Efficient Threshold RSA Signatures with General Moduli and No Extra Assumptions

Ivan Damgård and Kasper Dupont*

Dept. of Computer Science, Aarhus University

Abstract. We propose techniques that allow construction of robust threshold RSA signature schemes that can work without a trusted dealer using known key generation protocols and is as efficient as the best previous schemes. We do not need special conditions on the RSA modulus, extra complexity or set-up assumptions or random oracles. An "optimistic" variant of the scheme is even more efficient in case no faults occur. Some potential more general applications of our basic idea are also pointed out.

1 Introduction

In a threshold public-key system we have a standard public key (for the RSA system, for instance), while the private key is shared among a set of servers, in such a way that by collaborating, these servers can apply the private key operation to a given input, to decrypt it or sign it, as the case may be. If there are l servers, such schemes typically ensure that even if an active adversary corrupts less than $l/2$ servers, he will not learn additional information about the private key, and will be unable to force the network to compute incorrect results. Thus threshold cryptography is an important concept because it can improve substantially the reliability and security of applications in practice of public-key systems.

The most efficient known robust threshold RSA signature scheme was proposed by Shoup [25] (see [9] for some of the first work in this direction and [18] for a more efficient solution in case of passive attacks). Shoup's scheme needs the RSA modulus n to be a product of *safe primes*, that is, besides $n = pq$ where p, q are prime, we require $p = 2p' + 1, q = 2q' + 1$ and p', q' are also primes. When Shoup proposed his scheme, it was not known how to generate efficiently such an RSA key in a *distributed* way, i.e., such that the servers generate the key from scratch without the secret key ever becoming known to a single entity. Shoup's scheme therefore assumed a trusted dealer generating the keys – although a distributed key generation would of course have been more satisfactory since it completely avoids any single points of attack.

* Both authors supported BRICS, Basic Research in Computer Science, Center of the Danish National Research Foundation, and FICS, Foundations in Cryptography and Security, Center of the Danish Science Research Council.

S. Vaudenay (Ed.): PKC 2005, LNCS 3386, pp. 346–361, 2005.

It was already known how to generate *general* (random) RSA keys via a distributed protocol [2, 10], but such keys are safe prime products with only negligible probability. Later, in [1], Algesheimer, Camenisch and Shoup propose a RSA key generation protocol that can also generate safe prime products in a reasonable amount of time, in the sense that their method will be much faster than employing generic multiparty computation methods.

Despite this result, there are good reasons for considering threshold RSA schemes that can use general RSA keys: we do not know if there are infinitely many safe primes, and in any case, safe prime products constitute a small fraction of the possible RSA keys. Thus it could in principle be the case that safe prime products are easy to factor, while the general RSA assumption is still true. We stress that nothing is known to suggest that this is the case, but in general most experts agree that the most sound approach is to use RSA keys with as few special constraints as possible. Furthermore, generating safe primes is slower than generating random primes, simply because there are so few of them: to generate a random k-bit prime, we need to examine $O(k)$ candidates before finding a prime, but (from heuristic arguments) we need $O(k^2)$ candidates before finding a safe prime. Most candidates can be ruled out using trial division, so the extra cost for safe primes may not be so significant in a traditional scenario where a single party generates keys. But it is much more painful in a distributed key generation protocol, since here even a simple trial division costs communication.

It is in fact possible to use more general RSA moduli: in [8], Damgård and Koprowski propose a threshold RSA scheme which is as efficient as Shoup's and which can use a much more general class of moduli. However, this comes at the expense an extra and non-standard intractability assumptions, on top of the basic RSA assumption (which is of course necessary). Independently, Fouque and Stern [14] suggested a different approach that is based only on the RSA assumption, but is significantly less efficient than [25, 8]. All these schemes need the random oracles to make the signing protocol be non-interactive. One can do without them at the expense of extra interaction, but doing it in a constant number of rounds requires extra set-up assumptions.

More recently, Cramer and Damgård propose a technique known as secret-key zero-knowledge[5]. They suggest applying this to threshold RSA, this way one obtains non-interactive protocols without random oracles. On the other hand, the modulus is restricted in the same way as for Shoup's scheme and extra key set-up assumptions are needed.

In this paper, we propose new threshold RSA schemes which are as efficient as [25, 8], they do not need the extra intractability assumptions introduced in [8], nor extra key set-up assumptions. To understand how this is possible, recall that in virtually any proposed threshold RSA scheme, each server must contribute a partial result computed from its own share of the private key, plus it must prove in zero-knowledge to the client requesting the signature that this partial result is correct. We then observe that a minor change in the algorithm that computes the signature from all contributions allows us to make do with a much larger (non-negligible) error probability for the zero-knowledge proofs. This is a very generic

idea that can be applied to most known threshold RSA schemes. Now, since the restrictions on the RSA moduli in previous schemes were typically nedeed to have a negligible error probability, we no longer need these restrictions.

Working out the details of this can be more or less straightforward, depending on which of the previous RSA schemes we start from. For instance, if we start from Shoup's scheme, there are indeed a few technicalities to sort out, and we do this in detail in the last part of the paper. Since we want to avoid random oracles and extra set-up assumptions, we cannot get a protocol that is always non-interactive, but we can get one that requires at most 3 moves, and only 1 if servers behave correctly (as they would most of the time in practice). Note that this would not have been possible if we had used zero-knowledge proofs in the standard way. In any case, the total communication and computational complexity is comparable to that of [25, 8]. Our schemes comes in several variants:

- The most efficient variant can be proved secure, based on an assumption that implies the RSA assumption. We conjecture that they are in fact equivalent (and we can prove this in the random oracle model). The modulus $n = pq$ must satisfy that $(p-1)/2, (q-1)/2$ have no prime factors smaller than $3t^2$, where t is the maximal number of corrupted severs.
- A slightly more complex version that is slower than the basic one by a constant factor, but can be proved secure under the RSA assumption. It uses the same clas of moduli as the basic one.
- A variant that can use *any* RSA modulus and is secure under the RSA assumption. Its complexity is higher than the basic one by a factor of $\log_2 3 + 2\log_2 t$ – in practice, this is usually a rather small price to pay.

In the last section of the paper, we point out some more general applications of our basic idea, in particular, any threshold signature scheme, but also threshold cryptosystems based on polynomial secret sharing could benefit from our technique.

2 Model

Here we describe the model for threshold signature schemes we use, rather informally, due to space limitations. In the type of schemes we consider there are l *servers* and one *client*. In the *generation phase* on input a security parameter k the public key pk and *secret key shares* $s_1, ..., s_l$ are created, where s_i belongs to server number i. There is a *signing protocol* defined for the servers and the client, which takes a message M as input and outputs (publically) a signature σ.[1] Finally, there is a *verification predicate* V, which is efficiently computable, takes pk, message M and signature σ as inputs, and returns *accept* or *reject*. Both

[1] Thus, we are in fact asuming (as usual in threshold signature schemes) that the client and servers agree on which message is to be signed. In practice, the implementation or application will have to ensure this. This is reflected in the model by not allowing the adversary to send inconsistent signing requests to servers, even if the client is corrupted.

the signing protocol and the verification predicate may make use of a random oracle (although most of the schemes we consider here do not).

To define security, we assume a polynomially bounded static and active adversary $\mathcal{A}$, who corrupts initially $t < l/2$ of the l servers, and possibly the client. Thus, the adversary always learns pk and the s_i's of corrupted servers. As the adversary's algorithm is executed, he may issue two types of requests:

- An *oracle request*, where he queries the random oracle used, he is then given the oracle's answer to the query he specified. Of course, this is only relevant if the protocol uses a random oracle.
- A signature request, where the adversary specifies a message M. This causes the signing protocol to be executed on input M, where the adversary controls the behaviour of corrupted servers and of the client if he is corrupt. The adversary will of course see whatever information is made public by honest servers.

At the end, $\mathcal{A}$ outputs a message M_0 and a signature σ_0.

We say the scheme is secure if the following two conditions hold for any adversary A:

Robustness: If the client is honest, each signature request results in the client computing a correct signature on M in expected polynomial time.

Unforgeability: The following happens with probability negligible in k: A outputs M_0, σ_0 such that M_0 was not used in a previous signature request, and $V(pk, M_0, \sigma_0) = accept$

3 Some Observations on Error Probabilities

Our first observations can be understood without bothering about the lower level details of threshold RSA schemes. So assume we start from the schemes of [25] or [8] which on a high level are completely similar: we are given as RSA public key n, e, and each of the l servers hold a share of the private key, s_i for the i'th server. In addition, there are some public verification keys, a global one v and a special verification key v_i for each server. These are used to verify that servers behave correctly. It is assumed that an adversary may initially corrupt up to $t < l/2$ servers and make them behave as he likes.

Given an input x to sign. We assume that x is the message as it looks after possible hashing and padding, so that the purpose is simply to compute the correct RSA root of x. We denote by H whatever process that leads from the actual message M to x, so that $x = H(M)$. Now, server i computes a signature share x_i, and gives a zero-knowledge proof $proof_i$, that x_i was correctly computed. Now, computing the signature takes place in two steps: First we discard all x_i corresponding to proofs that were rejected, leaving a set $\{x_i |\ i \in S_0\}$ of signature shares, where $S_0 = \{i |\ proof_i \text{ was accepted}\}$. We have $|S_0| \geq t + 1$, since at least $t + 1$ servers are honest. Second, we run an algorithm *Combine* on inputs $\{x_i |\ i \in S_0\}, m, n, e$ which is guaranteed to output the correct signature on m, if $|S_0| \geq t + 1$ and all shares in $\{x_i |\ i \in S_0\}$ are correct. This last condition is

satisfied except with negligible probability since in [8, 25] the proofs are designed such that the adversary can give an acceptable proof for an incorrect x_i with only negligible probability.

Now, an initial observation – first made in [23] – is that since one can always verify if the output from R is correct, one can always compute the signature, even if no proofs were available:

– For every subset $S \subset \{1, 2, ..., l\}$ of size $t+1$, do: Compute $sig := R(\{x_i |\ i \in S\}, m, n, e)$. If sig is a correct signature on m w.r.t. public key n, e, output sig and stop.

The problem with this algorithm is of course that it is inefficient for large l, t, in fact it takes exponential time in l, if $t \approx l/2$ because there are exponentially many subsets to try, so this may be unpleasant already for moderately large t, l.

However, a similar idea might still work, if we first use the proofs to reduce the number of incorrect signature shares from t down to something "sufficiently close" to 0. Our main point is that this can be done, even if there is a *non-negligible* chance of giving an acceptable proof for a bad signature share. We do the following:

Algorithm **Extended-Combine**

– For all $i = 1..l$ receive signature share x_i from server i, and let server i give a proof $proof_i$ that x_i is correct. Let $S_0 = \{i |\ proof_i$ was accepted$\}$.
– For every subset $S \subset S_0$ of size $t+1$, do: Compute $sig := Combine(\{x_i |\ i \in S\}, m, n, e)$, where $Combine$ is the algorithm mentioned above. If sig is a correct signature on m w.r.t. public key n, e, output sig and stop.

Assume a worst case situation, where t is maximal, so that we have only $t+1$ honest servers, and furthermore all corrupt servers supply incorrect signature shares. Let $p(t)$ be the soundness error for the interactive proofs that servers use to prove correctness of signature shares. In other words, if a signature share x_i is incorrect, then the proof given by server i is accepted with probability at most $p(t)$ (we assume we can control this error probability so that it is some function of t). Then, for any $i \in \{0, 1, ..., t\}$, the probability that i of the t proofs for incorrect signature schares are accepted is at most $\binom{t}{i} p(t)^i (1 - p(t))^{t-i}$. If i proofs are accepted, S_0 contains $t + 1 + i$ signature shares, and we need to find the subset of size $t+1$ corresponding to the $t+1$ correct signature shares. Hence, the expected time spent to compute the signature using the algorithm sketched above is proportional to (at most)

$$E(t) := \sum_{i=0}^{t} \binom{t+1+i}{t+1} \binom{t}{i} p(t)^i (1 - p(t))^{t-i}$$

Lemma 1. *If $p(t) \leq 1/ct^2$ for some constant $c > 2$, then (as a function of l and and t), the expected number of subsets tested by $Extended - Combine$ is in $O(1)$.*

Proof. It is sufficient to show that $E(t)$ is in $O(1)$. Choose a c' such that $2 < c' < c$, then for any $t > 1/(c'-2)$:

$$E(t) = \sum_{i=0}^{t} \binom{t+1+i}{t+1} \binom{t}{i} p(t)^i (1-p(t))^{t-i} \tag{1}$$

$$\leq \sum_{i=0}^{t} \binom{t+1+i}{t+1} \binom{t}{i} p(t)^i \tag{2}$$

$$\leq \sum_{i=0}^{t} (t+1+i)^i t^i p(t)^i \tag{3}$$

$$\leq \sum_{i=0}^{t} (c't)^i t^i p(t)^i \tag{4}$$

$$= \sum_{i=0}^{t} (c't^2 p(t))^i \tag{5}$$

$$\leq \sum_{i=0}^{t} (c't^2 \frac{1}{ct^2})^i \tag{6}$$

$$= \sum_{i=0}^{t} (\frac{c'}{c})^i \tag{7}$$

$$= \frac{1-(c'/c)^{t+1}}{1-c'/c} \tag{8}$$

$$\leq \frac{1}{1-c'/c} \tag{9}$$

In (3) we use the fact that $\binom{a}{b} \leq a^b$. In (4) we use the assumption about t and the fact $i \leq t$. In (6) we use the assumption about $p(t)$ In (8) we use the well known formula $\sum_{i=0}^{n} x^i = \frac{1-x^{n+1}}{1-x}$ for $x \neq 1$.

We remark that concrete caluations suggest that it may be possible to prove this lemma assuming $p(t) \leq 1/ct^a$ for a slightly smaller than 2, but that $a = 1$ would not work.

4 A Threshold RSA Scheme

The scheme we describe in this section follows to a large extent the approach of [8, 25]. The new ingredient is the way in which signatures shares are verified, where we use the observations we made in the previous section. Concretely, this means that the zero-knowledge proofs given for correctness of signature shares are interactive, using 3 moves. But on the other hand, since we can make do with a non-negligible soundness error, we only need short random challenges from the verifier, and this means that the proofs can be shown to be zero-knowledge and sound in the standard model without using random oracles.

4.1 Key Set-Up

We describe here the setup of keys as a trusted dealer D would do it. However, this dealer can be replaced by any of the known distributed RSA key generation protocols. Using a particular such protocol may affect slightly the way the secret sharing of the secret exponent is done (see below). Any such change can easily be accomodated, however.

1. D chooses a k-bit RSA modulus $n = pq$ where the primes p, q satisfy that $(p-1)/2, (q-1)/2$ have no prime factors less than $3t^2$, where t is the maximal number of corrupted servers we want to tolerate. Let l be the total number of servers, and $\Delta = l!$. In addition, $\phi(n)$ must be prime to the public exponent e, which must be a prime such that $e > l$. We set $d = e^{-1} \bmod \phi(n)$.
2. D chooses a random polynomial $f(x)$ of degree at most t with integer coefficients, such that $f(0) = d$;

$$f(x) = d + c_1 x + ... + c_t x^t$$

   where the c_i's are random independent integers chosen from the interval $[0..\Delta n 2^t 2^L]$, where L is a secondary security parameter. The secret share of the i'th server is $s_i = f(i)$. With the given choice of coefficients, it can be shown that, if we compare the distribution of any t shares resulting from sharing d with the one resulting from sharing any other d', the statistical distance between the two is at most 2^{-L} [19].
3. D chooses a random square v modulo n. For the i'th server, the verification key v_i is $v_i = v^{\Delta s_i} \bmod n$.
4. The public information is now $n, e, v, v_1, ..., v_l$, while each server i has s_i as its private information.

Note that by construction of n, any square modulo n has order not divisible by any prime less than $3t^2$.

4.2 An Auxiliary Protocol

Suppose we are given elements in Z_n^*, v, w, α, β. Also given is an integer B of size polynomial in k, the security parameter. We assume it is guranteed that the orders of v, w, α and β are not divisible by any prime less than B. Finally, a prover P is given an integer s such that $v^s = w \bmod n$, and now P wants to convince us that $\alpha^s = \beta \bmod n$. We use the following variant of a standard protocol (a similar protocol was used in [13, 8, 25]):

1. P chooses r, a random integer of bitlength $\log_2(s) + \log_2(B) + L$. and sends $a = v^r, b = \alpha^r$.
2. The verifier chooses a random challenge c, with $0 \leq c < B$.
3. P replies by sending $z = r + cs$
4. To check the proof, one verifies that $v^z = aw^c$ and $\alpha^z = b\beta^c$.

It is trivial to see that if $\alpha^s = \beta$ and P follows the protocol, the verifier will always accept. Moreover, a standard rewinding argument shows that the protocol

is statistical zero-knowledge, since the number of challenges is polynomial and r is chosen to be exponentially (in L) larger than cs. For the soundness part, we have the following result, which asserts that if the claim is wrong, then the prover can only have the verifier accept with unusually large probability if he can solve a supposedly hard problem:

Lemma 2. *Let $n, v, w, \alpha, \beta, s, B$ be given as described above, and suppose $\alpha^s \neq \beta \bmod n$. Let P^* be any prover in the above protocol, and fix any set of random coins for P^*. If the probability (given these random coins) that the verifier accepts is larger than $1/B$, then: using P^* as oracle, one can easily compute either a μ'th root of v modulo n , $1 < \mu < B$, or a multiple of the order of v in Z_n^*.*

Remark 1. A very similar protocol and analysis was presented in [13]. The difference is that here, we are satisfied with a non-negligible error probability and hence we can use small challenges. This is what implies that to break the soundness, the adversary must find a root of v with "public exponent" in a very small set (between 1 and B). We can therefore base security essentially on the standard RSA assumption, rather than the strong RSA assumption as in [13].

Proof. The claimed algorithm will simply send all possible challenges to P^* (rewinding in between) and record all answers. Since the number of possible challenges is B and the accept probability was larger than $1/B$, we must get good answers to at least 2 distinct challenges c, c'. So we have values a, b, c, c', z, z' such that $v^z = aw^c$, $\alpha^z = b\beta^c$, $v^{z'} = aw^{c'}$ and $\alpha^{z'} = b\beta^{c'}$ (all equations modulo n). It follows that

$$v^{z-z'} = w^{c-c'}, \quad \alpha^{z-z'} = \beta^{c-c'}.$$

Now, let $d = gcd(z - z', c - c') < B$. By assumption on the orders of v, w, α, β

$$v^{(z-z')/d} = w^{(c-c')/d}, \quad \alpha^{(z-z')/d} = \beta^{(c-c')/d}.$$

Now, if $d < c - c'$, take integers γ, δ such that $1 = \gamma(z - z')/d + \delta(c - c')/d$. Using the relation we just derived, we get:

$$v = v^{\gamma(z-z')/d+\delta(c-c')/d} = \left(w^{\gamma} v^{\delta}\right)^{(c-c')/d}$$

which is, as promised, a non trivial μ'th root of v, where $\mu = (c - c')/d$. On the other hand, if $d = c - c'$, we have in fact that

$$v^{(z-z')/(c-c')} = w, \quad \alpha^{(z-z')/(c-c')} = \beta.$$

It must be the case that $s \neq (z - z')/(c - c')$ since otherwise we get a contradiction with $\alpha^s \neq \beta$, so this and $v^s = w$ implies that the order of v divides $s - (z - z')/(c - c')$.

4.3 Signing a Message

We can now describe how the threshold RSA scheme will work. We give first a basic version, which we later show how to modify to get a more practical scheme or to reduce the necessary assumptions.

- When a client requests that message $x = H(M)$ be signed, server i will compute a signature share as $x_i = x^{2\Delta s_i} \bmod n$.
- The server then proves that x_i is correct. This proof will consist of proving in ZK that the discrete log of x_i^2 base x^4 equals the discrete log of v_i base v (namely Δs_i), although the proof will have a non-negligible error probability. For this we may use the auxiliary protocol given above, with parameters $n, B = 3t^2, v, w = v_i, \alpha = x^4 \bmod n, \beta = x_i^2 \bmod n$ and $s = \Delta s_i$. It is easy to see that with our choices of n, v, this will satisfy the conditions we stated for the auxiliary protocol.
- The client now attempts to find a $t+1$-subset of signature shares with accepted proofs that leads to the correct signature being computed, i.e., we use the $Extended - Combine$ from the previous section. Concretely, let S be a $t+1$-subset of the indices $1, .., l$, and define interpolation coefficients

$$\lambda_{0,j}^S = \Delta \prod_{i \in S \setminus \{j\}} \frac{i}{i-j}$$

  These are integers, and we have $d\Delta = f(0)\Delta = \sum_{j \in S} \lambda_{0,j}^S s_j$. If for all signature shares in S, it is indeed the case that server's claim about x_i is true, i.e., $(x^4)^{\Delta s_i} = x_i^2 \bmod n$, then we can compute

$$\omega = \prod_{j \in S} x_j^{2\lambda_{0,j}^S} = x^{4\Delta^2 d} \bmod n$$

- Note that $\omega^e = x^{4\Delta^2} \bmod n$. But since e is prime to $4\Delta^2$, we can take integers a, b such that $a4\Delta^2 + be = 1$. It now follows easily that $y = \omega^a x^b \bmod n$ is the desired RSA signature, provided the subset we tried consisted of correct signature shares. If the signature does not verify, we try the next subset.

We base the security on the following assumptions. First a variant of the standard RSA assumption:

Conjecture 1. Let a k-bit RSA modulus n and t chosen as described above be given, and let $w \in Z_n^*$ be uniformly chosen. Given this input, any probabilistic polynomial time algorithm computes a μ'th root of $w \bmod n$, where $1 < \mu < 3t^2$, with negligible probability (in k).

The only difference to standard RSA is that the public exponent is not fixed to a single value but must be in a small given set. This is in contrast to the *strong* RSA assumption, where the adversary can choose an arbitrary public exponent.

Conjecture 2. Let n, e, t be chosen as described above be given and let $w \in Z_n^*$ be uniformly chosen. Suppose an oracle is also given that on input message M will return y such that $y^e = H(M) \bmod n$. Given this input and oracle, any probabilistic polynomial time algorithm computes a μ'th root of $w \bmod n$, where $1 < \mu < 3t^2$, with negligible probability.

The second assumption is clearly at least as strong as the first, but they may well be equivalent, namely if access to the e'th root oracle does not help to compute μ'th roots. This seems reasonable since e is by assumption prime to any allowed μ-value, and the adversary cannot even choose freely the numbers on which e'th roots are computed. Indeed, if we model H as a (full domain)random oracle, the assumptions are provably equivalent since then, using standard tricks, the e'th root oracle is easy to implement without knowing the factors of n.

The basic variant of the threshold RSA scheme that we already presented can be proved secure under Conjecture 2. Before doing this, we need two auxiliary lemmas:

Lemma 3. *Let n, e, distributed as the honest dealer chooses them, be given. Furthermore, let w, a random square in Z_n^* be given. Based on this, the information the adversary learns from the honest dealer initially can be simulated efficiently with a statistically close distribution, and with $v = w^e \bmod n$.*

Proof. Note that the information seen by the adversary is n, e, v, the shares of corrupted players, and the public verification values v_i of all players.

We begin by setting $v = w^e \bmod n$, and so we have $w = v^d \bmod n$. Without loss of generality, assume the adversary corrupts the first t players. Perform now a sharing of an arbitrary value d' (say, $d' = 1$) according to the algorithm used by the dealer, and let $s_1, ..., s_t$ be the shares for the corrupted players resulting from this. By the privacy of the secret sharing, this is statistically close to the distribution resulting from sharing the correct d. Hence, except with negligible probability, there exists a polynomial $f(x)$ of degree at most t and with coefficients in the correct range, such that $f(0) = d$ and $f(i) = s_i$, $i = 1...t$. So we have $w = v^d = v^{f(0)} \bmod n$. Define $S = \{0, 1, ..., t\}$. Recall that we earlier defined $\lambda_{i,j}^S$, the standard Lagrange interpolation coefficients multiplied by Δ. We can now compute, for honest plyer i:

$$w \cdot \prod_{j=1}^{t} (v^{s_j})^{\lambda_{i,j}^S} = \prod_{j=0}^{t} (v^{f(j)})^{\lambda_{i,j}^S} = v^{\Delta f(i)}$$

which is by definition exactly the public verification values that results for honest players when the dealer chooses $f(x)$ for the sharing of d. We can therefore output $n, e, s_1, ..., s_t, v, v^{\Delta f(1)}, ..., v^{\Delta f(l)}$.

Lemma 4. *Assume we are given a set of values distributed by the honest dealer to the adversary, i.e., $n, e, v, v_1, v_2, ..., v_l$ and the s_i's sent to the corrupt servers. Let also a message M, and the signature $H(M)^d \bmod n$ be given. Based on this, the contributions from honest servers in the protocol where M is signed can be simulated efficiently with the correct distribution.*

Proof. Let $f(x)$ be the polynomial used by the dealer to share d. Since we are given $H(M)^d = H(M)^{f(0)} \bmod n$, and we know the shares of corrupted players, we can compute what we need by interpolation "in the exponent" similarly to the proof of the previous lemma. Assume without loss of generality that the first t players are corrupt. We then compute, for honest player i:

$$\prod_{j=0}^{t}(H(M)^{f(j)})^{\lambda_{i,j}^{S}} = H(M)^{\Delta f(i)}$$

which is by definition the signature share contributed by this player.

Theorem 1. *The threshold RSA scheme defined in this section is secure under Conjecture 2 and assuming the underlying RSA signature scheme is chosen message attack secure. In the random oracle model, we can replace Conjecture 2 by Conjecture 1.*

Proof. We will show that if there exists an adversary A that breaks the above threshold RSA scheme, there exists an expected poly-time adversary A' that either breaks the underlying RSA signature scheme under a chosen message attack, or contradicts Conjecture 2.

Our claimed adversary A' will be given just public key n, e and access to an e'th root oracle, or equivalently, a chosen message attack on the underlying signature scheme. Then A' will start a copy of A, choose w as a random square modulo n and run the simulation from Lemma 3 and in this way produce values v, v_i for everyone, and s_i for those t servers A wanted to corrupt.

When A wants to have some x signed, A' uses the chosen message attack to get a signature on x, uses Lemma 4 to compute the x_i of honest servers and the zero-knowledge property to simulate their proofs. Note that if the client is corrupt, this involves simulating the proofs of honest servers where A acts as verifier. If the client is honest, A' just executes the normal client algorithm. In this way, A' simulates an entire execution of A and outputs whatever A outputs.

Note that to break the threshold scheme, A must either violate robustness or unforgeability. Since A' does a statistically close simulation of A's attack, either event happens with essentially the same probability in A''s execution as in real life.

Assume first that A violates robustness. Since the client by definition keeps going until it finds a correct signature (and eventually it will always succeed), A can only violate this property by creating a situation whete the expected time spent by the client is larger than specified. By Lemma 1, this can only happen, if at least one incorrect signature share is accepted with probability larger than $1/ct^2$, which is $1/3t^2$ in this particular case. By Lemma 2, this implies that either that we can find a μ'th root of v where $1 < \mu < 3t^2$, and hence of w since $w = v^e \bmod n$ and e is guaranteed to be prime to μ. Or we can find a multiple of the order of v and hence of w. Note that instead of choosing w, A' could take a random input and use this as w. Such an input will be a square with large probability (1/4). Hence, in the first case, we can directly break Conjecture 2, in the second case we note that ability to find the order of random elements in Z_n^* implies we can factor n using a well known reduction, and so we can in particular break Conjecture 2.

Now, assume A violates unforgeability and not robustness. This clearly means that A' runs in expected polynomial time, and produces with non-negligible probability a new message with valid signature. Since A' runs only with access to a chosen message attack on the underlying signature scheme, we have broken this scheme.

4.4 An "Optimistic" Variant

We show how to modify the basic scheme so it becomes more efficient and also non-interactive in case the severs behave correctly, which in a practical scenario is likely to be the case almost all the time.

The client will send requests to the servers to sign x. Each server returns x_i, a_i where a_i is the first message in the proof of correctness for x_i. Moreover, the randomness used in computing a_i is computed by applying a pseudorandom function on x. That is, the random coins are computed as $\phi_{K_i}(x)$ where K_i is a secret key server i stores together with its secret share s_i, and $\phi.(\cdot)$ is a pseudorandom function (say, built from AES encryption). The client tries to compute the signature, assuming all servers sent correct x_i's. If it fails, it sends requests to the servers for proofs that the x_i's were correct, including x, x_i and challenge e in the request to server i. The servers can verify that indeed $x_i = x^{2\Delta s_i} \bmod n$, and if so, recompute a_i using the pseudorandom function. The proofs can then be completed exactly as in the original scheme.

Note that servers do not need to remember anything from the intial request to sign x, the proof can be conducted only from the public data, x, x_i and the private values s_i, K_i. This idea can also be used with the two variants given below.

4.5 A Variant Based on Reduced Assumptions

In this subsection, we present a variant that can be proved secure, only from Conjecture 1, without relying on random oracles. It is less efficient than the basic one, but only by a constant factor. We only sketch the solution informally.

The idea is to use 2 RSA moduli n, n', chosen independently but of the same form as in previous sections. The public exponent e is defined w.r.t. n, the secret key is shared as before, and signature shares are still computed as $x_i = x^{2\Delta^2 s_i} \bmod n$.

The change applies to the way in which we verify the signature shares. We generate a public key modulo n' for the integer commitment scheme of [6], i.e., $h, g \in Z^*_{n'}$, such that h is random square modulo n' and $g = h^z \bmod n'$ for secret z. Then the verification key v_i is a commitment to s_i under public key n', g, h, i.e., $v_i = g^{s_i} h^{r_i} \bmod n'$ for random r_i of suitable size, given to server i initially. This commitment scheme is unconditionally hiding, and is binding relative to the RSA assumption.

Note that since commitments are always random elements in the group generated by h (no matter the value committed to) it is easy to simulate the v_i's without knowing the s_i's. Therefore results analogous to Lemmas 3,4 also hold in this scenario.

Now, in [6], a protocol of the standard 3-move form is presented for proving knowledge of how to open a commitment. An easy modification of this protocol also allows proving that the contents of a commitment is the same as a given discrete log, for instance the one defined by input x and a signature share x_i.

Soudness of this protocol is proved in [6] relative to the strong RSA assumption. However, in our scenario, we can make do with a larger error probability, in

particular, challenges for this protocol are chosen between 0 and B. Therefore, the proof of soundness from [6] now works relative to Conjecture 1 (w.r.t n'). We can therefore prove security of this new scheme following the same strategy as for the basic one. In particular, if we are given an adversary that breaks robustness, this implies we can break Conjecture 1 w.r.t. n', assuming we are given an oracle that generates signatures w.r.t. n. But we can then do a reduction that it takes n' as input, generates n with known factors, and does the generation of secret exponent and secret shares itself. This means that requests to sign messages can be handled without any oracle access, and so security follows from Conjecture 1 alone.

4.6 A Variant Using Any RSA Modulus

The basic scheme imposes some restrictions on the RSA moduli that can be used. In this section we describe a variant that can use a completely arbitrary RSA modulus and can be proved secure assuming only that the underlying RSA signature scheme is secure.

This is done using the basic scheme we already described, with only one difference: in the proofs of correctness of a signature share, we use the auxiliary protocol from Section 4.2 with a 1-bit challenge, instead of choosing it in $[0..B[$. The protocol is then repeated in parallel $\log B$ times (where we choose as before $B = 3t^2$, and t is the number of corruptible servers). More precisely, given x_i that is to be verified against x, v, v_i, the server starts $\log B$ copies of the auxiliary protocol, sends the initial messages $a_i^{(1)}, ..., a_i^{(\log B)}$, the client sends a random $\log B$-bit challenge $b_1, ..., b_{\log B}$ and the server answers this, using b_j as challenge in the j'th instance of the protocol.

Now, if a server can answer more than one challenge, there is at least one instance j where it can answer both $b_j = 0$ and $b_j = 1$. It is now trivial to see from the proof of Lemma 2 that given such answers, and if the server's claim is false, one can find a multiple of the order of v, and hence factor n, without assuming anything about the form of n, except that it is a valid RSA modulus. Hence the proof of security of this modified scheme goes through in exactly the same way as before. The only difference is that in this case we know that if the verification of the signature shares fail, the adversary can factor n, and hence also break the underlying signature scheme (in the basic scheme, we can only prove he can break Conjecture 2). We obtain:

Theorem 2. *The modified threshold RSA scheme defined in this section is secure no matter how the RSA modulus is chosen, assuming the underlying RSA signature scheme is chosen message attack secure.*

Note that it is not known how to use arbitrary RSA moduli for threshold RSA unless 1-bit challenge proofs are used. Furthermore, without the observations we made earlier about the required error probability, one would need to repeat the 1-bit challenge protocol enough times to make the error probability be negligible, e.g., k repetitions where k is the security parameter. Being able able to do with $\log B = \log 3 + 2 \log t$ iterations will be a significant advantage in most practical cases.

5 General Applications of the Main Idea

Let us try to generalize the basic idea from this paper to other threshold cryptosystems or signature schemes. Let the input be x, and suppose server i contributes string x_i that hopefully enables decryption or signing of x. We will assume that we have $t+1$ honest servers and at most t corrupt ones. Suppose finally that the servers prove interactively that each x_i is correct, that the soundness error for these proofs satisfy the bound stated in Lemma 1, and that from any set of $t+1$ correct contributions, we can easily decrypt or sign x.

Lemma 1 now guarantees us that we can compute the correct output efficiently, by searching exhaustively through all $t+1$-subsets of the accepted contributions; *assuming, however, that we can recognize the correct subset when we get to it during the exhaustive search.*

For a signature scheme, this is easy because we can tentatively assume that the subset is correct, attempt to produce a signature, and verify the output value using the public verification key. The same is true for most RSA encryption schemes, namely those that map a message m to some number $y(m,r) \in Z_n$ where r denotes some random coins chosen internally by the decryption process, n is the modulus, and where the ciphertext is $x = y(m,r)^e \bmod n$. In such a case, each guess at a subset will produce a candidate value for $y(m,e)$ which can be checked by verifying the relation $x = y(m,r)^e \bmod n$.

For a probabilistic encryption scheme such as El-Gamal, however, the situation is less clear. The problem is that for El Gamal and related encryption schemes, one cannot easily check whether a given plaintext is contained in a given ciphertext. Simply encrypting the suggested plaintext m under the public key most likely results in a ciphertext different from x even if m was the correct answer.

However, if $t < l/3$ where l is the total number of servers, there is an alternative way to recognize the set of correct contributions: for threshold El-Gamal, like for most threshold cryptosystems, we have for a correct contribution x_j that $x_j = x^{s_j}$ where s_j is a secret exponent held by server i, and where in fact $s_j = f(j)$ where f is a polynomial of degree at most t over some finite field, typically $GF(q)$ for a prime q in case of threshold El-Gamal encryption.

Now, by the assumption $t < l/3$ we have in the worst case t incorrect contributions and $2t+1$ correct ones. Some of the incorrect contributions will be discarded by the proofs of correctness, and among the remaining ones, we can search for a subset of $2t+1$ values x_j, such that all x_j's in the subset are of form $x_j = x^{f'(j)}$ for a polynomial f' of degree at most t. This can be verified by Lagrange interpolation: assume without loss of generality that $\{x_i | i = 1...2t+1\}$ is the set we are checking. Then, for $i > t+1$, and any polynomial f' of degree at most t, we have $f'(i) = \sum_{j=1}^{t+1} \alpha_{i,j} f'(j)$ for fixed and public coefficients $\alpha_{i,j}$. We can therefore verify for all $i = t+2, ..., 2t+1$ that $x_i = \prod_{j=1}^{t+1} x_j^{\alpha_{i,j}}$. Assuming this verifies for all $i = t+2, ..., 2t+1$, we know that the subset is of the required form. It is now easy to see that since $t < l/3$, the polynomial f' we implicitly define here must agree with the polynomial f defined by the honest players in

at least t+1 points, hence $f = f'$ and therefore the plaintext suggested by this subset is correct.

For this situation, we can do a computation similar to the one for Lemma 1. We obtain that is this case, the expected number of subsets to test is in $O(1)$ if the soundness error of the correctness proofs is at most $1/ct^2$ where $c > 3$.

References

1. Algesheimer, Camenisch and Shoup: *Efficient computations modulo a shared secret with applications to generation of shared safe-prime products*, proc. of Crypto 2002.
2. D. Boneh and M. Franklin *Efficient generation of shared RSA keys*, Proc. of Crypto' 97, Springer-Verlag LNCS series, nr. 1233.
3. R. Canetti, *Security and Composition of Multiparty Cryptographic Protocols*, Journal of Cryptology, vol.13, 2000. On-line version at http://philby.ucsd.edu/cryptolib/1998/98-18.html.
4. R.Canetti, *A unified framework for analyzing security of protocols* , Cryptology Eprint archive 2000/67, http://eprint.iacr.org/2000/067.ps
5. Cramer and Damgård: *Secret-Key Zero-Knowledge*, Proc. of TCC 2003, Springer Verlag LNCS.
6. Damgård and Fujisaki: *A statistically hiding integer commitment scheme based on groups with hidden order*, proc. of AsiaCrypt 2002.
7. Damgård and Jurik: *A Generalization and some Applications of Paillier's Probabilistic Public-key System*, to appear in Public Key Cryptography 2001.
8. Damgård and Koprowski: *Practical threshold RSA signatures without a trusted dealer*, Proc. of EuroCrypt 2001.
9. Alfredo De Santis, Yvo Desmedt, Yair Frankel, Moti Yung: *How to share a function securely*, STOC 1994: 522-533
10. Yair Frankel, Peter Gemmell, Philip D. MacKenzie and Moti Yung *Optimal-Resilience Proactive Public-Key Cryptosystems* Proc. of FOCS 97.
11. Yair Frankel, Philip D. MacKenzie and Moti Yung *Robust Efficient Distributed RSA-Key Generation*, Proc. of STOC 98.
12. P. Fouque, G. Poupard, J. Stern: *Sharing Decryption in the Context of Voting or Lotteries*, Proceedings of Financial Crypto 2000.
13. E. Fujisaki and E. Okamoto: *Statistical Zero-Knowledge Protocols to prove Modular Polynomial Relations*, proc. of Crypto 97, Springer Verlag LNCS series 1294.
14. Pierre-Alain Fouque and Jacques Stern: *Fully Distributed Threshold RSA under Standard Assumptions*, IACR Cryptology ePrint Archive: Report 2001/008, February 2001
15. Gennaro, Jarecki, Krawczyk and Rabin: *Secure Distributed Key Generation for Discrete-Log Based Cryptosystems*, Proc. of EuroCrypt 99, Springer Verlag LNCS series, nr. 1592.
16. Gennaro, Rabin, Jarecki and Krawczyk: *Robust and Efficient Sharing of RSA Functions*, J.Crypt. vol.13, no.2.
17. Shingo Miyazaki, Kouichi Sakurai and Moti Yung *On Threshold RSA-Signing with no Dealer*, Proc. of ICISC 1999, Springer Verlag LNCS series, nr.1787.
18. Brian King: *Improved Methods to Perform Threshold RSA.*, ASIACRYPT 2000, pp.359-372, Springer Verlag LNCS.
19. M. Koprowski: *Threshold Integer Secret Sharing*, manuscript, 2003.

20. P.Pallier: *Public-Key Cryptosystems based on Composite Degree Residue Classes*, Proceedings of EuroCrypt 99, Springer Verlag LNCS series, pp. 223-238.
21. Pedersen: *A Threshold cryptosystem without a trusted third party*, proc. of EuroCrypt 91, Springer Verlag LNCS nr. 547.
22. T.Rabin: *A Simplified Approach to Threshold and Proactive RSA*, proc. of Crypto 98, Springer Verlag LNCS 1462.
23. M.K.Reiter and K.P.Birman: *How to securely replicate services*, ACM Transactions on programming languages and systems 1994, vol 16, nr.3, pp.986–1009.
24. J. B. Rosser and L. Schoenfeld: *Approximate formulas for some functions of prime numbers*, Ill. J. Math. 6 (1962), 64–94.
25. Victor Shoup *Practical Threshold Signatures*, Proceedings of EuroCrypt 2000, Springer Verlag LNCS series nr. 1807.