texttt{CPK}\rangle$;

Securely delete $S_t.\texttt{SK}$;

$(S_{t+1}.\texttt{SK}, S_{t+1}.\texttt{PK}) \leftarrow \Sigma.\mathbf{KeyGen}(1^k)$;

$\texttt{if}\ (\texttt{c} = \texttt{sk\&pk})\ \texttt{then}$

$\quad (\texttt{C.SK}_{t+1}, \texttt{C.PK}) \leftarrow \Psi.\mathbf{KeyGen}(1^k)$;

$\quad \mu_t \leftarrow \langle \mu'_t, \texttt{C.PK}\rangle;\ \mu'_{t+1} \leftarrow \emptyset$

$\texttt{CPK} \leftarrow \langle S_{t+1}.\texttt{PK}, \texttt{C}_{t+1}.\mathbf{Sign}(S_{t+1}.\texttt{PK})\rangle$;

$\mu'_{t+1} \leftarrow \mu'_t || S_{t+1}.\texttt{PK}$

$\texttt{C.SK}_{t+2} \leftarrow \Psi.\mathbf{SUpd}(\texttt{C.SK}_{t+1})$;

$\mathcal{S}.\texttt{SK}_{t+1} = \langle t+1, \texttt{C.SK}_{t+2}, \mu'_{t+1}, S_{t+1}.\texttt{SK}, \texttt{CPK}\rangle$;

$\texttt{if}\ (\texttt{c} = \texttt{sk\_only})\ \texttt{then return}\ (\mathcal{S}.\texttt{SK}_{t+1})$;

$\texttt{elseif}\ (\texttt{c} = \texttt{sk\&pk})\ \texttt{then return}\ (\mathcal{S}.\texttt{SK}_{t+1}, \mu_t)$;

$\mathcal{S}.\mathbf{PUpd}(\mathcal{S}.\texttt{PK}_t, \mu_t)$:

Let $\mathcal{S}.\texttt{PK}_t = \langle \vec{\texttt{PK}}_i, \texttt{C.PK}\rangle;\ \ \mu_t = \langle \vec{\texttt{PK}}_{[i+1,t]}, \texttt{C}'.\texttt{PK}\rangle$

$\texttt{return}\ (\ \mathcal{S}.\texttt{PK}_{t+1} \stackrel{def}{=} \langle \vec{\texttt{PK}}_t, \texttt{C}'.\texttt{PK}\rangle\ )\qquad \%\ \vec{\texttt{PK}}_t = \vec{\texttt{PK}}_i || \vec{\texttt{PK}}_{[i+1,t]}$

$\mathcal{S}.\mathbf{Sign}(\mathcal{S}.\texttt{SK}_t, m)$:

Let $\mathcal{S}.\texttt{SK}_t = \langle t, \texttt{C.SK}_{t+1}, \mu'_t, S_t.\texttt{SK}, \texttt{CPK}\rangle$;

$\texttt{return}\ (\ sig = \langle t, \texttt{CPK}, S_t.\mathbf{Sign}(m)\rangle\ )$

$\mathcal{S}.\mathbf{Ver}(\mathcal{S}.\texttt{PK}_t, m, sig)$:

Let $\mathcal{S}.\texttt{PK}_t = \langle \vec{\texttt{PK}}_j, \texttt{C.PK}\rangle;\ sig = \langle i, \texttt{CPK} = \langle S.\texttt{PK}, \langle i', cs\rangle\rangle, s\rangle$

$\texttt{if}\ i \le j\ \texttt{then return}\ (\ S_i.\mathbf{Ver}(m)\ )\quad \%\ S_i.\texttt{PK}\ is\ in\ \vec{\texttt{PK}}_j$

$\texttt{else return}\ (\ i = i' \bigwedge \Psi.\mathbf{Ver}(\texttt{C.PK}, S.\texttt{PK}, \langle i', cs\rangle) \bigwedge \Sigma.\mathbf{Ver}(S.\texttt{PK}, m, s)\ )$

PERFORMANCE. We use $|PK_\Sigma|$ (resp. $|PK_\Psi|$) to denote the size of the public key for the Σ (resp. Ψ) scheme, $|s_\Sigma|$ (resp. $|s_\Psi|$) for the size of the signature under the Σ (resp. Ψ) scheme (wlog, assume that it does not depend on the message signed), l_t for the size of the representation of the time period t, and $|SK_\Sigma|$ (resp. $|SK_\Psi|$) is the size of the secret key for Σ (resp. Ψ). Let t be an arbitrary time period and j be the first time period which has no public key updates between itself and t: $j = \min\{i : P(i) = P(t)\}$. Then our GKS scheme above has the following performance characteristics:

The size of public key at time t is $|\mathcal{S}.\texttt{PK}_t| = j \cdot |PK_\Sigma| + |\texttt{PK}_\Psi|$.

A signature sig generated at time t has length $|sig| = |s_\Sigma| + |s_\Psi| + |PK_\Sigma| + l_t$ (only the last term depends on t).

The size of the secret key at time t is $|\mathcal{S}.\texttt{PK}_t| = l_t + |SK_\Sigma| + |SK_\Psi| + |s_\Psi| + (t - j + 1) \cdot |PK_\Sigma|$.

The time complexities of all the $\mathcal{S}$ functions are independent of t: $\mathcal{S}.\mathbf{KeyGen}$ requires only a single $\Psi.\mathbf{KeyGen}$. $\mathcal{S}.\mathbf{Sign}$ requires only a single $\Sigma.\mathbf{Sign}$. And $\mathcal{S}.\mathbf{Ver}$ needs at most one $\Sigma.\mathbf{Ver}$ and one $\Psi.\mathbf{Ver}$ computations. $\mathcal{S}.\mathbf{SUpd}$ requires one of each $\Sigma.\mathbf{KeyGen}$, $\Psi.\mathbf{SUpd}$, $\Psi.\mathbf{Sign}$ and some trivial data (list) manipulations, plus possibly one $\Psi.\mathbf{KeyGen}$. $\mathcal{S}.\mathbf{PUpd}$ has O(1) time complexity — independent of Σ and Ψ complexities, as well as of t.

4.2 Concatenation Scheme Security in the Full Reveal Model

Without essential loss of generality, let the clearing period δ below be $\delta = 1$.

Theorem 1. *Let Σ and Ψ be an ordinary and forward-secure signature schemes, respectively; let the probability of forgery be $\leq \frac{\epsilon(k)}{T}$ for each. Then the GKS concatenation scheme $\mathcal{S}$ (Sec. 4.1) is $\epsilon(k)$-secure:*

$$\forall \; ppt \; F' \; \mathsf{Prob}[Forge_{\mathcal{S}}(F', k, T, \delta) = 1] \leq \epsilon(k)$$

Proof sketch: Suppose for some forger F', $\mathsf{Prob}[Forge_{\mathcal{S}}(F', k, T, \delta) = 1] > \epsilon(k)$. Then we construct a *GKS* adversary F, which uses F' as an oracle and with probability $> \epsilon(k)/T$ breaks either Σ or Ψ.

Let $\hat{S}$ and $\hat{C}$ be instances of Σ and Ψ, respectively, given to F. Then, F selects t uniformly from $\{1, \ldots, T\}$ and sets $S_t = \hat{S}$; the other instances of S_i in the experiment *Forge* are generated randomly.

For Ψ, F does similarly, but since the number of public key updates is unknown, F does the following: Whenever, public key update is requested by F' in the *Forge*, F sets $C = \hat{C}$ with probability $1/T$, and or generates C randomly otherwise. Also, after $\hat{C}$ is used once, all subsequent instances of C are generated randomly.

Then, intuitively, the experiment *Forge* returns 1 if F' forges either a Σ- or a Ψ- signature. With probability $1/T$, it will be the forgery for $\hat{S}$ or $\hat{C}$. □

4.3 Partial Reveal Schemes

A SIMPLE TRADE-OFF. The schemes of [18] all use the partial reveal model. Still, they have both public keys and signatures linear in the maximum number of public key updates T_P.

In our full reveal scheme, the signature size is reduced to constant while the public key size is linear in the current time period.[7]

Furthermore, the size of the signatures in the schemes of [18] gives some indication to the attacker of what T_P might be equal to. Thus the attacker may force the signer to reveal all or most of the secrets in one attack. In contrast our concatenation scheme has no T_P and thus can be continued indefinitely, and the signer has no hidden secrets for the attacker to extort — all the secrets to be used in the future are generated as needed.

We note that the schemes of [18] can be directly optimized to reduce the public key size to constant, while keeping the signature size linear in T_P; the

[7] To be fair, we must note that in our scheme the number of updates may be larger than in the monotone signatures: since we update the secret key every time there is a threat of exposure. On the other hand, this also allows us to deal with full-reveal inconspicuous exposures, which the monotone signatures are unable to protect against. If inconspicuous exposures are not a threat, then the `sk_only` updates can be implemented using a forward-secure signature scheme Σ, reducing the size of the public key to (nearly) match the monotone signatures.

verification would also be reduced to constant — one ordinary signature verification. Indeed, intuitively each monotone signature is verified with respect to all the published public key components. But for security, it is sufficient to verify the signatures only against the last public key component published. Thus the previously published public keys can be deleted. Of course, for this optimized scheme it is possible to generate signatures that could be repudiated at any time later on. This scheme also allows the signer to violate monotonicity: generate signatures which are invalid at the time of generation, but valid at the later times.

In fact, combining similar optimization with our concatenation scheme, it is possible to achieve a linear trade-off between the signature and public key sizes. However, such trade-off is only of theoretical value, since it is very unlikely that the savings of the public key length might justify the proportional increase of the signature length.

Interval Crossing Scheme (*IX*).

Intuition. This section presents a *GKS* scheme for the partial reveal model exponentially improving the asymptotic performance of the above schemes, as well as the schemes of [18]. We call it *interval crossing scheme* or *IX* for short.

IX is based on the forward-secure scheme of [13]. It also uses two security parameters: k and l. Intuitively, k is the length of the modulus used for GQ-like signatures [9]; l is the length of the outputs of the random oracle used for the scheme.

Let T be an upper-bound on the number of all the key updates. The scheme of [13] divides the interval $[2^l, 2^{l+1}]$ into T subintervals $I_t = [2^l + (t-1)2^l/T,\ 2^l + t2^l/T)$ — one per each time period t. A prime exponent $e_t \in I_t$ is used in the GQ-like the signatures for the time period t. *IX* further subdivides each I_t into C intervals, for the parameter C of the given *IX* scheme. It then offers an option to specify the sub-interval of e_t with greater precision: The *IX* public key may include index $1 \le i_t \le C$ for each time period t; if the exponent e_t used in a signature for time t is not from the i_t-th subinterval of I_t, then the signature is invalidated. For convenience we identify the index i_t with the i_t-th subinterval of I_t.

Intuitively, C corresponds to the upper-bound on the number of ways the signer may try to cheat the attacker. While C is a parameter of the scheme, and is thus assumed to be known to the attacker, the actual number of the spare versions of e_t prepared for each interval by the signer is unknown. Indeed, as will become apparent from below, it may be wise to select both T and C to be fairly large — the cost to the signer is only logarithmic in C (recording the index i_t in the public key) and independent of T. In fact these two parameters may be set for some fixed large values, same for all users (e.g., $T = 10^{10},\ C = 2^{10}$).

Thus, if no extortion attacks took place in period t, then no index i_t for that period needs to be published in the *IX* public key. However, if the signer is attacked during the time t then he can reveal some secrets for one or more $e'_t \in I_t$, leaving at least one $e_t \in I_t$ not revealed. Upon release, the signer can update the public key, publishing i_t; all signatures for time t using $e'_t \notin i_t$ are then disqualified. Clearly, if the indistinguishability of the "fake" and "real" secrets in this case can be obtained only if the attacker does not have any legitimate signatures for the same period t. This can be achieved, for example, by the signer performing an update (thus incrementing t) as soon as he comes under the attack.

This mechanism is essentially similar to "black-listing" the extorted keys (more accurately, "white-listing" a non-extorted key) when needed. Applied directly, this method would not offer significant improvement over the concatenations scheme. However we achieve an exponential improvement by using a compact representation of the "white-list".

Compact set representation. Consider the following problem stated here informally: We need to generate a set S of t_{max} elements so that for any $t \leq t_{max}$, the subset $S_t \subseteq S$ of t elements can be represented compactly and in such a way that no information about $S - S_t$ is revealed. In particular, no information about the size of S is leaked, nor is it possible to deduce whether a given $x \notin S_t$ is actually in S.

We propose the following, slightly simplified approach: Let most of the elements of the sets S_t be leaves of a small (logarithmic in t) number of trees (e.g., at most one of each height). Each tree can be represented by its height and the value at its root. All the tree leaves are computed from the root as in the pseudo-random function (PRF) tree of [7]. A small —bounded by constant— number of elements of S_t are not leaves in such trees, but are listed individually.

We use this approach as follows: as the i_t values are published in the IX public key, they are initially listed individually. As more of them are collected, they are aggregated into the trees. At any moment of time, the set of the revealed i_t's gives no indication of whether it has reached t_{max}, the maximum number of periods for which the signer had prepared the secret key. Moreover, the latest revealed "fake" secrets cannot be distinguished from the "real" ones.

IX-Scheme: Formal Description. For the sake of simplicity in describing the algorithms, we assume that $C = 2$. Thus, each i_t can be specified with a single bit (in addition to specifying t). Let $H : \{0,1\}^* \rightarrow \{0,1\}^l$ be a random function, $\Delta \stackrel{def}{=} 2^l/(TC)$, and t_{max} be the maximum number of the time periods for the give scheme instance. Let P be a length-doubling PRG, $P : \{0,1\}^L \rightarrow \{0,1\}^{2L}$, wlog assume $L = l$. We use P to generate PRF trees. Consider a balanced PRF tree with T leaves, and let R_t be the smallest set of the tree nodes such that the set of the maximal subtrees rooted in R_t contains exactly the leftmost t leaves of the tree, and contains no trees with exactly two leaves and at least one tree of size one. Let the j-th leftmost leaf determine i_j. Say $(e, j) \in R_t$ whenever $e \in i_j$ as determined by R_t.

KeyGen (k, l, T)

Generate random $(\lceil \frac{k}{2} \rceil - 1)$-bit primes q_1, q_2 s.t. $p_i = 2q_i + 1$ are both primes.

$n \leftarrow p_1 p_2$

For a PRF tree of depth $\lceil \lg T \rceil$, generate $R_{t_{max}}$.
For each $t : 1 \le t \le t_{max}$, generate primes e_t, e'_t such that $e_t \in i_t$ and $e'_t \in I_t - i_t$.
(let $f_i \stackrel{def}{=} e_i e'_i \ldots e_{t_{max}} e'_{t_{max}}$, $F'_i \stackrel{def}{=} f_{i+1} \cdot e'_i$, $F_i \stackrel{def}{=} f_{i+1} \cdot e_i$, and $\vec{e}_{[i,j]} \stackrel{def}{=} \langle e_i, e'_i, \ldots, e_j, e'_j \rangle$)
$b_1 \stackrel{R}{\leftarrow} Z_n^*$
$v \leftarrow 1/b_1^{f_1} \bmod n$
$s_1 \leftarrow b_1^{F'_1} \bmod n$
$s'_1 \leftarrow b_1^{F_1} \bmod n$
$b_2 \leftarrow b_1^{e_1 e'_1} \bmod n$
$\mathtt{SK}_1 \leftarrow \langle R_{t_{max}}, 1, t_{max}, n, s_1, e_1, b_2, \vec{e}_{[2,t_{max}]} \rangle$ % *real secret*
$\mathtt{SK}'_1 \leftarrow \langle \neg R_1, 1, 1, n, s'_1, e'_1, v, \vec{e}_{[2,1]} = \oslash \rangle$ % *fake secret, to be revealed in key exposure*
$\mathtt{PK}_1 \leftarrow (n, v, \oslash)$
return $(\mathtt{SK}_1, \mathtt{SK}'_1, \mathtt{PK}_1)$ % *while formally* $\mathtt{SK}'$*, should be part of* $\mathtt{SK}$ *we keep them separate for clarity; the signer disposes of* $\mathtt{SK}'$ *before issuing the first signature of the period*

SUpd (SK_j)
Let $\mathtt{SK}_j = \langle R_{t_{max}}, j, t_{max}, n, s_j, e_j, b_{j+1}, \vec{e}_{[j+1,t_{max}]} \rangle$
if $j = t_{max}$ then **return** $\oslash$
$s_{j+1} \leftarrow b_{j+1}^{F'_{j+1}} \bmod n$;
$s'_{j+1} \leftarrow b_{j+1}^{F_{j+1}} \bmod n$;
$b_{j+2} \leftarrow b_{j+1}^{e_{j+1} e'_{j+1}} \bmod n$
return $\mathtt{SK}_{j+1} = \langle R_{t_{max}}, j+1, t_{max}, n, s_{j+1}, e_{j+1}, b_{j+2}, \vec{e}_{[j+2,t_{max}]} \rangle$ and fake secret key
$\mathtt{SK}'_{j+1} = \langle R_j \cup e'_{j+1}, j+1, n, s'_{j+1}, e'_{j+1}, v, \oslash \rangle$;
$\mu \leftarrow R_j$; % *it is sufficient to include the part of* R_j *only for the periods of extortion*

PUpd $(\mathtt{PK}_j, \mu_j)$
Let $\mathtt{PK}_j = \langle n, v, \ldots \rangle$.
return $\mathtt{PK}_{j+1} = \langle n, v, \mu \rangle$

Sign (SK_j, M)
let $\mathtt{SK}_j = \langle R_{t_{max}}, j, t_{max}, n, s_j, e_j, b_{j+1}, \vec{e}_{[j+1,T]} \rangle$
$r \stackrel{R}{\leftarrow} Z_n^*$
$y \leftarrow r^{e_j}$ mod n
$\sigma \leftarrow H(j, e_j, y, M)$
$z \leftarrow r s_j^{\sigma}$ mod n
return (z, σ, j, e_j)

Ver $(\mathtt{PK}_t, M, (z, \sigma, j, e))$
Let $\mathtt{PK}_t = \langle n, v, R_{t'} \rangle$.
if e is even **or** $e \notin I_j$ **or** $z \equiv 0 \bmod n$ **then return** 0
if $j \le t'$ **and** $(e, j) \notin R_{t'}$ **then return** 0
$y' \leftarrow z^e v^{\sigma}$
if $\sigma = H(j, e, y', M)$ **return** 1 **else return** 0

Reveal

When face dangers, the signer just needs to update his secret key and begins new time period, j. For time period j, the signer reveals the fake secret key $\mathtt{SK}'_j$.

The proof of security of the above scheme is similar to the proof in [13]. Intuitively, if the forger breaks in some time period, the signer can reveal fake secret key for that time period, this fake secret is indistinguishable from the real secret key under the current public key. Furthermore, this fake secret key doesn't help the forger to guess the real secret key by the variant of the strong RSA assumption in [4]. The probability that the forger succeeds in generating valid signature for other time periods is negligible. In the above algorithm, we always use the fixed fake secret. Actually, the signer can reveal several fake secret keys. The attacker have no idea how many secret keys in each interval because the T and s is unknown to the attacker, and the number of secret candidates is probabilistic.

From the construction, we can see that the size of the signature is constant and the size of the public key grows logarithm with time. This is in contrast to the full reveal scheme where each time period added the security parameter number of bits.

5 Lower Bounds: Full Reveal Model

In this section we consider the *GKS* schemes in the full reveal model, and prove the lower bound for the public key length matching that of our scheme in sec. 4.1.

First, a simple probability observation:

Claim. For any events A, B, C, $\mathsf{Prob}[A \bigwedge B|C] = \mathsf{Prob}[A|C] \cdot \mathsf{Prob}[B|A \bigwedge C]$.

Indeed, starting with the right-hand side and using the conditional probability definition we get $\mathsf{Prob}[A|C] \cdot \mathsf{Prob}[B|A \bigwedge C] = \frac{\mathsf{Prob}[A \bigwedge C]}{\mathsf{Prob}[C]} \cdot \frac{\mathsf{Prob}[B \bigwedge A \bigwedge C]}{\mathsf{Prob}[A \bigwedge C]} = \frac{\mathsf{Prob}[A \bigwedge B \bigwedge C]}{\mathsf{Prob}[C]} = \mathsf{Prob}[A \bigwedge B|C]$. □

Now, let $\mathcal{S}$ be an *GKS*-secure scheme and F be a forger. Fix some time period t, immediately after a public key update.[8] For $i \leq t$, let f_i denote an event that an attacker generates a signature valid for $\mathcal{S}.\mathtt{PK}_t$, and some message and time period (m, i) pair (without querying the $Osig(m, i)$ oracle). Let b_i be the event of the attacker break-in at period i (in other words, a query to $Orvl_i$ oracle); assume no other key exposures except those mentioned explicitly. Recall that k is the security parameter and $\mathsf{Prob}[f_i|\neg b_i] \leq \epsilon(k)$ (see definition 1).

[8] The next lemma, and thus the subsequent theorem, rely on the assumption that the clearing period is $\delta = 1$. Adjusting them to arbitrary δ would require that we talk below about the public key $\mathtt{PK}_{t'}$ such that t' is δ public key updates after t: $P(t') \geq P(t) + \delta$. Then the lower bound would be shown for the public key $\mathtt{PK}_{t'}$ instead of $\mathtt{PK}_t$. The rest of the section would remain intact.

Lemma 1. $\mathsf{Prob}[(f_1 \wedge f_2 \wedge \ldots f_t)|b_0] \leq \epsilon(k)^t$, *where k is the security parameter.*

Proof: $\mathsf{Prob}[(f_1 \wedge f_2 \wedge \ldots f_t)|b_0]$
$=\mathsf{Prob}[f_1|b_0] \times \mathsf{Prob}[(f_2 \wedge f_3 \ldots f_t)|(f_1 \wedge b_0)]$ (by the claim above)
$\leq \mathsf{Prob}[f_1|b_0] \times \mathsf{Prob}[(f_2 \wedge f_3 \ldots f_t)|(b_1 \wedge b_0)]$
$=\mathsf{Prob}[f_1|b_0] \times \mathsf{Prob}[f_2|(b_1 \wedge b_0)] \times \mathsf{Prob}[(f_3 \wedge f_4 \ldots f_t)|(f_2 \wedge b_1 \wedge b_0)]$
$\vdots$
$\leq \mathsf{Prob}[f_1|b_0] \times \mathsf{Prob}[f_2|(b_1 \wedge b_0)] \times \mathsf{Prob}[(f_3|b_2 \wedge b_1 \wedge b_0] \ldots \mathsf{Prob}[f_t|(b_{t-1} \wedge b_{t-2} \ldots b_0)]$
$\leq \epsilon(k) \times \epsilon(k) \ldots \epsilon(k) = \epsilon(k)^t$ by the definition of *GKS*. □

For the full reveal *GKS* signatures, we assume that the attacker receives all the secrets of the system at the time period of the break-in. Thus immediately after b_i, the real signer and the attacker F have exactly the same information. The difficulty of $f_{j>i}$ relies on the fact that evolutions of the signer and F are likely to diverge. If somehow F, emulating evolution of the real signer, arrives at the same public key $\mathcal{S}.\mathtt{PK}_j$ as the real signer, then F can generate signatures for all messages and all time periods $i' : i \leq i' \leq j$ and valid for $\mathcal{S}.\mathtt{PK}_j$.

Thus, probability of F emulating real signer evolution and arriving at the same public key $\mathcal{S}.\mathtt{PK}_t$ as the real signer is no greater than $\mathsf{Prob}[(f_1 \wedge f_2 \wedge \ldots f_t)|b_0]$. Which means that this probability of evolving into $\mathcal{S}.\mathtt{PK}_t$ is $\leq \epsilon(k)^t$. But since both signer and F use the same probability distributions, this implies the following theorem:

Theorem 2. *Let $\mathcal{S}$ be a GKS signature scheme secure (with security parameter k) in the full reveal model and let t immediately follow a public key update. Then $|\mathcal{S}.\mathtt{PK}_t| \geq \lg^{\frac{1}{\epsilon(k)}} t$ (e.g., if $\epsilon(k) = \frac{1}{2^k}$, then $|\mathcal{S}.\mathtt{PK}_t| \geq kt$).*

The theorem implies optimality of our scheme in Sec. 4.1 at least with respect to the length of the public keys.

6 Acknowledgments

The authors are grateful to Shai Halevi and Robert McNerney for helpful discussions.

References

1. *Third Conference on Security in Communication Networks (SCN'02)*, Lecture Notes in Computer Science. Springer-Verlag, September 12–13 2002.
2. Michel Abdalla and Leonid Reyzin. A new forward-secure digital signature scheme. In Tatsuaki Okamoto, editor, *Advances in Cryptology—ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 116–129, Kyoto, Japan, 3–7 December 2000. Springer-Verlag. Full version available from the Cryptology ePrint Archive, record 2000/002, `http://eprint.iacr.org/`.
3. Ross Anderson. Invited lecture. Fourth Annual Conference on Computer and Communications Security, ACM (see http://www.ftp.cl.cam.ac.uk/ftp/users/rja14/forwardsecure.pdf), 1997.

4. Niko Barić and Birgit Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In Walter Fumy, editor, *Advances in Cryptology—EUROCRYPT 97*, volume 1233 of *Lecture Notes in Computer Science*, pages 480–494. Springer-Verlag, 11–15 May 1997.
5. Mihir Bellare and Sara Miner. A forward-secure digital signature scheme. In Michael Wiener, editor, *Advances in Cryptology—CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 431–448. Springer-Verlag, 15–19 August 1999. Revised version is available from `http://www.cs.ucsd.edu/~mihir/`.
6. Yevgeniy Dodis, Jonathan Katz, Shouhuai Xu, and Moti Yung. Strong key-insulated signature schemes. Unpublished Manuscript.
7. Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, October 1986.
8. Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, April 1988.
9. Louis Claude Guillou and Jean-Jacques Quisquater. A "paradoxical" indentity-based signature scheme resulting from zero-knowledge. In Shafi Goldwasser, editor, *Advances in Cryptology—CRYPTO '88*, volume 403 of *Lecture Notes in Computer Science*, pages 216–231. Springer-Verlag, 1990, 21–25 August 1988.
10. Johan Håstad, Jakob Jonsson, Ari Juels, and Moti Yung. Funkspiel schemes: an alternative to conventional tamper resistance. In *Proceedings of the 7th ACM conference on Computer and communications security*, pages 125–133. ACM Press, 2000.
11. Gene Itkis. Cryptographic tamper evidence. Submitted. Avaliable from `http://www.cs.bu.edu/ itkis/papers/`, 2002.
12. Gene Itkis. Intrusion-resilient signatures: Generic constructions, or defeating strong adversary with minimal assumptions. In *Third Conference on Security in Communication Networks (SCN'02)* [1].
13. Gene Itkis and Leonid Reyzin. Forward-secure signatures with optimal signing and verifying. In Joe Kilian, editor, *Advances in Cryptology—CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 332–354. Springer-Verlag, 19–23 August 2001.
14. Gene Itkis and Leonid Reyzin. Intrusion-resilient signatures, or towards obsoletion of certificate revocation. In Moti Yung, editor, *Advances in Cryptology—CRYPTO 2002*, Lecture Notes in Computer Science. Springer-Verlag, 18–22 August 2002. Available from `http://eprint.iacr.org/2002/054/`.
15. Anton Kozlov and Leonid Reyzin. Forward-secure signatures with fast key update. In *Third Conference on Security in Communication Networks (SCN'02)* [1].
16. Hugo Krawczyk. Simple forward-secure signatures from any signature scheme. In *Seventh ACM Conference on Computer and Communication Security*. ACM, November 1–4 2000.
17. Tal Malkin, Daniele Micciancio, and Sara Miner. Efficient generic forward-secure signatures with an unbounded number of time periods. In Lars Knudsen, editor, *Advances in Cryptology—EUROCRYPT 2002*, Lecture Notes in Computer Science. Springer-Verlag, 28 April–2 May 2002.
18. David Naccache, David Pointcheval, and Christophe Tymen. Monotone signatures. In P. Syverson, editor, *Financial Cryptography*, volume 2339 of *Lecture Notes in Computer Science*, pages 305–318. Springer-Verlag, 2001.