ProSiBIR: Proactive Signer-Base Intrusion Resilient Signatures

Philip Atzemoglou, Tal Malkin

April 25, 2007

Abstract

The notion of Signer-Base Intrusion-Resilient (SiBIR) signatures was introduced in [IR02] as a scheme that can withstand an arbitrary number of key-exposures, as long as both of its modules are not compromised simultaneously. This was achieved by dividing time into predefined time periods, each corresponding to a different time-evolving secret key, while maintaining a constant public key. The two modules of this scheme consist of a signer that can generate signatures on its own, and a base that is used to update the signer's key as it evolves through time.

The purpose of this paper is to provide a model for multi-signer, multi-base intrusion-resilient signatures. This proactive SiBIR scheme essentially breaks the preexisting notions of signer and base, to an arbitrary number of signer and base modules. This tends to implementations where multiple parties need to agree for a document to be signed. An attacker needs to break into all the signers at the same time in order to forge a signature for that period. Moreover, he needs to break into all the bases as well, at that same time period, in order to "break" the scheme and generate future signatures. Thereby, by assuming a large number of bases, the risk of our scheme being compromised becomes arbitrarily small.

We provide an implementation that's provably secure in the random oracle model, based on the strong RSA assumption. We also yield a modest improvement in the upperbound of our scheme's insecurity function, as opposed to the one presented in [IR02].

Keywords: intrusion resilience, signature schemes, proactive signatures, key evolving signatures

1 Introduction

1.1 Background and Motivation

Digital signatures play an essential role for security on the Internet. Electronic commerce, private and authenticated communication, and secure storage are but a few of the multitude of applications that rely on signatures to assert authenticity, ownership, or delegation. However, signature-based systems are very vulnerable to the *key exposure problem*, which is currently a far more likely cause of compromise than cryptanalysis. Once a private key has been exposed, all signatures (past and future) associated with it are invalidated. The damage of these compromises can be enormous, both in terms of overhead in revoking and reissuing keys and signatures, and in terms of the new security vulnerabilities introduced by the possibility of repudiation. Indeed, it is very easy for a dishonest signer to leak his secret key in order to repudiate a previously signed document. These problems are especially serious given the crucial role that signatures play for applications such as certificate authorities (i.e. Verisign), digital signing of legal contracts, electronic checkbooks, etc.

There have been several approaches towards limiting the effect of key exposure. Techniques such as the use of short lived keys and centralized time-stamping mechanisms do not scale well; the former requires many public keys to be certified, distributed, and maintained, while the latter is interaction heavy and requires a trusted third party who itself must never be compromised in order for security to be maintained. Key revocation mechanisms, while very important, are problematic, their current schemes being very complex, cumbersome, suffering from various disadvantages and presenting some of the most challenging problems with public key infrastructure. Moreover, they do not solve the problem, as they do not prevent forgeability of past signatures and the repudiation problem. Two promising approaches to mitigating the damage caused by key exposure are those of threshold or proactive schemes, and of key evolving schemes (see below). These approaches come with their own advantages and disadvantages, each being appropriate for different applications.

In this paper we present a new model and an efficient protocol for intrusion-resilient proactive signature schemes, thereby combining the advantages of both approaches. Before describing our contributions, however, we start with a brief overview of these two lines of research.

Threshold and Proactive Signatures. This approach uses a distributed multi-party system in order to increase the difficulty of key exposure that allows for signature forgery. Threshold signature schemes [DF89] share the secret key among a number of signers, using a multi-party protocol to generate a signature. In this model, a compromise (such as a key exposure) of up to a certain threshold of the parties does not allow the adversary to forge signatures. Proactive signatures [OY91, HJKY95, HJJ+97] take this one step further by dividing the lifetime of the scheme to time periods. In order for an adversary to be able to forge signatures, he will have to break into more than a threshold of the parties, during the same period. This is done by providing a procedure that rearranges the secret key shares at the beginning of every time period.

Key Evolving Signatures and SiBIR. The basic idea behind this approach is to extend a standard digital signature algorithm with a *key update* algorithm, so that the *secret key* can be changed frequently, while the public key stays the same. The first such notion was that of **forward-secure** signatures [And97, BM99]. [IR01] later provided with a very efficient implementation, followed by [MMM02] with a more generic implementation. Forward security ensures the authenticity of all past signatures, by dividing time into time periods and evolving the secret key along the progression of every period. This means that a key exposure would not enable the adversary to backtrack and reproduce past signatures, although it would yield all the information necessary to forge present and future signatures. In order to overcome this, the model of **Key-Insulated Security** [DKXY03] was proposed. This model uses two separate modules (later named signer and base). The signer generates signatures, while the base helps the signer update the key every time period. As long as the adversary only breaks into one of the modules, security for both past and future signatures is maintained. A compromise of the signer's key alone yields forgeries for the current time period, whereas a compromise of the base's key alone yields nothing. However, if the adversary ever breaks into both modules, even during different time periods, the security of all signatures is compromised.

The key-evolving model with the strongest security guarantee to date is that of **Signer-Base Intrusion Resilience (SiBIR)** [IR02, Itk02], combining the benefits of forward secure, key insulated, and proactive security for two parties. Here, there are again two modules, a signer and a base. The signer module generates signatures as signers are oft to do. The base module is used by the signer in order to update a time-evolving secret key whenever we advance to the next time period, as well as to rearrange the key shares during regular refresh intervals in a manner akin to proactive security (such refresh operations are transparent to the verifier, while the time period is included as part of the signature to be verified). This notion achieves security for both past and future signatures even when both of its modules are compromised, as long as there has been a refresh operation between the compromise of each module. Moreover, even in the case where both modules are compromised simultaneously, the scheme maintains forward-security.

A different extension of the forward-security model was given by [AMN01], who provided a scheme for **Proactive Forward Security**, combining the benefits of forward-secure and proactive signatures. Here, as in the proactive setting, there are multiple parties, and security is maintained as long as the adversary has not compromised more than a threshold of the parties. If the adversary does compromise more than that threshold, however, forward-security is still maintained. Namely, signatures for the current and future time periods are invalidated and their key needs to be revoked, whereas the security of past signatures is still maintained.

1.2 Our Contributions

Model. We introduce the notion of **ProSiBIR**, proactive signer-base intrusion resilient signatures. Our new model uses an n-out-of-n approach to expand on the concept of intrusion resilience, by adding to it the multiparty aspects of proactive security. Instead of requiring a large number of signing parties to maintain security, our framework allows the number of signers to be set according to the needs of the application. For example, signing a credit card bill may require one person, while selling a house that's collectively owned by 7 brothers requires 7 parties. When there is only one signer, she can generate signatures by herself offline. When there are multiple signers, some interaction between them is needed to generate signatures, though the exchanged messages are not sensitive (i.e. do not need to be kept secret). In retrospect, it seems counter intuitive to artificially beef up the number of signers in order to increase security. Instead, our scheme achieves better results by increasing the number of bases. These parties are not needed for signature generation and can be provided by one or more trusted independent organizations, hosted on different software platforms, in different parts of the world. This can protect us from exposures due to vendor-specific vulnerabilities, legal or geopolitical factors.

An adversary would need to break into all the signers, at the same time and refresh period, in order to forge present signatures, while past and future would remain protected. Even if that takes place, however, as soon as we progress to a new time period, that signing key will be of no use to him, as it will have evolved during the update phase. The only way for an adversary to keep forging signatures, short of intercepting all of the update messages for ever that is, would be to compromise the bases' keys. In order to do that, though, he would need to break into all the bases at the same time, while having compromised all the signers during that period. That can be really hard for the adversary; given a sufficiently large number of bases, the risk of our scheme being compromised becomes arbitrarily small. Finally, even if the adversary did manage to procure all the secrets from all the parties involved simultaneously, the scheme would still be forward secure, thereby preserving the security of all past signatures. A syllogism similar to [AMN01] will tell us that such a reduced usefulness of signers' keys and key shares, will be a disincentive and will act as a deterrent for potential adversaries.

Construction. We provide a ProSiBIR scheme, based on the SiBIR scheme of [IR02], which in turn is based on the Guillou-Quisquater signature scheme [GQ88]. Our scheme is as efficient as the ordinary signatures of [GQ88]; every signing party needs to calculate just two modular exponentiations and is required to send $2\ell - 2$ non-sensitive messages (where ℓ is the number of signers) to the other signers so that they form a product that's needed for the signature. Similarly, verifying only requires two modular exponentiations.

Our scheme proves to be secure in the random oracle model, based on the strong RSA assumption. In addition to that, through our proof of security, we achieve a somewhat tighter security bound than that of [IR02] (in fact, we observe that the [IR02] analysis can be improved to achieve this tighter bound).

From Certificate Revocation to Offline Verification. In order to deal with the danger presented by key exposures in public key signature systems, people have turned to certificate revocation mechanisms. If a signer's secret key is compromised, the certificate is revoked, thereby informing other people that signatures relying on that certificate should no longer be considered valid. The process of revoking a signature, however, is very demanding, both computationally and in terms of storage for the required infrastructure. Essentially, this means that verifiers will need to be online, to consult a certificate revocation list, to know if a signature is valid or not. Our scheme, however, in a manner similar to that stated in the intrusion resilient scheme presented by [IR02], minimizes the need for certificate revocation. Key exposures become much harder and rarer, while at the same time only compromising signatures for the present time period. Our key update, turns out to be a much faster and smarter reaction to key exposure than a key revocation. This way, with the revocation infrastructure becoming so obsolete, signature verification can even be performed offline, since there won't be any need to consult revocation lists any more.

Comparison. We compare our work to the two main ones previously known in this domain.

Compared to the work of [AMN01], our scheme achieves intrusion resilience, as opposed to the simple forward security of proactive signatures. This means that even if all the signers were to be compromised[1], only the current time period would be rendered unsafe, as opposed to the current and future periods of forward security. This is achieved through the use of the base modules, whose keys can be better protected, as they are not needed for signing, not used as frequently and can be held in a non-mobile, more secure environment. Furthermore, computationally, the [AMN01] scheme relies on [BM99], so it's inherently not as efficient in terms of computations as our scheme, which relies on [IR02], which is as efficient as [GQ88].

We next compare our work to that of [IR02]. Despite incorporating the benefits of proactive schemes by using a base and a signer, [IR02] does not take advantage of their multiparty aspects, thus limiting itself to a single signer and a single base. Single signer schemes may not suit particular implementations where more than one party is needed to authorize a document. An example of this could be the movie scenario where the president and the vice-president of a country need to turn their keys simultaneously to activate a nuclear weapon. Single base schemes carry another disadvantage; considering that key-exposures and intrusions are inevitable these days, it may not be as hard for an attacker to compromise the base module at the same time as the signer module.

2 Model

The definitions we use are based on those of [IR02]. Before we delve into any further formalities, however, allow us to present an intuitive overview of the model, to make what we are doing clearer.

[1]This is not an unlikely scenario, if one comes to consider that a plurality of signers might likely share some common affiliation. Such groups of people could be susceptible to an insider attack.

This model expands the notions of signer and base, to include an arbitrary number of parties for each. Each of these parties, be it a signer or a base, holds a share of the signer or base key, respectively. All the signers are required to collaborate, each contributing with her share of the secret key, in order to sign messages.

A key update operation is performed at the beginning of every time period, every base contributing with its share of the key, in order to evolve all the keys. These updates are always followed by a key refresh; a process where messages are transmitted between all parties, leading to a re-distribution of the shares for the secret key. Key refresh operations can also occur at any time in between updates (i.e. especially when there is a compromise).

The adversary is defined as capable of obtaining the shares of individual signers and bases, for time periods that he chooses. He can also gain access to update and refresh information, by intercepting messages between parties of his choice.

Note that, as in [IR02], though our definitions are given in the standard model, they can easily incorporate random oracles as per the model used for our proof of security.

2.1 Functional Definition

Borrowing from the notation used in [IR02], we'll use $RN(t)$ to refer to the number of times a key refresh is performed at time period t.

$SKS_*^{t.r}$ is used to denote the signing key which is shared among k signing parties. When referring to these shares individually, we'll use $SKS_i^{t.r}$ to refer to the i^{th} signer's key at time period t, refresh period r.

$SKB_*^{t.r}$ is similarly shared among ℓ bases and holds the secrets of the update process. $SKB_j^{t.r}$ would refer to the j^{th} base's key, at time period t, refresh period r.

$SKU_{i.j}^{t}$ is the message transmitted during the update phase, from the j^{th} base to the i^{th} signer, at time period t.

$SKR_{i.j}^{t.r}$ refers to the message transmitted during the refresh phase, from the j^{th} base to the i^{th} signer, at time period t, refresh period r.

In addition to the notation outlined above, our scheme is going to consist of the following algorithms:

1. *Gen*: Key generation algorithm
   Input: Security parameters k, ℓ, κ, λ and the total number of time periods T.
   Output: Preliminary key shares, $SKS_i^{0.0}$, for signers, and $SKB_j^{0.0}$, for bases.

2. *UB*: Algorithm to update a base's key share
   Input: The base's key share: $SKB_j^{t.r}$.
   Output: The base's new key share: $SKB_j^{t+1.0}$, along with k update messages:

$$SKU_{1.j}^{t}, SKU_{2.j}^{t}, \cdots, SKU_{k.j}^{t}$$

3. *US*: Algorithm to update a signer's key share
   Input: The signer's key share: $SKS_i^{t.r}$, as well as ℓ update messages:

$$SKU_{i.1}^{t}, SKU_{i.2}^{t}, \cdots, SKU_{i.\ell}^{t}$$

   Output: The signer's new key share: $SKS_i^{t+1.0}$

4. RB: Algorithm to refresh a base's key share
   Input: The base's key share: $SKB_j^{t.r}$.
   Output: A new key share for that base: $SKB_j^{t.r+1}$, along with k distinct refresh messages: $SKR_{i.j}^{t.r}$

5. RS: Algorithm to refresh a signer's key share
   Input: The signer's key share: $SKS_i^{t.r}$, as well as ℓ refresh messages: $SKR_{i.j}^{t.r}$.
   Output: A new key share for that signer: $SKS_i^{t.r+1}$

6. $ProSign$: The signing algorithm
   Input: The signers' shares of the signing key: $SKS_i^{t.r}$ and a message m
   Output: A valid, for time period t, signature of message m.

7. Ver: Signature verification algorithm
   Input: A message m, its corresponding signature and the public key PK
   Output: "valid" or "invalid"

2.2 Definition of Security

We define the attacker, F, as a probabilistic polynomial-time oracle Turing Machine with access to the following oracles:

O_{sig} is our signing oracle, mapping all input (m, t, r) (where $1 \leq t \leq T$ and $1 \leq r \leq RN(t)$) to corresponding signatures

$$ProSign(SKS_1^{t.r}, ..., SKS_k^{t.r}, m)$$

O_{sec} is the key exposure oracle; it will expose keys (analogous to intrusions) corresponding to the input it receives:

- On input $(``s_i", t.r)$, where $1 \leq t \leq T$ and $1 \leq r \leq RN(t)$, the oracle retrieves $SKS_i^{t.r}$.
- On input $(``b_j", t.r)$, where $1 \leq t \leq T$ and $1 \leq r \leq RN(t)$, the oracle retrieves $SKB_j^{t.r}$
- On input $(``u_j", t)$, where $1 \leq t \leq T-1$, the oracle retrieves SKU_j^t, as well as base j's refresh messages $SKR_{i.j}^{t.1}$ to all of the signers.
- On input $(``r_{i.j}", t.r)$, where $1 \leq t \leq T$ and $1 \leq r \leq RN(t)-1$, the oracle retrieves $SKR_{i.j}^{t.r}$

Let Q be a set of queries to the O_{sec} oracle. For any such set Q, $t \geq 1$ and $1 \leq r \leq RN(t)$, we say that the signing key $SKS_*^{t.r}$ is $Q-exposed$ if any of the following three conditions holds:

- All of the signers' shares have been compromised: $(``s_i", t.r) \in Q, \forall i : 1 \leq i \leq k$
- The keys have been refreshed at least once during this time period, all the refresh messages were compromised, and the signing key was $Q-exposed$ before the refresh (recursive): $r > 1, SKS_*^{t.(r-1)}$ is $Q-exposed$, and $(``r_{i.j}", t.(r-1)) \in Q, \forall i : 1 \leq i \leq k, \forall j : 1 \leq j \leq \ell$
- The keys were just updated, all the update messages were compromised and the signing key was $Q-exposed$ before the update (recursive): $r = 1$, $SKS_*^{(t-1).RN(t-1)}$ is $Q-exposed$, and $(``u_j", t-1) \in Q, \forall j : 1 \leq j \leq \ell$.

The same definition of $Q - exposure$ can be applied when the bases' keys are exposed; just replace SKS with SKB. If $SKS_*^{t.r}$ and $SKB_*^{t.r}$ both become $Q - exposed$, then the attacker will be able to forge present and future signatures. In such a case we say that the scheme is $(t, Q) - compromised$. The scheme is $(t, Q) - compromised$ if either of these two conditions hold:

- $SKS_*^{t.r}$ is $Q - exposed$
- $SKS_*^{t'.r}$ and $SKB_*^{t'.r}$ are both $Q - exposed$ for some $t' < t$

To finish defining our attacker's modus operandi, we will consider the following: The attacker's algorithm will return a 1 if the attacker manages to forge a signature, using the information obtained from Q (the set of queries to O_{sec}), without rendering the scheme $(t, Q) - compromised$, generating any illegal or out of bounds queries, or generating this signature by using the O_{sig} oracle. In all other cases it will return a 0. So, to define this more formally, for a k number of signers, ℓ bases, security values κ and λ, a T number of time periods and a map RN that maps time periods to the corresponding maximal number of refresh intervals at that time period; let the attacker's algorithm be such that:

$Run - Adversary(F, k, \ell, \kappa, \lambda, T, RN)$

returns 1 when:

- The attacker manages to forge a signature, using the information obtained from his set of queries to O_{sec}.
- **and** in doing so, the attacker doesn't render the scheme $(t, Q) - compromised$, nor does he generate any illegal or out of bounds queries.
- **and** the attacker hasn't generated this signature by using the O_{sig} oracle.

returns 0 on all other cases.

Now, to actually define security. Let $ProSiBIR[k, \ell, \kappa, \lambda, T, RN]$ be a proactive SiBIR scheme. We define $\mathbf{Succ}^{IR}(F, ProSiBIR[k, \ell, \kappa, \lambda, T, RN])$ to be the probability an adversary's algorithm will return 1 (i.e. satisfy all of the conditions listed above).

$$\mathbf{Succ}^{IR}(F, ProSiBIR[k, \ell, \kappa, \lambda, T, RN]) = PR[Run - Adversary(F, k, \ell, \kappa, \lambda, T, RN) = 1]$$

$\mathbf{InSec}^{IR}(ProSiBIR[k, \ell, \kappa, \lambda, T, RN], \tau, q_{sig})$ is defined to be the maximum possible value of $\mathbf{Succ}^{IR}(F, ProSiBIR[k, \ell, \kappa, \lambda, T, RN])$, over all possible adversary algorithms; so it is the probability of the "best possible" adversary algorithm returning a 1. Our goal is to show that our scheme's insecurity value is negligible:

$$\mathbf{InSec}^{IR}(ProSiBIR[k, \ell, \kappa, \lambda, T, RN], \tau, q_{sig}) < \epsilon$$

3 Scheme Construction

The scheme we propose, *ProSiBIR1*, is based on the SiBIR scheme. We expand the notions of signer and base to include multiple parties that function collaboratively as signers and bases. Our scheme utilizes four security parameters: k, ℓ, κ and λ. We use k and ℓ to refer to the number of signers and bases, respectively. Our scheme can easily be reduced to an equivalent of the SiBIR1 scheme, by setting the number of signers to 1 and the number of bases to 1. When referring to the product of all the messages sent by base j, we are going to denote this by $SKU^t_{*.j} = \prod_{i=1}^{k} SKU^t_{i.j}$. Similarly, the product of all the messages received by a signer i will be denoted by $SKU^t_{i.*} = \prod_{j=1}^{\ell} SKU^t_{i.j}$, whereas the product of all update messages will be denoted by $SKU^t_{*} = \prod_{i=1}^{k} \prod_{j=1}^{\ell} SKU^t_{i.j}$.

Implementing *Gen*:

- Generate a modulo n: n is the product of two $(\kappa/2)$-bit *Sophie Germain* primes, p and q:

$$n = pq = (2p' + 1)(2q' + 1)$$

  where p' and q' are also primes. Once n is computed, we keep it public and delete its factorization.

- Take a hash function, $H : \{0, 1\}^* \rightarrow \{0, 1\}^\lambda$.

- Generate some $(\lambda+1)$-bit exponents: $e_1, e_2, ..., e_T$, making sure that they be relatively prime. The product of all exponents, from a time period t to the last time period T, is denoted by $e_{[t,T]}$. It follows that the product of all the exponents is represented by $e_{[1,T]}$. Note that none of these exponents are secret.

- Every party generates a random number. This number is assigned to $s_{i[1,T]}$ if the party is the i^{th} signer, or to $b_{j[1,T]}$ if the party is the j^{th} base. Every signer i computes $v_{s_i} = \frac{1}{s_{i[1,T]}^{e_{[1,T]}}}$, every base j computes $v_{b_j} = \frac{1}{b_{j[1,T]}^{e_{[1,T]}}}$.

- The product of of all the v_{s_i}'s and all the v_{b_j}'s is assigned to $v = (v_{s_1} v_{s_2} ... v_{s_k})(v_{b_1} v_{b_2} ... v_{b_\ell})$. This v is public and is to be used during signature verification.

- The time and refresh periods are both initialized: $t = 0$, $r = 0$.

- Every signer i sets $SKS^{0.0}_i = \{s_{i[1,T]}, \emptyset, \emptyset\}$

- Every base j sets $SKB^{0.0}_j = \{b_{j[1,T]}\}$

The next step in completing key generation is to perform our very first update and refresh. Let's have a look at how these are done.

Implementing *UB*: A new time period always starts with an update operation to evolve the secret keys, bases first.

- The time period is incremented: $t = t + 1$

- First we update some values we'll need next time we try performing an update. Every base j computes $b_{j[t+1,T]} = b^{e_t}_{j[t,T]}$. and sets its key to $SKB^{t.1}_j = \{b_{j[t+1,T]}\}$

- Next, we update the values to be used in this time period. Every base computes $b_{jt} = b^{e_{[t+1,T]}}_{j[t,T]}$

- Every base then generates $k-1$ random numbers in Z_n^*, taking care that they be relatively prime to our modulus n, and assigns them to all but one of the update messages it is going to send out. As for the i^{th} update message that is left, every base j will compute it as follows: $SKU_{i.j}^t = \frac{b_{jt}}{SKU_{1.j}^t \cdots SKU_{i-1.j}^t SKU_{i+1.j}^t \cdots SKU_{k.j}^t}$. This essentially breaks the key into k multiplicative shares, one for each signer, using them as the update messages to be transmitted. Note that $SKU_{*.j}^t = \prod_{i=1}^{k} SKU_{i.j}^t = b_{jt}$.

Implementing US: As soon as the bases are done with their update, the signers use the update messages received to evolve their keys.

- Every signer multiplies the update messages she has received to compute $SKU_{i.*}^t = \prod_{j=1}^{\ell} SKU_{i.j}^t$
- As with the bases, all signers now have to update some values they'll need for the next update. Every signer i computes $s_{i[t+1,T]} = s_{i[t,T]}^{e_t}$.
- Next the signers update the value they'll use during this time period. Every signer computes $s_{it} = s_{i[t,T]}^{e_{[t+1,T]}}$.
- Every signer sets her key to $SKS_i^{t.1} = \{s_{i[t+1,T]}, s_{it}, SKU_{i.*}^t\}$

Implementing RB: Every update operation is followed by a refresh, though we can also have additional refreshes in between. Refresh operations start with the bases.

- The refresh period is incremented: $r = r + 1$
- Every base j generates a random number $SKR_{i.j}^{t.r} \in Z_n^*$ for every signer i. Note that this key needs to be relatively prime to our modulus n.
- Each base j refreshes its key by multiplying with each of the k random numbers it generated: $SKB_j^{t.r} = \{b_{j[t+1,T]} \prod_i SKR_{i.j}^{t.r}\}$
- All of the $SKR_{i.j}^{t.r}$ are then transmitted to their respective signers.

Implementing RS: When the bases transmit their refresh messages, all signers have to refresh their keys accordingly, so as to complete the redistribution of shares.

- Every signer i refreshes its key by dividing with each of the ℓ random numbers it has received from the bases:
$$SKS_i^{t.r} = \left\{ \frac{s_{i[t+1,T]}}{\prod_i SKR_{i.j}^{t.r}}, s_{it}, SKU_{i.*}^t \right\}$$

Now that we have finished describing how key generation, update and refresh are performed, we can proceed to talk about signing.

Implementing $ProSign$:

- Every signer generates a random $x_i \in Z_n^*$. The product of all the x_i's is denoted by $x = \prod_i x_i = x_1 x_2 ... x_k$. Note here that, while each signer knows her own x_i, nobody ever learns x in its entirety. Every signer also computes $y_i = x_i^{e_t}$.
- All the y_i's are pooled to compute $y = \prod_i y_i$, which in essence, is the same as x^{e_t}. y is known to all the signers and is then used to compute the hash $\sigma = H(t, e_t, y, M)$
- Every signer computes $z_i = x_i s_{it}^{\sigma} (SKU_{i.*}^t)^{\sigma} = x_i s_{it}^{\sigma} (\prod_{j=1}^{\ell} SKU_{i.j}^t)^{\sigma}$
- The z_i's are then pooled to compute:

$$z = \prod_{i=1}^{k} z_i = x s_t^{\sigma} \left(\prod_{i=1}^{k} \prod_{j=1}^{\ell} SKU_{i.j}^t \right)^{\sigma} = x s_t^{\sigma} \left(\prod_{j=1}^{\ell} b_{jt} \right)^{\sigma} = x \left(s_t b_t \right)^{\sigma}$$

- The signature is now ready. We simply return: $\{z, \sigma, t, e_t\}$

Implementing Ver:
Compute $y' = z^{e_t} v^{\sigma}$
The signature is valid iff: $H(t, e_t, y', M) = \sigma$
This makes sense because, if the signature is valid, then y' will be: $y' = x^{e_t} \left(s_t b_t \right)^{\sigma e_t} v^{\sigma} = y$

4 Security

4.1 Complexity Assumption

The strong RSA assumption states that given a number $n = pq$, where p and q are both prime, as well as an $\alpha \in Z_n^*$, it is computationally intractable find a $\beta \in Z_n^*$ and an $r \geq 3$ such that $\beta^r \equiv \alpha \pmod{n}$. Its use was introduced by [BP97] and [FO97], though [IMS03] provides a more rigorous study of the complexity of the assumption. We are going to base our proof on a variant identical to the one used by [IR01] and [IR02]. This variant restricts our modulus n, so that it is formed by the product of two *Sophie Germain* primes. It also places an upperbound κ on the length of our modulus and limits the possible values of $r : r \leq 2^{\lambda+1}$. Both [IR01] and [IR02], do an excellent job of explaining why these restrictions don't strengthen our assumption and why $\mathbf{InSec}^{SRSA}(\kappa, \lambda, \tau)$ is negligible.

4.2 Proof of Security

Theorem 1. *Suppose there exists a forger, F, with a running time $\leq \tau$, asking at most q_{hash} hash queries and q_{sig} signing queries, whose probability of success is $\mathbf{Succ}^{IR}(F, ProSiBIR[k, \ell, \kappa, \lambda, T, RN]) \geq \epsilon$. We claim that, from that forger, we can construct an algorithm A that, for $\alpha \in Z_n^*$ and a random modulus n, can represent α as a perfect power:*

i.e. $\quad b^r \equiv \alpha \pmod{n}$

Such that:

$$\epsilon' = \frac{(T-1)\left(\epsilon - 2^{2-\kappa}q_{sig}(q_{hash}+1)\right)^2}{T^2(q_{hash}+1)} - \frac{(T-1)\left(\epsilon - 2^{2-\kappa}q_{sig}(q_{hash}+1)\right)}{2^{\lambda}T}$$

and τ' is polynomial on τ. Where τ' is the algorithm's running time, and its probability of success is $\geq \epsilon'$.

Proof. Let us start by examining the way in which A will use F, as a subroutine, to break strong RSA. A picks a random w between 1 and T, guessing F will forge a signature corresponding to that time period. A generates values according to our scheme construction; starting with the exponents $e_1, ..., e_T$, then proceeding to calculate $v = \frac{1}{\alpha^{e_1 \cdots e_{w-1} e_{w+1} \cdots e_T}} \pmod n$.

A then calls the forger, F, using the v he just computed. Signature and hash queries are to be answered randomly, while using respective query tables to enforce consistency in case the same query is asked twice. A will also be responsible of addressing any of F's queries to the signing and key exposure oracles. A more detailed discussion of how A will answer key exposure queries can be found in the O_{sec} section of the Appendix.

If F does not succeed in forging a signature for time period w, then A fails and needs to try again. In the case where F does succeed in producing (z, σ, w, e), however, we use it to compute $y \equiv z^e v^{\sigma} \pmod n$. This signature will have required a hash value $\sigma = H(w, e, y, m)$ which we can spot on the query table once we've computed y.

A, then, runs the forger again on the same input, returning the same results on all queries, according to the query tables he kept last time. When, during this second run, the forger asks A for the hash value of (w, e, y, m), A will return another value σ' instead of the σ returned in the first run. If this σ' is not the one used to forge the signature, then A fails and needs to try again. If the forger succeeds for a second time, the forged signature (z', σ', w, e) will be such that $y \equiv z^e v^{\sigma} \equiv z'^e v^{\sigma'}$. This means that $(\frac{z}{z'})^e \equiv v^{\sigma'-\sigma} \equiv \alpha^{(\sigma-\sigma')e_1 \dots e_{w-1}e_{w+1} \dots e_T}$. We know that e, also referred to as e_w, is by definition relatively prime to $e_1 \dots e_{w-1} e_{w+1} \dots e_T$. We also know that $\sigma - \sigma'$ is at most one bit smaller than e. Hence, it holds that $\gcd(e_1 \dots e_{w-1}e_{w+1} \dots e_T(\sigma - \sigma'), e) = \gcd(\sigma - \sigma', e) < e$ (since the σ and σ' provided by A were distinct). Thereby, by means of Lemma 1 from [IR01], we deduce that A can generate a non-trivial root of α, its degree being $\frac{e}{\gcd(f_{w+1}(\sigma'-\sigma), e) > 1}$

Now we can put all the probabilities together to derive the value of ϵ'. An analysis of why the probabilities turn out as they do can be found in the Appendix. This completes the proof of Theorem 1. □

Let $u = (\epsilon - 2^{2-\kappa}q_{sig}(q_{hash}+1))/T$. This enables us to transform our equation to

$$\epsilon' = \frac{(T-1)u^2}{q_{hash}+1} - \frac{(T-1)u}{2^{\lambda}} \Leftrightarrow$$

$$u^2 - (q_{hash}+1)2^{-\lambda}u - \frac{\epsilon'(q_{hash}+1)}{T-1} = 0$$

Now we can solve this quadratic equation to get

$$u \quad = \quad \frac{(q_{hash}+1)2^{-\lambda} + \sqrt{(q_{hash}+1)^2 2^{-2\lambda} + \frac{4\epsilon'(q_{hash}+1)}{T-1}}}{2}$$

$$
\begin{aligned}
&= (q_{hash}+1)2^{-\lambda-1} + \sqrt{(q_{hash}+1)^2 2^{-2\lambda-2} + \frac{\epsilon'(q_{hash}+1)}{T-1}} \\
&\leq (q_{hash}+1)2^{-\lambda-1} + \sqrt{(q_{hash}+1)^2 2^{-2\lambda-2}} + \sqrt{\frac{\epsilon'(q_{hash}+1)}{T-1}} \\
&= (q_{hash}+1)2^{-\lambda-1} + (q_{hash}+1)2^{-\lambda-1} + \sqrt{\frac{\epsilon'(q_{hash}+1)}{T-1}} \\
&= 2^{-\lambda}(q_{hash}+1) + \sqrt{\frac{\epsilon'(q_{hash}+1)}{T-1}}
\end{aligned}
$$

Now we can solve for ϵ (based on the way we initially defined u) to get that $\epsilon = Tu + 2^{2-\kappa} q_{sig}(q_{hash}+1)$. Substituting u with the inequality we have computed, we have that

$$\epsilon \leq T\sqrt{\frac{\epsilon'(q_{hash}+1)}{T-1}} + T2^{-\lambda}(q_{hash}+1) + 2^{2-\kappa} q_{sig}(q_{hash}+1)$$

This can also be expressed as:

$$\epsilon \leq T\sqrt{\frac{(q_{hash}+1)\mathbf{InSec}^{SRSA}(\kappa,\lambda,\tau')}{T-1}} + 2^{-\lambda}T(q_{hash}+1) + 2^{2-\kappa} q_{sig}(q_{hash}+1)$$

Hence, we have managed to upper-bound our scheme's insecurity; it is close to or less than the insecurity of strong RSA. This clearly shows that our scheme's insecurity is negligible.

References

[AMN01] Michel Abdalla, Sara Miner and Chanathip Namprempre. *Forward-secure threshold signature schemes.* In David Naccache, editor, Progress in Cryptology - CT-RSA 2001, volume 2020 of Lecture Notes in Computer Science, pages 143-158. Springer-Verlag, April 8-12 2001.

[And97] 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.

[BM99] Mihir Bellare and Sara Miner. *A forward-secure digital signature scheme.* In Michael Wiener, editor, Advances in Cryptology - CRYPTO 1999, 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/.

[BP97] Niko Bari'c and Birgit Pfitzmann. *Collision-free accumulators and failstop signature schemes without trees.* In Walter Fumy, editor, Advances in Cryptology - EUROCRYPT 1997, volume 1233 of Lecture Notes in Computer Science, pages 480494. Springer-Verlag, 1115 May 1997.

[DF89] Yvo Desmedt and Yair Frankel. *Threshold cryptosystems.* In G. Brassard, editor, Advances in Cryptology - CRYPTO 1989, volume 435 of Lecture Notes in Computer Science, pages 307-315. Springer-Verlag, 1990, 20-24 August 1989.

[DKXY03] Yevgeniy Dodis, Jonathan Katz, Shouhuai Xu and Moti Yung. *Strong Key-Insulated Signature Schemes.* In Y. Desmedt, editor, Proceedings of the 6th Annual International Workshop on Practice and Theory in Public Key Cryptography (PKC 2003), volume 2567 of Lecture Notes in Computer Science, pages 130-144. Springer-Verlag, 2003.

[FO97] Eiichiro Fujisaki and Tatsuaki Okamoto. *Statistical zero knowledge protocols to prove modular polynomial relations.* In In Burton S. Kaliski Jr., editor, Advances in Cryptology - CRYPTO 1997, volume 1294 of Lecture Notes in Computer Science, pages 1630. Springer-Verlag, 1721 August 1997.

[GQ88] Louis Claude Guillou and Jean-Jacques Quisquater. *A "paradoxical" identity-based signature scheme resulting from zero-knowledge.* In Shafi Goldwasser, editor, Advances in Cryptology - CRYPTO 1988, volume 403 of Lecture Notes in Computer Science, pages 216-231. Springer-Verlag, 1990, 21-25 August 1988.

[HJKY95] A. Herzberg, M. Jarecki, H. Krawczyk, and M. Yung. *Proactive secret sharing or: How to cope with perpetual leakage.* In D. Coppersmith, editor, Proc. of CRYPTO 1995, volume 963 of Lecture Notes in Computer Science, pages 339-352. Springer-Verlag, August 1995.

[HJJ+97] Amir Herzberg, Markus Jakobsson, Stanislaw Jarecki, Hugo Krawczyk, and Moti Yung. *Proactive public key and signature systems.* In Fourth ACM Conference on Computer and Communication Security, pages 100-110. ACM, April 1-4 1997.

[IMS03] Shintaro Itagaki, Masahiro Mambo and Hiroki Shizuya. *On the Strength of the Strong RSA Assumption.* In IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, Vol.E86-A No.5, pp.1164-1170, 2003.

[IR01] Gene Itkins 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.

[IR02] Gene Itkins 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/.

[Itk02] Gene Itkins. *Intrusion-Resilient Signatures: Generic Constructions, or Defeating Strong Adversary with Minimal Assumptions.* In Third Conference on Security in Communication Networks (SCN 2002), Lecture Notes in Computer Science, vol. 2576, pp. 102–118, 2002.

[MMM02] Tal Malkin, Daniele Micciancio, and Sara Miner. *Efficient generic forward-secure signatures with an unbounded number of time periods.* In Third Conference on Security in Lars Knudsen, editor. Advances in Cryptology - EUROCRYPT 2002, Lecture Notes in Computer Science. Springer-Verlag, 28 April-2 May 2002.

[OY91] Rafail Ostrovsky and Moti Yung. *How to withstand mobile virus attacks.* In 10^{th} Annual ACM Symp. on Principles of Distributed Computing, pages 5159, 1991.

Appendix A: O_{sec} Simulation

To begin with, we will need to use the notions of *points* and *intervals*, as they were defined in the SiBIR paper. For the sake of clarity, we have included these definitions here. In order to simplify notation, from now on, valid time/refresh periods of the form $t_i.r_j : 0 < r_j < RN(t_i)$ will be referred to as *points*. Let π_1, π_2 be two points, such that $\pi_1 < \pi_2$. We define an *interval* I to be the set of all points between π_1 and π_2, iff every point $\pi \in I$ is such that, $("r_{i.j}", \pi) \in Q, \forall P : 1 \leq P \leq k + \ell$.

Lemma 1. *Let $\pi_1, \pi_2 \in I : \pi_{1,2} \leq t$. If $("s_i", \pi_1) \in Q \forall i : 1 \leq i \leq k$ and $("b_j", \pi_2) \in Q \forall j : 1 \leq j \leq \ell$, then the scheme is $(t, Q) - compromised$.*

Proof. SKS_t and SKB_t are both $Q - exposed$; this follows trivially from the definitions of $Q - exposure$ and $intervals$. Consequently, proving the lemma that the scheme is indeed $(t, Q) - compromised$. □

If, for any $\pi \in I$, $("s_i", \pi) \in Q$ $\forall i : 1 \leq i \leq k$, then I is called a *signer interval.* A similar definition applies for what we call a *base interval*; $("b_j", \pi) \in Q$ $\forall j : 1 \leq j \leq \ell$. For the purposes of proving security, no interval can be both a signer and a base interval at the same time, as this would render the scheme $(w, Q) - compromised$ (neither F, nor A, can succeed if the scheme is $(w, Q) - compromised$).

Now that these notions are in place, we can proceed by generating all the values that A will need to respond to F's key exposure queries. First we randomly pick $b_{j[1,T].0} \in Z_n^*$, for every base j, making sure that these $b_{j[1,T].0}$ are relatively prime to n. Then we use the already defined base update and refresh functions of our scheme to generate all the SKB's, SKU's and SKR's required to answer F's queries. This gives us all the $b_{j[t+1,T].r}$ for $1 \leq t \leq T$ and $0 \leq r < RN(t)$.

We randomly pick $s_{i[1,T].0} \in Z_n^*$, for every signer except the k-th, making again sure they are relatively prime to n. For the k-th signer we compute:

$$s_{k[1,T].0} = \left(\prod_{j}^{\ell} b_{j[1,T].0} \prod_{i}^{k-1} s_{i[1,T].0} \right)^{-1}$$

We use the signer update and refresh functions of our scheme, along with the previously computed values, to generate all the SKS's until time period w. This gives us all the $s_{i[t+1,T].r}$ for $1 \leq t < w$ and $0 \leq r < RN(t)$. We randomly pick $s_{i[w+1,T].0}$ for every signer except the k-th so that they are relatively prime to n. We set

$$s_{k[w+1,T].0} = \alpha^{e_{[1,w-1]}} \left(\prod_{i}^{k-1} s_{i[w+1,T].0} \prod_{j}^{\ell} b_{j[w+1,T].0} \right)^{-1}$$

and we use the signer update and refresh functions again to generate the rest of the SKS's. This will give us the $s_{i[t+1,T].r}$ for $w \leq t < T$ and $0 \leq r < RN(t)$.

Now that everything is computed, A is ready to answer all of F's "s", "b" and "r" queries. As far as update queries go, A will need to check Q to find out the interval in which t resides in a $("u_j", t)$ query. If it is in a base interval, A responds to the query by sending $SKU_j^t = b_{jt}$. If it is in a signer interval and $t \geq w$, then A responds with $SKU_j^t = b_{jt}$. If, however, it is a signer interval where $t < w$; A will respond with $SKU_j^t = b_{jt}$ for $j \neq k$ and $SKU_k^t = \alpha^{e_{[1,T]}/e_t e_w} b_{kt}$ for the

k-th signer. Note that, when $t \neq w$, $\left(\prod_i s_{it} \prod_j SKU_j^t\right)^{e_t} = 1/v$. Answering queries in such a way creates a simulated run, so that it is indistinguishable from an actual run in the eyes of the forger, which is exactly what we need for F to be able to break the scheme. Details on why this simulated run is consistent with that of a true signer, are very similar to [IR02].

Appendix B: Counting Probabilities

In the interest of completeness, we include this probability analysis, working along the same lines as [IR01], while at the same time focusing on certain stages for the sake of clarity.

Let $p_{h,b,S}$ be the probability of F forging a signature based on the h-th hash query amongst the $q_{hash}+1$ it can ask, following a break-in at time period b, where a string S (of length m) is used to determine F's random tape.

Our algorithm is supposed to run F twice, while issuing different responses (σ and σ') to the h-th hash query. During the second run, however, there is a $2^{-\lambda}$ probability that the two hash values will collide. This means that instead of $p^2_{h,b,S}$, the probability of F forging a signature based on the h-th hash query in both runs but answered differently in the second run, following a break-in query at time period b and using the string S, is $p_{h,b,S}(p_{h,b,S} - 2^{-\lambda})$

Let $q_{h,b}$ be the probability of this over all possible strings of length m. It will then be such that

$$q_{h,b} = \sum_{S\in\{0,1\}^m} 2^{-m} p_{h,b,S}(p_{h,b,S} - 2^{-\lambda}) = 2^{-m}\left(\sum_{S\in\{0,1\}^m} p^2_{h,b,S} - 2^{-\lambda}\sum_{S\in\{0,1\}^m} p_{h,b,S}\right)$$

Let $p_{h,b}$ be the probability of F producing a forgery based on the h-th hash query after a break-in query at time period b. It therefore follows that the sum of the $p_{h,b,S}$ over the 2^m possible strings of length m is $\sum_{S\in\{0,1\}^m} p_{h,b,S} = 2^m p_{h,b}$

Hence, by applying Lemma 2 from [IR01], we have that

$$\begin{aligned} q_{h,b} &\geq 2^{-m}\left(\frac{\left(\sum_{S\in\{0,1\}^m} p_{h,b,S}\right)^2}{2^m} - 2^{-\lambda}\sum_{S\in\{0,1\}^m} p_{h,b,S}\right) \\ &= \frac{2^{-m}(2^m p^2_{h,b})}{2^m} - 2^{-\lambda}2^{-m}2^m p_{h,b} = p^2_{h,b} - 2^{-\lambda} p_{h,b} \end{aligned}$$

$\therefore q_{h,b} \geq p_{h,b}(p_{h,b} - 2^{-\lambda})$

Let ε_b be the probability of F producing a valid forgery after a break-in query at time period b. In essence it's the sum of the $p_{h,b}$ over all possible hash queries. So $\varepsilon_b = \sum_{h=1}^{q_{hash}+1} p_{h,b}$

The probability of F forging a signature on both runs based on the same hash query, while that query gets answered differently in the second run, and following a break-in query at period b is

$$\begin{aligned}
q_b = \sum_{h=1}^{q_{hash}+1} q_{h,b} &\geq \sum_{h=1}^{q_{hash}+1} p_{h,b}^2 - \sum_{h=1}^{q_{hash}+1} 2^{-\lambda} p_{h,b} \\
&\geq \frac{\left(\sum_{h=1}^{q_{hash}+1} p_{h,b}\right)^2}{q_{hash}+1} - \sum_{h=1}^{q_{hash}+1} 2^{-\lambda} p_{h,b} \text{ (Lemma 2, [IR01])} \\
&= \frac{\varepsilon_b^2}{q_{hash}+1} - 2^{-\lambda}\varepsilon_b
\end{aligned}$$

The sum of the e_b over all possible time periods for the break-in, is $\delta = \sum_{b=1}^{T} e_b$.

Note that since this is an intrusion resilient scheme, F's break-in query at time period b cannot take place during the same time period as the forgery. The probability of A picking a $w \neq b$ is $\frac{T-1}{T}$. The analysis of [IR02] mentions a factor of $\frac{1}{T}$ in place of this probability. A closer look at [IR01], however, which is the paper from which both this and the proof of [IR02] stem, suggests that since these are not simply forward secure schemes, but also intrusion resilient, a tighter bound of $\frac{T-1}{T}$ is forthcoming.

If we calculate the sum of the q_b over all possible time periods for a break-in and take into account the probability of $w \neq b$, we'll get

$$\begin{aligned}
\varepsilon' = \frac{T-1}{T}\sum_{b=1}^{T} q_b &\geq \frac{T-1}{T}\sum_{b=1}^{T}\left(\frac{\varepsilon_b^2}{q_{hash}+1} - 2^{-\lambda}\varepsilon_b\right) \\
&= \frac{(T-1)\sum_{b=1}^{T}\varepsilon_b^2}{T(q_{hash}+1)} - \frac{(T-1)\sum_{b=1}^{T}\varepsilon_b}{2^\lambda T} \\
&\geq \frac{(T-1)\left(\sum_{b=1}^{T}\varepsilon_b\right)^2}{T^2(q_{hash}+1)} - \frac{(T-1)\delta}{2^\lambda T} \text{ (Lemma 2, [IR01])} \\
&= \frac{(T-1)\delta^2}{T^2(q_{hash}+1)} - \frac{(T-1)\delta}{2^\lambda T}
\end{aligned}$$

We know that $\delta = \varepsilon - q_{sig}(q_{hash}+1)2^{2-\kappa}$, because F's chance of success is affected by the probability of collision between a hash defined in a signature query and that of a hash query. From this we deduce that

$$\varepsilon' \geq \frac{(T-1)\left(\varepsilon - 2^{2-\kappa}q_{sig}(q_{hash}+1)\right)^2}{T^2(q_{hash}+1)} - \frac{(T-1)\left(\varepsilon - 2^{2-\kappa}q_{sig}(q_{hash}+1)\right)}{2^\lambda T}$$