The extended abstract of this work appears in D. Naccache, editor, *Topics in Cryptology – CT-RSA 2001*, Volume 2020 of Lectures Notes in Computer Science, San Francisco, CA, USA, Apr. 8–12, 2001. Springer-Verlag, Berlin, Germany. This is the full version.

Forward-Secure Threshold Signature Schemes

Michel Abdalla   Sara Miner   Chanathip Namprempre

Department of Computer Science & Engineering
University of California at San Diego
9500 Gilman Drive
La Jolla, California 92093, USA
{mabdalla,sminer,cnamprem}@cs.ucsd.edu
http://www-cse.ucsd.edu/users/{mabdalla,sminer,cnamprem}

Abstract

We construct *forward-secure threshold* signature schemes. These schemes have the following property: even if more than the threshold number of players are compromised, it is not possible to forge signatures relating to the past. This property is achieved while keeping the public key fixed and updating the secret keys at regular intervals. The schemes are reasonably efficient in that the amount of secure storage, the signature size and the key lengths do not vary proportionally to the number of time periods during the lifetime of the public key. Both proposed schemes are based on the Bellare-Miner forward-secure signature scheme. One scheme uses multiplicative secret sharing and tolerates mobile eavesdropping adversaries. The other scheme is based on polynomial secret sharing and tolerates mobile halting adversaries. We prove both schemes secure via reduction to the Bellare-Miner scheme, which is known to be secure in the random oracle model assuming that factoring is hard. Finally, we sketch modifications which would allow our polynomial-sharing-based scheme to tolerate malicious adversaries.

Keywords: threshold cryptography, forward security, signature schemes, proactive cryptography.

Contents

1 Introduction

Exposure of a secret key for "non-cryptographic" reasons —such as a compromise of the underlying machine or system, human error, or insider attacks— is, in practice, the greatest threat to many cryptographic protocols. The most commonly proposed remedy is distribution of the secret key across multiple servers via secret sharing. For digital signatures, the primitive we consider in this paper, the main instantiation of this idea is threshold signature schemes [12]. The signature is computed in a distributed way based on the shares of the secret key, and a sufficiently large set of servers must be compromised in order to obtain the key and generate signatures.

Distribution of the key makes it harder for an adversary to learn the secret key, but does not remove this risk. Common mode failures —flaws that may be present in the implementation of the protocol or the operating system being run on all servers— imply that breaking into several machines may not be much harder than breaking into one. Thus, it is realistic to assume that even a distributed secret key can be exposed.

Proactive signatures address this to some extent, requiring all of the break-ins to occur within a limited time frame [17]. This again only ameliorates the key exposure problem. Once a system hole is discovered, it can quite possibly be exploited across various machines almost simultaneously.

A common principle of security engineering is that one should not rely on a single line of defense. We suggest a second line of defense for threshold signature schemes which can mitigate the damage caused by complete key exposure, and we show how to provide it. The idea is to provide *forward security*.

Forward security for digital signature schemes was suggested by Anderson [2], and solutions were designed by Bellare and Miner [3]. The idea is that a compromise of the *present* secret signing key does not enable an adversary to forge signatures pertaining to the *past*. (In this light, the term "backward security" may have been more appropriate, but we decide to be consistent with existing terminology in the literature here.) Bellare and Miner [3] focus on the single-signer setting and achieve this goal through the *key evolution paradigm*: the user produces signatures using different secret keys during different time periods while the public key remains fixed. Starting from an initial secret key, the user "evolves" the current secret key at the end of each time period to obtain the key to be used in the next. She then erases the current secret key to prevent an adversary who successfully breaks into the system at a later time from obtaining it. Therefore, the adversary can only forge signatures for documents pertaining to time periods after the exposure, but not before. The integrity of documents signed before the exposure remains intact.

Combining forward security and threshold cryptography will yield a scheme that can provide some security guarantees *even if* an adversary has taken control of *all* servers and, as a result, has completely learned the secret. In particular, she cannot forge signatures as if they were legitimately generated *before* the break-in. The complete knowledge of the secret signing key is useless for her with regard to signatures from "the past." [1]

It is worth noting that, at first glance, forward-secure signature schemes and signature schemes based on secret sharing can be viewed as two different alternatives for addressing the same problem, namely the key exposure problem. However, they do, in fact, provide complementary security properties. Forward security prevents an adversary from forging documents

[1] A related idea involving key evolution and distribution of secrets was presented in the context of key escrow [7]. However, in their work, the public key needs to be updated, which, in turn, requires the participation of a trusted third party in every time period.

pertaining to the past *even if* he is able to obtain the current secret key. On the other hand, threshold and proactive signature schemes make it harder for an adversary to learn the secret key altogether. The crucial distinction between the two notions is that forward security involves *changing the actual secret* while a secret sharing scheme distributes the secret which remains *unchanged* throughout the execution of the protocol. This is true for *both* threshold *and* proactive schemes. In particular, the refresh steps performed in a proactive scheme update the shares of the secret, but not the secret itself. Therefore, without forward security, if an adversary ever successfully obtains this secret, the validity of all documents signed by the group can be questioned, regardless of when the documents were claimed to have been signed.

Furthermore, one can think of the addition of forward security to threshold schemes as a deterrent to attempts at exposing the key. Specifically, in a forward-secure scheme, a stolen key is less useful to an adversary (i.e. it can't help her forge past signatures) than in a non-forward-secure scheme, since it only yields the ability to generate signatures in the future. In fact, as time progresses, the potential benefits of exposing the key at the current time dwindle, since there are fewer time periods in which it can generate a signature. Thus, an adversary's "cost-benefit analysis" may prevent her from attacking such a scheme in the first place.

Not only does forward security provide security improvements to an existing threshold signature scheme, it can do so without adding any "online cost" to the scheme, as is the case for both of our schemes. (By "online cost," we mean the cost incurred during signing such as the number of interactions or rounds in the protocol.) That is, with some pre-computation performed off-line, no more interactions are required to sign a message beyond those needed in the non-forward-secure threshold version of the scheme. This makes forward security an especially attractive improvement upon a distributed signature scheme.

Constructing a Candidate Scheme. Designing a forward-secure threshold scheme would be an easy task if one could ignore efficiency issues. In particular, an *efficient* forward-secure threshold signature scheme should incur

— only minimal interactions among the players, and

— small *cost parameters* (e.g. amount of storage, the size of signatures, and the key lengths) such as ones that do not vary proportionally to the number of time periods,

in addition to maintaining the basic security property of a forward-secure signature scheme, i.e. it should still be "hard" to forge signatures pertaining to the past.

Often times the two goals listed above are in conflict, and trade-offs need be made. For example, one can simply let a dealer pick T pairs of secret keys and public keys where T is the number of time periods for a lifetime of the public key, then distribute the secret keys to the players using a secret-sharing scheme. The jth secret key is then used to distributedly sign documents during the time period j, and the jth public key is used to verify documents signed during the time period j. Clearly, key evolution under this scheme requires *no* interactions among the players (each player simply deletes its own share of the secret key of the previous time period), and thus, our first goal is satisfied. However, the key lengths are proportional to T, thereby violating our second goal. With a technique suggested by Anderson[2], one can reduce the length of the public key, but the storage of the secret key will still be proportional to T [3].

As pointed out in [3], there are other alternatives. However, they either require the amount of *secure* storage or the signature size to be proportional to the number of time periods. Clearly, if these costs are not of major concern, a scheme such as the simple scheme presented above would be appropriate. Otherwise, one needs to consider different alternatives.

Our goal is to construct a forward-secure threshold signature scheme that satisfies the aforementioned criteria for efficiency. Individually, though, performing secure distributed computation and achieving forward security are difficult problems to solve efficiently. Therefore, our approach is to combine existing solutions for each problem, rather than attempting to re-invent the wheel.

Factoring-Based Schemes. In this paper, we present two forward-secure threshold signature schemes whose cost parameters do not grow proportionally to the number of time periods during the lifetime of a public key. Both schemes are based on the Bellare-Miner scheme [3], which in turn is based on the schemes proposed in [13] and [20]. The Bellare-Miner signature scheme is proven forward-secure in the random oracle model assuming that factoring a composite number into two large prime factors is hard. Consequently, we are able to prove the schemes proposed in this paper secure in the random oracle model under the same assumption.

The first scheme uses multiplicative secret sharing [10, 11] to distribute the secrets while the second scheme uses the standard polynomial sharing of secrets. Figure 1 summarizes the properties relevant to evaluating the schemes. In particular, a desirable forward-secure threshold scheme should be able to tolerate a high number of compromises by powerful adversaries, should require a small number of players to sign a message and to update a key, and should incur a small number of rounds of interactions among players for both signing and updating. These criteria are listed in the first column of the table.

Scheme Characteristic	Multiplication-based	Polynomial-based
$t =$ Number of compromises tolerated	$t = n - 1$	$t = (n-1)/3$
Type of adversary tolerated	mobile eavesdropping	mobile halting
$k_s =$ Number of players needed to sign	n	$(2n+1)/3$
Rounds of (on-line) communication to sign	1	$2l$
$k_u =$ Number of players needed to update	n	$(2n+1)/3$
Rounds of communication to update	0	2

Figure 1: Comparing our two schemes. The value n represents the total number of players in the scheme, and l is a security parameter.

As indicated in the figure, the multiplication-based scheme tolerates an optimal number of compromises with only one round of interactions to sign a message and no interactions to update a key. These gains come at a cost of requiring a large number of participants both for signing and updating in addition to tolerating only eavesdropping adversaries (albeit mobile). In contrast, the polynomial-based scheme can tolerate more powerful adversaries while requiring a more reasonable number of honest players for signing and updating. The number of rounds of interactions among the players, however, is higher than that of the multiplication-based scheme.

The multiplication-based scheme we propose here is simple and efficient. This makes it an attractive way to achieve forward security with distribution of secrets in the presence of passive adversaries. It is not clear, however, how to extend the scheme to handle more powerful adversaries.

The polynomial-based scheme is more involved than the multiplication-based scheme. In order to tolerate mobile halting adversaries, we need to be able to generate random secrets and to reconstruct the secrets when some of the players are halted, in addition to being able to perform distributed computation involving the secrets without leaking them. Furthermore, since we assume the presence of mobile adversaries, we also need to ensure that a player can re-join the group even though it has been previously halted during crucial periods such as a key update phase.

Fortunately, active research in the area of secure distributed computation has yielded powerful techniques that address these issues [5, 15, 16, 18, 23]. Consequently, we are able to apply these existing techniques in a straightforward way to construct a solution that can cope with these problems and to rigorously prove its security. Moreover, it is also possible to extend the scheme to cope with malicious adversaries. We sketch the idea of this extension in Appendix A.

Overall, our schemes are reasonably efficient. Clearly, compared to the single-user setting, there are additional costs due to the interactions incurred in sharing secrets. However, as previously mentioned, with a small amount of pre-computation performed off-line, forward security adds no additional online cost to the threshold (but non-forward-secure) version of the underlying scheme. (We note that this threshold scheme is of independent interest.)

Open Problems. The current online cost in round complexity of the signing protocol of the polynomial-based scheme is $2l$ rounds of interactions among players where l is the security parameter. This cost stems mostly from the need to distributedly multiply $l+1$ values using the distributed multiplication protocol of [16], which can multiply only two numbers at a time. With some optimization, we can cut down this cost to $\lg l$ rounds in the worst case. However, a secure multi-party protocol that can efficiently compute a product of more than two numbers can dramatically cut down this signing cost. A protocol that can do so in one round would be ideal. Alternatively, one could try to design a new forward secure signature scheme that lends itself more naturally to distributed computation. For example, a scheme that requires less computation involving secret information in a single-user setting will improve the efficiency of the scheme dramatically in a distributed setting.

A Word about Time-Stamping. A property similar to that provided by forward security can also be obtained via a time-stamping service. In particular, the signers could ask a trusted third party to time-stamp the document and then sign the resulting document themselves. Relying on such a service, however, may be costly and, more importantly, can introduce a single point of failure. In terms of the latter shortcoming, we stress that relying on a trusted third party to time-stamp every single document to be signed introduces a single point of failure that could be much more vulnerable compared to the trusted dealer used for key generation. The reason is that key generation is done only once in the beginning of the entire lifetime of the public key whereas a time-stamping service is requested every time a document needs be signed. As a result, an adversary has a much larger window of opportunity to attack the scheme via the time-stamping service than via the trusted dealer.

2 Definitions and Notation

In this section, we describe our communication model and the capabilities of different types of adversaries. We then explain what is meant by a forward-secure threshold signature scheme, using definitions relating to forward security based heavily on those provided in [3]. Finally, we formalize our notion of security, and describe notation used in the remainder of the paper.

Communication Model. The participants in our scheme include a set of n players who are connected by a broadcast channel. Additionally, they are capable of private point-to-point communication over secure channels. (Such channels might be implemented on the broadcast channel using cryptographic techniques.) Furthermore, we assume that there exists a trusted dealer during the setup phase and that the players are capable of both broadcast and point-to-point communication with him. Finally, we work in a synchronous communication model; that is, all participating players have a common concept of time and, thus, can send their messages simultaneously in a particular round of a protocol.

Types of Adversaries. We assume that any adversary attacking our scheme can listen to all broadcasted information and may compromise the players in some way to learn their secret information. However, the adversary might work in a variety of contexts. We categorize the different types of adversaries here. In both categories described below, the last option listed describes the most powerful adversary, since it always encompasses the preceding options in that category.

The first category we consider is the power an adversary can have over a compromised player. We list the options, as outlined in [14]. First, an adversary may be *eavesdropping*, meaning that she may learn the secret information of a player but may not affect his behavior in any way. A more powerful adversary is one that not only can eavesdrop but can also stop the player from participating in the protocol. We refer to such an adversary as a *halting* adversary. Finally, the most powerful notion in this category is a *malicious* adversary, who may cause a player to deviate from the protocol in an unrestricted fashion.

The second category which defines an adversarial model describes the manner in which an adversary selects the set of players to compromise. The first type is a *static* adversary, who decides before the protocol begins which set of players to compromise. An *adaptive* adversary, on the other hand, may decide "on the fly" which player to corrupt based on knowledge gained during the run of the protocol. Finally, a *mobile* adversary is traditionally one which is not only adaptive, but also may decide to control different sets of players during different time periods. In this case, there may be no player which has not been compromised throughout the run of the protocol, but the adversary is limited to controlling some maximum number of players at any one time.

Forward-Secure Threshold Signature Schemes. A (t, k, n)-*threshold* signature scheme is one in which the secret signing key is distributed among a set of n players, and the generation of any signature requires the cooperation of some size-k subset of honest players. In addition, any adversary who learns t or fewer shares of the secret key is unable to forge signatures. It is often the case that $k = t + 1$; that is, the number of honest players required for signature generation is exactly one more than the number of compromised shares that the scheme can tolerate. A threshold scheme has the advantages of a distributed secret while often not requiring all n players to participate each time a signature is generated.

In this paper, we are concerned with *forward-secure* threshold signature schemes. These schemes make use of the key evolution paradigm, and their operation is divided into time periods. Throughout the lifetime of the scheme, the public key is fixed, but the secret key changes at each time period. As in standard signature schemes, there is a key generation protocol, a signing protocol, and a verification algorithm. In a forward-secure scheme, however, there is an additional component known as the evolution or update protocol, which specifies how the secret key is to evolve throughout the lifetime of the scheme. A (t, k_s, k_u, n) key-evolving threshold signature scheme can tolerate at most t corrupted players and works as follows.

First, there is a key generation phase. Given a security parameter κ, the public and the secret keys are generated and distributed to the players. This can be accomplished with a dealer or jointly by the players.

At the start of a time period, an update protocol is executed among any subset of k_u non-corrupted players. The protocol modifies the secret key for the signature scheme. After the update protocol is executed, each non-corrupted player (whether part of the subset actively taking part in the update protocol or not) will have a share of the new secret for that time period.

To generate signatures, a subset of k_s players executes the signing protocol, which generates a signature for a message m using the secret key of the current time period. The players which sign can be any subset from the set of players not corrupted by the adversary since the beginning of the previous update period. The signature is a pair consisting of the current time period and a tag. Assuming that all players behave honestly, the signature will be accepted as valid by the verification algorithm.

Verification works the same as in a normal digital signature scheme. The verifying algorithm can be executed by any individual who possesses the public key. It returns either "Accept" or "Reject" to specify whether a particular signature is valid for a given message. We say that $\langle j, tag \rangle$ is a *valid* signature of m during time period j if performing the verification algorithm on the message-signature pair returns "Accept."

Furthermore, in a forward-secure threshold signature scheme, if an adversary learns more than t shares of the secret signing key for a particular time period γ, it should be computationally infeasible for her to generate a signature $\langle j, tag \rangle$ for any message m such that $\mathsf{verify}_{PK}(m, \langle j, tag \rangle) = 1$ and $j < \gamma$, where verify is the scheme's verification algorithm. That is, the adversary should not gain the ability to generate signatures for time periods prior to the time the secret key is compromised. Forward-secure schemes require that the secret key from the previous time period be deleted from the user's machine as part of the update protocol. Otherwise, an adversary who breaks into the user's machine will learn signing keys from earlier time periods, and hence have the ability to generate signatures for earlier time periods.

Formalizing The Security of Forward-Secure Threshold Schemes. Below, we formalize the property of forward security in terms of threshold signature schemes. The security properties we desire for such a scheme are two-fold. First, as in any other threshold scheme, no adversary with access to t or fewer shares of the secret key should be able to forge signatures. Second, in order for the scheme to be forward-secure, no adversary who gains $t + 1$ or more shares of the secret in a particular time period should be able to generate signatures for time periods *earlier* than that one. Our notion of security, given below, addresses forward security directly *and* captures threshold security as a special case.

An adversary against a forward-secure threshold signature scheme $\mathsf{KETS} = (\mathsf{KETS.keygen}, \mathsf{KETS.update}, \mathsf{KETS.sign}, \mathsf{KETS.verify})$ functions in three stages: the chosen message attack phase (denoted cma), the over-threshold phase (denoted $\mathsf{overthreshold}$), and the forgery phase (denoted forge).

In the chosen message attack phase, the adversary submits queries to the $\mathsf{KETS.sign}$ protocol on messages of her choice. She is also allowed oracle access to H, the public hash function used in the $\mathsf{KETS.sign}$ protocol. During this phase, she may be breaking into servers and learning shares of the secret, but we assume that no more than t of them are compromised during any one time period. Note that if a player is corrupted during the update protocol, we consider that player to be compromised in *both* the current time period and the immediately preceding one. This is a standard assumption in threshold schemes, since the secret information a player holds during the update protocol includes the secrets of both of the time periods.

In the over-threshold phase, for a particular time period b, the adversary may learn shares of the secret key for a set of players of size $t + 1$ or greater. This allows the adversary to compute the secret key. For simplicity in the simulation, we give the adversary the entire current state of the system (e.g. actual secret key and all shares of the key during this phase). If the adversary selects b to be a time period *after* the very last one, the secret key is defined to be an empty string, so the adversary learns no secret information.

In the forgery phase, the adversary outputs a message-signature pair $(M, \langle k, tag \rangle)$ for some message M and time period k. We consider an adversary *successful* if M was not asked as a query in the chosen message attack phase for time k and *either* of the following holds: (1) her output is accepted by $\mathsf{KETS.verify}$, and k is earlier than the time period b in which the adversary entered the over-threshold phase; (2) she is able to output a message-signature pair accepted by $\mathsf{KETS.verify}$ without compromising more than t players.

NOTATION. There are n players in our protocols, and the total number of time periods is denoted by T. The overall public key is denoted PK, and is comprised of l values, denoted $U_1, \ldots, U_l$. In each time period j, the corresponding l components of the secret key, denoted by $S_{1,j}, \ldots, S_{l,j}$, are shared among all players. The share of the i-th secret key value $S_{i,j}$ for time period j held by player ρ is denoted $S_{i,j}^{(\rho)}$ and the overall secret information held by player ρ in that time period (all l values) is denoted $SK_j^{(\rho)}$. In general, the notation $X^{(\rho)}$ indicates the share of X held by player ρ.

3 Forward security based on multiplicative secret sharing

Here, we introduce a simple $(t, t+1, t+1, t+1)$-threshold scheme, which is based on multiplicative sharing [10, 11]. It is forward-secure against eavesdropping adversaries. When sharing a value X multiplicatively, each player ρ holds a random share $X^{(\rho)}$ subject to $X \equiv X^{(1)} X^{(2)} \cdots X^{(n)} \pmod N$, for a given modulus N. The main advantage of this scheme is that no information about the secret is compromised, even in the presence of up to $n-1$ corrupted players (out of n total players). A disadvantage of the scheme, on the other hand, is that n honest players are required to participate in the signing and the refreshing protocols.

3.1 Construction

In Figure 2, we give a version of the scheme that can handle (static and) adaptive eavesdropping adversaries. Here, we point out the interaction of players required in each portion of the scheme. The key generation protocol is executed by a trusted dealer, who generates and sends a share of the initial secret key to each player. Key evolution is executed by each player individually; it does not require any player interaction. Signing, as mentioned earlier, requires the participation of (and interaction between) all n players. Finally, verification of a signature may be performed by any party in possession of the public key. No interaction of parties is required.

The scheme described in Figure 2 is secure against adaptive eavesdropping adversaries (although we do not present the proof here). To deal with mobile eavesdropping adversaries, we simply add a refresh protocol that is executed at the end of every refreshing period (which may or may not coincide with the key evolution). This renders any knowledge about the shares that an adversary may have gained prior to the execution of the refresh protocol useless, and thus, makes the scheme proactive. To accomplish the refreshing of shares, each player distributes a sharing of 1 and then multiplies its current share by the product of all shares received during the refreshment phase (including the share it generated for itself).

REFRESH. Each player i participates in the refresh protocol by picking n random numbers $x_1^{(i)}, \ldots, x_n^{(i)}$ such that $\prod_{j=1}^{n} x_j^{(i)} \equiv 1 \pmod N$. Then, for each j between 1 and n, it sends the value $x_j^{(i)}$ to player j through a private channel. Once a player j receives these values from all other players, it computes its share of the new secret by multiplying its current share by $\prod_{i=1}^{n} x_j^{(i)}$.

`protocol` MFST-SIG.keygen(κ, T)

1) Dealer picks random, distinct $k/2$-bit primes p, q, each congruent to 3 mod 4
2) Dealer sets $N \leftarrow pq$
3) `for` $i = 1, \ldots, l$ `do`
   a) `for` $\rho = 1, \ldots, n$ `do`
      Dealer sets $S_{i,0}^{(\rho)} \stackrel{R}{\leftarrow} Z_N^*$
   b) Dealer sets $S_{i,0} \leftarrow \prod_{\rho=1}^{n} S_{i,0}^{(\rho)}$
   c) Dealer sets $U_i \leftarrow {S_{i,0}}^{2^{(T+1)}}$
4) `for` $\rho = 1, \ldots, n$ `do`
   a) Dealer sets $SK_0^{(\rho)} \leftarrow (N, T, 0, S_{1,0}^{(\rho)}, \ldots, S_{l,0}^{(\rho)})$
   b) Dealer sends $SK_0^{(\rho)}$ to player ρ
5) Dealer sets $PK \leftarrow (N, T, U_1, \ldots, U_l)$ and publishes PK.

`protocol` MFST-SIG.sign(m, j)

1) `for` $\rho = 1, \ldots, n$ `do`
   a) Player ρ sets $R^{(\rho)} \stackrel{R}{\leftarrow} Z_N^*$
   b) Player ρ computes $Y^{(\rho)} \leftarrow (R^{(\rho)})^{2^{T+1-j}}$ and broadcasts it.
2) All players individually:
   a) Compute $Y \leftarrow Y^{(1)} Y^{(2)} \cdots Y^{(n)}$
   b) Compute $c_1 \ldots c_l \leftarrow H(j, Y, m)$
3) `for` $\rho = 1, \ldots, n$ `do`
   a) Player ρ computes $Z^{(\rho)} \leftarrow R^{(\rho)} \prod_{i=1}^{l} (S_{i,j}^{(\rho)})^{c_i}$ and broadcasts it.
4) All players compute $Z \leftarrow Z^{(1)} Z^{(2)} \cdots Z^{(n)}$
5) The signature of m is set to $\langle j, (Y, Z) \rangle$, and is made public.

`algorithm` MFST-SIG.verify$_{PK}(m, \sigma)$

1) Parse σ as $\langle j, (Y, Z) \rangle$.
2) `if` $Y \equiv 0$, `then return` 0.
3) $c_1 \ldots c_l \leftarrow H(j, Y, m)$
4) `if` $Z^{2^{(T+1-j)}} \equiv Y \cdot \prod_{i=1}^{l} U_i^{c_i}$,
   `then return` 1
   `else return` 0

`protocol` MFST-SIG.update(j)

1) `if` $j = T$, `then return` the empty string. Otherwise, proceed.
2) `for` $\rho = 1, \ldots, n$ `do`
   a) `for` $i = 1, \ldots, l$ `do`
      Each player ρ computes $S_{i,j}^{(\rho)} \leftarrow (S_{i,j-1}^{(\rho)})^2$.
   b) Each player ρ deletes $SK_{j-1}^{(\rho)}$ from his machine.

Figure 2: *Our threshold signature scheme forward-secure against adaptive eavesdropping adversaries. The scheme is based on multiplicative secret sharing. With the addition of the refresh protocol given in Section 3.1, it becomes secure against mobile eavesdropping adversaries. All computation other than the generation of N is performed modulo N.*

3.2 Security

The correctness of the construction of our MFST-SIG scheme follows from the underlying Bellare-Miner scheme. Furthermore, the threshold parameter values can be easily verified. Below, we state a theorem which relates the forward security of this construction to that of the underlying signature scheme given in [3]. The proof can be found in Appendix B.

Theorem 3.1 Let MFST-SIG be our $(t, t+1, t+1, t+1)$-threshold digital signature scheme making use of the refresh protocol given in Section 3.1. Let FS-SIG be the single-user digital signature scheme given by Bellare and Miner [3]. Then, MFST-SIG is a forward-secure threshold digital signature scheme in the presence of *mobile eavesdropping* adversaries as long as FS-SIG is a forward-secure signature scheme in the standard (single-user) sense.

4 Forward security based on polynomial secret sharing

In this section, we present PFST-SIG, our $(t, 2t+1, 2t+1, 3t+1)$-threshold scheme based on polynomial secret sharing, forward-secure against mobile halting adversaries. Its main advantage is that it does not require the presence of all players during signing or key update; only about two thirds of the players are needed in any of these cases. This, however, comes at the cost of more interaction among the players and a lower threshold in the total number of faults that we can tolerate in comparison to the scheme in Section 3. Its construction is shown in Figure 3 and relies on several standard building blocks tailored for our purposes. These tools are described in Section 4.2. Finally, Section 4.3 gives details about the security of our scheme.

4.1 Construction

The key generation protocol is executed by a trusted dealer, who shares the secret key among all n participants using a modified version of Shamir's secret sharing as described in Section 4.2. Each player's share of the base key $SK_0^{(\rho)}$ includes each of his shares of the $S_{i,0}$ values (there are l of them), so player ρ's secret key is then $(N, T, 0, S_{1,0}^{(\rho)}, \ldots, S_{l,0}^{(\rho)})$.

At the beginning of each time period, the evolution of the secret key is accomplished via the key update protocol in which exactly $2t+1$ players must participate. We call these $2t+1$ players the *active* players. (Note the difference from our earlier scheme, which uses multiplicative-sharing and needs *all* players to participate.) At the start of the protocol in time period j, each player who participated in the previous update protocol has $SK_{j-1}^{(\rho)}$, i.e. his share of the previous time period's secret. The new secret key is computed by squaring the l values in the previous secret key. The players compute this new secret key using the `Modified-Mult-SS` protocol (as described in Section 4.2) l times. At the end of the protocol, player ρ holds $SK_j^{(\rho)}$, and each player immediately deletes his share of the secret key from the previous time period. It is important to note that all "un-halted" players, *including those who had been halted by the adversary during the previous update protocol,* will now be given a share of the new secret.

Like the update protocol, signing does not require participation by all of the n players– only $2t+1$ active players are required. Because it is the threshold version of Bellare-Miner [3], this protocol is based on a commit-challenge-response framework, but the various steps are accomplished by the group using subprotocols described in Section 4.2. In order to distribute the Bellare-Miner scheme [3] across many players, we made one important modification to the underlying signature protocol, which we highlight here. In the Bellare-Miner scheme, R is a random element in Z_N^*, while here R is a random value in Z_N. As explained in Section 4.2, `protocol` PFST-SIG.keygen(κ, T)

1) Dealer picks random, distinct $k/2$-bit primes p, q, each congruent to 3 mod 4
2) Dealer sets $N \leftarrow pq$
3) **for** $i = 1, \ldots, l$ **do**
   a) Dealer sets $S_{i,0} \stackrel{R}{\leftarrow} Z_N^*$
   b) Dealer sets $U_i \leftarrow {S_{i,0}}^{2^{(T+1)}}$
   c) Dealer uses `Shamir-SS` over Z_N to create shares $S_{i,0}^{(1)}, \ldots, S_{i,0}^{(n)}$ of $S_{i,0}$.
4) **for** $\rho = 1, \ldots, n$ **do**
   a) Dealer sets $SK_0^{(\rho)} \leftarrow (N, T, 0, S_{1,0}^{(\rho)}, \ldots, S_{l,0}^{(\rho)})$
   b) Dealer sends $SK_0^{(\rho)}$ to player ρ
5) Dealer sets $PK \leftarrow (N, T, U_1, \ldots, U_l)$ and publishes PK.

`protocol` PFST-SIG.sign(m, j)

1) Using `Joint-Shamir-RSS` , players generate random value $R \in Z_N$ so that player ρ gets share $R^{(\rho)}$ of R.
2) Players compute $Y \leftarrow R^{2^{(T+1-j)}}$ using `Modified-Mult-SS` and their shares of R.
3) Each player ρ computes $c_1 \ldots c_l \leftarrow H(j, Y, m)$.
4) Each player ρ executes $Z^{(\rho)} \leftarrow R^{(\rho)}$, so that Z is initialized to R.
5) **for** $i = 1, \ldots, l$ **do**
   a) **if** $c_i = 1$, **then** players compute $Z \leftarrow Z \cdot S_{i,j}$ using `Modified-Mult-SS` .
6) The signature of m is set to $\langle j, (Y, Z)\rangle$, and is made public.

`algorithm` PFST-SIG.verify$_{PK}(m, \sigma)$

1) Parse σ as $\langle j, (Y, Z)\rangle)$.
2) **if** $Y \equiv 0$, **then return** 0.
3) $c_1 \ldots c_l \leftarrow H(j, Y, m)$
4) **if** $Z^{2^{(T+1-j)}} \equiv Y \cdot \prod_{i=1}^{l} U_i^{c_i}$,
   then return 1
   else return 0

`protocol` PFST-SIG.update(j)

1) **if** $j = T$, **then return** the empty string. Otherwise, proceed.
2) Players compute updated secret key shares $S_{1,j}, \ldots, S_{l,j}$ by squaring the previous values $S_{1,j-1}, \ldots, S_{l,j-1}$ using `Modified-Mult-SS` .
3) Each player ρ deletes $SK_{j-1}^{(\rho)}$.

Figure 3: *Our threshold signature scheme based on polynomial secret sharing is forward-secure against halting adversaries. All computation other than the generation of N is performed mod N.*

the signature generated by the signing algorithm is still valid. The verification portion of our scheme is identical to that of the Bellare-Miner scheme, because verification is an algorithm which requires only one party.

4.2 Building Blocks

Shamir-SS . Shamir's standard secret sharing protocol [23] operates over a finite field. A *dealer* chooses a secret value a_0 and a random polynomial $p(x)$ of degree k whose coefficients are denoted a_0 to a_k. He then sets the coefficient of the constant term to be the secret a_0 and sends to a *shareholder* i the value of $p(i)$. The proof of the privacy of this scheme is typically based on the fact that the computations are performed over a finite field. However, the computations in our scheme are performed over Z_N, which is not a field. Nevertheless, we can still guarantee that the system has a unique solution over Z_N by ensuring that the determinant of the Vandermonde matrix is relatively prime to N, and therefore, the matrix is invertible modulo N.

First, we require that the number of players in the protocol must be less than both p and q. Second, the share of the protocol given to player i must be $f(i)$. This way, none of the x_i's in the shares used to reconstruct contain a factor of p or q. Next, we recognize that all elements in the $k+1 \times k$ Vandermonde matrix are relatively prime to N since none of them contains a factor of p or q. Finally, the determinant of the Vandermonde matrix is given by $\prod_{1\le j<k\le k+1}(x_{i_k}-x_{i_j}) \bmod N$, and therefore the determinant must be relatively prime to N. Note that a similar approach has been taken by Shoup [24] when sharing an RSA key over $Z_{\Phi(N)}$.

Modified-Mult-SS . In our scheme, we need the ability to jointly multiply two distributed secrets. We use such a protocol in several places in our scheme, namely, during the generation of signatures and also during the updates of the secret key.

We formulate the problem as follows: two secrets α and β are shared among n players via degree-t polynomials $f_\alpha(x)$ and $f_\beta(x)$, respectively, so that $f_\alpha(0)=\alpha$ and $f_\beta(0)=\beta$. The goal is for the players to jointly compute a sharing of a new polynomial G, such that $G(0)=\alpha\beta$. Several previous works have addressed this problem, starting with the observation by [5] that simple multiplication by player P_i of his individual secrets $f_\alpha(i)$ and $f_\beta(i)$ determines a non-random polynomial with degree $2t$. We describe a modified version of a protocol proposed in [16], which describes a step accomplishing degree-reduction and randomization in a model with only eavesdropping adversaries.[2] In contrast, our model allows halting adversaries.

The degree reduction and randomization step in [16] assumes that the $2t+1$ participating players are those with indices $1,2,...,2t+1$, and therefore make use of precomputed constants in this step. However, in our model, the adversary may arbitrarily choose which players to halt, so we cannot assume that the participants are a particular subset of players. Instead, during the run of the protocol, we can jointly determine which players are available to participate. To do this, every player P_i who is functioning *and* was not halted during the most recent update phase will broadcast an "I'm alive" message. From the set of those that respond, we will select the players with the $2t+1$ smallest indices to actually perform the computation. Then, the constants corresponding to that subset of players can be computed efficiently, in time $O(2t+1)$.

We point out that, if at any time during the execution of the `Modified-Mult-SS` protocol, a participating player is halted by the adversary, this will be noticed by at least one other participant, and the protocol can be aborted and restarted with a different subset of (currently) participating players. Furthermore, the multiplication protocol will never need to be restarted more than t times, due to the bound on the number of players the adversary can halt during one time period. In addition, in the case of a `Modified-Mult-SS` restart, we stress that the entire update or signature protocol which is using the `Modified-Mult-SS` protocol *need not* be restarted. This is true because at each multiplication step of these protocols, we ensure that all n players are sent shares of the input of the next step. That is, when a new set of $2t+1$ players is selected during the restart of the multiplication protocol, we are guaranteed to find a sufficient set of players which possess the required input information for the multiplication. In particular, every player, whether active in the key update protocol or not, will be sent enough information to allow it to compute its share of the new secret.

Joint-Shamir-RSS . Standard joint-random secret-sharing protocols such as that proposed in [18] and [15] allow a group of players to jointly generate a secret without a trusted dealer. In the instantiation used in our scheme, each participant chooses a random secret and a polynomial to share the secret as in Shamir's secret sharing scheme. Each participant then plays the role of a dealer by distributing its secret using Shamir's secret sharing scheme. The jointly defined secret value is then the sum of the secrets of all participants.[3] Furthermore, we require that the

[2]A second protocol is given in [16] which requires players to commit to their input shares, so that it tolerates even malicious adversaries. In our model, however, we do not need this functionality, so we have modified their simpler protocol to meet our needs.

[3]Note that this scheme is secure for our purpose since only halting adversaries are allowed. It is not secure, however, under attacks by malicious adversaries as pointed out in [15].

shares from player P_i be dealt out in one broadcast message, with the share for each player P_j encrypted under P_j's public key. This ensures an "atomic" sharing, so that, regardless of when the adversary chooses to halt players, all players receive shares from the same subset of players. If no such message is broadcast from a particular player P_j, he is assumed to be halted, and the sum of shares for any individual player will clearly not include a share from P_j.

Our scheme requires that the jointly created random value R belongs in Z_N^*, but clearly, this protocol does not provide such a guarantee. However, the probability that R, which is known to be in Z_N, is *not in* Z_N^* is negligible. Specifically, the numbers in Z_N which are not in Z_N^* are precisely those numbers which are multiples of p and q. There are approximately $p+q$ of these, out of a total of pq values in Z_N. Therefore, the probability of finding an R which is in Z_N but not Z_N^* is approximately $\frac{1}{q} + \frac{1}{p}$, a negligible probability.

4.3 Security

In this section, we give several statements about the security of our PFST-SIG scheme. Proofs of the statements are omitted here and can be found in Appendix B. First, Lemma 4.1 demonstrates the correctness of our construction. Then, Lemma 4.2 states the threshold-related parameters of our scheme. Finally, Theorem 4.3 relates the forward security of our construction to that of underlying signature scheme given in [3]. It shows that, as long as we believe that the Bellare-Miner scheme is secure, any adversary working against our scheme would have only a negligible probability of success forging a signature with respect to some time period prior to that in which it gets the secret key.

Lemma 4.1 Let $PK = (N, T, U_1, \ldots, U_l)$ and $SK_0^{(j)} = (N, T, 0, S_{1,0}^{(j)}, \ldots, S_{l,0}^{(j)})$ $(j = 1, \ldots, n)$ be the public key and player j's secret key generated by PFST-SIG.keygen, respectively. Let $\langle j, (Y, Z)\rangle$ be a signature generated by PFST-SIG.sign on input m when all n players participated in the distributed protocol. Then PFST-SIG.verify$_{PK}(m, \langle j, (Y, Z)\rangle) = 1$

Lemma 4.2 PFST-SIG is a *key-evolving* (t, k_s, k_u, n)*-threshold digital signature scheme* for $n = 3t+1$, $k_s = 2t+1$, $k_u = 2t+1$. That is, it tolerates up to t halting faults when the total number of players $n = 3t+1$, requires the involvement of $2t+1$ players to evolve the secret key, and requires the involvement of $2t+1$ players to generate a valid signature.

Theorem 4.3 Let PFST-SIG be our key-evolving $(t, 2t+1, 2t+1, 3t+1)$-threshold digital signature scheme and let FS-SIG be the (single-user) digital signature scheme given in [3]. Then, PFST-SIG is a forward-secure threshold digital signature scheme in the presence of *halting* adversaries as long as FS-SIG is a forward-secure signature scheme in the standard (single-user) sense.

5 Discussion

COST ANALYSIS AND COMPARISONS. Distributed computation can be somewhat costly, but our signature schemes are quite efficient compared to the forward-secure single-user scheme of [3]. For example, in the multiplicative-sharing based scheme, the only added cost for the key generation protocol, which uses a trusted dealer, is the actual sharing of the secret. The update protocol is also very efficient, requiring l local multiplications and *no interactions*. Finally, the signing protocol requires only one round of interaction.

It is clear that our multiplicative-sharing-based scheme is very simple, efficient, and highly resilient, i.e. it can protect the secret even in the presence of up to $n-1$ corrupted players where n is the number of players. Furthermore, the costs of signing and updating are very low. The price of this simplicity and low overhead, however, is that the scheme can only cope with eavesdropping adversaries.

In contrast, the proposed scheme based on polynomial secret sharing can tolerate the more powerful halting adversaries, although it can tolerate fewer of them. It is also not as efficient as the multiplicative-sharing-based scheme. We can improve the performance of this scheme, however, by speeding up the communication required in multiplication. In particular, during the update, we can perform all l computations in parallel, and thus, use only one instantiation of the multiplication protocol. Signing can also be expedited by moving some of the computation off-line. Specifically, since the generation of the random value R and the computation of the commitment Y do not depend on the message or the current time period, they can be precomputed. This is a significant improvement, since the computation of Y is costly, given its $\frac{(T+1)}{2}$ squarings on average. With this optimization, the on-line signing costs of our new threshold scheme are the same as those in [3]. We can improve upon this slightly, by multiplying pairs of numbers together, and using their product as input into the next round of multiplication. In this way, on average we still perform $\frac{l}{2}$ multiplications, but only use $\lg \frac{l}{2}$ rounds of communication among players. The verification costs of our two schemes and the base scheme are identical, since the verifying algorithm is the same in all cases.

In terms of space efficiency, the sizes of the public keys in our two schemes are identical to that of the Bellare-Miner scheme. It is not surprising that our schemes require a larger amount of secret key memory overall, since the secret is distributed among a group of players. However, the secret key memory required per player is the same in both our schemes and the base scheme.

Interestingly, in our schemes, the size of the actual secret (as opposed to the size of the set of *shares* of the secret) is not any larger than that of the base scheme. This indicates that actual storage space required for players' shares of the secret in our schemes is the same as that required for the related threshold schemes without forward security. Therefore, with these improvements, adding forward security to the schemes imposes no additional online costs.

Adding robustness. In order to make our polynomial-secret-sharing based threshold scheme resilient to malicious adversaries, we need to make quite a few changes to it. We list the most significant ones here.

First, all secret sharings would need to be verifiable so that malicious behavior can be detected. For that purpose, we can use Pedersen's Verifiable Secret Sharing, `Pedersen-VSS`, protocol [22] in place of the `Shamir-SS` protocol. This protocol has the advantage of being information-theoretically secure in terms of privacy. The `Pedersen-VSS` protocol works as follows. Let $z \in Z_{q'}$ be the secret value being shared, where q' is a large prime. Let p' be a large prime such that q' divides $p'-1$, let $g \in Z_{p'}^*$ be an element of order q', and let h be a random element generated by g. In order to share z, we first generate two random polynomials $f(x) = a_0 + a_1 x + \ldots + a_t x^t$ and $f'(x) = a'_0 + a'_1 x + \ldots + a'_t x^t$ over $Z_{q'}$ with $a_0 = z$, then commit to the coefficients by broadcasting the values $C_\alpha = g^{a_\alpha} h^{a'_\alpha} \bmod p'$ for $\alpha = 0, \ldots, t$, and then give to player i the shares $z^{(i)} = f(i)$ and ${z'}^{(i)} = f'(i)$. In our case, however, we need $q' = N$, where N is the product of two unknown large primes. Fortunately, as in the case of the modified `Shamir-SS` protocol, this is not a problem since Z_N is an excellent approximation of a field.

Second, the `Joint-Shamir-RSS` protocol would also have to be replaced by a robust version. For that purpose, we can use the Joint-Pedersen Verifiable Secret Sharing, `Joint-Pedersen-VSS`, protocol [8, 15, 21]. This protocol works as follows. In a first step, each player i commits to a random value in $z_i \in Z_q'$ using `Pedersen-VSS` protocol. Then, in a second step, each player verifies the commitments sent by all other players and builds a set of good players containing all those players whose commitments passed the test. It can be shown [8, 15] that all good players will agree on the same set, which we call $QUAL$. At a last step, all players i in $QUAL$ will compute their shares $x^{(i)}$ of the common shared secret $x = \sum_{i \in QUAL} z_i \bmod q'$ by using the secret information they received in the first step. More specifically, if ${z_i}^{(j)}$ is the share of z_i received by player j from player i, then $x^{(i)} = \sum_{j \in QUAL} {z_i}^{(j)} \bmod q'$. Again, the only modification we would need to make in our case is to have $q' = N$.

Lastly, we need to modify the `Modified-Mult-SS` protocol so that it works even in the presence of malicious adversaries. In order to do so, we suggest a variant of the robust multiplication protocol given in [16], which we call `Robust-Mult-SS` , in which secrets are shared using the `Pedersen-VSS` protocol. This variant is very similar in spirit to the multiplication protocol given in [8]. Let α and β be the two shared secret values being multiplied. Let $\alpha^{(i)}$ and $\beta^{(i)}$ be the shares of α and β held by player i respectively. The protocol works as follows. First, each player i commits to the value $\alpha^{(i)}\beta^{(i)}$ by using the `Pedersen-VSS` protocol and then proves in Zero Knowledge (ZK) that the value it has just committed is the correct one. Examples of such proofs can be found in [9, 16]. Having done that, then each player can compute locally their shares of the new secret $\alpha\beta$ by computing a linear combination of correct shares it has received. As in the modified version of the multiplication protocol presented in the previous section, the exact values of the coefficients for the linear combination will depend on which set of qualified players is considered in the computation. Like in previous cases, we would have to use $q' = N$ in the `Pedersen-VSS` protocol. Clearly, this protocol involves a lot of interaction due to the ZK proofs contained in it. To avoid that, one can resort to solutions for robust multiplication like the one presented in [1] which avoid the use of ZK proofs altogether.

Acknowledgments

The authors would like to thank Mihir Bellare for his advice and comments on earlier drafts of this paper. Michel Abdalla was supported in part by CAPES under Grant BEX3019/95-2. Sara Miner and Chanatip Namprempre were supported in part by Mihir Bellare's 1996 Packard Foundation Fellowship in Science and Engineering and NSF CAREER Award CCR-9624439.

References

[1] M. Abe. Robust distributed multiplication without interaction. In M. J. Wiener, editor, *Advances in Cryptology – CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 130–147, Santa Barbara, CA, USA, Aug. 15–19, 1999. Springer-Verlag, Berlin, Germany.

[2] R. Anderson. Two remarks on public-key cryptology. Manuscript. Relevant material presented by the author in an invited lecture at the ACM CCS 97: 4th Conference on Computer and Communications Security, Zurich, Switzerland, Apr. 1–4, 1997, Sept. 2000.

[3] M. Bellare and S. Miner. A forward-secure digital signature scheme. In M. J. Wiener, editor, *Advances in Cryptology – CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 431–448, Santa Barbara, CA, USA, Aug. 15–19, 1999. Springer-Verlag, Berlin, Germany.

[4] M. Bellare and B. Yee. Forward security in private key cryptography. In M. Joye, editor, *Topics in Cryptology – CT-RSA 2003*, volume 2612 of *Lecture Notes in Computer Science*, pages 1–18, San Francisco, CA, USA, Apr. 13–17, 2003. Springer-Verlag, Berlin, Germany.

[5] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for noncryptographic fault-tolerant distributed computations. In *20th Annual ACM Symposium on Theory of Computing*, pages 1–10, Chicago, Illinois, USA, May 2–4, 1988. ACM Press.

[6] D. Boneh and M. K. Franklin. Efficient generation of shared RSA keys. In B. S. K. Jr., editor, *Advances in Cryptology – CRYPTO'97*, volume 1294 of *Lecture Notes in Computer Science*, pages 425–439, Santa Barbara, CA, USA, Aug. 17–21, 1997. Springer-Verlag, Berlin, Germany.

[7] M. Burmester, Y. Desmedt, and J. Seberry. Equitable key escrow with limited time span (or how to enforce time expiration cryptographically). In K. Ohta and D. Pei, editors, *Advances in Cryptology – ASIACRYPT'98*, volume 1514 of *Lecture Notes in Computer Science*, pages 380–391, Beijing, China, Oct. 18–22, 1998. Springer-Verlag, Berlin, Germany.

[8] R. Canetti, R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Adaptive security for threshold cryptosystems. In M. J. Wiener, editor, *Advances in Cryptology – CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 98–115, Santa Barbara, CA, USA, Aug. 15–19, 1999. Springer-Verlag, Berlin, Germany.

[9] R. Cramer and I. B. Damgård. Zero-knowledge proofs for finite field arithmetic, or: Can zero-knowledge be for free? In H. Krawczyk, editor, *Advances in Cryptology – CRYPTO'98*, volume 1462 of *Lecture Notes in Computer Science*, pages 424–441, Santa Barbara, CA, USA, Aug. 23–27, 1998. Springer-Verlag, Berlin, Germany.

[10] Y. Desmedt. Threshold cryptosystems. In J. Seberry and J. Pieprzyk, editors, *Advances in Cryptology – AUSCRYPT ' 92*, volume 718 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Germany, 1993.

[11] Y. Desmedt, G. Di Crescenzo, and M. Burmester. Multiplicative non-abelian sharing schemes and their application to threshold cryptography. In J. Pieprzyk and R. Safavi-Naini, editors, *Advances in Cryptology – ASIACRYPT'94*, volume 917 of *Lecture Notes in Computer Science*, Wollongong, Australia, Nov. 28 – Dec. 1, 1994. Springer-Verlag, Berlin, Germany.

[12] Y. Desmedt and Y. Frankel. Shared generation of authenticators and signatures. In J. Feigenbaum, editor, *Advances in Cryptology – CRYPTO'91*, volume 576 of *Lecture Notes in Computer Science*, pages 457–469, Santa Barbara, CA, USA, Aug. 11–15, 1991. Springer-Verlag, Berlin, Germany.

[13] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In A. M. Odlyzko, editor, *Advances in Cryptology – CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194, Santa Barbara, CA, USA, Aug. 1987. Springer-Verlag, Berlin, Germany.

[14] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust threshold DSS signatures. In U. M. Maurer, editor, *Advances in Cryptology – EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 354–371, Saragossa, Spain, May 12–16, 1996. Springer-Verlag, Berlin, Germany.

[15] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. In J. Stern, editor, *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 295–310, Prague, Czech Republic, May 2–6, 1999. Springer-Verlag, Berlin, Germany.

[16] R. Gennaro, M. O. Rabin, and T. Rabin. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In *17th ACM Symposium Annual on Principles of Distributed Computing*, pages 101–111, Puerto Vallarta, Mexico, June 28 – July 2, 1998.

[17] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung. Proactive secret sharing or: How to cope with perpetual leakage. In D. Coppersmith, editor, *Advances in Cryptology – CRYPTO'95*, volume 963 of *Lecture Notes in Computer Science*, pages 339–352, Santa Barbara, CA, USA, Aug. 27–31, 1995. Springer-Verlag, Berlin, Germany.

[18] I. Ingemarsson and G. J. Simmons. A protocol to set up shared secret schemes without the assistance of a mutually trusted party. In I. Damgård, editor, *Advances in Cryptology – EUROCRYPT'90*, volume 473 of *Lecture Notes in Computer Science*, pages 266–282, Aarhus, Denmark, May 21–24, 1990. Springer-Verlag, Berlin, Germany.

[19] H. Krawczyk. Simple forward-secure signatures from any signature scheme. In *ACM CCS 00: 7th Conference on Computer and Communications Security*, pages 108–115, Athens, Greece, Nov. 1–4, 2000. ACM Press.

[20] H. Ong and C.-P. Schnorr. Fast signature generation with a Fiat Shamir–like scheme. In I. Damgård, editor, *Advances in Cryptology – EUROCRYPT'90*, volume 473 of *Lecture Notes in Computer Science*, pages 432–440, Aarhus, Denmark, May 21–24, 1990. Springer-Verlag, Berlin, Germany.

[21] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In J. Feigenbaum, editor, *Advances in Cryptology – CRYPTO'91*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140, Santa Barbara, CA, USA, Aug. 11–15, 1991. Springer-Verlag, Berlin, Germany.

[22] T. P. Pedersen. A threshold cryptosystem without a trusted party. In D. W. Davies, editor, *Advances in Cryptology – EUROCRYPT'91*, volume 547 of *Lecture Notes in Computer Science*, pages 522–526, Brighton, UK, Apr. 8–11, 1991. Springer-Verlag, Berlin, Germany.

[23] A. Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, Nov. 1979.

[24] V. Shoup. Practical threshold signatures. In B. Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 207–220, Bruges, Belgium, May 14–18, 2000. Springer-Verlag, Berlin, Germany.

A Adding forward security to any threshold signature scheme

In this section, we present several ways one could add forward security to any existing threshold signature scheme. All the solutions presented here will have at least one of its parameters (i.e. public key, secret key, signature length, or memory size) growing linearly with the total number of time periods T. Except for the last one, all of methods described here follow ideas given in [3].

Long secret and public keys. In the first method, we have separate secret and public keys for each time period. We achieve this by running the key generation protocol of the underlying threshold scheme T times, once for each time period. The secret share of each player is then the collection of the shares for all T time periods. Likewise, the public key is the collection of the the public keys of each period. Signature generation and verification are done as in the underlying scheme, by using keys for the current time period to sign and verify. The only difference is that we attach to the original signature an index j indicating the time period in which the signature was generated. The update protocol deletes those shares which are no longer needed. As one can see, this clearly achieves forward security, but at the cost of very long secret and public keys. We note that proactivity can be easily added in this case by updating the shares of each player at the beginning of each time period.

Reducing the public key length. The second method is based on an idea suggested by Anderson [2], which trades public key length for storage space. As before, we create secret and public key pairs for each time period, but we think of these public keys as verification keys, since they will not actually be included in the scheme's public key. We also generate an additional pair (sk, pk) which we use to certify each of the verification keys. We then delete this additional secret key sk and set pk to be the public key of the scheme. The secret shares of each player are the same as in the previous method. Additionally, however, we need to store all the verification keys for each time period, and their respective certificates. These will be attached to the signature, so any verifying party can check the validity of the the verification key (using pk and the certificate) as well as the validity of the signature itself (using the verification key). As before, the update protocol only involves deleting the secret shares of old time periods. Proactivity can be added to this scheme in the same way suggested for the previous method.

Reducing the secret key length. A slight but significant variation to the previous method is proposed in [19]. It assumes a base threshold scheme with distributed key generation protocol in order to avoid the need for a trusted third party during a key update. Such protocols are known for both RSA [6] and discrete-log [15] based cryptosystems. The operation of the scheme is similar to the method we described immediately above, except that the coins used by each player in the key generation protocol are actually generated by means of a forward-secure pseudorandom number generator [4]. This type of generator differs from a standard one in that its seed is periodically updated and its output in previous stages are indistinguishable from random. The method which uses this type of generator works as follows.

Each player i will initially hold a random seed s_i for a forward-secure pseudorandom generator. This seed is then used to generate all the pseudorandom bits needed by i in its participation in the distributed key generation protocol in all T time periods. Then, as above, we run the key generation protocol of the base scheme for each time period with the difference that each player uses its own sequence of pseudorandom bits. We then generate an additional pair (sk, pk) of secret and public keys, use sk to create certificates for each of the public keys, delete sk, and set the public key to pk. The certificates are then stored and all the secret keys deleted. The initial secret share of player i is simply s_i. In the update protocol, each player i will run the forward-secure pseudorandom generator using its seed as input to obtain a new block of pseudorandom bits and a new seed. Then, all players engage in a key generation protocol using their new blocks of pseudorandom bits to obtain shares of the (recreated) secret key for the new period and the matching public key. Signing and verification work as in the previous method. One can easily see that this scheme achieves forward security against eavesdropping adversaries.

Sharing the seeds. Despite being very efficient both in terms of signing costs and key sizes, the previous method requires all the players to be present during a key update so that the exact same sequence of keys get generated. The absence of even a single player during a key update may be enough to obstruct its operation for that time period (since the key generation protocol most likely would not generate the secret-public key pair for which we have a certificate). To get around this constraint and thereby tolerate halting adversaries, we add one more level of secret sharing to the scheme. Let $s_{i,j}$ denote the seed held by player i in time period j. Now, in each time period j, each user would have a share of the seed $s_{i,j}$ held by player i (by working over a sufficiently large prime field). This way, if a player i becomes faulty in a time period j, thus not taking part in the key generation protocol for that period, then a threshold k of players can reconstruct the seed $s_{i,j}$ for that player, compute its sequence of pseudorandom bits, and then play its role in the the key generation protocol. As a result, only a threshold k of players is required to generate keys of signatures at any time. The main drawback of such an approach, which might be too severe in some cases, is that once a player becomes faulty, its secret is revealed to all other players. These players can then compute the sequence of pseudorandom bits for that player for all subsequent time periods.

B Proofs of Security

Proof of Lemma 4.1: In order to verify that $(m, \langle j, (Y, Z)\rangle)$ is a valid signature, we need to check whether $Z^{2^{(T+1-j)}} \equiv Y \prod_{i=1}^{l} U_i^{c_i} \pmod N$. From the description of the protocol, we know that R is a random element in Z_N, $Y \equiv R^{2^{(T+1-j)}} \pmod N$, $c_1 \ldots c_l = H(j, Y, m)$, and $Z \equiv R \prod_{i=1}^{l} S_{i,j}^{c_i} \pmod N$. Hence,

$$\begin{aligned} Z^{2^{(T+1-j)}} &\equiv (R \prod_{i=1}^{l} S_{i,j}^{c_i})^{2^{(T+1-j)}} \equiv R^{2^{(T+1-j)}} \prod_{i=1}^{l} S_{i,j}^{c_i 2^{(T+1-j)}} \\ &\equiv Y \prod_{i=1}^{l} S_{i,0}^{c_i 2^{(T+1)}} \equiv Y \prod_{i=1}^{l} U_i^{c_i} \pmod N , \end{aligned}$$

and $\mathsf{PFST\text{-}SIG.verify}_{PK}$ returns 1 on input $(m, \langle j, (Y, Z)\rangle)$ as desired. ▮

Proof of Lemma 4.2: The proof follows directly from the description of PFST-SIG and Lemma 4.1. ▮

Proof of Theorem 4.3: As the security of our threshold scheme is based on that for the forward security of the single-user scheme in [3], familiarization with their notion of security might be helpful in understanding our proof. We also use ideas from [14] regarding the simulation of the adversary's view of the protocol.

Let F be the adversary working against the security of our scheme. We want to construct an algorithm against the forward security of the underlying scheme using F as a subroutine. Following the model of [3], our algorithm runs in three phases: the chosen message attack phase, cma; the over-threshold phase, overthreshold; and the forgery phase, forge. It also has access to both a signing oracle, $\mathcal{O}$, and a hashing oracle, $\mathcal{H}$. Let the public key $PK = (N, T, U_1, \ldots, U_l)$ be the input of our algorithm during the cma phase. We then start running F in its cma phase, feeding it PK.

During the cma phase, F is allowed to make queries to a PFST-SIG.sign oracle and to a H oracle. Therefore, we need to simulate both of these oracles. In doing so, we have to simulate

F's view of the protocol. F is allowed to corrupt up to t players (of its choice) per time period in this phase, and it can corrupt by either simply eavesdropping or actually halting the player. Therefore, we need to be able to simulate the actions and views of the corrupted players.

At the beginning of each time period, F has the option of either staying in cma phase or switching to an overthreshold phase. If it chooses the first option, so does our algorithm. We only switch to a breakin phase when F switches to its overthreshold phase.

Distributing shares of the secret keys. Let $\mathcal{B}_j$ denote the set of corrupted players in the current time period j. We know that $|\mathcal{B}_j| \leq t$. We need to provide each player $b \in \mathcal{B}_j$ with a share of the current secret key $S_{i,j}$ for $i = 1, \ldots, l$. However, we do not know these secret values. Fortunately, we can get around this problem by simply picking a value for the share $S_{i,j}^{(b)}$ of $S_{i,j}$ at random from Z_N for each player $b \in \mathcal{B}_j$. We are allowed to do so because the sharing is information-theoretically secure, and all sets of t shares have the same probability. Moreover, because the `Modified-Mult-SS` protocol we use in the update protocol not only reduces the degree of the polynomial used to share the new secret key but also re-randomizes the shares, the values that we pick for the shares of the secret key of each corrupted player in different time periods are independent as long as at most t are corrupted. (Here we make use of our assumption that if a player is corrupted at the beginning of the update protocol, then it is corrupted in both the previous and current time periods.) Nevertheless, if we define t shares of $S_{i,j}$, then all other shares are implicitly defined (although we cannot compute them because we do not know the value of $S_{i,j}$ itself).

Simulating the signing oracle. Using our signing oracle $\mathcal{O}$, we can easily simulate F's signing oracle PFST-SIG.sign as well as F's view of the signing protocol. Let m be the message being queried to the signing oracle. We first query our oracle $\mathcal{O}$ for a signature $\langle j, (Y, Z) \rangle$ on m. This is the signature we return to F as the answer to its signing query.

In order to simulate F's view of the signing protocol, we do the following. First, we need to simulate the generation of R and then simulate the successive runs of the `Modified-Mult-SS` protocol until we get Y. But the problem is that we do not know the value R such that $R^{2^{T+1}} \equiv Y \pmod{N}$. However, we can get around this problem in a way similar to the method we used with the shares of the secret key. In the generation of R, each player picks a random value and shares it with the other players. Since there are at most t players in $\mathcal{B}_j$, we can pick their shares of R at random in Z_N. Since R is implicitly defined by Y, if $|\mathcal{B}_j| = t$, then all the other shares are also implicitly defined. But that poses no problem, because we can still provide all players $b \in \mathcal{B}_{j-1}$ with up to t random shares of these other shares without actually knowing their values.

The simulation of the run of the `Modified-Mult-SS` protocol in the computation of the shares of R^2 from the shares of R can also be done in a way similar to that of the update protocol. Let $R^{(b)}$ denote the share of R of a player $b \in \mathcal{B}_j$. We can compute the shares of R^2 as follows. For each player $b \in \mathcal{B}_j$, we create shares $(R^2)^{(b)} = (R^{(b)})^2$ using the `Shamir-SS` protocol and give them to all other players in $\mathcal{B}_j$. We also give each player $b \in \mathcal{B}_j$ a set of $n - \mathcal{B}_j$ random values in Z_N representing the shares of the shares of R^2 of all other players participating in the protocol. This defines, for each player not in $\mathcal{B}_j$, a set of $|\mathcal{B}_j|$ shares of the share of R^2 held by that player. And since $|\mathcal{B}_j| \leq t$, that player's share of R^2 is equally distributed in Z_N. We then repeat the same process in order to compute all the shares (and "sub-shares" thereof) of $R^4, \ldots, R^{2^{T+1}} \equiv Y \pmod{N}$ viewed by the corrupted players in $\mathcal{B}_j$. At this point, we also need to compute all the other shares and broadcast them so that Y can be computed by all players.

Let $R^{2^{T+1}(b)}$ denote the resulting share of $R^{2^{T+1}}$ held by a player $b \in \mathcal{B}_j$. We can compute the values of the other shares of $R^{2^{T+1}}$ by using both the value Y we obtained above from our signing query, and the values of the shares $R^{2^{T+1}(b)}$ of each player $b \in \mathcal{B}_j$. If $|\mathcal{B}_j| < t$, then we pick $t - |\mathcal{B}_j|$ shares for $R^{2^{T+1}}$ at random, then compute the rest of the shares from the ones we have. Having done this, we give Y and all shares of $R^{2^{T+1}}$ to F. Now, it still remains to simulate the adversary's view during the last part of the signing protocol, in which the shares of Z are computed from the challenge $c_1 \ldots c_l$ and the shares of R and $S_{i,j}$ $(i = 1, \ldots, l)$. However, we can do this using an argument similar to the one above since we know the value Z and the values of the shares $R^{(b)}$ and $S_{i,j}^{(b)}$ held by each player $b \in \mathcal{B}_j$.

Simulating the H oracle. We can easily simulate the hashing oracle H using our oracle $\mathcal{H}$. For each query (j, Y, m) made by F, we query $\mathcal{H}$ on the same input and return to F the answer we receive.

Obtaining a forgery. Let a be the time period in which F decides to switch to an overthreshold phase. At this point, we must provide F with the current secret key. To do so, we first switch to a breakin phase to get the current secret $(S_{1,a}, \ldots, S_{l,a})$ and then return it to F. Let $(m, \langle a, (Y, Z)\rangle)$ be the forgery output by F. We simply return $(m, \langle a, (Y, Z)\rangle)$ as the output of our algorithm.

Probability Analysis. The probability of success of our algorithm is very close to that of F. The only difference is that in simulating the signing oracle above, all the values we use are in Z_N^* (since the single-user version of the protocol works over Z_N^*) while, in the real signing oracle, it is possible for some of the values it outputs to be in Z_N but not Z_N^*. But since the value R in the signing protocol is picked at random from Z_N, the probability that it is not in Z_N^* is negligible. Given that $N = pq$, the probability is at most $p + q/(pq) = 1/q + 1/p$. Hence, if the total number of queries to the signing oracle is q_s, then $q_s(1/p + 1/q)$ is exactly the amount by which the probability of success of our algorithm is reduced with respect to that of F. ∎