Robust Efficient Distributed RSA-Key Generation

Yair Frankel[*] Philip D. MacKenzie[†] Moti Yung[‡]

Abstract

We solve a central open problem in distributed cryptography, that of robust efficient distributed *generation of RSA keys*. An *efficient protocol* is one which is independent of the primality test "circuit size", while a *robust protocol* allows correct completion even in the presence of a minority of arbitrarily misbehaving *malicious parties*. Our protocol is shown to be secure against any minority of malicious parties (which is optimal). The above problem was mentioned in various works in the last decade and most recently by Boneh and Franklin [BF97].

The solution is a crucial step in establishing sensitive distributed cryptographic function sharing services (certification authorities, signature schemes with distributed trust, and key escrow authorities), as well as other applications besides RSA (namely: composite ElGamal, identification schemes, simultaneous bit exchange, etc.). Of special interest is the fact that the solution can be combined with recent proactive function sharing techniques to establish the first efficient, optimal-resilience, robust and proactively-secure RSA-based distributed trust services where the key is *never* entrusted to a single entity (i.e., distributed trust totally "from scratch").

Our solution involves new efficient "robustness assurance techniques" which guarantee "correct computations" by mutually distrusting parties with malicious minority. These distributed-value representation and manipulation techniques are of independent interest.

1 Introduction

The notion of distributed cryptographic protocols has been central in cryptography for over 15 years. Some protocols have been designed to solve communication problems which are impossible from an information-theoretic perspective, like the coin-flipping protocol [B82] and the millionaire-problem protocol [Y82]. Other protocols have been designed to solve generic problems. These protocols (called "general compiler protocols") can securely compute any public function on secure inputs. The first such protocols were developed by Yao [Y86] and Goldreich, Micali and Wigderson [GMW], and various developments were made in subsequent works, e.g. [GHY, K, BGW, CCD].

Recently there has been a thrust to construct more efficient protocols for specific problems, and in particular, problems involving the distributed application of cryptographic functions (surveyed in [Gw97]). These efficient *(proactive) function sharing* protocols are needed to provide increased memory security, distributed trust, and flexible management (i.e., adding and deleting trustees) of crucial functions like certification authorities and consortium signatures.

A major efficiency difference between a general compiler protocol (which should be thought of as a plausibility result–see [Gr97]) and a function sharing protocol is due to the fact that the communication complexity of the former depends linearly on the actual size of the circuit computing the cryptographic functions, while the communication complexity of the latter is independent of the circuit size (and is typically a polynomial in the input/output size and the number of participants). This difference (pointed out first in [FY93, DDFY94]) is crucial to practitioners who require efficient protocols. A function sharing protocol involves a protocol for applying the function (based on distributed shares), and sometimes (in what is called a "proactive model") also a protocol for re-randomizing the function shares.

Another important step regarding "distributed cryptographic functions" is the (efficient) distributed generation of the function (the key shares). For cryptographic functions based on modular exponentiation over a field (whose inverse is the discrete logarithm which is assumed to be a one-way function), a protocol for the distributed generation of keys was known for quite a while [P2]. However, for the RSA function, which requires the generation of a product of two primes and an inverse of a public exponent, this step was an open problem for many years. Note that Yao's central motivation [Y86] in introducing general compiler protocols that "compute circuits securely in communication" was the issue of distributed generation of RSA keys. Indeed the results of [Y86, GMW] show the plausibility of this task.

A major step forward was recently achieved by Boneh and Franklin [BF97] who showed how a set of participants can actually generate an RSA function efficiently, thus detouring the inefficient compiler. They developed many important new protocol techniques, and showed that their protocol was secure in the limited model of "trusted but curious" parties. They left open the issue of *robustness*, i.e., generation in the presence of misbehaving (malicious) parties. [1]

In this work we solve the open problem of robust, efficient and

[*] CertCo, frankely@certco.com

[†] Boise State University, Boise, ID, philmac@cs.idbsu.edu. Initial work in this area was performed at Sandia National Laboratories under U.S. Department of Energy contract number DE-AC04-94AL85000.

[‡] CertCo, N.Y., NY, moti@certco.com, moti@cs.columbia.edu.

[1] Note that assuming adversaries misbehave arbitrarily, the Boneh-Franklin protocol may be prevented from ever generating a shared RSA key. Note also that more generally, robustness has motivated basic notions in cryptography such as verifi able secret sharing [CGMA] and general zero-knowledge proofs [GMR].

secure generation of shared RSA keys (or more generally, keys based on multiplying two large primes and exponentiation). Our solution assumes $n \geq 2t+1$, where at most t parties misbehave in any malicious and arbitrary way; i.e., we achieve optimal resilience, since a majority of good participants is required. If $n \geq 3t+1$ we have a slightly more efficient variant of the protocol. The techniques we present solve numerous other problems, since they can be employed to distributively initiate schemes based on composite numbers, such as: composite ElGamal encryption/signature, identification schemes where no participant is allowed to know the factorization of N (as in Feige, Fiat, Shamir [FFS]), and an efficient version of Yao's simultaneous bit exchange protocol [Y86].

The solution is built upon the following techniques:

- Multiplication protocols for shared "sum-of-poly" representations of values drawn from (1) a prime field (with robustness and security based solely on the hardness of the discrete logarithm problem), or (2) a subset of the integers (with robustness and security based solely on the hardness of the RSA problem, but without requiring a shared RSA modulus). The technique has potential in other applications since it is information-theoretically secure but also produces publicly verifiable witnesses which are held by the "community of servers".

- Techniques for "chained-consistency" of shared information and its associated checking information, i.e., forcing checking information to be consistent over various computational tasks (e.g., generation protocols, multiplication protocols, and double primality tests) even when the representation of that checking information changes. This involves

  - the idea of "cross-checking" information, i.e., maintaining consistency by verifying *share information* through *checking information*, and moreover, verifying new checking information (perhaps a different representation) through share information. This duality of "checking and computing" is promising and is perhaps of independent interest;
  - efficient zero-knowledge arguments which verify that checking information is consistent; and
  - a bootstrap technique assuring global checking by using a multitude of checking information w.r.t. individual keys.

- A commitment mechanism, called *Simulator-Equivocal Commitments*, which are as secure and efficient as normal commitments, but allow for certain simulatability arguments on committed data which could not be accomplished with standard commitments. The mechanism leads to a new proof technique of security for result-producing protocols.

The resulting protocol assures that distributed systems employing the RSA function (and the other functions mentioned above) can be initiated distributively as well. The protocol is relatively efficient (it does not depend on the size of the circuit of primality tests as in the general compilers). In addition, the number of rounds can be made about the same the number of rounds in the non-robust protocol, and the computation complexity (measured in the number of modular exponentiations) can be brought to about 100 times the computational complexity of the non-robust protocol, given reasonable values for the number of shareholders and security requirements, and a few modifications for efficiency discussed in Section 16.

These results show that our protocol is a feasible protocol that can be used in system initiation and key replacement in numerous systems (as mentioned above) and in various settings which require distributed trust. Note, for example, that shared public key replacement in a certification authority may be performed every three or five years, and thus need not be "real time" operation. Therefore, a somewhat long (say, two-three week) protocol is reasonable. On the other hand a "general compiler" protocol which will take more than the five year period itself is unreasonable!

Remark: The techniques developed here can be used to construct a robust threshold DSS as in [GJKR], but with optimal resilience and with no additional cryptographic assumptions.

2 The Model

The Network: We use the a model similar to various recent works and also [BF97]. We assume a group of n (probabilistic) servers, all connected to a common broadcast medium C, called the communication channel. We assume that messages sent on C instantly reach every party connected to it. The system is synchronized (and w.l.o.g. that servers act synchronously).

The Adversary: The adversary is computationally bounded (i.e., it can not break the underlying cryptographic primitives) and it can corrupt servers at any moment by viewing the memories of corrupted servers and/or modifying their behavior. The adversary decides on whom to corrupt at the start of the protocol. We assume that the adversary corrupts no more than t out of n servers throughout the protocol, where $n \geq 2t+1$. (or $n \geq 3t+1$ for the more efficient protocol variant.) Our model does not differentiate malicious faults from "normal" server failures (e.g., crashes). We also assume that the adversary is connected to the broadcast channel C, which means he can hear all the messages and inject his own. He cannot, however, modify messages sent to C by a server that he does not control, nor can he prevent a non-corrupted server from receiving a message sent on C.

In fact we can strengthen the adversary in the spirit of a more mobile adversary [OY91], where we assume the protocol proceeds in "trials" each trial independent of the past. During a trial a proper key can be generated or the remaining servers (which were not caught deviating from the protocol) restart a new trial. Provided that the eliminated misbehaving servers till now are t', we allow the adversary to occupy $t-1-t'$ new servers at the start of the new trial.

Notation: Let h be the security parameter, and let $H = 2^h$. Say N_* is the product of two unknown primes, each in the range $[\sqrt{H}, 2\sqrt{H}]$, and g_* and h_* are generators whose discrete log $\bmod N_*$ with respect to each other is unknown. Indeed finding such an N_* may be difficult since that is the whole subject of this paper! In this case, each server S_k, can choose a triple (N_k, g_k, h_k) and broadcast it; we, in turn, can bootstrap a solution based on these local values.

3 Basic Techniques

We assume familiarity with secret sharing [Sh], verifiable secret sharing [F], unconditionally secure verifiable secret sharing [P91], basic commitments using discrete logs over prime groups [P91], basic proofs of knowledge of discrete logs [GHY85, CEG, CEGP], and how a number of parties can generate a random value by all committing to private random values, and then revealing those values.

We also use the recent "share representation transformation" techniques from [FGMYa]. Specifically, we employ the following two methods:

- A "poly-to-sum" technique which transforms a function shared by a t degree polynomial amongst n servers into a t-out-of-t additive (sum) sharing.

- A "sum-to-poly" technique which transforms a function shared additively t-out-of-t into a t-out-of-n polynomial sharing.

4 Sum-of-Poly Multiplication over a Prime Field

We now discuss the new techniques which provide for secure and robust RSA key generation. We first develop a multiplication protocol over a prime field which we later modify to work over the integers. The protocol uses a sum-of-poly representation of values. We note that related protocols (over prime fields only) are described in [GJKR], however, as opposed to those protocols, our protocol is unconditionally secure, and is based on the difficulty of discrete logs with no additional assumptions.

We use semantically-secure public-key encryption for sending private messages between servers. We develop a protocol in which severs (shareholders) $S_1, \ldots, S_n$ (for $n \geq 2t+1$) can perform robust multiplication over a prime field while maintaining unconditional security. The goal of the protocol is to compute $C \equiv (A_1 + A_2 + \ldots + A_n)(B_1 + B_2 + \ldots + B_n) \bmod P'$ where A_i and B_i are chosen by S_i.

4.1 The protocol

1. **Set up:** Let $P = 2P' + 1$ be a strong prime, and let g and h be generators of Z_P^* (such that the discrete log of h over base g is unknown). Server S_i determines $A_i, B_i \in Z_{P'}$, and performs a Shamir secret sharing of those values with random polynomials $a_i(x) = \sum_{j=0}^{t} a_{i,j}x^j \bmod P'$ and $b_i(x) = \sum_{j=0}^{t} b_{i,j}x^j \bmod P'$, where $a_i(0) = A_i$ and $b_i(0) = B_i$. (If during the protocol any shareholder S_i is determined to be corrupt, we assume that $A_i = B_i = 0$ and that throughout the protocol shares for A_i and B_i are equal to zero.)

2. **Pedersen Sharing of A_i and B_i.**
While Pedersen used his sharing scheme for implementing a verifiable secret sharing, we use the sharing as a base for efficient and robust secure multiplication. We assume the inputs to the multiplication protocol were generated as follows:

   (a) Each server S_i generates two companion secrets $A'_i, B'_i \in_R Z_{P'}$ and shares them with Shamir secret sharing using random polynomials $a'_i(x) = \sum_{j=0}^{t} a'_{i,j}x^j \bmod P'$ and $b'_i(x) = \sum_{j=0}^{t} b'_{i,j}x^j \bmod P'$, where $a'_i(0) = A'_i$ and $b'_i(0) = B'_i$. It also publishes the Pedersen verification shares for each pair $(a_i(x), a'_i(x))$ and $(b_i(x), b'_i(x))$: $\alpha_{i,j} \equiv g^{a_{i,j}}h^{a'_{i,j}} \bmod P$ and $\beta_{i,j} \equiv g^{b_{i,j}}h^{b'_{i,j}} \bmod P$. Each server S_k verifies its shares with the verification shares.

   (b) Define
      - $A \equiv A_1 + A_2 + \ldots + A_n \bmod P'$,
      - $A' \equiv A'_1 + A'_2 + \ldots + A'_n \bmod P'$, and
      - $B \equiv B_1 + B_2 + \ldots + B_n \bmod P'$.
      - $a(x) \equiv \sum_{i=1}^{n} a_i(x) \bmod P'$, and $a'(x) \equiv \sum_{i=1}^{n} a'_i(x) \bmod P'$.

   Observe that the zeroth coefficients of these polynomials are A and A', respectively. Also note that the verification shares for pair $(a(x), a'(x))$ can be computed by all shareholders as follows: $\alpha_j \equiv \prod_{i=0}^{n} g^{a_{i,j}}h^{a'_{i,j}} \bmod P$.

3. **Generation of randomizing polynomials:** S_i generates random polynomials $z_i(x) \equiv \sum_{j=1}^{2t} z_{i,j}x^j \bmod P'$ and $z'_i(x) \equiv \sum_{j=1}^{2t} z'_{i,j}x^j \bmod P'$, distributes shares of these polynomials, and broadcasts verification shares $\rho_{i,j} = g^{z_{i,j}}h^{z'_{i,j}}$ for $1 \leq j \leq 2t$. (Note that $z_i(0) = z'_i(0) = 0$, and that $\rho_{i,0} = 1$, for $1 \leq i \leq n$.) Each server j verifies its shares with the verification shares.

   Each S_i also generates a random polynomial
   $r_i(x) \equiv \sum_{j=0}^{2t} r_{i,j}x^j \bmod P'$, and distributes shares of this polynomial.

4. **Generate and verify shares of randomized $A(x)B_i(x)$ and $A(x)B'_i(x)$:** Let $v_i(x) \equiv a(x)b_i(x) + z_i(x)$ and $v'_i(x) \equiv a'(x)b_i(x) + z'_i(x) + r_i(x)$. (Note that all shareholders can compute their shares of $v_i(x)$ and $v'_i(x)$ using previously received shares.) S_i broadcasts verification shares for the polynomial pair $(v_i(x), v'_i(x))$:

$$V_{i,j} = \prod_{u+v=j} (\alpha_u)^{b_{i,v}} \rho_{i,j} h^{r_{i,j}}$$

   On a disputes as to the correct $v_i(j)$ values, S_j challenges S_i to reveal the shares $b_i(j)$ and $r_i(j)$. All shareholders can determine if $b_i(j)$ fits the verification shares for $b_i(x)$, and they can check if the the share $v_i(j)$ fits the verification shares for $v_i(x)$ by computing $(g^{a(j)}h^{a'(j)})^{b_i(j)}(g^{z_i(j)}h^{z'_i(j)})h^{r_i(j)} \bmod P$, where $g^{a(j)}h^{a'(j)} \bmod P$ can be computed from the α verification shares for $(a(x), a'(x))$, and $g^{z_i(j)}h^{z'_i(j)} \bmod P$ can be computed from the ρ verification shares for $(z_i(x), z'_i(x))$.

5. **Prove correctness of verification shares (only necessary for $2t+1 \leq n \leq 3t$):** For $1 \leq i \leq n$, S_i proves to all others that for $1 \leq j \leq n$, it knows representations of $g^{b_i(j)}h^{b'_i(j)} \bmod P$ and $(g^{a(j)}h^{a'(j)})^{b_i(j)}h^{r_i(j)} \bmod P$, where the discrete logs of g in the first and $g^{a(j)}h^{a'(j)} \bmod P$ in the second are the same.

6. **Output:** Let $v(x) \equiv \sum_{i=1}^{n} v_i(x) \bmod P'$ and $v'(x) \equiv \sum_{i=1}^{n} v'_i(x) \bmod P'$. Each S_k reveals $v(k)$ and $v'(k)$ and interpolates the resulting values to get $v(0)$. (Observe $v(0) \equiv AB \equiv C \bmod P'$.) Note that the verification shares for the polynomials $v(x)$ and $v'(x)$ can be computed from the verification shares from the previous step. All revealed shares are verified using the verification shares.

5 Basic Techniques over The Integers

Secret Sharing over the Integers[FGMYa] This is a variant of Shamir secret sharing [Sh]. Let $L = n!$. For sharing a secret $s \in [0, K]$, a random polynomial $a(x) = \sum_{j=0}^{t} a_j x^j$ is chosen such that $a_0 = L^2 s$, and each other $a_j \in_R \{0, L, 2L, \ldots, L^3K^2\}$, Each shareholder $i \in \{1, \ldots, n\}$ receives a secret share $s_i = a(i)$. Any set Λ of cardinality $t+1$ can compute s using Lagrange interpolation.

Pedersen Unconditionally Secure VSS over the Integers.

This is a variant of Pedersen Unconditionally Secure VSS [P91]. Assume h be the security parameter, and let $H = 2^h$. Say N_* is the product of two unknown primes, each in the range $[\sqrt{H}, 2\sqrt{H}]$, and g_* and h_* are generators whose discrete log modN_* with respect to each other is unknown. Indeed finding such an N_* may be difficult since that is the whole subject of this paper! In this case, each member S_k, can choose a triple (N_k, g_k, h_k) and broadcast it.

The protocol begins with $n+1$ secret sharings: the first being a Shamir sharing of the secret over the integers, and the next n being sharings of companion secrets using a variant of Shamir sharing over the integers. Specifically, for a secret $s \in [0, K]$ a random polynomial $a(x) = \sum_{j=0}^{t} a_j x^j$ is chosen such that $a_0 = L^2 s$, and each other $a_j \in_R \{0, L, 2L, \ldots, L^3 K^2]$, Then for each triple (N_k, g_k, h_k), a random polynomial $a'_k(x) = \sum_{j=0}^{t} a'_{j,k} x^j$ is chosen with each $a'_{j,k} \in_R [0..L^3 K^3]$. Then shares of each polynomial are sent to each shareholder ($n+1$ total shares to each shareholder) and the verification shares $\{g_k^{a_j} h_k^{a'_{j,k}} \bmod N_k\}_{0 \le j \le t, 1 \le k \le n}$, are published. Note that each shareholder S_k can verify its shares $a(k)$ and $a'_{k'}(k)$ using the verification shares over $N_{k'}$ (for all $1 \le k' \le n$).

6 Sum-of-Poly Multiplication Over The Integers

Here we develop a multiplication protocol over the integers. The protocol uses a sum-of-poly representation of values. The protocol is unconditionally secure, but only in the statistical sense.

We use semantically-secure public-key encryption for sending private messages between servers. This protocol allows servers (shareholders) $S_1, \ldots, S_n$ (for $n \ge 2t+1$) to perform robust multiplication over the integers while maintaining unconditional security. The goal of the protocol is to compute $C \equiv (A_1 + A_2 + \ldots + A_n)(B_1 + B_2 + \ldots + B_n) \bmod P'$ where A_i and B_i are chosen by S_i from the range $[\frac{1}{2}\sqrt{H}, \sqrt{H}]$. (The adversary may choose values from outside this range.)

The protocol is the same as that of the previous section except that the share computation is performed over the integers rather than $\bmod P'$, the verification and check share computations are performed $\bmod N_*$ rather than $\bmod P$, and with the following specific changes:

- **Step 1:** For each i, the zeroth coefficients of $a_i(x)$ and $b_i(x)$ will be $L^2 A_i$ and $L^2 B_i$, respectively.
- **Step 2b:** The zeroth coefficients of $a(x)$ will be $L^2 A$ instead of A.
- **Step 3:** The coefficients of $z_i(x)$ and $z'_i(x)$ will be drawn as follows: $z_{i,j} \in_R [0, L^{11} H^3]$ and $z'_{i,j} \in_R [0, L^{11} H^4]$. The coefficients of $r_i(x)$ will be drawn as follows: $r_{i,j} \in_R [0, L^{11} H^4]$.
- **Step 6:** Finally, $v(0)$ will be $L^4 AB$ instead of AB, so we also divide by L^4.

7 Simulator-Equivocal Commitments

We next show Simulator-Equivocal Commitments. Other robustness tools are given in Section 12.

We show a party B can commit to a value for A, such that the commitment is binding, but a simulator could produce a commitment that is non-binding. The earliest use of a similar mechanism that we are aware of (although it is a less efficient one) is in [IY87]. Simulator-Equivocal commitments are basically trapdoor commitments [FS89] combined with a proof of knowledge of the trapdoor.

We use the following setup protocol, which only needs to be run once for any number of commitments:

1. A strong prime P and a generator g for Z_P^* are distributively chosen.
2. A chooses a value $g' \in Z_P^*$ to be used for B's commitments to A, transmits g' to B and proves to B that it knows the discrete of g' base g. (This can be done using the Basic Proof of Knowledge of Discrete Logs.)

A commitment is constructed just as in the Basic Commitment protocol, except that g and g' are used. That is, B commits to a value $x \in Z_{P'}$ by choosing $x' \in_R Z_{P'}$ and publishing $Commit = g^x g'^{x'} \bmod P$.

When B wishes to open the commitment, B reveals x and x', and A may check if $Commit \equiv g^x g'^{x'} \bmod P$.

Theorem 1 *The Simulator-Equivocal commitment protocol is a commitment protocol which, moreover, can be simulated by a polynomial time simulator*

Proof: Equivocalness: To be able to open up a commitment to any value, the simulator uses backtracking in the proof of knowledge of a discrete log a of g' in the setup to obtain the actual discrete log. (In the setup, the simulator can be thought of as an *extractor* [BCLL].) After the simulator has committed to x with $g^x g'^{x'} \bmod P$, if it wishes to open the commitment as z, it reveals z and $x' + ((x - z)/a) \bmod P'$.

Binding: If B is able to open its commitment two different ways, then it would know the discrete log of g' base g, and that probability is negligible. $\square$

8 Robust Distributed Double-Primality Test

We use the double-prime test scheme of Boneh-Franklin but we make it robust and "chain" the robustness tools to the preceding generation of the value n. To get robustness, for each i we use the polynomials $f_i(x) = a_i(x) + b_i(x) = \sum_{j=0}^{t} f_{i,j} x^j$. where $a_i(x)$ and $b_i(x)$ were used to distribute p_i and q_i, respectively, in the Distributed Computation of N, and we use the corresponding check shares used in the Pedersen sharings. Call these check shares $\gamma_{i,j}$ for $0 \le j \le t$, with $\gamma_{i,0} = g_*^{L^2(p_i+q_i)} h_*^{p'_i+q'_i} \bmod N_*$, where p'_i and q'_i are the companion secrets to p_i and q_i respectively.

The following steps are repeated h times to get the desired security level:

1. The shareholders randomly choose g (technique mentioned in Section 3), such that $\left(\frac{g}{n}\right) = 1$.
2. S_1 broadcasts $Q_1 = g^{(N+1-p_i-q_i)/4} \bmod N$. Then it proves knowledge of the discrete log of Q_i and a corresponding representation of $g_*^{L^2 N+1} \gamma_{1,0}^{-1}$ (over $g_*^{4L^2}$ and h_*) using the protocol in Section 12.1. For each $i > 1$, S_i broadcasts $Q_i = g^{(p_i+q_i)/4} \bmod N$. Then it proves knowledge of the discrete log of Q_i and a corresponding representation of $\gamma_{i,0}$ (over $g_*^{4L^2}$ and h_*) using the protocol in Section 12.1.
3. Now all shareholders verify that $Q_1 / \prod_{i=2}^{n} Q_i \equiv \pm 1 \bmod N$. If it is not equivalent, they declare that N is not a product of two primes.

9 Robust Generation of Public and Private Keys

Remember that $\phi(N) = N - \sum_{i=1}^{n}(p_i + q_i) + 1$. As in Boneh and Franklin, we give two procedures, a simple one for small public keys and a more complicated one for general public keys.

What is interesting in our procedure is that certain operations can be easily done (while maintaining checking information to assure robustness) if a change of representation of the shared value is first performed. The recent sharing representation changes "sum-to-poly" and "poly-to-sum" [FGMYa] will be employed.

9.1 Small Public Keys

We will assume $e = 3$; the procedure for other small public keys is similar.

1. Shareholders jointly choose $g \in_R [1, N-1]$ using Simulator-Equivocal Commitments.
2. For each i, S_i broadcasts $g^{p_i+q_i} \bmod N$.
3. Each shareholder checks that value against $Q_i^4 \bmod N$ (Actually S_1 checks it against $g^{N+1}/Q_1^r \bmod N$.
4. For each i, S_i reveals $x_i = p_i + q_i \bmod 3$.
5. S_i proves knowledge of the discrete log of $g^{p_i+q_i}g^{-x_i} \bmod N$ with base g^3.
6. Note that $\phi(N) \equiv N + 1 - \sum_{i=1}^n x_i \bmod 3$. Let $r = N + 2 - \sum_{i=1}^n x_i$, and $r' = 2N + 3 - 2\sum_{i=1}^n x_i$.

   Similar to Boneh and Franklin, if $\phi(N) \equiv 2 \bmod 3$, S_1 computes its share of d as $d_1 = \frac{r-(p_1+q_1-x_1)}{3}$, and for $2 \le i \le n$, S_i computes its share of d as $d_i = \frac{-(p_i+q_i-x_i)}{3}$. If $\phi(N) \equiv 1 \bmod 3$, S_1 computes its share of d as $d_1 = \frac{r'-2(p_1+q_1-x_1)}{3}$, and for $2 \le i \le n$, S_i computes its share of d as $d_i = \frac{-2(p_i+q_i-x_i)}{3}$.

   All shareholders broadcast check shares $g^{d_i} \bmod N$.
7. All shareholders check that the check shares cubed are correct.
8. A sum-to-poly is performed to construct a (t, n)-secure polynomial sharing of d.

9.2 Large Public Keys

For e large, we use the idea from Boneh and Franklin of finding $(\phi(N))^{-1} \bmod e$. For now, we assume e is a prime, with $E = 2e + 1$ a (strong) prime. Recall that for $1 \le i \le n$, $f_i(x) = a_i(x) + b_i(x)$. Let $f(x) = L^2(N + 1) - \sum_{i=1}^n f_i(x)$. Then $f(0) = L^2\phi(N)$.

1. Shareholders jointly choose e randomly, and test to see that e and $2e + 1$ are prime.
2. Shareholders jointly choose $g_e, h_e \in_R Z_E^*$.
3. Shareholders jointly choose $g, h \in Z_N^*$ using Simulator-Equivocal Commitments. (Actually $polylog$ of them.)
4. S_i chooses $m_i \in_R [0, H]$, and performs a Shamir sharing of m_i over the integers. Say $m = m_1 + \cdots + m_n$.
5. The multiply protocol is run to calculate $D = L^4\phi(N)m \bmod e$ (with all values of the Shamir sharing of each m_i taken mod$e$). Let $INV = D^{-1} \bmod e$, which is easily calculated from D.
6. For each i, j, each S_i multiplies its integer share of m_j by INV to get integer sharings whose sum contains a secret $W \equiv (L^4\phi(N))^{-1} \bmod e$.
7. Perform a multiplication over the integers with check shares over N to get a polynomial sharing with $-L^4W\phi(N) + 1$ in the zero coefficient. (To add one, simply add one to all the resulting shares.) However, instead of revealing the shares of the resulting polynomial, reveal those shares only in the exponent. For example, assuming the resulting polynomial is $v(x)$ and the resulting companion polynomial is $v'(x)$, instead of S_i revealing $v(i)$ and $v'(x)$, it reveals $g^{v(i)} \bmod N$ and $h^{v'(i)} \bmod N$, and proves that it knows the actual shares using the protocol from Section 12.1. Note that $v(0) \equiv 0 \bmod e$, and $v(0) \equiv 1 \bmod \phi(N)$.
8. The shareholders perform a poly-to-sum to $t + 1$ shareholders.
9. Those $t+1$ shareholders divide their additive shares by e, and publish the remainders. Also they publish the check shares for the new additive shares. (Everyone should check these check shares by raising them to the e power and comparing them to the original additive shares.) The remainders will sum to a multiple of e. Add this multiple to the additive share of one of the shareholders. Then these shareholders hold an additive sharing of d.
10. The $t + 1$ shareholders perform a sum-to-poly to construct a (t, n)-secure polynomial sharing of d.

10 Protocol

The protocol:

1. The parties run the setup protocol for the Simulator-Equivocal Commitments.
2. **Re-Starting point**
   At this point the majority agrees which servers are "honest" and what is the upper bound on the number of misbehaving parties (given that some parties have been eliminated).
3. **Run the Distributed Computation of N**
   There are $n \ge 2t + 1$ (or $n \ge 3t + 1$ for the more efficient multiplication protocol) shareholders $S_1, \ldots, S_n$. Let $L = n!$. Let h be the security parameter, and let $H = 2^h$. The goal of the protocol is to compute $N = (p_1 + p_2 + \ldots + p_n)(q_1 + q_2 + \ldots + q_n)$, where the p's and q's are chosen randomly by the shareholders.

   (a) Each S_i chooses $p_i, q_i \in_R [\frac{1}{2}\sqrt{H}, \sqrt{H}]$.

   (b) The Shamir scheme over the integers is used to distribute shares of p_i and q_i. Say the polynomial used to share p_i is $a_i(x)$, and the polynomial used to share q_i is $b_i(x)$.

   (c) The multiplication scheme over the integers is run to compute $N = (p_1+p_2+\ldots+p_n)(q_1+q_2+\ldots+q_n)$.

   (d) Each S_i proves that p_i and q_i are in the range $[0, \frac{3}{2}\sqrt{H}]$ using the protocol in Section 12.1 (over $g_*^{L^2}$ and h_*) with the zero-coefficient verification shares from the multiplication scheme.

   (e) If any party misbehaves, and the majority agrees that it is misbehaving, it is excluded from the rest of the protocol.
4. **Run the Robust Distributed Double-Primality test of N**
   If N fails restart the protocol with the current "honest" parties.
5. **Run the Robust Distributed Generation of Public and Private Keys**
   Decide whether to use a small or large public exponent and run the appropriate protocol to determine public and private keys e and d.

11 Simulatability and Robustness of Multiplication over a Prime Field

We omit all proofs in this section due to space considerations.

Definition 2 *A (t,n)-restricted adversary may corrupt up to t out of n shareholders.*

Definition 3 *A multiplication protocol M is* perfectly (statistically) (t,n)-simulatable *if for any probabilistic polynomial-time (t,n)-restricted adversary $\mathcal{A}$, for any possible initial setup S (consisting of share values $a_i(j)$ and $b_i(j)$ for $1 \le i \le n$ and $1 \le j \le t$, and initial polynomials $a_i(x)$ and $b_i(x)$ for $1 \le i \le t$) and for any possible product C (that could result from that initial setup), there is a probabilistic polynomial-time simulator* $\mathrm{simu}(S,C,\mathcal{A})$ *for $M(S,C)$ such that* $\mathrm{view}_{\mathcal{A}}^{\mathrm{simu}(S,C,\mathcal{A})}$ *is perfectly (statistically) indistinguishable from* $\mathrm{view}_{\mathcal{A}}^{M(S,C)}$.

We only deal with a an adversary that can corrupt up to t shareholders at the start of the protocol (or a start of a trial). We assume we have secure channels. (The proof for channels with semantically-secure encryptions uses an additional standard hybrid walking proof.)

Lemma 4 *The multiplication protocol is perfectly (t,n)-simulatable, assuming that the Shamir polynomials of the non-corrupt shareholders are chosen randomly as discussed above.*

Definition 5 (Robustness of multiplication) *Assume that for $1 \le i \le n$, shareholder S_i shares the secrets A_i and B_i using Shamir (unverifiable) sharing, and assume the adversary does not know y such that $g \equiv h^y \bmod P$. A protocol is a (t,n)-robust multiplication protocol if for any probabilistic polynomial-time (t,n)-restricted adversary $\mathcal{A}$, with all but negligible probability, within polynomial-time, the product $C \equiv (A_1+\cdots+A_n)(B_1+\cdots+B_n) \bmod P'$ is output.*

Lemma 6 *The multiplication protocol is (t,n)-robust.*

Lemma 7 *The multiplication over the integer protocol is statistically (t,n)-simulatable, assuming that the Shamir polynomials chosen have coefficients drawn randomly as discussed above.*

Lemma 8 *The multiplication protocol over the integers is (t,n)-robust.*

12 Robustness Tools

Note that in all the proofs of knowledge, we use the commitment tools chosen in the set up. These tools allow us to perform efficient constant-round zero-knowledge proofs here. Due to space considerations, we only include the basic proof of knowledge of a discrete log.

12.1 Proof of knowledge of a discrete log

A zero-knowledge interactive proof in which a prover $\mathcal{P}$ proves knowledge to a verifier $\mathcal{V}$ of a discrete log (or any isomorphic function) of $X = g^x \bmod P$ was presented in [GHY85]. Below we demonstrate a similar proof based on a composite modulus, N, and input $X = g^x \bmod N$, where $x \in [0..Z]$, for some $Z \ge 1$.

1. $\mathcal{P}$ commits to a randomly chosen h-bit string $c = c_1 || \cdots || c_h$ using a Simulator-Equivocal commitment, and also transmits $(g^{q_1} \bmod N, \ldots, g^{q_h} \bmod N)$ where $q_i \in_R [0..ZN]$.
2. $\mathcal{V}$ sends a randomly chosen h-bit string $c' = c'_1 || \cdots || c'_h$ to $\mathcal{P}$.
3. $\mathcal{P}$ opens its commitment. Let $d = c \oplus c'$. For $k = 1 \ldots h$, $\mathcal{P}$ transmits $v_k = d_k x + q_k$.
4. For $k = 1 \ldots h$, $\mathcal{V}$ verifies $X^{d_k} g^{q_k} \overset{?}{\equiv} g^{v_k} \bmod N$

Theorem 9 *The above protocol is a statistical zero-knowledge interactive proof of knowledge of a discrete log modulo a composite.*

Proof: Omitted due to space considerations. $\square$

13 Protocol Simulation

Our adversary is assumed to control up to t shareholders (chosen non-adaptively at the start). Thus we assume we know which t shareholders the adversary will corrupt. WLOG, assume they are the first t shareholders.

We also use semantically-secure encryption which can be simulated (using a standard hybrid proof). Thus, we can assume we have secure channels. We also assume the adversary knows the first $n-1$ shares. Namely, for each protocol instance the t shares of the corrupt shareholders are chosen, then an result oracle will be queried, it will generate the other shares at random and will give the simulator the resulting N and the $n-1$ first shares. (This is the "minimum knowledge" strategy to simulate "result producing protocol" [GHY]). We will have to prove that the information enables "simulation" of the view of the "adversary". In addition we will prove (see Lemma 13 below) that the information given out by the oracle does not give more than polynomial "computational advantage" beyond the knowledge of N itself in the sense of the zero-knowledge methodology [GMR].

13.1 Distributed Computation of N

Given $p_1, \ldots, p_{n-1}, q_1, \ldots, q_{n-1}, N$, we must simulate the adversary's view.

The simulation given here is one in which the protocol succeeds and the value of N is found. Of course, we can easily simulate the case when the protocol fails.

1. Shareholders S_1 through S_{n-1} are given their values $\hat{p}_1$ through $\hat{p}_{n-1}$ and $\hat{q}_1$ through $\hat{q}_{n-1}$. (Note that the adversary knows all these values.) Let $\hat{p}_n = \hat{q}_n = 0$.
2. All shareholders perform Shamir sharing as in the protocol.
3. The simulation for the multiplication protocol is performed.
4. The proof that the values are in the correct range is faked by the simulator.

13.2 Distributed Primality Test

For $1 \le i \le n-1$,

1. For $1 \le i \le t$, the adversary chooses and broadcasts C_i and C'_i. For $t+1 \le i \le n-1$, the simulator chooses a random $r_i \in [0, LH^3]$, and broadcasts $C_i = g_*^{(p_i+q_i)/4} h_*^{r_i} \bmod N_*$ and $C'_i = C_i^{4L^2} \bmod N_*$. The simulator then chooses a random C_n and broadcasts C_n and $C_n^{4L^2}$.
2. The proofs are run normally, except for the one for C_n, in which the simulate fakes the proof, using backtracking.

The following simulated steps are repeated h times:

1. The shareholders randomly choose g as in the real protocol.

2. For $1 \leq i \leq t$, S_i broadcasts Q_i. For $t+1 \leq i \leq n-1$, S_i broadcasts $Q_i = g^{(p_i+q_i)/4} \bmod N$. The simulator chooses a bit b randomly, and broadcasts

$$(-1)^b g^{(N+1-\sum_{i=1}^{n-1}(p_i+q_i))/4}$$

for S_n. The proofs are run as in the real protocol, except for the the one by S_n, in which the simulator fakes the proof.

3. The test for $Q_1/\prod_{i=2}^{n} Q_i \equiv \pm 1 \bmod N$. is performed as in the real protocol.

13.3 Generation of Public and Private Keys

13.3.1 Small Public Keys

Again we assume $e = 3$.

1. The simulator generates a random $r \in [0, N]$, and using simulator-equivocal commitments, ensure that the g chosen is actually $r^e \bmod N$. Thus $r = g^d \bmod N$.
2. For each $i < n$, S_i broadcasts $g^{p_i+q_i} \bmod N$, but S_n broadcasts $g^{N+1}/\prod_{i=1}^{n-1} g^{p_i+q_i} \bmod N$
3. The proof are the same, except that the simulator fakes the proof by S_n.
4. For each $i < n$, S_i reveals $x_i = p_i + q_i \bmod 3$, but S_n reveals 0, 1 or 2. (The simulation is run for all three.)
5. Again, the proofs are the same except that the simulator fakes the proof by S_n.
6. $\phi(N) \bmod 3$ is computed as in the real protocol, and all shareholders except S_n compute shares of the private key d. Then all shareholders except S_n broadcast their shares $g^{d_i} \bmod N$, except S_n broadcasts $r/\prod_{i=1}^{n-1} g^{d_i} \bmod N$.
7. Finally, the sum-to-poly is simulated.

13.3.2 Large Public Keys

1. e is chosen as in the real protocol.
2. $g_e, h_e \in_R Z_E^*$ are generated as in the real protocol.
3. The simulator chooses $r \in_R [0, N]$, and performs the simulation (as explained in that section) such that the resulting generated value is $g = r^e \bmod N$. Then $r \equiv g^d \bmod N$. h is chosen as in the real protocol.
4. For all $i \in \{1, \ldots, n-1\}$, $\hat{m}_i$ is generated and shared as in the real protocol. We assume the adversary knows all of these. Then the simulator sets $\hat{m}_n = 0$, and shares it as in the real protocol ($\bmod e$).
5. A random value $D \in Z_e$ is chosen by the simulator to be the product ($L^4 m\phi(N) \bmod e$), and the simulation for the multiply protocol is run. Then INV is calculated as in the real protocol.
6. The shares of each polynomial sharing of $\hat{m}_i$ is multiplied by INV as in the real protocol.
7. The multiplication over the integers is simulated, with the last step revealing the check shares over g and h individually. This can be simulated using $g^{v(0)} \equiv g \bmod N$, and $g^{v(i)} \bmod N$ for $1 \leq i < n$. The proofs are run as in the real protocol, except the proof for S_n is faked by the simulator.
8. The poly-to-sum is simulated.
9. The division by e is simulated using the correct value to achieve a multiple of e with all leftovers. The check shares are simulated by having the simulator compute the last one as $r = g^d \bmod N$ divided by the others.
10. The sum-to-poly is simulated.

14 Security Proof

Here we will sketch the proof of security. First we define the RSA function.

Definition 10 *Let h be the security parameter. Let key generator GE define a family of RSA functions to be $(e, d, N) \leftarrow GE(1^h)$ such that N is a composite number $N = P * Q$ where P, Q are prime numbers of $h/2$ bits each. The exponent e and modulus N are made public while $d \equiv e^{-1} \bmod \lambda(N)$ is kept private.*[2]

The **RSA encryption function** *is public, defined for each message $M \in Z_N$ as: $C = C(M) \equiv M^e \bmod N$. The* **RSA decryption function** *(also called signature function) is the inverse: $M = C^d \bmod N$. It can be performed by the owner of the private key d. The security of RSA is given in Definition 11.*

For naming convenience, we will assume our system is used for direct RSA signing of messages; however, the same protocol could be used for decryption. Our results simply concern the application of the RSA function in its assumed intractable direction as a one-way function (as assumed in protocols with formal security proofs which employ RSA, e.g. [ACGS]).

Define $\mathrm{hist}(d, N, L)$ to be a "history" of messages/signature pairs with messages taken by L, and signatures generated using the RSA secret key (d, N).

Definition 11 The RSA security assumption*:*
Let h be the security parameter. Let key generator GE define a family of RSA functions (i.e., $(e, d, N) \leftarrow GE(1^h)$ be an RSA instance with security parameter h). For any probabilistic polynomial-time adversary $\mathcal{A}$, given a polynomial-size list L of messages chosen uniformly at random:

$$\begin{aligned}\Pr[u^e \equiv w \bmod N \quad : \quad & (e, d, N) \leftarrow GE(1^h); w \in_R \{0,1\}^h; \\ & u \leftarrow A(1^h, w, e, N, \mathrm{hist}(d, N, L))]\end{aligned}$$

is negligible.

Definition 12 Modified RSA security assumption*:*
Let h be the security parameter, and let $P', Q' \geq 0$ be chosen by an adversary. Let key generator GE define a family of RSA functions (i.e., $(e, d, N) \leftarrow GE(1^h, P', Q')$ is an RSA instance in which $N = (P' + P)(Q' + Q)$, with P', Q' drawn from $[0, k\sqrt{H}]$, for some $k = O(h)$, and then P, Q drawn randomly from $[0, \sqrt{H}]$ until $P' + P$ and $Q' + Q$ are found to be prime). For any probabilistic polynomial-time adversary $\mathcal{A}$, given a polynomial-size list L of messages chosen uniformly at random:

$$\begin{aligned}\Pr[u^e \equiv w \bmod N \quad : \quad & (e, d, N) \leftarrow GE(1^h, P', Q'); w \in_R \{0,1\}^h; \\ & u \leftarrow A(1^h, w, e, N, P', Q', \mathrm{hist}(d, N, L))]\end{aligned}$$

is negligible.

[2] $\lambda(N) = \mathrm{lcm}(P-1, Q-1)$ is the smallest integer such that any element in Z_N^* raised by $\lambda(N)$ is the identity element. RSA is typically defined using $\phi(N)$, the number of elements in Z_N^*, but it is easy to see that $\lambda(N)$ can be used instead. We use it because it gives an explicit way to describe an element of maximal order in Z_N^*. Note that $\phi(N)$ is a multiple of $\lambda(N)$, and that knowing any value which is a multiple of $\lambda(N)$ implies breaking the system.

Lemma 13 *If the RSA Security Assumption is true, then the Modified RSA Security Assumption is true.*

Proof: Assume that the Modified RSA Security Assumption were false. Then there would be a P', Q' such that the probability of breaking the system is non-negligible. Note that with probability $\Omega(k^{-2})$, the P and Q chosen by the generator $GE(1^{2^h \log k})$ will be within $\sqrt{H}$ of P' and Q', and thus with non-negligible probability, the original RSA system would be broken. □

The above demonstrate that if we get all the additive pieces of P but one, and the same for Q, still given the resulting N, the RSA assumption regarding this N holds just as well (up to a polynomially small advantage). This will determine a simulation strategy, where a result oracle will be queried with the shares of the "corrupted parties" and will draw the rest of the pieces to generate N and will then output N and all but one piece for P and one for Q. Given the output of that "result oracle" the RSA function is still secure, which justified our simulation above.

Now we define what it means for a system to be robust. In these definitions we assume the security parameter h is large enough so that the analysis holds.

Definition 14 (Robustness) *Let h be the security parameter. A system S is a robust RSA-key-generation system if for any probabilistic polynomial-time t-out-of-n adversary $\mathcal{A}$, with all but negligible probability, within polynomial-time, an RSA key is generated such that the public key (e, N) is known, and the secret d is shared among the n shareholders by a degree-t polynomial.*

Next we define what it means for the system to be secure (RSA is treated as a one-way function).

Definition 15 (Security) *Let h be the security parameter. A system S is a (t, n)-secure RSA-key-generation system if for any probabilistic polynomial-time (t, n)-restricted adversary $\mathcal{A}$, when $S_{\mathcal{A}}$ (the system running with that adversary) generates an RSA instance, for any polynomial-size list L of randomly chosen messages submitted to $S_{\mathcal{A}}$ to be signed, for any probabilistic polynomial-time function A, with $S_{\mathcal{A}}(e, d, N)$ denoting the system S with the adversary $\mathcal{A}$ in which the RSA instance (e, d, N) was generated:*

$$\begin{aligned}\Pr[u^e \equiv w \bmod N \quad : \quad & (e, d, N) \leftarrow S_{\mathcal{A}}; w \in_R \{0,1\}^h; \\ & u \leftarrow A(1^h, w, \mathrm{view}_{\mathcal{A}}^{S_{\mathcal{A}}(e,d,N)}, \mathrm{hist}(d, N, L))]\end{aligned}$$

is negligible.

Theorem 16 *The protocol is (t, n)-secure.*

Proof: Recall by the argument above, to prove breaking our system implies breaking the RSA Security Assumption, we simply need to prove that breaking our system implies breaking the Modified RSA Security Assumption.

WLOG, assume that the adversary controls shareholders 1 through t. Also, assume that the adversary knows $p_{t+1}, \ldots p_{n-1}$ and $q_{t+1}, \ldots, q_{n-1}$. It is easy to see that the security with this assumption implies security without this assumption.

We construct a simulator that reduces the (modified) security of the RSA function to the security of our system.

We prove that given the values $p_1, \ldots, p_{n-1}, q_1, \ldots, q_{n-1}, N$, the simulator generates a view of the protocol that is statistically indistinguishable from the real protocol (assuming secure channels). We will again only use the set (N_*, g_*, h_*) in our proof, but it easily extends to using all (N_i, g_i, h_i).

Note that all shareholders behave exactly as in the protocol except shareholder S_n, who distributes shares for $\hat{p}_n = 0$ and $\hat{q}_n = 0$. We show that with high probability, any secret values p_n and q_n will correspond to the shares given to the first t shareholders.

To see that a value p_n is possible, note that one needs a polynomial $a_n(x)$ in which $a_n(0) = L^2 p_n$, and $a_n(i) = \hat{a}_n(i)$ for $i \in \{1, \ldots, t\}$, The only question is whether the polynomial has all coefficients in the correct range. To see this, examine the difference polynomial $d(x) = a_n(x) - \hat{a}_n(x)$, where $d(0) = L^2 p_n$, and $d(i) = 0$ for $1 \leq i \leq t$. Using interpolation, one can see that $d(x)$ has coefficients that are multiples of L with values in the range $[-L^3 H, L^3 H]$ (use Lemma 2 from [FGMYa]). Then if the coefficients of $\hat{a}_n(x)$ are all in the range $[L^3 H, L^3 H^2 - L^3 H]$, all coefficients of $a_n(x)$ will be in a valid range (and those shares for shareholders $\{1, \ldots, t\}$ would correspond to $a_n(x)$).

The same argument follows for q_n.

Finally, we have already shown a simulation for the multiplication protocol over the integers. (Actually, using the conditions from step 2 that the polynomials $\hat{a}_n(x)$ and $\hat{b}_n(x)$ both have secret 0 and coefficients drawn from $[L^3 H, L^3 H^2 - L^3 H]$, part of the simulatability argument for multiplication is avoided.)

For the distributed primality test, the values $g^{(p_i+q_i)/4} \bmod N$ are revealed for all i. Note that

$$g^{(p_n+q_n)/4} \equiv (\pm 1) g^{-(N - \sum_{i=1}^{n-1}(p_i+q_i))/4} \bmod N,$$

assuming N is the product of two primes, so it can be simulated.

Proofs of knowledge and equivalence with the simulated values $\hat{\gamma}_{n,0}$ are simulated using backtracking. with statistical indistinguishability.

For the generation of private keys from small public keys, say $e = 3$, the simulator can run three times, once for each $p_n + q_n \equiv i \bmod 3$, $1 \leq i \leq 3$. The proof for S_n non-corrupt shareholders is simulated using backtracking, with statistical indistinguishability.

Omitted is the security proof for the generation of private keys from large public keys. □

15 Robustness and Correctness Proof

Lemma 17 *Let h be the security parameter. Let modulus generator GE define a family of modulus generating functions (i.e., $N \leftarrow GE(1^h)$ be an RSA modulus with security parameter h). For any probabilistic polynomial-time adversary $\mathcal{A}$, $\Pr[u^e \equiv w^d \bmod N; (e \neq 0) \vee (d \neq 0) : N \leftarrow GE(1^h); u, w \in_R \{0,1\}^h; e, d \leftarrow A(1^h, w, u)]$ is negligible.*

Proof: Similar to [B84]. □

Note that if we determine a shareholder is corrupt, we simply restart the protocol with that shareholder excluded. Also each remaining shareholder chooses new (N_i, g_i, h_i) values.

Theorem 18 *The protocol is (t, n)-robust.*

Proof: **Distributed Computation of N:** Robustness basically follows from the robustness of multiplication over the integers. Note that any bad sharings will be caught using the verification shares in the multiplication over the integers protocol.

Also, if the resulting N is not a Blum integer, all shares are revealed to find the corrupt shareholders.

Distributed primality test: Note that there are $t + 1$ good shareholders that own shares of $p_i + q_i$. They define the polynomial $f_i(x)$. S_i broadcasts Q_i and proves that it knows the discrete log of Q_i, and that it is the same as the first part of a representation of $\gamma_{i,0}$ over $g_*^{4L^2}$ and h_*. Say the representation used by S_i was (z, z'). If $z \neq (p_i + q_i)/4$, this would imply that S_i knows two representations for $\gamma_{i,0}$, i.e. $g_*^{4L^2 z} h_*^{z'}$ and $g_*^{L^2(p_i+q_i)} h_*^{y}$, and thus it knows values, $e = z - (p_i + q_i)/4$ and $d = z' - y$ such that $g_*^e = h_*^d \bmod N$. The probability of this is small, by Lemma 17. (A slightly different argument is used for S_1.)

Generation of Public and Private Keys:

Small Public Keys: The generation of g is robust using the binding property of the Simulator-Equivocal Commitments. Every shareholder can compute $g^{p_i+q_i} \bmod N$ for itself, using the value of Q_i. Say S_i reveals x_i. If it is not $p_i + q_i \bmod 3$, and if S_i proves knowledge of the discrete log of $g^{p_i+q_i}g^{-x_i} \bmod N$ with base g^3, then it knows x, y, z such that $g^{3z} \equiv g^{p_i+q_i-x_i} \equiv g^{3x+y} \bmod N$, where $0 \leq y \leq 3$. In this case, $3z - 3x - y$ is a non-trivial multiple of $\phi(N)$, since $3z - 3x - y \neq 0$, and $g^{3z-3x-y} \equiv 1 \bmod N$.

Large Public Keys: Omitted.

□

16 Efficiency of the Protocol and Variations

In typical systems, we expect h to be on the order of 1000 (i.e. 1024-2048); working over the integers will add less than 400 to the size of exponents, assuming that assurance of 2^{-40} is reasonable. We would expect n to be less than 10, and 2^{-k} chance of error on proofs with $k = 40$ to be sufficient assurance of correctness.

The probability of generating an RSA modulus from two random primes of $h/2$ bits each is about $(h/2)^{-2}$, so we will have about $h^2/4$ rounds. The communication complexity of each round is bounded by $O(nkh)$ and the computational complexity of each round is about $O(n(k+t))$ modular exponentiations. Given realistic values of k and t, the computational complexity is dominated by the multiplication protocol and would be about $24n(t+1)$ modular exponentiations. Unfortunately, we cannot gain much efficiency by performing trial division as discussed in Boneh and Franklin, since each trial against a small prime would involve a robust multiplication, and thus around $O(ntB/\ln B)$ modular exponentiations for each distributed integer tested.

Nevertheless, numerous practical improvements can be made. First, trial division can be done once a possible N is generated, and this would eliminate many distributed double-primality tests. Also, if the test that each p_i and q_i are in the correct range is done after this trial division, then many of those can be saved. (In this case, N needs to be tested to make sure it is in a possible range, so that flagrant misbehavior of shareholders can be determined.)

Perhaps the largest improvement would come from revealing the value of each p_i mod the product J of small primes up to some B, and assuming the shared value is a multiple of the J. The sum of the revealed values could be tested to see if it is divisible by any of the small primes up to B. If so, the distributed integer would not be prime and would be thrown out. Of course, this reduces the security of the problem, so it would probably be done for primes up to, say, 7. This reduces the security by about 6 bits ($\log((2-1)\cdot(3-1)\cdot(5-1)\cdot(7-1))$), but would increase the probability that the number is prime by a factor of about 5 and thus reduce the expected number of rounds for the protocol by a factor of about 25. (If the reduction in security is worrisome, h could be slightly increased without affecting the running time significantly. Or we could only use primes 2 and 3, reducing the security by about 1 bit, while reducing the expected number of rounds for the protocol by about 6.)

With the improvements described above, the total number of modular exponentiations in our protocol will be about $24n(t+1)$ times $(h/10)^2$ (reduced from $(h/2)^2$ because of our substitute for trial division), which is about 10000. For the case $n = 4$ and $t = 1$, we get $2,000,000$ modular exponentiations. The non-robust protocol of Boneh-Franklin using trial division up to $8,103$ performs about 484 modular exponentiations, about a factor of 4000 less. Next we show further interesting improvements.

If it is likely that all parties are honest, one can increase performance by performing what is known as "optimistic execution". The idea is to run the protocol in this paper (plus the trial division from Boneh-Franklin) but without any checking information. That is, there would be no companion polynomials or verification shares generated. At the end of each round in which the participants failed to produce an RSA modulus, they reveal all their information from the round. If all participants agree that the information revealed is correct, then there has been no cheating in that round, and they proceed. If cheating is ever found, they revert to the robust protocol described in this paper. If an RSA modulus is found, then they rerun the round using the same polynomials sharings, but including the robustness checks (i.e., companion polynomials, verification shares, proofs of knowledge). If all the information is verified correctly, then they use that RSA modulus. Otherwise, cheating has occurred and they revert to the robust protocol.

The problem with this "mode" of operation is that cheating can be detected, but it is impossible to determine exactly which participant cheated. To determine who cheated, we can require that participants sign their messages to other participants, and have recipients of messages either accept the signature or ask the sender to reveal the message to everyone. Assuming the signatures are RSA signatures on participants' private keys, we can assume that they require a single modular exponentiation. Furthermore, we could use one signature per round, or even over multiple rounds, to reduce the number of modular exponentiations. In all, this method will still be robust and secure (with some standard assumptions about security and robustness of signatures and encryptions), but require only about n^2 times the modular exponentiation of the non-robust protocol, where n is the number of participants. We expect this variant to be used in practice. (Note that this only works when $n \geq 3t+1$, because we cannot use ZK proofs to guarantee correct shares in the multiplication protocol.)

16.1 Variations on the Protocol

Once a composite N is established there are various ways to use it in various cryptographic protocols other than as a distributed RSA system:

1. The parties can initiate a "composite ElGamal" system. They draw a random element g and generate public key $y = g^x$ where x is shared among the parties using either an additive or polynomial sharing scheme.

2. Users register for an identification protocol by using the availability of N to generate a quadratic residue for which they keep the root as a secret.

3. Users can engage in the Yao's secret bit exchange protocol, where they both encrypt the value under N and simultaneously reveal the decryption bit by bit.

4. Using (e, N), users can use RSA as a public commitment scheme (i.e. using it as a one-way function) which cannot be opened unless the majority wants to do it. One application for this is the escrow of plaintexts.

References

[ACGS] W. Alexi, B. Chor, O. Goldreich and C. Schnorr. RSA and Rabin Functions: Certain Parts are as Hard as the Whole. In *SIAM Journal of Computing*, volume 17, n. 2, pages 194–209, April 1988.

[B84] E. Bach, Discrete Logarithms and Factoring. Tech. Report No. UCB/CSD 84/186. Computer Science Division (EECS), University of California, Berkeley, CA. June 1984.

[BGW] Ben-Or M., S. Goldwasser and A. Wigderson, *Completeness Theorem for Non cryptographic Fault-tolerant Distributed Computing*, STOC 1988, ACM, pp. 1-10.

[B82] M. Blum, "Coin flipping by telephone: a protocol for solving impossible problems," IEEE Computer Conference 1982, 133–137.

[BF97] D. Boneh and M. Franklin, *Effi cient Generation of Shared RSA Keys*, Crypto 97, pp. 425–439.

[B88] C. Boyd, *Digital Multisignatures*, IMA Conference on Cryptography and Coding, Claredon Press, 241–246, (Eds. H. Baker and F. Piper), 1986.

[BCLL] G. Brassard, C. Crépeau, S. Laplante, C. Léger. *Computationally Convincing proofs of knowledge*, In Proceedings of the 8th Symp. on Theoretical Aspects of Computer Science (Springer, Berlin, 1991), pp. 251–262.

[BGM] E. Brickell, D. Gordon and K. McCurley. *Fast Exponentiation with Precomputation Advances in Cryptology – Eurocrypt* 92 *Proceedings*, Lecture Notes in Computer Science Vol. 658, R. Rueppel ed., Springer-Verlag, 1992.

[CCD] D. Chaum, C. Crepeau and I. Damgard, *Multiparty Unconditionally Secure Protocols*, STOC 1988, ACM, pp. 11-19.

[CEG] D. Chaum, J.-H. Evertse and J. van de Graaf, *Multiparty computations ensuring privacy of each party's input and correctness of the result*, *Advances in Cryptology – Eurocrypt* 88 *Proceedings*, Lecture Notes in Computer Science Vol. 330, C. Gunther ed., Springer-Verlag, 1988, pp. 87–119.

[CEGP] D. Chaum, J.-H. Evertse, J. van de Graaf and R. Peralta, *An improved protocol for demonstrating possession of discrete logarithms and some generalizations*, *Advances in Cryptology – Crypto* 86 *Proceedings*, Lecture Notes in Computer Science Vol. 263, A. Odlyzko ed., Springer-Verlag, 1986, pp. 200–212.

[CGMA] B. Chor, S. Goldwasser, S. Micali and B. Awerbuch, *Verifi able Secret Sharing and Achieving Simultaneous Broadcast*, Proceedings *of the* 26th *Symposium on Foundations of Computer Science*, IEEE, 1985, pp. 335-344.

[DDFY94] A. De Santis, Y. Desmedt, Y. Frankel, and M. Yung, *How to Share a Function Securely*, ACM Proceedings *of the* 26th *Annual Symposium on Theory of Computing*, ACM, 1994, pp. 522-533.

[DF89] Y. Desmedt and Y. Frankel, *Threshold cryptosystems*, *Advances in Cryptology – Crypto* 89 *Proceedings*, Lecture Notes in Computer Science Vol. 435, G. Brassard ed., Springer-Verlag, 1989, pp. 307-315.

[DH] W. Diffi e and M. Hellman, *New Directions in Cryptography* , IEEE Trans. on Information Theory 22 (6), 1976, pp. 644-654.

[FFS] U. Feige, A. Fiat and A. Shamir, *Zero-Knowledge Proof of Identity*, Proceedings of the Nineteenth annual ACM Symp. Theory of Computing, 1987, pp 210–217.

[F] P. Feldman, *A Practical Scheme for Non-Interactive Verifi able Secret Sharing*, Proceedings *of the* 28th *Symposium on Foundations of Computer Science*, IEEE, 1987, pp.427-437

[FS86] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identifi cation and signature problems," in *Advances in Cryptology—CRYPTO '86 Proceedings* (Lecture Notes in Computer Science, Vol. 263), ed. A. Odlyzko, 186–194, Springer-Verlag, New York, 1987.

[F89] Y. Frankel, *A practical protocol for large group oriented networks*, In J. J. Quisquater and J. Vandewalle, editor, *Advances in Cryptology, Proc. of Eurocrypt '89, (Lecture Notes in Computer Science 773)*, Springer-Verlarg, pp. 56-61.

[FGY] Y. Frankel, P. Gemmell and M. Yung, *Witness Based Cryptographic Program Checking and Robust Function Sharing*. Proceedings *of the* 28th *Annual Symposium on Theory of Computing*, ACM, 1996, pp. 499-508.

[FGMY] Y. Frankel, P. Gemmel, P. MacKenzie and M. Yung. *Proactive RSA*, Crypto 97.

[FGMYa] Y. Frankel, P. Gemmel, P. MacKenzie and M. Yung. *Optimal Resilience Proactive Public-Key Cryptosystems*, FOCS 97.

[FS89] U. Feige and A. Shamir. Zero knowledge proofs of knowledge in two rounds. CRYPTO 1989, 20–24.

[FY93] M. Franklin and M. Yung, *Secure and Effi cient Off-Line Digital Money*, Proc. of the 20th Int. Col. on Automata, Languages and Programming (ICALP), 1993, LNCS 700, Springer Verlag, pp. 265-276.

[GHY] Z. Galil, S. Haber and M. Yung, *Minimum-Knowledge Interactive Proof for Decision Problems*, SIAM J. Comp., 18, 1989, pp 711–739.

[GHY85] Z. Galil, S. Haber and M. Yung, *Symmetric Public-Key Cryptography* Crypto 85.

[GHY87] Z. Galil, S. Haber and M. Yung, *Cryptographic Computations: Secure Fault Tolerant Protocols in the Public Key Model*, Crypto 87, pp. 135-155.

[GJKR] R. Gennaro, S. Jarecki, H. Krawczyk, T. Rabin, *Robust Threshold DSS Signatures*, *Advances in Cryptology – Eurocrypt* 96 *Proceedings*, Lecture Notes in Computer Science Vol. 1070, U. Maurer ed., Springer-Verlag, 1996, pp. 354-371.

[Gr97] O. Goldreich, *On Foundations of Modern Cryptography*, an invited paper, Crypto 97.

[GMW86] O. Goldreich, S. Micali, and A. Wigderson, "Proofs that yield nothing but their validity and a methodology of cryptographic protocol design," IEEE FOCS 1986, 174–187.

[GMW] O. Goldreich, S. Micali, and A. Wigderson, *How to play any mental game*, Proceedings of the Nineteenth annual ACM Symp. Theory of Computing, 1987, pp 218–229.

[Gw97] S. Goldwasser, *A New Directions in Cryptography: Twenty something years after*, an invited paper, FOCS 97.

[GMR] S. Goldwasser, S. Micali and C. Rackoff, *The Knowledge Complexity of Interactive Proof-Systems*, Siam J. on Computing, 18(1) (1989), pp. 186-208.

[HW] G. Hardy and E. Wright *An introduction to the theory of numbers*, Oxford Science Publications, London, Great Britain, fi fth ed., 1985

[HJJKY] A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk, M. Yung, *Proactive Public-Key and Signature Schemes Proceedings of the Fourth Annual Conference on Computer and Communications Security*, ACM, 1996.

[IY87] R. Impagliazzo, and M. Yung, "Direct minimum-knowledge computation," in *Advances in Cryptology—CRYPTO '87 Proceedings* (Lecture Notes in Computer Science, Vol. 293), ed. C. Pomerance, 40–51, Springer-Verlag, New York, 1988.

[K] J. Kilian, "Founding cryptography on oblivious transfer," ACM STOC 1988, 20–31.

[M76] G. Miller, *Riemann's Hypothesis and Test of Primality*, J. of Comp. and Syst. Sciences, 13, 300–317, 1976.

[OY91] R. Ostrovsky and M. Yung, *How to withstand mobile virus attacks*, Proc. of the 10th ACM Symposium on the Principles of Distributed Computing, 1991, pp. 51-61.

[P] T.P. Pedersen, *Distributed Provers with Applications to Undeniable Signatures*, *Advances in Cryptology – Eurocrypt* 91 *Proceedings*, Lecture Notes in Computer Science Vol. 547, D. Davies ed., Springer-Verlag, 1991, pp. 221-242.

[P2] T.P. Pedersen, *A threshold cryptosystem without a trusted party*, *Advances in Cryptology – Eurocrypt* 91 *Proceedings*, Lecture Notes in Computer Science Vol. 547, D. Davies ed., Springer-Verlag, 1991, pp. 129-140.

[P91] T.P. Pedersen, *Non-interactive and information theoretic secure verifi able secret sharing*, *Advances in Cryptology – Crypto* 91 *Proceedings*, Lecture Notes in Computer Science Vol. 576, J. Feigenbaum ed., Springer-Verlag, 1991, pp. 129-140.

[RSA] R. Rivest, A. Shamir and L. Adleman, *A Method for Obtaining Digital Signature and Public Key Cryptosystems*, Comm. of ACM, 21 (1978), pp. 120-126.

[Sh] A. Shamir, *How to share a secret*, Comm. of ACM, 22 (1979), pp. 612-613.

[Y82a] A. C. Yao, *Theory and Applications of Trapdoor functions*, Proceedings of the 23th Symposium on the Foundation of Computer Science, 1982, pp. 80-91.

[Y82] A. C. Yao, "Protocols for secure computations," IEEE FOCS 1982, 160–164.

[Y86] A. C. Yao, "How to generate and exchange secrets," IEEE FOCS 1986, 162–167.