Threshold RSA for Dynamic and Ad-Hoc Groups (Extended Abstract*)

Rosario Gennaro, Shai Halevi, Hugo Krawczyk, Tal Rabin

IBM T.J.Watson Research Center
Hawthorne, NY USA

Abstract. We consider the use of threshold signatures in ad-hoc and dynamic groups such as MANETs ("mobile ad-hoc networks"). While the known threshold RSA signature schemes have several properties that make them good candidates for deployment in these scenarios, none of these schemes seems practical enough for realistic use in these highly-constrained environments. In particular, this is the case of the most efficient of these threshold RSA schemes, namely, the one due to Shoup. Our contribution is in presenting variants of Shoup's protocol that overcome the limitations that make the original protocol unsuitable for dynamic groups. The resultant schemes provide the efficiency and flexibility needed in ad-hoc groups, and add the capability of incorporating new members (share-holders) to the group of potential signers without relying on central authorities. Namely, any threshold of existing members can cooperate to add a new member. The schemes are efficient, fully non-interactive and do not assume broadcast.

1 Introduction

A distributed signature scheme is a protocol where the ability to sign is distributed among a group of entities, so that a sufficiently large subset can produce valid signatures while a "small" subset cannot generate such a signature. These schemes are often referred to as *t-out-of-n threshold signatures* where n is the total number of entities and t is the "threshold". Namely, $t+1$ cooperating parties can produce a valid signature, but t or less cannot (even if they depart maliciously from the protocol). Threshold signature schemes are known for standard signatures such as RSA and DSS. One major appeal of these schemes is that the verification of a signature uses a regular public key and a standard verification procedure; hence the verifier of a signature does not need to be aware of the form (centralized or distributed) in which the signature was generated, or who were the parties involved, nor does the signature increase in size as a function of the number of signers.

Typically, in these systems each signing entity holds a share of the signing key, that it uses to produce a "fragment" of a signature on a given message.

* Full version available in [15]

When a sufficient number of such fragments are collected (i.e., a threshold), the fragments are combined in some prescribed manner to generate the resultant (standard) signature on the given message.

Threshold signature were traditionally motivated by applications requiring the protection of highly-valuable signature keys, such as in the cases of a certification authority signing public-key certificates or a bank minting electronic coins. In such applications, threshold signatures increase the security of the key by preventing "a single point of failure", and also increase the reliability and availability of the service (since disabling some nodes does not disrupt the service as long as there are $t + 1$ good functioning nodes). A substantial body of work has been devoted to designing efficient threshold signature schemes [3, 5, 9], especially for standard algorithms such as RSA [11, 8, 19, 14, 13, 24] and DSS [6, 17]. Given this motivation, prior works were mostly concerned with scenarios with a small number of nodes, with static configuration and tight coordination. (Typically the protocol would be implemented by a small set of nodes, all of which are governed by one administrative entity.)

A more recent application of threshold signatures has emerged in the area of networking and distributed computing, such as in the setting of mobile and ad-hoc networks (MANETs). In these cases, relatively small subsets of very large (and dynamic) groups report data, that is aggregated and "certified" by means of a signature. Examples include vehicular networks where cars report traffic conditions, sensor networks that report aggregate data such as temperature or radiation levels, military devices transmitting information to be reported to various commands, and more. Threshold signatures provide a robust, flexible and secure way for the nodes to report and certify data that can be verified by third parties regardless of the specific reporters. All that is needed is the assurance that a large enough subset of authorized reporters agreed on the data.

In these environments the set of parties is formed in dynamic and ad-hoc manners. Nodes may be dynamically added to the system, their share of the secret key can be either installed by trusted authorities before deployment, or added "on-the-fly" by a qualified subset of nodes already in the network. Large numbers of nodes may be deployed in the network, and yet at a given time a node may be in the communication range of only a few other nodes. In many such applications, communication bandwidth may be constrained (e.g., due to energy limitations), transmitting large amount of data or heavy interaction may be infeasible, and expensive communication primitives like broadcast may not be available. Adapting threshold signature schemes to work in such environments is challenging. In principle, threshold RSA signatures [11, 8, 24] are appealing in this case due to two important properties:

Standard signatures. They implement standard RSA signatures. Namely, the end-result of running these protocols is a standard RSA signature on the given message, and anyone can verify that signature as if it was generated by a standard centralized signer.

Non interactive. Given a message and its share of the secret key, each party locally computes a "signature fragment" without any interaction with the other parties. Then there is a public combination function that takes all these signature fragments (together with the message and public key) and turns them into a standard RSA signature.

Note that we would like the scheme to remain non-interactive even when misbehaving parties provide wrong signature fragments. For these cases, techniques from [19, 18] can be used to obtain fragment verifiability without losing the non-interactive nature of the protocols.

From the existing protocols with the above properties, the most practical is the one due to Shoup [24], which is very efficient in scenarios with static, relatively small sets of parties. But Shoup's protocol has a parameter n, which is a global upper-bound on the number of (potential) parties in the protocol, and the computation of signature fragments and the combining function use the number $n!$ in the calculations (specifically as an exponent in a modular exponentiation operation). This means that the parameter n must be fixed and known to all parties, and the computation takes time at least linear in n. As the parameter n grows, as is likely in the dynamic applications that we mentioned, these computations become expensive or even infeasible.

This problem is even more serious, since in Shoup's protocol n is an upper bound on the *identities* of parties in the protocol, namely it is assumed that no party has an identity whose value (as an integer) is larger than n. For example, if the identities are arbitrary 32-bit numbers, the protocol must use $n = 2^{32}$ so $n!$ is a 2^{37}-bit number! (Clearly, using network addresses or serial numbers of 64 bits or 160-bit hash values is incompatible with this protocol.) The range of identities can be reduced via tight coordination of the identity name space, but such tight coordination flies in the face of the flexibility that is expected in dynamic groups. Further, tight coordination may be impossible in applications that need flexible addition of nodes, either by loosely-coordinated "trusted authorities" or even by completely un-coordinated groups of members within the group itself.

This work. We describe two results that extend Shoup's protocol to dynamic ad-hoc groups:

- We present a variant of Shoup's protocol that keeps all the appealing properties of the original scheme but frees the protocol from any dependency on the total number of parties. Technically, the dependency on n as discussed before is replaced with a dependence on t (where $t + 1$ is the number of parties that needed to generate a signature).
- We show the practicality of our scheme in the dynamic group scenarios by extending our threshold scheme to support the addition of new members without the need to centrally coordinate this join operation. Basically, we allow any set of $t+1$ or more parties to cooperate (non-interactively!) in order to add a new member into the group, without having to invoke any "trusted authority". This is done by adapting to our case the elegant non-interactive solution of Saxena et al. [25].

We believe that the combination of both results leads to the first practical solution for dynamic and communication-constrained scenarios described above.

2 Background

In this work we use RSA moduli of special form, namely, the product of *safe primes*: $N = pq$ such that $p = 2p' + 1$ and $q = 2q' + 1$ with p', q' also primes. We assume that $|p| = |q| = k$, where k is the security parameter. With Z_N^* we denote the set of non-negative integers smaller than N which are relatively prime to N. This is a group with respect to multiplication mod N. Define $\lambda(N) \stackrel{\text{def}}{=} 2p'q'$ (the Carmichael function of N). We also denote $m \stackrel{\text{def}}{=} p'q'$.

The RSA function [21] is defined by an integer e which is relatively prime to $\lambda(N)$:

$$\forall x \in Z_N^* \;\; RSA_{N,e}(x) = x^e \bmod N$$

Since e is relatively prime to $\lambda(N)$ this is a permutation over Z_N^*. It is believed that for a randomly-generated composite N (which is a product of two large enough safe primes), inverting the RSA function on random inputs is infeasible. On the other hand, inverting this function is easy given some trapdoor information. The trapdoor could be the prime factorization of N, since it would allow to compute $d = e^{-1} \bmod \lambda(N)$ and hence

$$\text{if } y = x^e \bmod N \text{ then } x = y^d \bmod N$$

Assumption 1 (RSA) *Consider a generation procedure* Gen *for generating pairs* (N, e) *as above. The RSA Assumption (with parameters* (T, ϵ)*) says that for any algorithm* $\mathcal{A}$ *that runs in time* T*:*

$$\Pr[(N, e) \leftarrow \mathsf{Gen},\; x \in_R Z_N^* \;;\; \mathcal{A}(N, e, y = x^e \bmod N) = x] \leq \epsilon$$

RSA Signatures. We recall the *Full Domain Hash RSA* (FDH-RSA) signatures. The public key is (N, e), the secret key is $d = e^{-1} \bmod \lambda(N)$, and a hash function H which maps arbitrary messages to Z_N^* is also part of the public key. To sign a message M, the signer computes $y = H(M)$ and the signature $\sigma = y^d \bmod N$. To verify a message/signature pair (M, σ) under public key (N, e), the receiver checks if $\sigma^e = H(M) \bmod N$. It is well known that the security of FDH-RSA can be reduced to the RSA Assumption if we model H as a random oracle [4, 7].

2.1 Threshold Cryptography

In a *threshold cryptographic* scheme, the secret key of a cryptographic scheme is stored in a shared form among several parties [3, 5, 10]. The goal is to prevent compromise of the secret key by an attacker who breaks into parties and reads their memory. Indeed if the secret key were stored in a single party, then a single break-in would compromise the security of the entire scheme. The idea of threshold cryptography is to share the key among several parties so that the attacker must break into several of them before learning the secret key.

Threshold cryptography schemes are thus composed of two phases. The first is a *sharing phase* in which the secret key is installed in this shared form among the parties; this part can be performed by a trusted dealer who temporarily knows the secret key, and shares it among the parties, before erasing it from its memory. (Alternatively the parties jointly generate the key directly in a shared form.) The second phase is a *computation phase*, where the parties run some protocol to jointly perform the cryptographic computations in which the secret key is employed.

We assume to have a set of upto n *parties* that can communicate with each other. We do not assume physically secure channels or a broadcast channel among them. In addition to the n parties, we also consider a *dealer* who will share the secret key among them (we do assume that for this phase there are private channels between the dealer and the parties).

We assume a computationally bounded adversary, $\mathcal{A}$, who can corrupt up to t of the n parties in the network, where t is a parameter. (In our application domain we typically have $t \ll n$, but the protocol that we describe works for any $t < n$.) We call an adversary that corrupts no more than t parties a t-adversary. We say that the adversary is *static* if it corrupts all its parties at the beginning of the protocol, otherwise we say it is *adaptive*. An adversary is *honest-but-curious* if it does not modify the code of the corrupted parties, but just reads their memory. A *malicious* adversary on the other hand may also cause corrupted parties to behave in any (possibly malicious) way. Yet, the adversary can never corrupt the dealer.

Threshold Signature Schemes The following definitions of secure threshold signature schemes are essentially taken from [17]. A threshold signature scheme is a pair of protocols ($\mathsf{Thresh\text{-}Key\text{-}Gen}$, $\mathsf{Thresh\text{-}Sig}$) for a set of n parties and a dealer, and a verification algorithm Ver.

$\mathsf{Thresh\text{-}Key\text{-}Gen}$ is a key generation protocol carried out by a designated dealer to generate a pair (pk, sk) of public/private keys. At the end of the protocol the private output of party P_i is a value sk_i (related to the private key sk). The public output of the protocol contains the public key pk, and possibly some additional verification information v.

$\mathsf{Thresh\text{-}Sig}$ is the distributed signature protocol. The private input of P_i is the value sk_i. The public inputs for all parties consist of a message M, the public key pk, and the verification information v (if any) . The output of the protocol is a signature sig on M (relative to the public key pk). The verification algorithm Ver, on input M, sig, pk, checks if sig is a valid signature of M under pk.

The definition of security is adapted from the centralized case. Specifically, we consider a t-adversary that can interact with the honest parties and ask them to generate signatures on messages $M_1, M2, \ldots$ that the adversary chooses adaptively. In the malicious model the adversary can also actively participate in these signature-generation protocols (via the set of t parties that it controls), and in either model the adversary can choose the set of parties that would participate in the current signature generation. As usual, the goal of the adversary is to produce a signature on any message M that was not obtained via one of these runs of the protocol.

Definition 1. *We say that a threshold signature scheme (*Thresh-Key-Gen*,* Thresh-Sig*,* Ver*) is t-*secure *(with parameters (T, ϵ)), if no t-adversary $\mathcal{A}$ that runs in time T can produce a signature on any message M without the participation of at least one honest party, except with probability ϵ.*

In the semi-honest model this should hold even if $\mathcal{A}$ can watch the signature protocol being run on messages $M_1, M_2, \dots$ of $\mathcal{A}$'s choosing (so long as $M \neq M_i$ for all i), and in the malicious model this should hold even if the adversary can actively participate in these protocol executions.

2.2 Secret Sharing vs. Threshold Cryptography

Notice the difference between threshold cryptography and the simpler task of secret sharing [22, 2]. In secret sharing, the secret is first shared among the parties who later reconstruct it. In threshold cryptography, the secret is never reconstructed, but rather employed as a shared input into a cryptographic computation. However the sharing mechanisms are usually similar in both areas.

Here we recall Shamir's scheme for secret sharing [22]. Let q be a prime and the secret is an integer $s \in Z_q$. The goal is to share s among n parties in such a way that t or less of them have no information about s, while $t+1$ of them can easily reconstruct it.

The dealer chooses t random values $a_t, \dots, a_1$ in Z_q and considers the polynomial $f(x) = a_t x^t + \ldots + a_1 x + s$. Assume each party is given as a "name" an integer between 1 and n. Then party i is given the *share* $s_i = f(i) \bmod q$. Notice how t shares give no information about s (for every possible secret s' there is a polynomial f' consistent with s' and the t shares). On the other hand $t+1$ shares completely define s, via polynomial interpolation. Notice that if $s_{i_1}, \dots, s_{i_{t+1}}$ are $t+1$ shares corresponding to the points in $S = \{i_1, \dots, i_{t+1}\}$, then the secret can be computed from S and these shares as

$$s = f(0) = \sum_{j=1}^{t+1} L_S(0, i_j) s_{i_j} \bmod q$$

where $L_S(0, i_j)$ are the appropriate Lagrangian coefficients, namely

$$L_S(\alpha, \beta) \stackrel{\text{def}}{=} \frac{\prod_{\gamma \in S, \gamma \neq \beta} (\alpha - \gamma)}{\prod_{\gamma \in S, \gamma \neq \beta} (\beta - \gamma)} \bmod q \qquad (1)$$

Notice that Shamir's secret sharing has an interesting homomorphic property that allows the shares to be used to jointly compute exponentiations of the form y^s where y is known, without reconstructing the secret in the clear. Assume for example that y is an element in a cyclic group G of known prime order q: then each party could publish the value $\gamma_i = y^{s_i}$ in G and then

$$y^s = y^{\left(\sum_{j=1}^{t+1} L_S(0, i_j) s_{i_j} \bmod q\right)} = \prod_{j=1}^{t+1} \gamma_{i_j}^{L_S(0, i_j)} \quad \in G$$

2.3 Shoup's Threshold RSA

The homomorphic property of Shamir's secret sharing described above would seem ideal for the setting of threshold RSA, as it would allow the parties to jointly compute RSA signatures with secret key d, without recovering d itself (recall that to compute a signature on M, the parties have to jointly compute $\sigma = y^d \bmod N$ where $y = H(M)$). However, a closer look reveals that this approach does not immediately work in the case of threshold RSA.

The problem is that the sharing must be conducted modulo $\lambda(N)$, which must be kept secret from the parties themselves (since knowing $\lambda(N)$ or a multiple of it would allow any single party to factor N and compute the secret key d for itself). This issue arises in the computation of the Lagrangian coefficients modulo $\lambda(N)$, since they require the computation of inverses modulo $\lambda(N)$ (which are equivalent to knowing a multiple of $\lambda(N)$). This problem received considerable attention in the threshold cryptography literature (e.g. [11, 8]) and various solutions were proposed. Shoup in [24] presented the most efficient solution to date.

Shoup observed that if we denote $\Delta = n!$ then the values $\Delta \cdot L_S(0, j)$ for all S, j are integers and no inverse computation modulo $\lambda(N)$ is required if we compute the linear combination of the shares using $\Delta \cdot L_S(0, i_j)$ instead of $L_S(0, i_j)$. However doing so we will reconstruct the value $\sigma' = y^{\Delta d} \bmod N$ rather than $\sigma = y^d \bmod N$. But if e relatively prime to Δ, we can apply the extended Euclidean algorithm in the exponent [23] to recover σ from σ'. Details of Shoup's scheme follow.

Sharing Phase: Given the public key N, e, the dealer (who knows the factorization of N) computes $d = e^{-1} \bmod m$. It also chooses t random values $a_1, \ldots, a_t$ in Z_m and defines the polynomial $f(z) = a_t z^t + \ldots + a_1 z + d$. Party i is given the *share* $d_i = f(i) \bmod m$. The (maximal) number n of parties is fixed and public and with it the value $\Delta = n!$.

Signature Computation Phase: On input a message M, party i computes $y = H(M) \in Z_N^*$ and the *signature fragment* $\sigma_i = y^{2\Delta \cdot d_i} \bmod N$ which it then publishes. (See the next section for an explanation of the 2Δ factor in the exponent.) Then given any $t+1$ of these values, $\sigma_{i_1}, \ldots, \sigma_{i_{t+1}}$, anybody can compute

$$\sigma' = \prod_{j=1}^{t+1} \sigma_{i_j}^{2\Delta \cdot L_S(0,i_j)} \bmod N \tag{2}$$

By simple algebra we have that

$$\sigma' = \prod_{j=1}^{t+1} y^{4\Delta^2 \cdot L_S(0,i_j) \cdot d_{i_j}} \bmod N =$$
$$(y^{4\Delta^2})^{\sum_{j=1}^{t+1} L_S(0,i_j) d_{i_j} \bmod m} \bmod N = y^{4\Delta^2 \cdot d} \bmod N$$

Notice that the operations (polynomial interpolation) "in the exponent" are performed mod m because the value $y^{4\Delta^2}$ has order m.

At this point we need to show how to compute $\sigma = y^{1/e}$, given σ'. If $GCD(e, 4\Delta^2) = 1$ (which we ensure by choosing e as a prime larger than n), we compute integers a, b such that $ae + 4b\Delta^2 = 1$ and set $\sigma = y^a(\sigma')^b \bmod N$. To see that $\sigma = y^{1/e} \bmod N$ note the following equalities mod N:

$$\sigma = y^a(\sigma')^b = (y^{1/e})^{ae} \cdot (y^{1/e})^{4b\Delta^2} = (y^{1/e})^{ae+4b\Delta^2} = y^{1/e}.$$

3 Threshold RSA Signatures for Ad-Hoc Groups

As discussed in the Introduction, we are interested in constructing a threshold RSA signature for "ad hoc" groups, that can be very large, dynamic, and decentralized, and where parties from very different domains aggregate temporarily to perform a specific ("ad hoc") task. In these networks, because of the large number of parties and the loose coordination among them, it is impractical (if not impossible) to maintain a coherent centralized naming scheme, i.e., one where if there are n parties in the network their identities will be integers from 1 to n.

Consider for example the task of adding parties to the network and giving them shares of the secret key. In Section 5 we describe a solution in which $t+1$ honest parties can provide a new party with a new share of the secret key. To do that, the $t+1$ parties must give this new party a unique identity ID and its share $f(ID)$, where f is the t degree polynomial such that $f(0) = d$. In a very decentralized network, it is basically impossible to keep track of the IDs issued by various subsets of parties in the network, and therefore it is infeasible to maintain these identities in a small subset of integers.

What is most likely to happen in ad hoc networks is that parties have an ID already assigned to them, probably a large integer (say a 32-bit or a 160-bit integer, e.g. their serial number, their IP address or a hash of some other identity) and we would want to use that ID all across the board. In the example above, when adding parties to the network, the subset of parties could use the ID of the new party (if it has one) or generate a random one for it (if the ID are sufficiently long, it will most likely be unique).

Now consider what happens in Shoup's scheme when the IDs of parties are long integers say between 1 and 2^k for some parameter k. In this case the computation of the value $\Delta = n!$ is infeasible as one must set $n = 2^k$ in order for Δ to remove all possible denominators in Eq. (2) (note that $2^k! > 2^{2^k}$). There are two places where this factor Δ is used in Shoup's scheme:

(1) In the interpolation of partial signatures: Given the various shares $\sigma_i = y^{2\Delta f(i)}$, the players reconstruct the value $\sigma' = y^{4\Delta^2 \cdot f(0)}$ via interpolation "in the exponent". One of the two Δ factors in the exponent is needed for the correct operation of the scheme, as it is exactly what lets the parties replace the fractional Lagrangian coefficients with integers that they can compute.

Reducing this Δ factor is straightforward. Indeed, given the set S of parties in the threshold computation of the RSA signature, this factor can be easily replaced by

$$\Delta_S \stackrel{\text{def}}{=} \operatorname{lcm}\Big\{ \Big(\prod_{\substack{j \in S \\ j \neq i}} (i-j) \Big) \;:\; i \in S \Big\}.$$

Note that if S has $t+1$ elements, each a number between 0 and 2^k, then Δ_S is bounded by

$$\Delta_S \;<\; \prod_{\substack{i,j \in S \\ i \neq j}} (i-j) \;<\; (2^k)^{t^2} \;=\; 2^{kt^2},$$

so the bit-size of Δ_S is linear in k and at most quadratic in t.

(2) In the computation of partial signatures: Given public input y and their private share d_i, each party computes $\sigma_i = y^{2\Delta d_i} \bmod N$. This additional Δ factor in the exponent is introduced to obtain a provably secure scheme. Indeed we need to show that the values transmitted by the good parties do not help the adversary, beyond the computation of $\sigma = y^{1/e}$. This is done by a simulation argument in which we show that the adversary is able to compute the values transmitted by the good parties from any t shares (belonging to the parties the attacker controls) and the value $\sigma = y^{1/e}$. Assume that the adversary knows t shares $d_{i_1}, \ldots, d_{i_t}$, and denote $B = \{i_1, \ldots, i_t\}$ (where B stands for "bad") the set of corrupted parties. Denote $\tilde{B} = B \cup \{0\}$. Notice that any other share d_i for $i > t$ can be computed as

$$d_i = L_{\tilde{B}}(i,0)d + \sum_{j=1}^{t} L_{\tilde{B}}(i,i_j)d_{i_j} \bmod m$$

Thus the adversary given the value $\sigma = y^{1/e}$ can compute the value σ_i for $i > t$ output by a good party as follows:

$$\sigma_i = y^{2\Delta d_i} = (y^2)^{L_{\tilde{B}}(i,0)d + \sum_{j=1}^{t} L_{\tilde{B}}(i,i_j)d_{i_j} \bmod m} = \sigma^{2\Delta L_{\tilde{B}}(i,0)} \prod_{j=1}^{t} (y^{2d_j})^{\Delta L_{\tilde{B}}(i,j)}$$

Notice that the products $\Delta L_{\tilde{B}}(i,j)$ are all integer values and thus the adversary can perform this computation (since it does not require computing inverses modulo m). However, this argument would not hold if the parties transmitted just $\sigma_i = y^{d_i}$ (in which case we do not know if a simulation would be possible).

Eliminating this factor is not as easy as in case (1). The reason that we cannot replace Δ with some Δ_S as above is that the relevant set S here is the set $\tilde{B}$ of "bad parties" (i.e., those that were compromised by the adversary), and $\tilde{B}$ is not known to the honest parties. Fortunately, we show below that the analysis can still be carried out when Δ is replaced by the much smaller factor 2^{kt}. For this crucial reduction in the size of the exponent, we pay either by having to carry out the analysis in the random-oracle model or by strengthening the hardness assumption. (Note however that the basic Shoup scheme already relies on the random-oracle model for the non-interactive verification of signature fragments.)

Remark: Shoup presented an alternative protocol for threshold RSA in [24], where the share of party i is $d_i = \Delta^{-1} f(i) \bmod m$ (rather than $d_i = f(i)$). Thereafter the parties do not have to raise their partial signatures to a factor Δ, as it is already "factored in" in their shares. However the dealer still needs to compute $\Delta^{-1} \bmod m$, and the signature generation still needs to exponentiate to the power of $\Delta \cdot L_S(\cdot, \cdot)$. Hence both the dealer and the "signature combiner" work in time at least linear in n (which is exponential in settings where $n = 2^k$).

3.1 Our Threshold RSA Protocol for Ad-Hoc Groups

We now describe the details of our new scheme, which we call **TFDH-RSA**. We first describe a basic version that is only secure in the honest-but-curious attack model. *The extension to the malicious attack model is discussed in Section 6.*

Sharing Phase: Given the public key N, e (where e is a prime larger than $n = 2^k$) and the secret key d (such that $d = e^{-1} \pmod{m}$), the dealer chooses t random values $a_1, \ldots, a_t$ in Z_m and defines the polynomial $f(x) = a_t x^t + \ldots + a_1 x + d$. Each party i is given the share $d_i = f(i) \bmod m$.

Signature Computation Phase: On input a message M, party i computes $y = H(M) \in Z_N^*$ and the *signature fragment* $\sigma_i = y^{2^{kt} d_i} \bmod N$, which it then publishes.

Given $t+1$ of these values $\sigma_{i_1}, \ldots, \sigma_{i_{t+1}}$ the signature $\sigma = y^d \bmod N$ is computed as follows. Let

$$\Delta_S = \operatorname{lcm}\left\{ \left(\prod_{\substack{j \in S \\ j \neq i}} (i - j) \right) \ : \ i \in S \right\} \tag{3}$$

where $S = \{i_1, \ldots, i_{t+1}\}$, and set

$$\sigma' = \prod_{j=1}^{t+1} \sigma_{i_j}^{\Delta_S \cdot L_S(0, i_j)} \bmod N. \tag{4}$$

Using extended Euclidean algorithm find values a, b such that $ae + b(2^{kt}\Delta_S) = 1$. Finally, the signature $\sigma = y^d \bmod N$ is computed as $\sigma = y^a \cdot (\sigma')^b \bmod N$.

Note that for any j, the quantity $\Delta_S \cdot L_S(0, i_j)$ is an integer that can be computed just by knowing S and i_j (no need to know m or any inverse $\bmod m$). Moreover, this integer is smaller than 2^{kt^2} and then the computation of σ' is feasible. To see that the value σ we computed is indeed $y^d \bmod N$ note that similarly to the case of Eq. (2)

$$\sigma' = y^{\Delta_S \cdot 2^{kt} \cdot f(0)} = y^{\Delta_S \cdot 2^{kt} \cdot d} \bmod N \tag{5}$$

and then

$$\sigma \ = \ y^a \cdot (\sigma')^b \ = \ (y^{1/e})^{ae} \cdot (y^{1/e})^{b 2^{kt} \Delta_s} \ = \ y^{1/e} \pmod{N}. \tag{6}$$

Finally, note that the values a, b computed via the extended Euclidean algorithm exist since by choice of e, $GCD(e, 2^{kt}\Delta_S) = 1$.

Remark: the choice of e. The value e must be chosen as a prime larger than any possible identity (e.g., larger than 2^k). Alternatively, if the identities are random k-bit integers, the value e can be chosen somewhat smaller (but still a "large enough prime") and then rely on the fact that with high probability no two identities will have a difference that is divisible by e.

4 Security Analysis

Ideally, we would like to claim that the protocol above is as secure as the underlying signature scheme. In particular, we want to claim that seeing the signature fragments σ_i does not gives the adversary any more information than the signature σ itself, by presenting a simulator $\mathcal{S}$ that given σ can generate the view of the adversary in the protocol (i.e., the signature fragments of the honest parties $\sigma_j = x^{f(j)} \bmod N$).

Unfortunately, as we explained above, we do not know how to generate this view without the extra factor of Δ in the signature generation. Specifically, the simulator can only compute the related quantities $\sigma'_j = \sigma_j^{\Delta_{\tilde{B}}}$, where $\Delta_{\tilde{B}}$ is as defined in Eq. (3) (with respect to the set $\tilde{B}$ that consists of zero and the "bad parties"). The reason is that we can write

$$\sigma'_j \;=\; \sigma^{\Delta_{\tilde{B}} \cdot L_{\tilde{B}}(j,0)} \cdot \prod_{i \in B} y^{\Delta_{\tilde{B}} \cdot L_{\tilde{B}}(j,i) \cdot d_i},$$

and for all i, the quantity $\Delta_{\tilde{B}} \cdot L_{\tilde{B}}(j,i)$ is an integer that the simulator can efficiently compute from $\tilde{B}$, i and j. However, in this case we do not know a way to "extract" the $\Delta_{\tilde{B}}$ root out. In particular we do not know a "public exponent" corresponding to $f(j)$ that would let us use the GCD calculations in order to eliminate the extra factor of $\Delta_{\tilde{B}}$ in the exponent.

In dealing with this problem, we separate the treatment of the powers of two in $\Delta_{\tilde{B}}$ from the odd factors of it.[1] Write

$$\Delta_{\tilde{B}} = 2^{\ell(\tilde{B})} e(\tilde{B})$$

with $e(\tilde{B})$ odd. Clearly, for any set $\tilde{B}$ of $t+1$ elements in $[1..2^k]$, it must be that $\ell(\tilde{B}) < kt$.

Since in our protocol the parties compute signature fragments by raising $y = H(m)$ to the power $2^{tk} f(i)$, then the simulator does not need to take $2^{\ell(\tilde{B})}$ roots (see details below). On the other hand the problem of taking $e(\tilde{B})$ roots remains.

We present two solutions for this problem. In Theorem 2 we show that when we model the function H as a random oracle, our protocol can be proven t-secure (as per Definition 1) under the standard RSA assumption. This is a strong assurance of security, especially since random-oracle proofs are the only security assurance that is known for most standard RSA signatures. However, this proof

[1] See discussion after Theorem 1 about the reason for this separation.

does not say much about the security of our signature protocol when instantiated with any specific hash function. In particular, it still leaves open the possibility that there are hash functions H for which centralized RSA signature are secure but our protocol is not. Therefore, in Theorem 1 we provide a different analysis *in the standard model*, relating the security of our protocol with any specific function H to a slightly modified centralized RSA signature that uses the same function H. We argue informally that this "slightly modified" scheme is likely to be as secure as the original one, hence providing yet other assurance of the security of our protocol.

4.1 Security Theorems

Consider the following signature scheme D-RSA: the public key is (N, e) where e is prime, and relatively prime with $\lambda(N)$. The secret key is the factorization of N. The public key contains a hash function H which outputs elements of Z_N^*. Moreover we allow the adversary to specify an additional parameter e' that must be an odd integer co-prime with e.

On input a message M, the signer returns a pair (σ, σ') such that $\sigma^e = (\sigma')^{e'} = H(M) \bmod N$. Given an alleged signature (σ, σ') on M, however, the verifier only checks that $\sigma^e = H(M) \bmod N$.

We say that the D-RSA signature scheme using the hash function H is *two-phase secure* if no feasible forger $\mathcal{F}$ can win the following game: first $\mathcal{F}$ is given as input N, e and it specifies an odd integer e', that must be co-prime with e (Phase 1). Then $\mathcal{F}$ conducts a traditional adaptive chosen-message attack (i.e. $\mathcal{F}$ gets signatures on messages of its choice) and produces a valid signature on a message that it did not request before (Phase 2).

Theorem 1. *For any static, honest-but-curious, t-adversary $\mathcal{A}$ and for every hash function H, then TFDH-RSA scheme using H is a secure threshold signature scheme against $\mathcal{A}$ if the D-RSA signature scheme using the same H is two-phase secure.*

Interpretation of Theorem 1. Although we cannot prove that the protocol TFDH-RSA does not give the adversary any more power than just interacting with the underlying RSA signature scheme (with the same hash function H), Theorem 1 tells us that it does not give it more power than what it could get from the ability to get also e'-th roots (in addition to the e-th roots that it gets from the signature scheme). It is generally believed that when e, e' are co-primes, then extracting e'-th roots does not help in extracting e-th roots. In this light, Theorem 1 can be interpreted as asserting that any attack on the protocol TFDH-RSA must either break the underlying RSA signatures, or find a way to use e'-th roots in order to extract e-th roots.

Proof. Assume by contradiction that TFDH-RSA is not secure. Then there exists an adversary $\mathcal{A}$ that interacts with TFDH-RSA, statically corrupting at most t parties, such that $\mathcal{A}$ has a noticeable chance ϵ of forging an RSA signature. We want to use this $\mathcal{A}$ as a subroutine to construct a forger $\mathcal{F}$ that breaks the two-phase security of D-RSA. We want $\mathcal{F}$ to have a similar running time and similar success probability as $\mathcal{A}$.

The forger $\mathcal{F}$ basically simulates TFDH-RSA for the adversary $\mathcal{A}$. It is given a public key (N, e), and must run Phase 1. $\mathcal{F}$ starts by giving (N, e) to the adversary $\mathcal{A}$. The latter responds by asking to corrupt a set $B = \{i_1, \ldots, i_t\}$ of parties. ($\mathcal{A}$ can compromise upto t parties, and we assume w.l.o.g. that it compromises exactly t parties.) Then $\mathcal{F}$ chooses t values $d_{i_1}, \ldots, d_{i_t}$ at random in the interval $[N/4]$ and gives to $\mathcal{A}$ the value d_{i_j} as the secret-key share for party i_j. As observed above, the distributions of the shares given by $\mathcal{F}$ to $\mathcal{A}$, is statistically close to the distributions of the shares seen by $\mathcal{A}$ during a real execution of FDH-RSA.

With B the set of corrupted, parties, $\mathcal{F}$ sets $\tilde{B} \stackrel{\text{def}}{=} B \cup \{0\}$ and computes $\Delta_{\tilde{B}}$ as defined in Eq. (3), namely

$$\Delta_{\tilde{B}} \;=\; \mathrm{lcm}\Bigg\{ \Bigg(\prod_{\substack{j \,\in\, \tilde{B} \\ j \,\neq\, i}} (i-j) \Bigg) \;:\; i \in \tilde{B} \Bigg\}.$$

It also lets $\ell(\tilde{B})$ be the largest integer ℓ such that 2^ℓ divides $\Delta_{\tilde{B}}$, and sets $e(\tilde{B}) = \Delta_{\tilde{B}}/2^{\ell(\tilde{B})}$ and $\Gamma_{\tilde{B}} = 2^{kt} \cdot e(\tilde{B})$ (so $e(\tilde{B})$ is odd and $\Delta_{\tilde{B}}$ divides $\Gamma_{\tilde{B}}$). The forger $\mathcal{F}$ concludes Phase 1 by specifying the value $e' = e(\tilde{B})$. (Note that e' is co-prime with e, since e is a prime larger than 2^k and all the identities in $\tilde{B}$ are k-bit integers).

Now $\mathcal{F}$ starts Phase 2, the adaptive chosen message attack, by running $\mathcal{A}$'s attack. When $\mathcal{A}$ asks for message M to be signed, the forger $\mathcal{F}$ asks its own signature oracle for a signature on M, therefore receiving the values σ, σ' such that $\sigma^e = (\sigma')^{e'} = y = H(M) \bmod N$. Notice that σ is the signature on M that must be computed in the simulation of TFDH-RSA.

To complete the simulation for $\mathcal{A}$, the forger $\mathcal{F}$ must now use σ to produce the signature fragments of the good parties. Let $f(x)$ be the polynomial (modulo m) of degree t that satisfies $f(i) = d_i$ for every $i \in B$ and $f(0) = d$. Then, the value of σ_j for $j \notin B$ is

$$\sigma_j = y^{2^{kt} f(j)} \bmod N$$

Remember that

$$f(j) = \sum_{i \in \tilde{B}} L_{\tilde{B}}(j,i) \cdot f(i) \bmod m$$

but because the values $L_{\tilde{B}}(j,i)$ are fractions we cannot compute this value directly or in the exponent. But by multiplying the sum by $\Gamma_{\tilde{B}}$ will remove all the denominators. By using σ' the forger can also bypass the problem of taking $e(\tilde{B})$-roots:

Using Shamir's method of "GCD in the exponent", the forger $\mathcal{F}$ first computes a value w such that $w^{e' \cdot e} = y \bmod N$. Namely, since $GCD(e, e') = 1$ then $\mathcal{F}$ can find integers a, b such that $ae + be' = 1$, and setting $w = \sigma^b (\sigma')^a \bmod N$

yields the required value. Next, for each $i \in \tilde{B}$ the forger $\mathcal{F}$ computes the integer $\lambda_{j,i} \stackrel{\text{def}}{=} \Gamma_{\tilde{B}} \cdot L_{\tilde{B}}(j,i)$. (These are indeed integers since the denominator in each of the Lagrangian coefficients divides $\Delta_{\tilde{B}}$ and therefore also $\Gamma_{\tilde{B}}$.) Finally, the forger computes $\sigma_j = w^{\lambda_{j,0}} \cdot \prod_{i \in B} (\sigma')^{\lambda_{j,i} d_i} \pmod N$. To see that this is the correct value, observe that:

$$\begin{aligned}
w^{\lambda_{j,0}} \cdot \prod_{i \in B} (\sigma')^{\lambda_{j,i} d_i} &= w^{\Gamma_{\tilde{B}} L_{\tilde{B}}(j,0)} \cdot \prod_{i \in B} (\sigma')^{\Gamma_{\tilde{B}} L_{\tilde{B}}(j,i) d_i} \\
&\stackrel{(a)}{=} w^{(2^{kt} e') L_{\tilde{B}}(j,0)} \cdot \prod_{i \in B} (\sigma')^{(2^{kt} e') L_{\tilde{B}}(j,i) d_i} \\
&\stackrel{(b)}{=} (w^{2^{kt}})^{d \cdot e \cdot e' L_{\tilde{B}}(j,0)} \cdot \prod_{i \in B} (\sigma')^{2^{kt} e' L_{\tilde{B}}(j,i) d_i} \\
&= (w^{e' \cdot e})^{2^{kt} L_{\tilde{B}}(j,0) d} \cdot \prod_{i \in B} ((\sigma')^{e'})^{2^{kt} L_{\tilde{B}}(j,i) d_i} \\
&= \prod_{i \in \tilde{B}} y^{2^{kt} L_{\tilde{B}}(j,i) f(i)} \quad = \quad y^{2^{kt} \sum_{i \in \tilde{B}} L_{\tilde{B}}(j,i) f(i)} \\
&= y^{2^{kt} f(j)} \quad = \quad \sigma_j \pmod N
\end{aligned}$$

where Equality (a) holds since $2^{kt} e' = 2^{kt} e(\tilde{B}) = \Gamma_{\tilde{B}}$, and Equality (b) holds since $w^{2^{kt}}$ is a quadratic residue modulo N and hence its order divides m, and since $d = e^{-1} \bmod m$ then $(w^{2^{kt}})^{d \cdot e} = w^{2^{kt}} \bmod N$. (Note that since we set $d = e^{-1} \bmod m$ and not $d = e^{-1} \bmod \lambda(N)$, then $z^{de} = z$ does not necessarily hold when z is not a quadratic residue modulo N.)

It follows from the description above that the simulated view that $\mathcal{A}$ sees in this run of $\mathcal{F}$ is almost identical to its view in the interaction with the protocol (the only difference is the negligible difference in the distribution of the d_{i_j}'s). Hence with probability negligibly close to ϵ, $\mathcal{A}$ outputs a valid forgery $\hat{\sigma}$ on some message $\hat{M}$. Namely, $\hat{\sigma}^e = H(\hat{M}) \bmod N$. The forger $\mathcal{F}$ then chooses an arbitrary value $\hat{\sigma}' \in Z_N^*$ and the pair $(\hat{M}, \langle \hat{\sigma}, \hat{\sigma}' \rangle)$ is a valid forgery for D-RSA (recall that the second "signature" is not verified in D-RSA).

Remark. Note the reason for separating the powers of two in $\Delta_{\tilde{B}}$ from the odd factors: For the assumption that we make about security of D-RSA, it is crucial that the parameter e' be odd (since letting the adversary to extract even roots might allow it to factor the modulus N). Hence we must limit the extra help that the simulator can get to only odd roots, and the even factors must be handled in the protocol itself.

Theorem 2. *When the hash function H is modeled as a random oracle, then TFDH-RSA is a t-secure threshold signature scheme against static, honest-but-curious adversaries, under the RSA assumption.*

One way to prove Theorem 2 is to observe that under the RSA assumption, the D-RSA signature scheme is two-phase secure in the random-oracle model (where the proof is nearly identical to the security proof for Full-Domain-Hash signatures), and so we can use Theorem 1 from above.

5 Dynamic Additions of New Parties

We consider highly dynamic networks, where parties can join at any time and must be provided with shares of the signature key when they join. In some cases it may be reasonable to assume that such a share is installed by a trusted entity before the party is added to the network, but in other, less centralized situations, the parties already deployed will have to cooperate in order to furnish the new party with a share of the signing key.

In [25] an elegant method was introduced to solve the problem of player addition in secret sharing. The advantage of their scheme is that it is non-interactive, that is, each of the original $t+1$ parties only needs to send a single message to the new party P_{new}. The key idea of their approach is to use a bivariate polynomial. Similarly to the original Shamir's scheme, their scheme works over a prime field.

In this section we show how to adapt their scheme to work for the purpose of a threshold RSA scheme. The technical problems are the same at the ones we saw in the previous sections (the interpolation happens modulo a secret number m). We use some similar techniques to those used earlier, but the end result is somewhat different. When adding a new party P_{new} to the network, the new share given to it is not the same share as the one it would have received from the dealer in the sharing phase, but rather some multiple of that original secret. Thus, we need to modify the signature computation phase to incorporate these different shares. We show in the following that we are still able to generate signatures in a threshold fashion.

We modify our description of the threshold RSA from the previous sections so that the sharing of the secret is done via a bivariate polynomial. We note that if we do not consider newly added shares then this modification only affects the format in which the shares are represented but leaves the signature generation protocol exactly as it is in the case of the single-variate polynomial. The details follow. Again, we describe the protocol in the case of a honest-but-curious adversary. The details to add robustness (security against a malicious adversary) appear in Section 6.[2]

Sharing Phase: Given the public key N, e and the secret key d (such that $d = e^{-1} \pmod{m}$), the dealer chooses $(t+1)^2$ values $a_{i,j} \in Z_m$ (for $i, j \in [0, t]$), at random subject to $a_{i,j} = a_{j,i}$ and $a_{0,0} = d$. The dealer then defines the polynomial $f(x, y) = \sum_{i,j} a_{i,j} x^i y^j$. Note that since $a_{i,j} = a_{j,i}$ then the polynomial is symmetric, $f(x, y) = f(y, x)$. The share of party i is the polynomial $d_i(x) = f(x, i)$. (We note that only the free term $d_i(0) = f(0, i)$ will be used when computing the signature fragments, so reconstructing the signature from the fragments is unchanged from before.)

[2] We note that performing secret sharing via a bivariate polynomial has the added advantage, that it provides a check that the dealer is actually sharing a unique secret. This is called *verifiable secret sharing* (VSS) [1]. We do not discuss this feature in this paper, but we point out that it could be useful in some applications.

For the design of our protocol we also need to define an additional value δ_i. Initially, for the parties who receive shares from the dealer we have that $\delta_i = 1$. Thus each party P_i holds a polynomial $d_i(x)$ and an integer δ_i. The system invariant that we maintain is that for every party i, $d_i(x) \equiv \delta_i f(x,i) \bmod m$. Note that this trivially holds for all the original parties (for which $\delta_i = 1$).

Incorporating a New Party: When a new party P_{new} joins the group, it needs to receive its shares $d_{new}(x)$ and δ_{new}, while maintaining the system invariant that $d_{new}(x) \equiv \delta_{new} f(x, new) \bmod m$. This is done by having party P_{new}:

1. Receive from every P_{i_j}, $j = 1, \ldots, t+1$, the values $\alpha_{i_j} = d_{i_j}(new)$ and δ_{i_j}.
2. Let $\delta = \mathrm{lcm}(\delta_{i_1}, \ldots, \delta_{i_{t+1}})$ and recall the definition of Δ_S (Eq. 3) for the set $S = \{i_1, \ldots, i_{t+1}\}$. Compute its share as:

$$d_{new}(x) = \delta \Delta_S f_{new}(x) = \sum_{j=1}^{t+1} \Delta_S L_S(x, i_j) \frac{\delta}{\delta_{i_j}} \alpha_{i_j} \tag{7}$$

3. Stores as its share this interpolated polynomial $d_{new}(x)$ and the value $\delta_{new} = \delta \cdot \Delta_S$.

To see that this computation gives the new party its appropriate share preserving the invariant and that it is feasible to compute this value consider the following. By definition $f_{new}(x) = f(x, new)$. Note that due to the symmetry of $f(x,y)$ and the system invariant, if P_i is a party already in the network, then

$$f_{new}(i) = f(i, new) = f(new, i) = \frac{d_i(new)}{\delta_i} \bmod m \tag{8}$$

though we can't explicitly compute the last fraction mod m.

Given $\delta = \mathrm{lcm}(\delta_{i_1}, \ldots, \delta_{i_{t+1}})$ as defined in the protocol, multiplying both sides of Eq.8 by δ and specifying $i = i_j$ for all $j = 1, \ldots, t+1$ we have:

$$\delta f_{new}(i_j) = \frac{\delta}{\delta_{i_j}} d_i(new) \bmod m$$

Notice that now the fraction is an integer and can be computed even without knowing m.

Given the values $\alpha_{i_j} = d_{i_j}(new)$ and δ_{i_j} for $j = 1, \ldots, t+1$, we can interpolate the polynomial $\delta f_{new}(x)$ as

$$\delta f_{new}(x) \stackrel{\text{def}}{=} \sum_{j=1}^{t+1} L_S(x, i_j) \frac{\delta}{\delta_{i_j}} \alpha_{i_j} \bmod m \tag{9}$$

where $L_S(x, i_j)$ is the appropriate Lagrangian coefficient (see Eq. 1). Yet, as this computation needs to be computed mod m and requires the calculation of the Lagrangian coefficients mod m this cannot be done directly. To enable the computation we employ the techniques described in the previous sections. We multiply Eq.(9) by Δ_S resulting in the computation of Eq. (7). Notice that multiplying by Δ_S removes all the denominators on the right-hand side of the equation. Furthermore, setting $\delta_{new} = \delta \cdot \Delta_S$ preserves the system invariant.

Signature Computation Phase: Assume that we want to compute the signature σ on a message M. Let $y = H(M)$, then we have $\sigma = y^d \bmod N$.

Recall that each party P_i holds the polynomial $d_i(x)$ and the integer δ_i such that $d_i(x) = \delta_i f(x,i) \bmod m$. Thus each party P_i publishes as its signature fragment the pair $(\sigma_i = y^{2^{kt} d_i(0)}, \delta_i)$.

Given $t+1$ of these signature fragments published by parties $P_{i_1}, \ldots, P_{i_{t+1}}$, δ and Δ_S as above, set

$$\sigma' = \prod_{j=1}^{t+1} \sigma_{i_j}^{\frac{\delta}{\delta_{i_j}} \cdot \Delta_S \cdot L_S(0,i_j)} \bmod N \tag{10}$$

Let $e' = 2^{kt} \cdot \delta \cdot \Delta_S$, compute integers a, b such that $ae + be' = 1$. Set the signature to:

$$\sigma = y^a \sigma'^b \bmod N$$

We show that the above computation generates a proper signature. First note that all the exponents in Eq. (10) are integers (as Δ_S removes the denominators from the Lagrangians, and δ_{i_j} divides δ by definition). Furthermore,

$$\sigma' = y^{\sum_{j=1}^{t+1} 2^{kt} \cdot \frac{\delta}{\delta_{i_j}} \cdot \Delta_S \cdot L_S(0,i_j) d_{i_j}(0)} \bmod N \tag{11}$$

Let's focus on the exponent of y in the above equation: by using the invariant $d_{i_j}(0) = \delta_{i_j} f(0, i_j)$ we have that the exponent equals:

$$\sum_{j=1}^{t+1} 2^{kt} \cdot \delta \cdot \Delta_S \cdot L_S(0,i_j) f(0,i_j)$$

From the polynomial interpolation we have that $\sum_{j=1}^{t+1} L_S(0,i_j) f(0,i_j) = f(0,0) = d$ therefore

$$\sigma' = y^{2^{kt} \cdot \delta \cdot \Delta_S \cdot d} \bmod N \tag{12}$$

Assuming that $GCD(e, e') = 1$ we use the techniques from the previous section to extract the signature σ out of σ'. Indeed $\sigma' = \sigma^{e'} \bmod N$ and there exists integers a, b such that $ae + be' = 1$. Therefore

$$\sigma = \sigma^{ae+be'} = y^a \sigma'^b \bmod N$$

We remark that if we choose e as a prime larger than 2^k (the largest possible identity), then $GCD(e, e')$ is guaranteed to be 1 as the value e' can only be the product of differences of identities.

The proofs follow directly from the proofs of the previous section while incorporating the extra factor δ in the simulation.

Remark: Notice that the shares held by parties in this modified scheme are larger. Just because of the bivariate polynomial technique, each party holds $t+1$ values mod m rather than a single one. Moreover the size of the shares of parties added later in the system grow with the additive factor $\log \delta + \log \Delta_S$ (notice that Eq. (7) is computed over the integers by P_{new}).

Remark: Fazio *et al.* in [12] consider Shoup's original protocol and show how to add parties without using bivariate polynomials. Their work is not directly applicable to "ad hoc" networks as it requires *all* parties in the network to participate in assigning a share to a new party. On the other hand their solution does not increase the share size by a factor of t and may have more enhanced properties, such as proactive security.

6 Adding Robustness

The protocols described in the previous sections work only in the presence of an honest-but-curious adversary. Here we show how to tolerate a malicious adversary.

During the sharing phase, the dealer chooses a random value $g \in Z_N^*$ (with high probability g has order m) and publishes the values $G_{i,j} = g^{a_{i,j}} \bmod N$ for all the coefficients $a_{i,j}$ of the sharing polynomial f. Notice that this allows any party to compute $g^{f(i,j)}$ on any point (i,j) by "polynomial evaluation in the exponent".

When a new party P_{new} joins the network, it receives from an existing party P_i the values δ_i and $\alpha_i = d_i(new) = \delta_i f(new, i)$. Then P_{new} checks that $g^{\alpha_i} = [g^{f(new,i)}]^{\delta_i} \bmod N$ where $g^{f(new,i)}$ is computed using the values $G_{i,j}$ published by the dealer.

When computing a signature on a message M, where $y = H(M)$, a party P_i publishes the values δ_i and $\sigma_i = y^{2ktd_i(0)} \bmod N$, where $d_i(0) = \delta_i f(0,i)$ and proves that

$$\log_g [g^{f(0,i)}]^{\delta_i} = \log_{y^{2kt+1}} \sigma_i^2 \bmod m$$

(the extra squaring operation is needed to make sure that we are in the subgroup of order m in Z_N^*). Only signature fragments that pass the above verification test will be accepted.

Efficient zero-knowledge proofs for this language were presented in [19] and [18]. The ZK proof presented in [18] can be made non-interactive using the Fiat-Shamir heuristic in the random oracle model.

Acknowledgment. We are grateful to Stas Jarecki and the anonymous Eurocrypt reviewers for their helpful comments. Research was sponsored by US Army Research laboratory and the UK Ministry of Defense and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the US Army Research Laboratory, the U.S. Government, the UK Ministry of

Defense, or the UK Government. The US and UK Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

References

1. M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness Theorems for Non-cryptographic Fault-Tolerant Distributed Computations. In *Proc. 20th STOC*, pages 1–10. ACM, 1988.
2. G. R. Blakley. Safeguarding cryptographic keys. In *Proc. AFIPS 1979 National Computer Conference*, pages 313–317. AFIPS, 1979.
3. C. Boyd. Digital Multisignatures. In H. Baker and F. Piper, editors, *Cryptography and Coding*, pages 241–246. Claredon Press, 1989.
4. M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *1st ACM Conference on Computer and Communications Security*, pages 62–73, 1993.
5. R. A. Croft and S. P. Harris. Public-key cryptography and re-usable shared secrets. In H. Baker and F. Piper, editors, *Cryptography and Coding*, pages 189–201. Claredon Press, 1989.
6. M. Cerecedo, T. Matsumoto, and H. Imai. Efficient and secure multiparty generation of digital signatures based on discrete logarithms. *IEICE Trans. Fundamentals*, E76-A(4):532–545, 1993.
7. Jean-Sébastien Coron. On the Exact Security of Full Domain Hash. In *Advances in Cryptology - CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 229–235. Springer, 2000.
8. Alfredo De Santis, Yvo Desmedt, Yair Frankel, and Moti Yung. How to share a function securely. In *Proc. 26th STOC*, pages 522–533. ACM, 1994.
9. Y. Desmedt. Society and group oriented cryptography: A new concept. In *Crypto '87*, pages 120–127, 1987. LNCS No. 293.
10. Y. G. Desmedt. Threshold cryptography. *European Transactions on Telecommunications*, 5(4):449–457, July 1994.
11. Y. Desmedt and Y. Frankel. Shared generation of authenticators and signatures. In *Crypto '91*, pages 457–469, 1991. LNCS No. 576.
12. N. Fazio. Personal Communication. September 2007.
13. Y. Frankel, P. Gemmell, P. Mackenzie, and M. Yung. Proactive RSA. In *Crypto '97*, pages 440–454. LNCS No. 1294.
14. Y. Frankel, P. Gemmell, and M. Yung. Witness-based Cryptographic Program Checking and Robust Function Sharing. In *Proc. 28th STOC*, pages 499–508. ACM, 1996.
15. R. Gennaro, S. Halevi, H. Krawczyk, and T. Rabin. Threshold RSA for Dynamic and Ad-Hoc Groups. Full version available on-line from `http://eprint.iacr.org/2008/045`
16. R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust and efficient sharing of RSA functions. In *Crypto '96*, pages 157–172, 1996. LNCS No. 1109.
17. R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust Threshold DSS Signatures. *Information and Computation*, 164(1):54–84, 2001. Extended abstract in EUROCRYPT'96.
18. R. Gennaro, H. Krawczyk, and T. Rabin. RSA-based Undeniable Signatures. In *Crypto '97*, pages 132–149. LNCS No. 1294. Final version in *J. Cryptology* 13(4): 397-416 (2000).

19. R. Gennaro, H. Krawczyk, and T. Rabin. Robust and Efficient Sharing of RSA Functions. *Journal of Cryptology*, 13(2):273–300, 2000. Conference version [16].
20. Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. *SIAM J. Computing*, 17(2):281–308, April 1988.
21. Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. Communication of the ACM, 21(2):120–126, 1978.
22. A. Shamir. How to Share a Secret. *Communications of the ACM*, 22:612–613, 1979.
23. A. Shamir. On the Generation of Cryptographically Strong Pseudorandom Sequences. *ACM Trans. Comput. Syst.*, 1(1):38–44, 1983.
24. V. Shoup. Practical threshold signatures. In *Eurocrypt '00*, pages 207–220, 2000. LNCS No. 1807.
25. Nitesh Saxena, Gene Tsudik, and Jeong Hyun Yi. Efficient node admission for short-lived mobile ad hoc networks. In *ICNP, 13th IEEE International Conference on Network Protocols*, pages 269–278, 2005.