Efficient Generation of Shared RSA keys

Dan Boneh[*]
dabo@cs.stanford.edu

Matthew Franklin
franklin@cs.ucdavis.edu

Abstract

We describe efficient techniques for a number of parties to jointly generate an RSA key. At the end of the protocol an RSA modulus $N = pq$ is publicly known. None of the parties know the factorization of N. In addition a public encryption exponent is publicly known and each party holds a share of the private exponent that enables threshold decryption. Our protocols are efficient in computation and communication. All results are presented in the *honest but curious* settings (passive adversary).

Keywords: RSA, Threshold Cryptography, Primality testing, Multiparty computation.

1 Introduction

We present efficient protocols for a number of parties to jointly generate an RSA modulus $N = pq$ where p, q are prime. At the end of the computation the parties are convinced that N is indeed a product of two large primes. However, none of the parties know the factorization of N. We then show how the parties can proceed to compute a public exponent e and shares of the corresponding private exponent. Our techniques require a number of steps including a new distributed primality test. The test enables two (or more) parties to test that a random integer N is a product of two large primes without revealing the primes themselves.

Several cryptographic protocols require an RSA modulus $N = pq$ for which none of the participants know the factorization. A good example is the original Fiat-Shamir authentication protocol [20] where all parties use the same modulus N, but none of them know its factorization. For other examples see [19, 25, 33, 36, 37]. Usually a modulus N with an unknown factorization is obtained by asking a dealer to generate it. The dealer must be *trusted* not to reveal the factorization of N. Our results eliminate the need for a trusted dealer since the parties can generate the modulus N themselves.

Threshold cryptography is a concrete example where shared generation of RSA keys is very useful. We give a brief motivating discussion and refer to [26] for a survey. Let $N = pq$ be an RSA modulus and d, e be signing and verification exponents respectively, i.e. $de = 1 \bmod \varphi(N)$. A threshold RSA signature scheme involves k parties and enables any subset of t of them to generate an RSA signature of a given message. No subset of $t - 1$ parties can generate a signature. Unlike standard secret sharing [42], the signature is generated without having to reconstruct the private decryption exponent d. A simple approach for obtaining a k-out-of-k threshold signature scheme is as follows [21]: Pick random d_i's satisfying $d = \sum_{i=1}^{d} d_i \bmod \varphi(N)$ and give d_i to party i. At a high level, to sign a message m each party computes $s_i = m^{d_i} \bmod N$ and sends s_i to a combiner (that

---

[*] Supported by DARPA contract #F30602-97-C-0326

holds no secrets). The combiner multiplies the s_i's and obtains the signature. This way the parties are able to generate a standard RSA signature without having to reconstruct the private key d at a single location. This is clearly advantageous for securing a sensitive private RSA key such as the one used by a Certificate Authority. Constructions providing a t-out-of-k RSA threshold signatures schemes can be found in [43, 17, 14, 22, 41].

An important issue left out of the above discussion is the initial generation of the RSA modulus N and the shares d_i. Traditionally the modulus N and the shares of the private key are assumed to be generated by a *trusted dealer*. Clearly, the dealer, or anyone who compromises the dealer, can forge signatures. Our results eliminate the need for a trusted dealer since the k parties can generate N and the private shares themselves. Such results were previously known for the ElGamal public key system [38, 28], but not for RSA.

The paper is organized as follows. We give a high level description of our protocol in Section 2. In Sections 3–5 we explain the various stages for generating a modulus $N = pq$ with an unknown factorization. In Section 7 we describe protocols that given a public exponent e, generate k shares of the corresponding private key $d = e^{-1} \bmod \varphi(N)$. The resulting sharing is a k-out-of-k sharing, namely all k shares are needed to use the private key. To defend against loss of shares one often shares d using a t-out-of-k scheme. This allows any subset of t users to apply the private key d. We explain how to obtain a t-out-of-k sharing of d in Section 7.3. We discuss various practical improvements to our protocol in Section 8.

We note that generic secure circuit evaluation techniques, e.g. [46, 30, 4, 11] can also be used to generate shared RSA keys. After all, a primality test can be represented as a boolean circuit. However, such general techniques are too inefficient.

1.1 Communication and privacy model

Let k be the number of parties engaging in the distributed RSA key generation protocol. Throughout the paper we assume $k \geq 3$. The communication and privacy model assumed by our protocol are as follows:

Full connectivity: Any party can communicate with any other party.

Private and authenticated channels: Messages sent from party A to party B are private and cannot be tampered with en route. This can be achieved by having A and B share a secret key which they use for encryption, integrity, and authentication.

Honest parties: We assume all parties are honestly following the protocol. At the end of the protocol no threshold of parties have enough information to factor the generated modulus. This is often called the *honest but curious* scenario. Recently, Frankel, MacKenzie and Yung [24] showed how our protocol can be made robust against $\lfloor \frac{k-1}{2} \rfloor$ malicious parties.

Collusion: Our protocol is $\lfloor \frac{k-1}{2} \rfloor$ private. That is, any coalition of size at most $\lfloor \frac{k-1}{2} \rfloor$ learns no information about the factorization of $N = pq$. However, a coalition of more than $\lfloor \frac{k-1}{2} \rfloor$ can recover the factorization. The reason for this bound is our reliance on the BGW [4] protocol in several steps of the algorithm. The BGW protocol provides information theoretic security and hence is limited to achieving $\lfloor \frac{k-1}{2} \rfloor$ privacy (in the honest parties model). Note that we are only considering static (non-adaptive) adversaries. Recently, Poupard and Stern [39] and Gilboa [29] showed how to achieve $k-1$ privacy for RSA modulus generation by replacing the

BGW steps with a protocol based on 1-out-of-n oblivious transfer. These BGW-replacement protocols can be proven secure based on the security of the underlying oblivious transfer mechanism. However, these protocols are not as efficient as BGW. We note that one can also achieve $k-1$ privacy for RSA modulus generation by replacing the BGW steps with a heuristic protocol due to Cocks [12]. Unfortunately, Cocks' protocol is much slower and its security is based on a heuristic argument.

To prove the security properties (i.e. privacy) satisfied by our protocol we provide a simulation argument for each of its components. The simulation shows that given the inputs and outputs to the protocol the participants can simulate the transcript of the protocol. Hence, they learn nothing new from the transcript. See [31, 35] for more information on simulation arguments.

2 Overview

In this section we give a high level overview of the key generation protocol. The k parties wish to generate a shared RSA key. That is, they wish to generate an RSA modulus $N = pq$ and a public/private pair of exponents e, d where $e \cdot d = 1 \bmod \varphi(N)$. The factors p and q should be at least n bits each. At the end of the computation N and e are public, and d is shared between the k parties in a way that enables threshold decryption and signatures. All parties should be convinced that N is a product of two primes, but no coalition of at most $t = \lfloor \frac{k-1}{2} \rfloor$ parties should have any information about the factors of N (beyond what is revealed by N itself).

At a high level the protocol works as follows:

(1) pick candidates: The following two steps are repeated twice.

- **(a) secret choice:** Each party i picks a secret n-bit integer p_i and keeps it secret.
- **(b) trial division:** Using a private distributed computation the k parties determine that $p = p_1 + \ldots + p_k$ is not divisible by any prime less than some bound B_1. Details are given in Section 6. If this step fails repeat Step (a).

Denote the secret values picked at the first iteration by $p_1, \ldots, p_k$, and at the second iteration by $q_1, \ldots, q_k$.

(2) compute N: Using a private distributed computation the k parties compute

$$N = (p_1 + \ldots + p_k) \cdot (q_1 + \ldots + q_k)$$

Other than the value of N, this step reveals no further information about the secret values $p_1, \ldots, p_k$ and $q_1, \ldots, q_k$. Details are given in Section 5.

Now that N is public, the k parties can perform further trial divisions and test that N is not divisible by small primes in the range $[B_1, B_2]$ for some bound B_2.

(3) biprimality test: The k parties engage in a private distributed computation to test that N is the product of two primes. If the test fails, then the protocol is restarted from Step 1. We note that the biprimality test protocol is $k-1$ private and applies whenever two (or more) parties are involved. Details are in Section 4.

(4) key generation: Given a public encryption exponent e, the parties engage in a private distributed computation to generate a shared secret decryption exponent d. Details are in Section 7.

Notation Throughout the paper we adhere to the following notation. The RSA modulus is denoted by N and is a product of two n bit primes p, q. When $p = \sum p_i$ we denote by p_i the share in possession of party i. Similarly for q_i. When the p_i's themselves are shared among the parties we denote by $p_{i,j}$ the share of p_i that is sent to party j.

Performance issues Our protocol generates two random numbers p and q and tests that $N = pq$ is a product of two primes. By the prime number theorem the probability that both p and q are simultaneously prime is asymptotically $(1/\nu n)^2$ where $\nu = \ln 2$. Therefore, naively one has to perform $(\nu n)^2$ probes on average until a suitable N is found. This is somewhat worse than the expected $2\nu n$ probes needed in traditional generation of an RSA modulus (one first generates one prime using νn probes and then a second prime using another νn probes). This $\nu n/2$ degradation in performance is usually unacceptable (typically $n = 512$).

Fortunately, thanks to trial division things aren't so bad. Our trial division (Step 1b) tests each prime individually. Therefore, to analyze our protocol we must analyze the effectiveness of trial division. Suppose a random n-bit number p passes the trial division test where all primes less than B are tested. We take $B = c \cdot n$ for some constant c. How likely is p to be prime? Using a classic result due to Mertens, DeBruijn [13] shows that asymptotically

$$\Pr[p \text{ prime } \mid \text{ trial division up to } B] = e^{\gamma} \frac{\ln B}{\ln 2^n}(1 + o(1/n)) = 2.57 \frac{\ln B}{n}(1 + o(1/n))$$

Hence, when $n = 512$ bits and $\ln B = 9$ (i.e. $B = 8103$) the probability that p is prime is approximately $1/22$. Consequently, traditional RSA modulus generation requires 44 probes while our protocol requires 484 probes. This eleven fold degradation in performance is unfortunate, but manageable. We discuss methods to avoid this slowdown in Section 8.

Generation of shares In Step (1) of the protocol each party uniformly picks a random n bit integer p_i as its secret share. The prime p is taken to be the sum of these shares. Since the sum of uniform independent random variables is *not* uniformly distributed, p is picked from a distribution with slightly less entropy than uniform. We show that this is not a problem. The sum $p = \sum_i p_i$ is at most an $n + \log k$ bit number. One can easily show that p is chosen from a distribution with at least n bits of entropy (since the n least significant bits of p are a uniformly chosen n bit string). These $\log k$ bits of "lost" entropy cannot help an adversary, since they can be easily guessed (the number of parties k is small, certainly $k < n$). This is formally stated in the next lemma.

A second issue is the fact that the shares p_i themselves leak some information about the factors of N. For instance, party i knows that $p > p_i$. We argue that this information does not help an adversary either.

The two issues raised above are dealt with in the following lemma. Let $\mathbb{Z}_n^{(2)}$ be the set of RSA moduli $N = pq$ that can be output by our protocol above when k parties are involved. We assume $k < \log N$.

Lemma 2.1 *Suppose there exists a polynomial time algorithm $\mathcal{A}$ that given a random $N \in \mathbb{Z}_n^{(2)}$ chosen from the distribution above and the shares $\langle p_i, q_i \rangle$ of $k-1$ parties, factors N with probability at least $1/n^d$. Then there exists an expected polynomial time algorithm $\mathcal{B}$ that factors $1/4k^3n^d$ of the integers in $\mathbb{Z}_n^{(2)}$.*

Assuming the hardness of factoring, the lemma shows that even an adversary who is given N and the private shares of $k-1$ parties cannot factor the modulus N generated by the protocol.

The proof of the lemma is somewhat tedious and is given in Appendix A so as not to distract the reader from the main thrust of the paper.

3 Privately computing products in a group

We first review a simple protocol due to Benaloh [5] that will be used throughout. Let G be a finite group. We write the group action multiplicatively. Suppose each of the k parties has a $y_i \in G$. They wish to compute $y = \prod_{i=1}^{k} y_i$ without revealing any further information about their secret shares y_i. Benaloh's protocol, which is $k-1$ private, works as follows:

Step 1: Each party i chooses k random elements $y_{i,j} \in G$ for $j = 1, \dots, k$ such that $y_i = \prod_{j=1}^{k} y_{i,j}$. For example, party i could pick random $y_{i,1}, \dots, y_{i,k-1} \in G$ and set $y_{i,k} = y / \prod_{j=1}^{k-1} y_{i,j}$.

Step 2: Each party i sends $y_{i,j}$ to party j.

Step 3: Each party j receives k shares $y_{i,j}$ for $i = 1, \dots, k$. Party j computes $\hat{y}_j = \prod_{i=1}^{k} y_{i,j}$ and sends $\hat{y}_j$ to each of the other k parties.

Step 4: Each party i receives $\hat{y}_1, \dots, \hat{y}_k$, and computes the required product as $y = \prod_{j=1}^{k} \hat{y}_j$.

Observe that at the end of the protocol each party has the correct product $y = \prod_{i=1}^{k} y_i$. The protocol is $k-1$ private since each coalition of up to $k-1$ users can simulate its transcript of the protocol. Indeed, let $U \subset \{1, \dots, k\}$ be a coalition of size s. By symmetry we may assume $U = \{1, \dots, s\}$. The coalition's view of the protocol is $\langle y, y_i, y_{i,j}, \hat{y}_j \rangle$ where $i = 1, \dots, s$, and $j = 1, \dots, k$. Then given $\langle y, y_i \rangle$ the simulator simulates this transcript by picking $y_{i,j}$ as random elements in G and picking $\hat{y}_j$ as random elements in G satisfying $\prod_{j=1}^{k} \hat{y}_j = y$. This is a perfect simulation of the transcript, and hence the coalition learns nothing new from it. See [5] for more details.

4 Distributed biprimality test

We begin the detailed discussion of the protocol with the distributed biprimality test (Step 3 of Section 2). Party i has two secret n-bit integers p_i, q_i. All parties know N where $N = pq = (\sum p_i)(\sum q_i)$. They wish to determine if N is the product of two primes without revealing any information about the factors of N. We refer to this test as a distributed biprimality test. Our biprimality test is a probabilistic test [44, 40] carried out in both $\mathbb{Z}_N^*$ and a quadratic extension of $\mathbb{Z}_N^*$.

Throughout the section we are assuming that $p = q = 3 \bmod 4$ (hence the resulting $N = pq$ is a Blum integer). This can be arranged ahead of time by having party 1 pick shares $p_1 = q_1 = 3 \bmod 4$. All other parties pick shares $p_i = q_i = 0 \bmod 4$.

Before describing the test we briefly discuss the structure of the quadratic extension of $\mathbb{Z}_N^*$ we will be using. We will be working in the group $\mathbb{T}_N = (\mathbb{Z}_N[x]/(x^2+1))^*/\mathbb{Z}_N^*$. Suppose all prime factors of N are equal to 3 mod 4. In this case, x^2+1 is irreducible in $\mathbb{Z}_N[x]$ and $\mathbb{Z}_N[x]/(x^2+1)$ is a quadratic extension of $\mathbb{Z}_N$. A linear polynomial $f(x) = \alpha x + \beta$ in $\mathbb{Z}_N[x]/(x^2+1)$ is invertible if and only if $\gcd(\alpha, \beta, N) = 1$. It follows that elements of $\mathbb{T}_N$ can be viewed as linear polynomials

$f(x) = \alpha x + \beta \in \mathbb{Z}_N[x]$ with $\gcd(\alpha, \beta, N) = 1$. Two linear polynomials $f, g \in \mathbb{Z}_N[x]$ represent the same element of $\mathbb{T}_N$ if $f = ag$ for some $a \in \mathbb{Z}_N^*$.

Distributed biprimality test:

Step 1: The parties agree on a random $g \in \mathbb{Z}_N^*$. The value g is known to all k parties.

Step 2: Party 1 computes the Jacobi symbol of g over N. If $\left(\frac{g}{N}\right) \neq 1$ the protocol is restarted at Step (1) and a new random g is chosen.

Step 3: Otherwise, party 1 computes $v_1 = g^{(N-p_1-q_1+1)/4} \bmod N$. All other parties compute $v_i = g^{-(p_i+q_i)/4} \bmod N$. The parties use the protocol of Section 3 to compute $v = \prod_{i=1}^k v_i \bmod N$. They then check if

$$v = \prod_{i=1}^{k} v_i \stackrel{?}{=} \pm 1 \pmod N$$

If the test fails then the parties declare that N is not a product of two primes.

Step 4: The parties perform a Fermat test in the group $\mathbb{T}_N = (\mathbb{Z}_N[x]/(x^2+1))^*/\mathbb{Z}_N^*$. To carry out the Fermat test in $\mathbb{T}_N$ the parties pick a random $h \in \mathbb{T}_N$. Party 1 computes $u_1 = h^{N+p_1+q_1+1}$. All other parties compute $u_i = h^{p_i+q_i}$. The parties then use the protocol of Section 3 to compute $u = \prod_{i=1}^k u_i$. They then check if

$$u = \prod_{i=1}^{k} u_i = h^{N+p+q+1} \stackrel{?}{=} 1$$

If the test fails N is rejected. Otherwise they declare success.

Since $p_1 = q_1 = 3 \bmod 4$ and $p_i = q_i = 0 \bmod 4$ for $i > 1$, the exponents in the computation of the v_i's (Step 3) are guaranteed to be integers after division by 4. The correctness and privacy of the protocol is proved in the next two lemmas.

Lemma 4.1 *Let $N = pq$ be an integer with $p \equiv q \equiv 3 \bmod 4$. If N is a product of two distinct primes then success is declared in all invocations of the protocol. Otherwise, the parties declare that N is not a product of two primes with probability at least $\frac{1}{2}$ (over the random choice of g and h).*

Proof Observe that in Step (3) of the protocol we test that $\prod v_i = \pm 1 \bmod N$ which amounts to testing $g^{(N-p-q+1)/4} \equiv \pm 1 \bmod N$.

Suppose p and q are distinct primes. In Step (2) we verify that $\left(\frac{g}{N}\right) = 1$. This implies $\left(\frac{g}{p}\right) = \left(\frac{g}{q}\right)$. Also, since $\frac{q-1}{2}$ and $\frac{p-1}{2}$ are odd we have,

$$\begin{cases} g^{\varphi(N)/4} = \left(g^{\frac{p-1}{2}}\right)^{\frac{q-1}{2}} \equiv \left(\frac{g}{p}\right)^{\frac{q-1}{2}} = \left(\frac{g}{p}\right) \pmod p \\ g^{\varphi(N)/4} = \left(g^{\frac{q-1}{2}}\right)^{\frac{p-1}{2}} \equiv \left(\frac{g}{q}\right)^{\frac{p-1}{2}} = \left(\frac{g}{q}\right) \pmod q \end{cases}$$

Since $\left(\frac{g}{p}\right) = \left(\frac{g}{q}\right)$ it follows that $g^{\varphi(N)/4} \equiv \pm 1 \bmod N$. Since $\varphi(N) = N - p - q + 1$ when p and q are prime, it follows that the test in Step 3 always succeeds.

Similarly, we show that when p and q are distinct primes the test in Step 4 succeeds. Since $p \equiv q \equiv 3 \bmod 4$ the polynomial $x^2 + 1$ has no root in $\mathbb{F}_p$ and $\mathbb{F}_q$. Therefore, $\mathbb{F}_p[x]/(x^2 + 1)$

and $\mathbb{F}_q[x]/(x^2+1)$ are quadratic extensions of $\mathbb{F}_p$ and $\mathbb{F}_q$ respectively. It follows that the group $\mathbb{T}_p = (\mathbb{Z}_p[x]/(x^2+1))^*/\mathbb{Z}_p^*$ is of order $p+1$. Similarly, $|\mathbb{T}_q| = q+1$. By the Chinese Remainder Theorem, $\mathbb{T}_N = \mathbb{T}_p \times \mathbb{T}_q$ and hence $|\mathbb{T}_N| = (p+1)(q+1)$. We conclude that all $h \in \mathbb{T}_N$ satisfy $h^{(p+1)(q+1)} = 1$. Consequently, the test of Step 4 always succeeds.

To prove the converse, suppose $N = pq$ is not a product of two distinct primes. Then, either at least one of p, q is not prime, or $p = q$. Let $N = r_1^{d_1} \cdots r_s^{d_s}$ be the factorization of N into s distinct primes. Set $e = (N-p-q+1)/4 = (p-1)(q-1)/4$ to be the exponent used in Step (3). Note that e is odd since $p \equiv q \equiv 3 \bmod 4$. Define the following two subgroups of $\mathbb{Z}_N^*$:

$$G = \{g \in \mathbb{Z}_N \text{ s.t. } \left(\frac{g}{N}\right) = 1\} \quad \text{and} \quad H = \{g \in G \text{ s.t. } g^e = \pm 1 \bmod N\}$$

First note that H is a subgroup of G: the set H is closed under multiplication, $1 \in H$, and $H \subseteq G$. To see that $H \subseteq G$ observe that if $g^e = \pm 1 \bmod N$ then, since e is odd, it follows that $\left(\frac{g}{N}\right) = \left(\frac{g^e}{N}\right) = \left(\frac{\pm 1}{N}\right)$. But then $\left(\frac{g}{N}\right) = 1$ since $\left(\frac{1}{N}\right) = \left(\frac{-1}{N}\right) = 1$ when $N = 1 \bmod 4$. Hence, $g \in G$.

To prove the lemma we show that $|H| \leq \frac{1}{2}|G|$. This will prove that N is rejected in Step (3) with probability at least $\frac{1}{2}$. Since H is a subgroup of G it suffices to prove proper containment of H in G, i.e. prove the existence of $g \in G \setminus H$. There are four cases to consider.

Case 1. Suppose $s \geq 3$. Let r_2, r_3 be distinct prime factors of N such that $d_2 = d_3 \bmod 2$ (such a pair must exist by the pigeonhole principle). Let r_1 be a prime factor of N distinct from both r_2 and r_3 and let a be a quadratic non-residue modulo r_3. Define $g \in \mathbb{Z}_N$ to be an element satisfying

$$g \equiv 1 \bmod r_1 \quad \text{and} \quad g \equiv -1 \bmod r_2 \quad \text{and} \quad g \equiv \begin{cases} 1 \bmod r_3 & \text{if } \left(\frac{-1}{r_2}\right) = 1 \\ a \bmod r_3 & \text{if } \left(\frac{-1}{r_2}\right) = -1 \end{cases}$$

and $g \equiv 1 \bmod r_i$ for $i > 3$. Observe that $g \in G$. Since e is odd $g^e = g = 1 \bmod r_1$ and $g^e = g = -1 \bmod r_2$. Consequently, $g^e \neq \pm 1 \bmod N$ i.e. $g \notin H$.

Case 2. Suppose $\gcd(p, q) > 1$. Then there exists an odd prime r such that r divides both p and q. Then r^2 divides N implying that r divides $\varphi(N)$. It follows that in $\mathbb{Z}_N^*$ there exists an element g of order r. Since r is odd we have $\left(\frac{g}{N}\right) = \left(\frac{g^r}{N}\right) = \left(\frac{1}{N}\right) = 1$, i.e. $g \in G$. Since r divides both p and q we know that r does not divide $N-p-q+1 = 4e$. Consequently $g^{4e} \neq 1 \bmod N$ implying that $g^e \neq \pm 1 \bmod N$. Hence, $g \notin H$.

Case 3. The only way $N = pq$ does not fall into either of the two cases above is if $p = r_1^{d_1}$ and $q = r_2^{d_2}$ where r_1, r_2 are distinct primes and at least one of d_1, d_2 is bigger than 1 (Case 2 handles N that are a prime power $N = r^d$). By symmetry we may assume $d_1 > 1$. Since $\mathbb{Z}_p^*$ is a cyclic group of order $r_1^{d_1-1}(r_1-1)$ it contains an element of order $r_1^{d_1-1}$. It follows that $\mathbb{Z}_N^*$ also contains an element g of order $r_1^{d_1-1}$. As before, $\left(\frac{g}{N}\right) = 1$, i.e. $g \in G$. If $q \neq 1 \bmod r_1^{d_1-1}$ then $4e = N-p-q+1$ is not divisible by $r_1^{d_1-1}$. Consequently, $g^{4e} \neq 1 \bmod N$, i.e. $g \notin H$.

Case 4. We are left with the case $N = pq$ with $p = r_1^{d_1}$, $q = r_2^{d_2}$, $d_1 > 1$ as above and $q \equiv 1 \bmod r_1^{d_1-1}$. Since $p = q = 3 \bmod 4$ we know that $r_1 = r_2 = 3 \bmod 4$. In this case it may indeed happen that $H = G$. We show that in this case, Step (4) of the biprimality test will fail with probability at least half (over the choice of $h \in \mathbb{T}_N$).

Define the group $H' = \{h \in \mathbb{T}_N \text{ s.t. } h^{(p+1)(q+1)} = 1\}$. We show that $|H'| \leq \frac{1}{2}|\mathbb{T}_N|$. Since H' is a subgroup of $\mathbb{T}_N$ it suffices to prove proper containment, i.e. we must exhibit an element $w \in \mathbb{T}_N \setminus H'$. Since $p = r_1^{d_1}$ the group $\mathbb{T}_p$ has order $r_1^{d_1-1}(r_1+1)$. Therefore it contains an element h of order r_1. It follows that there exists an element $w \in \mathbb{T}_N$ of order r_1. Since by assumption $q \equiv 1 \bmod r_1$ we know that r_1 does not divide $q+1$. Hence r_1 does not divide $N+p+q+1$ and therefore $w^{N+p+q+1} = w^{(p+1)(q+1)} \neq 1$. This proves $w \in \mathbb{T}_N \setminus H'$ completing the proof of the lemma. □

We note that Step (4) of the protocol is needed to filter out integers that fall into Case 4 above. Indeed, such integers will pass Steps (1)-(3). For example, consider integers $N = pq$ with $p = 3^s$ and $q = 2 \cdot 3^{s-1} + 1$ with s odd and q prime. For these integers we have $H = G$ where H, G are as defined in the body of the proof above. Consequently, such integers always pass Steps (1)-(3) even though they are not a product of two distinct primes. However, they will be rejected in Step (4) with probability at least $\frac{1}{2}$. In Section 4.1 we give an alternate approach for filtering out integers that fall into Case 4.

The following lemma shows that when N is the product of two distinct primes, the biprimality test protocol reveals no other information about the private shares of the participants.

Lemma 4.2 *Suppose p, q are distinct primes with $p = q = 3 \bmod 4$. Then any coalition of up to $k-1$ parties can simulate their view of the biprimality testing protocol. Consequently, the protocol is $k-1$ private.*

Proof Since p, q are prime we know that $v = \prod_{i=1}^{k} v_i = \pm 1 \bmod N$ where the v_i's are defined as in Step (3) of the protocol. Also, $u = \prod u_i = 1$ where the u_i's are defined in Step (4). Let U be a coalition of at most $k-1$ parties. The coalition's view consists of $\langle N, p_i, q_i, u_i, v_i, g, h, u, v\rangle$ for $i \in U$ and the intermediate values generated by the protocol of Section 3.

We construct a perfect simulator for the coalition's view as follows. The simulator is given N, p_i, q_i for $i \in U$ as input (this is the coalition's input when the protocol is started). The simulator starts by picking g. It does so by picking a random $x \in \mathbb{Z}_N^*$ and $b \in \{0,1\}$. It then sets $g = x^2 \cdot (-1)^b \bmod N$. Since $p = q = 3 \bmod 4$ it follows that -1 is a quadratic non-residue in $\mathbb{Z}_N$ with Jacobi symbol 1. Thus g is uniformly distributed among the elements of $\mathbb{Z}_N^*$ with Jacobi symbol 1. This distribution on g is identical to the distribution on g induced by the protocol, as required. Note that g is a quadratic residue in $\mathbb{Z}_N$ if and only if $b = 0$.

Next, the simulator picks a random $h \in \mathbb{T}_N$. It then computes the u_i, v_i values for all $i \in U$ using the p_i, q_i values it is given as input. Next the simulator must generate u, v. Simulating u is trivial since $u = 1$. To simulate v, recall from the first part of the proof of Lemma 4.1:

$$\begin{cases} v = \prod_{i=1}^{k} v_i = 1 \pmod{N} & \text{implies } \left(\frac{g}{p}\right) = \left(\frac{g}{q}\right) = 1 \\ v = \prod_{i=1}^{k} v_i = -1 \pmod{N} & \text{implies } \left(\frac{g}{p}\right) = \left(\frac{g}{q}\right) = -1 \end{cases}$$

Hence, the 'sign' of v is determined by whether g is a quadratic residue modulo N. Thus the simulator simply sets $v = b$.

Once u, v and u_i, v_i for $i \in U$ are established, the simulator for the protocol of Section 3 can be used to simulate the intermediate values generated by that protocol. To conclude, given that N is the product of two distinct primes, the set of values $\langle N, p_i, q_i, u_i, v_i, g, h, u, v\rangle$ produced by the simulator is indistinguishable from a real transcript. Consequently, the coalition learns no information other than what it had at the beginning of the protocol. □

We note that Step (2) of the protocol is crucial. Without it the condition of Step (3) might fail (and reveal the factorization) even when p and q are prime. We also note that in practice the probability that a non RSA modulus passes even one iteration of this test is actually far less than a half.

4.1 An alternative to Step (4)

Step (4) of the distributed biprimality protocol is necessary to filter out integers that fall into Case 4 in the proof of Lemma 4.1. We describe an alternative simpler approach to filter out such integers. It requires less computation, although there is more communication between the parties.

Observe that if N falls into Case 4 then $\gcd(N, p+q-1) > 1$. The alternative to Step (4) is to directly test this condition. To test this condition with no information leakage the parties do the following. Each party picks a random $r_i \in \mathbb{Z}_N$ and keeps it secret. Using the protocol of the next section they compute

$$z = \left(\sum_{i=1}^{k} r_i\right)\left(-1 + \sum_{i=1}^{k}(p_i + q_i)\right) \pmod N$$

without leaking any information about the private shares p_i, q_i or r_i. Finally, the parties check if $\gcd(z, N) > 1$. If so, N is rejected. Using the BGW method (Section 5.1) this approach is $\lfloor \frac{k-1}{2} \rfloor$ private. All N that fall into case 4 are rejected. All N that are a product of two distinct primes pass the test with overwhelming probability. We note that this alternate test eliminates a few valid RSA moduli, i.e. moduli $N = pq$ with p, q prime and $q = 1 \bmod p$.

5 Distributed computation of N

Next we describe the computation of N (Step 2 of Section 2). Each party has a secret p_i, q_i. They wish to make the product $N = (\sum p_i)(\sum q_i)$ public without revealing any information about their private shares beyond what is revealed by the knowledge of N.

5.1 The BGW method

BenOr, Goldwasser and Wigderson [4] describe an elegant protocol for private evaluation of general functions for three or more parties. Their full technique is an overkill for the simple function we have in mind. We adapt their protocol to the computation at hand so as to minimize the amount of computation and communication between the parties. From here on, let $P > (k2^n)^2 > N$ be some prime. Unless otherwise stated, all arithmetic operations are done modulo P. The protocol works as follows:

Step 1: Let $\ell = \lfloor \frac{k-1}{2} \rfloor$. For all $i = 1, \ldots, k$ party i picks two random degree ℓ polynomials $f_i, g_i \in \mathbb{Z}_P[x]$ satisfying $f_i(0) = p_i$ and $g_i(0) = q_i$. In other words, the constant term of f_i, g_i are set to p_i, q_i and all other coefficients are chosen at random in $\mathbb{Z}_P$. Similarly, party i picks a random degree 2ℓ polynomial $h_i \in \mathbb{Z}_p[x]$ satisfying $h_i(0) = 0$.

Step 2: For all $i = 1, \ldots, k$ party i computes the $3k$ values:

$$\forall j = 1, \ldots, k: \quad p_{i,j} = f_i(j) \quad ; \quad q_{i,j} = g_i(j) \quad ; \quad h_{i,j} = h_i(j)$$

Party i then privately sends the triple $\langle p_{i,j}, q_{i,j}, h_{i,j} \rangle$ to party j for all $j \neq i$. Note that the $p_{i,j}$ for $j = 1, \ldots, k$ are standard ℓ-out-of-k Shamir secret sharings of p_i. The same holds for q_i.

Step 3: At this point each party i has all of $\langle p_{j,i}, q_{j,i}, h_{j,i} \rangle$ for $j = 1, \ldots, k$. Party i computes:

$$N_i = \left(\sum_{j=1}^{k} p_{j,i} \right) \cdot \left(\sum_{j=1}^{k} q_{j,i} \right) + \sum_{j=1}^{k} h_{j,i} \pmod P$$

Party i broadcasts N_i to all other parties.

Step 4: At this point each of the parties has all values N_i for $i = 1, \ldots, k$. Let $\alpha(x)$ be the polynomial

$$\alpha(x) = \left(\sum_{j=1}^{k} f_j(x) \right) \left(\sum_{j=1}^{k} g_j(x) \right) + \sum_{j=1}^{k} h_j(x) \pmod P$$

Observe that $\alpha(i) = N_i$ and by definition of f_i, g_i, h_i we have $\alpha(0) = N$. Furthermore, $\alpha(x)$ is a polynomial of degree 2ℓ. We note that ℓ is defined so that $k \geq 2\ell+1$. Consequently, since all parties have at least $2\ell+1$ points on $\alpha(x)$ they can interpolate it and discover its coefficients. Finally, each party evaluates $\alpha(0)$ and obtains $N \bmod P$. Since $N < P$ the parties learn the correct value of N.

From the description of the protocol it is clear that all parties learn the value N. We note that the protocol requires that at least three parties be involved. In the case of exactly three parties, linear polynomials are used and the protocol is 1-private.

The following lemma shows that no coalition of $\lfloor \frac{k-1}{2} \rfloor$ parties learns any further information about the private shares. This statement holds in the information theoretic sense – no complexity assumptions are needed. For completeness we sketch the proof of the lemma and refer to [4] for a more detailed discussion of the protocol.

Lemma 5.1 *Given N, any coalition of $\lfloor \frac{k-1}{2} \rfloor$ parties can simulate the transcript of the protocol. Consequently, the protocol is $\lfloor \frac{k-1}{2} \rfloor$ private.*

Proof Sketch Set $\ell = \lfloor \frac{k-1}{2} \rfloor$. By symmetry we may assume the coalition is made up of parties $1 \ldots \ell$. In what follows we consistently let the index i vary in $i = 1, \ldots, \ell$, the index j vary in $j = 1, \ldots, k$ and the index r vary in $r = \ell + 1, \ldots, k$. Then the coalition's view consists of $\langle p_i, q_i, f_i, g_i, h_i, p_{i,j}, q_{i,j}, h_{i,j}, p_{r,i}, q_{r,i}, h_{r,i}, N_j, N \rangle$.

To simulate the coalition's view, the simulator is given p_i, q_i, N as input. It first picks random $f_i, g_i, h_i \in \mathbb{Z}_P[x]$ with $f_i(0) = p_i$, $g_i(0) = q_i$ and $h_i(0) = 0$. Here f_i, g_i are of degree ℓ and h_i is of degree 2ℓ. Next, it computes $p_{i,j} = f_i(j)$, $q_{i,j} = g_i(j)$ and $h_{i,j} = h_i(j)$. It picks $p_{r,i}, q_{r,i}, h_{r,i}$ as random independent elements of $\mathbb{Z}_P$ and computes $N_i = (\sum_j p_{j,i})(\sum_j q_{j,i}) + \sum_j h_{j,i}$. Next, it picks a random degree 2ℓ polynomial $\alpha(x) \in \mathbb{Z}_P[x]$ satisfying $\alpha(i) = N_i$ and $\alpha(0) = N$. It completes the simulation by setting $N_r = \alpha(r)$. These values are a perfect simulation of the coalition's view. □

The above protocol consists of one phase of the full BGW method. It differs from the BGW protocol in that there is no need for a degree reduction step. Also, we combine the addition and multiplication stages into one phase. The resulting computation is surprisingly efficient. Essentially, there is only *one* multi-precision multiplication performed by each party (the one in Step 3).

5.2 BGW Modulo non primes

In our description of the BGW protocol all arithmetic operations were carried out modulo a prime $P > N$. In the coming sections it will be useful to run the BGW protocol while working modulo a non prime P. That is, the k parties wish to compute $N = (\sum p_i)(\sum q_i) \bmod M$ for some integer M, not necessarily prime. One can easily show that if M has no prime divisors smaller than k then the protocol can be used *as is*. Indeed, all Lagrange coefficients used during the interpolation at Step (4) exist. Lemma 5.1 remains correct.

Running the protocol modulo M containing small factors requires a slight modification. Write $M = M_1 \cdot M_2$ where M_1 has no prime factors smaller than k and M_2 has only prime factors smaller than k. As mentioned above the protocol immediately works modulo M_1. The problem when working modulo M_2 is that Shamir secret sharing (which is the basis of BGW) is not possible. For instance, consider the case when $M_2 = 3$. It is not possible to use Shamir secret sharing in $\mathbb{F}_3$ among $k > 3$ parties since $\mathbb{F}_3$ does not contain enough points (in Shamir secret sharing each party must be given its own unique point). A simple solution is to run the entire protocol in an algebraic extension of $\mathbb{F}_3$ that contains more than k points. The simulation argument immediately extends to this case. In general, one can factor M_2 into its prime factors and run the protocol in a large enough extension for each factor. Using the Chinese Remainder Theorem one can recover the value of $N \bmod M$.

5.3 Sharing the final outcome

In some cases (as in Section 7.2) we wish to have the parties evaluate the function $N = (\sum p_i)(\sum q_i)$; however the result should be additively shared among the parties rather than become publicly available. That is, at the end of the computation each party should have an M_i such that

$$\sum_{i=1}^{k} M_i = (\sum_{i=1}^{k} p_i)(\sum_{i=1}^{k} q_i) \pmod P$$

and no information is revealed about the private shares or the final result.

The modification to BGW in order to achieve the above goal is immediate. The parties do not perform Step 4 of the protocol and do not perform the broadcast described at the end of Step 3. Consequently, they each end up with a point on a polynomial $\alpha(x)$ of degree 2ℓ that evaluates to N at $x = 0$. Using Lagrange interpolation we know that

$$N = \alpha(0) = \sum_{i=1}^{k} \lambda_i(0) N_i \pmod P$$

where $\lambda_i(x)$ is the appropriate Lagrange coefficient (e.g., $\lambda_i(x) = \prod_{j \neq i} \frac{(x-j)}{(i-j)}$ when $k = 2\ell + 1$). Therefore, rather than broadcast N_i at the end of Step 3, party i simply sets $M_i = \lambda_i(0) N_i$. The resulting M_i's are an additive sharing of N as required. As before, there is a simple simulation argument showing that any minority of parties obtains no information from this protocol.

6 Trial division

In this section, we consider the trial division step (Step 1b in Section 2). Let $q = q_1 + \ldots + q_k$ be an integer shared among k parties. Let ℓ be a small prime. To test if ℓ divides q each party picks a random $r_i \in \mathbb{Z}_p$. Using the BGW protocol (as described in Section 5.1) they compute $qr = (\sum q_i)(\sum r_i) \bmod \ell$. If $qr \neq 0$ then ℓ does not divide q. Furthermore, since r is unknown to any minority of parties, qr provides no other information about q.

Note that in the above approach, a bad candidate q is always rejected. Unfortunately, a good candidate might also be rejected. Indeed, even if ℓ does not divide q it is still possible that $qr = 0 \bmod \ell$. To alleviate this problem one can repeat the above test twice for each small prime ℓ. Then the probability that a good candidate is rejected is at most $1 - \prod_{\ell < B}(1 - \frac{1}{\ell^2}) < \frac{1}{2}$.

One caveat in the above approach is that the BGW protocol as described in Section 5.1 cannot be applied to test divisibility of q by very small ℓ, namely $\ell \leq k$. The reason is that for such small ℓ the field $\mathbb{F}_\ell$ does not contain enough points to do Shamir secret sharing among k parties. For such small ℓ one must run the BGW protocol in an extension field of $\mathbb{F}_\ell$ that contains at least $k+1$ elements, as explained in Section 5.2.

7 Shared generation of public/private keys

Once the parties successfully construct an RSA modulus $N = pq = (\sum p_i)(\sum q_i)$ they may wish to compute shares of $d = e^{-1} \bmod \varphi(N)$ for a given encryption exponent e. We have two approaches for computing additive shares of d. The first only works for small e (say $e < 1000$) but is very efficient requiring very little communication between the parties. The second works for any e and is still efficient, but requires more communication.

Throughout the section we set $\varphi = \varphi(N)$. Since e is an RSA exponent we know that $\gcd(e, \varphi) = 1$. Recall that the public modulus $N = (\sum p_i)(\sum q_i)$ satisfies $\varphi(N) = \varphi = \sum_{i=1}^{k} \varphi_i$ where $\varphi_1 = N - p_1 - q_1 + 1$ and $\varphi_i = -p_i - q_i$ for $i = 2, \ldots, k$. Observe that for all $i = 1, \ldots, k$ party i can locally compute φ_i.

To compute shares of d the parties must invert e modulo $\sum \varphi_i$ without exposing their φ_i's. Unfortunately, traditional inversion algorithms, e.g. extended gcd, involve computations modulo $\sum \varphi_i$. We do not know how to efficiently perform modular arithmetic when the modulus is shared among the participants. Fortunately, there is a trick for computing $e^{-1} \bmod \varphi$ without using any reductions modulo φ. We compute the inverse of $e \bmod \varphi$ in three steps: (1) Compute $\zeta = \varphi^{-1} \bmod e$. (2) Set $T = -\zeta \cdot \varphi + 1$. Observe that $T \equiv 0 \bmod e$. (3) Set $d = T/e$. One can easily verify that $d = e^{-1} \bmod \varphi$ since $d \cdot e = T \equiv 1 \bmod \varphi$. Using this observation there is no need for reductions modulo φ. Both methods rely on this observation.

7.1 Small public exponent

We begin by describing an efficient technique for generating shares of d when the public exponent e is small. The protocol is $k-1$ private. However, it leaks the value of $\varphi(N) \bmod e$, and thus can only be used when e is small.

Step 1: The parties jointly compute the value of $\ell = \varphi \bmod e$. Since $\ell = \sum \varphi_i \bmod e$ it is possible to compute ℓ without revealing any other information about the private shares. To do so we use Benaloh's protocol [5] in the additive group $\mathbb{Z}_e$ as described in Section 3.

Step 2: Let $\zeta = \ell^{-1} \bmod e$. Then, as explained above, $d = (-\zeta \cdot \varphi + 1)/e$. Each party i locally computes:

$$d_i = \left\lfloor \frac{-\zeta \cdot \varphi_i}{e} \right\rfloor$$

As a result we have $d = \sum d_i + r$ where $1 \le r < k$.

Step 3: The above sharing of d enables shared decryption [21] using the equality $c^d \equiv c^r \prod c^{d_i} \bmod N$. Party 1 can determine the value of r by trying all possible values of $0 \le r \le k$ during a trial decryption.

The above approach leaks $\varphi(N) \bmod e$ and r. This is a total of $\log e + \log k$ bits. This information cannot help an opponent since it can be correctly guessed in time $O(ek)$.

The value of r is found using only one trial decryption: Party 1 first picks a random message $m \in \mathbb{Z}_N$ and computes $c = m^e \bmod N$. It then asks the parties to decrypt c. Each party computes $m_i = c^{d_i} \bmod N$ and sends the result to party 1. By performing at most k multiplications party 1 finds an $r \in [0, k)$ satisfying $m = (\prod m_i)c^r \bmod N$. It then updates d_1 by setting $d_1 = d_1 + r$. Then $\sum d_i = d \bmod \varphi$.

7.2 Arbitrary public exponent

Unlike the previous technique, our second method for generating shares of d works for arbitrary public exponent e. It is very efficient, but leaks $2 \log k$ bits. This information cannot help an opponent since it can be easily guessed. A simple modification to our protocol (Remark 7.2 below) shows how to generate d without leaking any information. However, this modification makes the protocol somewhat less efficient.

Rather than exposing $\varphi \bmod e$ and then inverting it (as we did above) we show how to invert $\varphi \bmod e$ while it is shared among the parties. As a result, no information about φ is revealed. The protocol is $\lfloor \frac{k-1}{2} \rfloor$ private.

Step 1. Each party i picks a random $r_i \in \mathbb{Z}_e$.

Step 2. Using the protocol of Section 5.2 the parties compute $\psi = (\sum r_i) \cdot (\sum \varphi_i) \bmod e$. At the end of the computation ψ is known to all parties. If ψ is not invertible modulo e the protocol is restarted at Step 1.

Step 3. Each party locally computes $\zeta_i = r_i \psi^{-1} \bmod e$. Observe $\sum \zeta_i = (\sum r_i)\psi^{-1} = \varphi^{-1} \bmod e$. Hence, the parties share $\varphi^{-1} \bmod e$ without revealing any information about their secret shares.

Step 4. Next, the parties agree on a prime $P > 2Ne$. They view the shares $0 \le \zeta_i < e$ as elements of $\mathbb{Z}_P$. Using the modified BGW protocol of Section 5.3 they compute an additive sharing

$$\sum_{i=1}^{k} T_i = -(\sum \zeta_i)(\sum \varphi_i) + 1 \pmod{P}$$

Each party has a $T_i \in \mathbb{Z}_P$ and any minority of parties learns no other information.

Step 5. From here on we regard the T_i's as integers $0 \le T_i < P$. Our objective is to ensure that over the integers

$$\sum_{i=1}^{k} T_i = -(\sum_{i=1}^{k} \zeta_i)(\sum_{i=1}^{k} \varphi_i) + 1 \tag{1}$$

We know that at the end of Step 4 we have $\sum T_i < kP$. Therefore, $\sum T_i = sP + u$ where $s < k$ and $0 \le u < P$. Given a candidate value of $s \in [0, k)$ party 1 can set $d_1 \leftarrow d_1 - sP$. If the given s is correct then $0 \le \sum T_i < P$ and the above equality holds over the integers. To determine the correct s the protocol proceeds to Step 6 with each possible value of s until the trial decryption in Step 6 succeeds.

Step 6. Assuming equality (1) holds over the integers we know that e divides $\sum T_i$. To see this observe that:

$$\sum T_i = -(\sum \zeta_i)(\sum \varphi_i) + 1 = -(\sum \varphi_i)^{-1}(\sum \varphi_i) + 1 \equiv 0 \pmod{e}$$

Therefore, $d = (\sum T_i)/e$. Each party i now sets $d_i = \lfloor T_i/e \rfloor$. As a result we have $d = \sum d_i + r \bmod \varphi(N)$ where $0 \le r \le k$. Party 1 can determine the value of r by trying all possible values of $0 \le r \le k$ during a trial decryption.

The protocol leaks the value of r and s. Hence, a total of at most $2 \log k$ bits is exposed. Player 1 determines the value of r and updates d_1 as described at the end of Section 7.1. To determine s this procedure is repeated for each of the k candidate values of s. Recall that k is typically small (e.g. less than 10).

Remark 7.1: Observe that Step 5 of the above protocol is needed due to the fact that the BGW method is carried out modulo P. However, it is possible to carry out the BGW protocol of Step 4 directly over the integers, avoiding Step 5 altogether. This can be done using a variant of Shamir secret sharing over the integers (see [22, 24]). Let $L = k!$. Then, each party i picks two polynomials $f_i, g_i \in \mathbb{Z}[x]$ of degree $\lfloor \frac{k-1}{2} \rfloor$ as follows. The constant terms are set to $f_0 = f(0) = L\zeta_i$ and $g_0 = g(0) = L\varphi_i$. All other coefficients are chosen at random from $\{0, L, 2L, \ldots, L^3N^2\}$. By interacting with the other parties, each party i computes a point on the polynomial $(\sum f_i)(\sum g_i)$ over the integers. These points, multiplied by the appropriate Lagrange coefficients, become the additive sharing $\sum T_i$ of $(\sum \zeta_i)(\sum \varphi_i)$ over the integers. Unfortunately, in this approach, the resulting shares d_i of the private key are of order N^2 rather than N as in the above protocol. This results in a factor of two slowdown during threshold signature generation. On the positive side, there is no need to leak the value of s, and there is no need to search for s by repeating Step 6 multiple times.

The computation of $\varphi^{-1} \bmod e$ (Steps 1-3 above) is based on a technique due to Bar-Ilan and Beaver [2, 3]. Recently, Catalano, Gennaro, and Halevi [10], introduced an alternate protocol for inverting a number modulo a shared modulus. The protocol leaks no secret information and is more efficient than the above protocol. The cost is that the resulting shares of d are larger.

7.3 t-out-of-k sharing

The previous two subsections explain how one can obtain a k-out-of-k sharing of d. However, to defend against loss of shares it is often desirable to have a t-out-of-k sharing enabling any subset of t parties to apply the private key. The simplest solution, due to T. Rabin [41], makes use of a generic technique for converting k-out-of-k sharing of a private RSA key into a t-out-of-k sharing scheme. T. Rabin's approach immediately applies once our k-out-of-k sharing of d is obtained.

8 Optimizations

We describe several practical techniques for improving the performance of our distributed protocol. These optimizations are incorporated in an implementation of our protocol [34].

Sieving In Step (1) of the protocol (Section 2) the parties repeatedly pick a random shared integer p until they find one that is not divisible by small primes. For this they engage in an interactive trial division protocol. We briefly outline a more efficient approach. Let M be the product of all odd primes less than some bound B_1. Suppose the k parties could generate an additive sharing $a = \sum a_i$ of a random integer a in $\mathbb{Z}_M^*$. Party i could then set its share of p to be $p_i = rM + a_i$ where r is some random number of the appropriate length to make p_i be an n-bit integer. The resulting candidate prime $p = \sum p_i$ is a random n-bit integer that is relatively prime to M. There is no need to run the trial division protocol on p with primes less than B_1. The only question that remains is how do the parties generate an additive sharing of a random integer $a \in \mathbb{Z}_M^*$? To do so, each party i generates a random element $b_i \in \mathbb{Z}_M^*$. Then $a = \prod b_i \bmod M$ is a random element of $\mathbb{Z}_M^*$. The parties then use the BGW method [4] with the modification described in Section 5.3 to convert the multiplicative sharing of a into an additive sharing. One caveat is that for this to work the BGW method must be made to work in $\mathbb{Z}_M$ (which is not a field). This is done as discussed in Section 5.2.

Load balancing In Step (3) of the biprimality test party 1 computes $v_1 = g^{(N+1-p_1-q_1)/4} \bmod N$ while all other parties only had to compute $v_i = g^{(p_i+q_i)/4} \bmod N$. Notice that $N+1$ is roughly $2n$ bits while $p_i + q_i$ is only about n bits. Consequently, party 1 works twice as hard as the other parties. To even things out it makes sense to test k candidate N in parallel. In each of these tests the role of party 1 is played by a different party. This way $g^{N+1} \bmod N$ is computed by a different party for each N. This results in better load balancing improving overall performance by up to a factor of two.

Parallel trial division Recall that once N is computed the parties perform trial division on it before invoking the distributed biprimality test (Step 2 in Section 2). The k parties can perform this trial division in parallel – each party is in charge of verifying that N is not divisible by some set of small primes. This can be efficiently done by hard-coding all primes $p_1, p_2, \ldots, p_l$ in the range $[B_1, B_2]$ in a list. Party i is in charge of testing that N is not divisible by any primes p_j for all $j \equiv i \bmod k$. This factor of k speedup enables us to use a larger bound B_2, increasing the effectiveness of trial division.

Fermat filter There is no need to run the full biprimality test of Section 4 on every candidate modulus N. Instead, one can do a Fermat test first and proceed to perform the full test only if the Fermat test succeeds. We refer to the test

$$g^{N-p-q+1} = g^{N+1-\sum(p_i+q_i)} = 1 \bmod N$$

as a Fermat test. If equality does not hold, then N is not a product of two distinct primes. To carry out this test in zero-knowledge all k parties locally compute $v_i = g^{p_i+q_i} \bmod N$.

They then use the protocol of Section 3 to compute $v = \prod_{i=1}^{k} v_i \bmod N$ and test that $v \overset{?}{=} g^{N+1} \bmod N$. This protocol is easily shown to be $k-1$ private. It saves the computation of the Jacobi symbol of g for most integers N.

Avoiding quadratic slowdown In Section 2 we noted that our protocols suffer from a quadratic slowdown in comparison to single user generation of an RSA key. The main reason is that both primes p, q are generated at once. We briefly outline a potential solution. Consider the case of three parties, Alice, Bob and Carol. They could generate N as

$$N = p_a q_b (r_a + r_b + r_c)$$

where Alice has $\langle p_a, r_a \rangle$ and p_a is an n-bit prime. Bob has $\langle q_b, r_b \rangle$ and q_b is an n-bit prime. Carol has $\langle r_c \rangle$. The number of probes until $r_a + r_b + r_c$ is found to be prime is just as in single user generation of N. Hence, we are able to avoid the quadratic slowdown. Furthermore, no single party knows the *complete* factorization of the resulting N. Unfortunately, this approach doesn't scale well. To enable t-privacy N must be a product of $t+2$ primes. Another drawback is that the parties must perform a distributed biprimality test to verify that N is the product of *three* primes. The techniques of this paper do not easily generalize to enable such a test. However, results of [8] can be used to test that a modulus shared among k parties is the product of three primes without revealing its factorization.

9 Summary and open problems

We presented techniques that allow three or more parties to generate an RSA modulus $N = pq$ such that all parties are convinced that N is indeed a product of two primes; however none of them can factor N. Our method achieves $\lfloor \frac{k-1}{2} \rfloor$ privacy. We also show show how the parties can generate shares of a private decryption exponent to allow threshold decryption. To test that N is the product of two primes we presented a distributed double-primality test. We note that our test was recently extended to a triple-primality test [8] enabling k parties to test that N is the product of *three* primes, without revealing any information about the factors.

To demonstrate the effectiveness of our protocols we implemented them. Generating a 1024 bits shared RSA key among three 300MhZ Pentium machines takes 90 seconds. Total network traffic is 1.16Mb. All communication is encrypted and authenticated using SSL. See [34] for a description of the implementation as well as detailed timing measurements.

Our protocol works when three or more parties are involved in key generation. It cannot be used ‘as is’ to generate a distributed key among two parties. This is due to our reliance on the BGW protocol in several steps. Recently, Poupard and Stern [39] and Gilboa [29] showed how to replace these BGW steps with an alternate protocol that achieves $k-1$ privacy. Using these techniques our protocol can generate shared RSA keys between two parties only. We also note that recently Frankel, MacKenzie and Yung [24] showed how our protocol can be made robust against $\lfloor \frac{k-1}{2} \rfloor$ *malicious parties.*

An important open problem is the generation of shared RSA keys of special form. For example, a modulus which is a product of “strong primes” (i.e., where both $\frac{p-1}{2}$ and $\frac{q-1}{2}$ are prime) has been considered for security purposes [32] as well as for technical reasons related to threshold cryptography [17, 27, 43]. Currently, our techniques do not enable shared generation of moduli of special form. Progress in this directions would be very helpful.

Recently, Shoup [43] presented a robust threshold RSA signature scheme. One would like to use the distributed key generation algorithm for generating a shared secret key for this scheme. Unfortunately, this is currently not possible. Shoup's scheme requires a modulus which is a product of strong primes. As mentioned above, distributed generation of such a modulus is currently an open problem. It is likely that strong primes are not inherently required for Shoup's scheme. Modifying Shoup's scheme so that it does not rely on strong primes would remove the current requirement for a trusted dealer.

Acknowledgments

We thank Don Beaver for helpful discussions, and the anonymous reviewers for many suggestions that improved the presentation of this paper.

References

[1] N. Alon, Z. Galil and M. Yung, "Dynamic-resharing verifiable secret sharing," Proc. 3rd Annual European Symposium on Algorithms (ESA), Springer-Verlag LNCS 979, pp. 523–537, 1995.

[2] J. Bar-Ilan, D. Beaver, "Non-cryptographic fault-tolerant computing in a constant number of rounds", Proc. 8th Annual ACM Symposium on Principles of Distributed Computing (PODC), pp. 201–209, 1989.

[3] D. Beaver, "Security, fault tolerance, and communication complexity in distributed systems," Ph.D. thesis, Harvard University, May 1990.

[4] M. Ben-Or, S. Goldwasser, A. Wigderson, "Completeness theorems for non-cryptographic fault tolerant distributed computation", Proc. 20th Annual ACM Symposium on Theory of Computing (STOC), pp. 1–10, 1988.

[5] J. Benaloh (Cohen), "Secret sharing homomorphisms: keeping shares of a secret secret," *Advances in Cryptology – Crypto '86*, Springer-Verlag LNCS 263, pp. 251-260, 1987.

[6] S. Blackburn, S. Blake-Wilson, M. Burmester, S. Galbraith, "Shared generation of shared RSA keys", tech. report CORR 98-19, available at `http://...gm....pm`

[11] D. Chaum, C. Crépeau, and I. Damgård, “Multiparty unconditionally secure protocols,” Proc. 20th Annual ACM Symposium on Theory of Computing (STOC), pp. 11–19, 1988.

[12] C. Cocks, “Split knowledge generation of RSA parameters”, Available from the author `cliff_cocks@cesg.gov.uk`.

[13] N. De Bruijn, ”On the number of uncanceled elements in the sieve of Eratosthenes”, Proc. Neder. Akad. Wetensch, vol. 53, 1950, pp. 803–812. Reviewed in LeVeque Reviews in Number Theory, Vol. 4, Section N-28, p. 221.

[14] A. DeSantis, Y. Desmedt, Y. Frankel, M. Yung, “How to share a function securely”, Proc. 26th Annual ACM Symposium on Theory of Computing (STOC), pp. 522–533, 1994.

[15] Y. Desmedt, “Threshold cryptography,” European Transactions on Telecommunications and Related Technologies, Vol. 5, No. 4, July-August 1994, pp. 35–43.

[16] Y. Desmedt, G. Di Crescenzo, M. Burmester, “Multiplicative non-abelian sharing schemes and their applications to threshold cryptography”, *Advances in Cryptology – AsiaCrypt '94*, Springer-Verlag LNCS 917, pp. 21–32, 1995.

[17] Y. Desmedt and Y. Frankel, “Shared generation of authenticators and signatures”, *Advances in Cryptology – Crypto '91*, Springer-Verlag LNCS 576, 457–469, 1992.

[18] T. ElGamal, “A public key cryptosystem and a signature scheme based on the discrete logarithm”, *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[19] U. Feige, A. Fiat, A. Shamir, “Zero-knowledge proofs of identity,” *Journal of Cryptology*, Vol. 1, pp. 77–94, 1988.

[20] A. Fiat, A. Shamir, “How to prove yourself: Practical solutions to identification and signature problems,” *Advances in Cryptology – Crypto '86*, Springer-Verlag LNCS 263, pp. 186-194, 1987.

[21] Y. Frankel, “A practical protocol for large group oriented networks”, *Advances in Cryptology – Eurocrypt '89*, Springer-Verlag LNCS 434, pp. 56–61, 1990.

[22] Y. Frankel, P Gemmell, P. MacKenzie, M. Yung, “Optimal resilience proactive public key cryptosystems”, Proc. 38th Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 384–393, 1997.

[23] Y. Frankel, P Gemmell, M. Yung, “Witness-based Cryptographic Program Checking”, Proc. 28th Annual ACM Symposium on Theory of Computing (STOC), pp. 499–508, 1996.

[24] Y. Frankel, P. MacKenzie, M. Yung, “Robust efficient distributed RSA key generation”, Proc. 30th Annual ACM Symposium on Theory of Computing (STOC), pp. 663–672, 1998.

[25] M. Franklin and S. Haber, “Joint encryption and message-efficient secure computation,” *Journal of Cryptology*, 9 (1996), 217-232.

[26] P. Gemmell, “An introduction to threshold cryptography”, in CryptoBytes, a technical newsletter of RSA Laboratories, Vol. 2, No. 7, 1997.

[27] R. Gennaro, S. Jarecki, H. Krawczyk, T. Rabin, “Robust and efficient sharing of RSA functions”, *Advances in Cryptology – Crypto '96*, Springer-Verlag LNCS 1109, pp. 157–172, 1996.

[28] R. Gennaro, S. Jarecki, H. Krawczyk, T. Rabin, “Secure Distributed Key Generation for Discrete-Log Based Cryptosystems”, *Advances in Cryptology – Eurocrypt '99*, Springer-Verlag LNCS 1592, pp. 295–310, 1999.

[29] N. Gilboa, “Two party RSA key generation”, *Advances in Cryptology – Crypto '99*, Springer-Verlag LNCS 1666, pp. 116–129, 1999.

[30] O. Goldreich, S. Micali, A. Wigderson, “How to play any mental game”, Proc. 19th Annual ACM Symposium on Theory of Computing, pp. 218–229, 1987.

[31] S. Goldwasser, S. Micali, C. Rackoff, “The knowledge complexity of interactive proof systems”, *SIAM J. of Computing*, Vol. 18, No. 1, pp. 186–208, 1989.

[32] J. Gordon, “Strong primes are easy to find”, *Advances in Cryptology – Eurocrypt '84*, Springer-Verlag LNCS 209, pp. 216–223, 1985.

[33] L. Guillou and J. Quisquater, “A practical zero-knowledge protocol fitted to security microprocessor minimizing both transmission and memory,” *Advances in Cryptology – Eurocrypt '88*, Springer-Verlag LNCS 330, 123-128, 1988.

[34] M. Malkin, T. Wu, D. Boneh, “Experimenting with shared RSA key generation”, In proceedings of the Internet Society's 1999 Symposium on Network and Distributed System Security (SNDSS), pp. 43–56, 1999.

[35] S. Micali, P. Rogaway, “Secure computation”, *Advances in Cryptology – Crypto '91*, Springer-Verlag LNCS 576, pp. 392–404, 1992.

[36] K. Ohta and T. Okamoto, “A modification of the Fiat-Shamir scheme,” *Advances in Cryptology – Crypto '88*, Springer-Verlag LNCS 403, 232-243, 1990.

[37] H. Ong and C. Schnorr, “Fast signature generation with a Fiat Shamir-like scheme,” *Advances in Cryptology – Eurocrypt '90*, Springer-Verlag LNCS 473, pp. 432-440, 1991.

[38] T. Pederson, “A threshold cryptosystem without a trusted party,” Advances in Cryptology – Eurocrypt 91, Springer-Verlag LNCS 547, pp. 522–526, 1991.

[39] G. Poupard, J. Stern, “Generation of Shared RSA Keys by Two Parties”, *Advances in Cryptology – AsiaCrypt '98*, Springer-Verlag LNCS 1514, pp. 11–24, 1998.

[40] M. Rabin, “Probabilistic algorithm for testing primality”, *J. of Number Theory*, vol. 12, pp. 128–138, 1980.

[41] T. Rabin, “A simplified approach to threshold and proactive RSA”, *Advances in Cryptology – Crypto '98*, Springer-Verlag LNCS 1462, pp. 89–104, 1998.

[42] A. Shamir, “How to share a secret”, *Comm. of the ACM*, Vol. 22, 1979, pp. 612–613.

[43] V. Shoup, “Practical threshold signatures”, *Advances in Cryptology – Eurocrypt 2000*, Springer-Verlag LNCS 1807, pp. 207–220, 2000.

[44] R. Solovay, V. Strassen, "A fast monte carlo test for primality", *SIAM Journal of Computing*, vol. 6, pp. 84–85, 1977.

[45] M. Wegman and J. Carter, "New hash functions and their use in authentication and set equality", *J. Comput. Syst. Sci.* 22 (1981), 265–279.

[46] A. Yao, "How to generate and exchange secrets", Proc. 18th Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 162–167, 1986.

A. Proof of Lemma 2.1

Let $\mathbb{Z}_n^{(2)}$ be the set of RSA moduli $N = pq$ that can be output by our protocol when k parties are involved and each party picks two n bit shares p_i and q_i. We assume $k < \log N$. Lemma 2.1 shows that even an adversary who is given the private shares of $k-1$ of the parties cannot factor N (assuming the hardness of factoring a non-negligible fraction of the RSA moduli in $\mathbb{Z}_n^{(2)}$).

Lemma 2.1 *Suppose there exists a polynomial time algorithm $\mathcal{A}$ that given: (1) a random $N \in \mathbb{Z}_n^{(2)}$ chosen from the distribution on $\mathbb{Z}_n^{(2)}$ induced by the protocol, and (2) the shares $\langle p_i, q_i \rangle$ of $k-1$ parties, factors N with probability at least $1/n^d$. Then there exists an expected polynomial time algorithm $\mathcal{B}$ that factors $1/4k^3n^d$ of the integers in $\mathbb{Z}_n^{(2)}$.*

Proof Given $N \in \mathbb{Z}_n^{(2)}$ algorithm $\mathcal{B}$ works by repeating the following two steps until N is factored: (1) Pick $2(k-1)$ random independent n-bit integers $\langle p_i, q_i \rangle_{i=1}^{k-1}$, and (2) Run algorithm $\mathcal{A}$ on the input N and $\langle p_i, q_i \rangle_{i=1}^{k-1}$. We show that for $1/4k^3n^d$ of the integers in $\mathbb{Z}_n^{(2)}$, after an expected polynomial number of iterations algorithm $\mathcal{B}$ will output a factorization of N.

Our protocol for generating an RSA modulus $N = pq$ induces a distribution on $\mathbb{Z}_n^{(2)}$: an element N is generated by picking random n-bit numbers $\langle p_1, \ldots, p_k \rangle$ and $\langle q_1, \ldots, q_k \rangle$ and outputting the product of their sums. First, we slightly restrict the choice of p and q by disallowing values close to the boundaries, i.e. the largest and smallest p's and q's. The total weight of the elements we disallow is sufficiently small so as not to hurt $\mathcal{A}$'s probability of factoring the remaining N's. Then we show that on the remaining N's algorithm $\mathcal{B}$ succeeds in factoring essentially the same N's as algorithm $\mathcal{A}$.

Let $S_p = [a, b]$ be the shortest interval such that $\Pr[\sum_{i=1}^{k} p_i \in S_p] \geq 1 - \frac{1}{4n^d}$ where the distribution is over the choice of random n-bit integers $\langle p_1, \ldots, p_k \rangle$ for which $\sum p_i$ is prime. The set S_p contains all possible prime factors produced by the protocol except those close to the boundaries. Define $S \subseteq \mathbb{Z}_n^{(2)}$ as

$$S = \{p \cdot q \mid p, q \in S_p\}$$

Then when a random $N \in \mathbb{Z}_n^{(2)}$ is chosen by the protocol we have that

$$\Pr[N \in S] > (1 - \frac{1}{4n^d})^2 > 1 - \frac{1}{2n^d}$$

Hence, when a random $N \in S$ and $k-1$ of the shares used to generate p and q are given to algorithm $\mathcal{A}$ it will factor N with probability at least $\frac{1}{n^d} - \frac{1}{2n^d} = \frac{1}{2n^d}$. It follows that there exists a subset $S' \subseteq S$ with $\Pr[N \in S'] > 1/4n^d$ such that algorithm $\mathcal{A}$ succeeds in factoring all $N \in S'$ with probability at least $1/4n^d$ (over the choice of the p_i's and q_i's and the random bits used by $\mathcal{A}$). We show that algorithm $\mathcal{B}$ will factor all $N \in S'$ in expected polynomial time. Since the size of S is at least $1/k^2$ of $\mathbb{Z}_2^{(n)}$ and S' is at least $1/4n^dk$ of S we have that S' contains at least $1/4n^dk^3$ of the elements in $\mathbb{Z}_2^{(n)}$.

Consider a hypothetical algorithm $\mathcal{B}'$ that takes an integer $N = pq \in S'$ as input. Algorithm $\mathcal{B}'$ is identical to algorithm $\mathcal{B}$ except that it invokes algorithm $\mathcal{A}$ only if $\sum_{i=1}^{k-1} p_i$ is in the range $[p - 2^n, p - 2^{n-1}]$ and $\sum_{i=1}^{k-1} q_i$ is in the range $[q - 2^n, q - 2^{n-1}]$. Otherwise, algorithm $\mathcal{B}'$ restarts the iteration picking new random $\langle p_i, q_i \rangle$. Note that we are assuming algorithm $\mathcal{B}'$ knows p and q. Clearly if $\mathcal{B}'$ succeeds in causing algorithm $\mathcal{A}$ to factor N then so will $\mathcal{B}$ (the only difference between $\mathcal{B}$ and $\mathcal{B}'$ is that algorithm $\mathcal{B}'$ applies a filter prior to running $\mathcal{A}$ while $\mathcal{B}$ always runs $\mathcal{A}$). We show that for $N \in S'$ algorithm $\mathcal{B}'$ will cause $\mathcal{A}$ to succeed in expected polynomial time.

Given $N = pq \in S'$ define a distribution $\mathcal{P}_N$ on tuples of n-bit integers $\langle p_1, \ldots, p_{k-1} \rangle$ as follows: the distribution $\mathcal{P}_N$ is obtained by uniformly picking a k tuple $\langle p_1, \ldots, p_k \rangle$ among all the k-tuples that sum up to p and outputting $\langle p_1, \ldots, p_{k-1} \rangle$. Define $\mathcal{Q}_N$ similarly. For $N \in S'$ we know that

$$\Pr_{\mathcal{P}_N, \mathcal{Q}_N}[\mathcal{A} \text{ factors } N] \geq 1/4n^d$$

Next, observe that the distribution $(\mathcal{P}_N, \mathcal{Q}_N)$ is identical to the distribution that algorithm $\mathcal{B}'$ feeds $\mathcal{A}$. Hence, algorithm $\mathcal{B}'$ succeeds in factoring an $N \in S'$ in polynomial time if the filtering step does not take too long. It remains to show that for $N \in S'$ the filtering step does not take too long. Let p be an integer in the range $[k \cdot 2^{n-1}, k \cdot 2^n]$ (this is the range of all possible values of p produced by the protocol). The question is, what is the probability that a random tuple $\langle p_1, \ldots, p_{k-1} \rangle$ of n-bit integers satisfies that $\sum_{i=1}^{k-1} p_i$ is in the range $[p - 2^n, p - 2^{n-1}]$. The definition of S_p implies that for all $p \in S_p$ this probability is at least $1/k^2n^d$. Hence, since the filtering step applies to both p and q, the expected number of tries until the filtering step is satisfied is $(k^2n^d)^2$. Consequently, algorithm $\mathcal{B}$ runs in polynomial time (in k and n). $\square$