Protocols for Secure Generation of Shared RSA Keys

DARPA Contract F30602-97-C-0326

Data Item Number A007: Technical Information Report.

Bill Aiello
aiello@bellcore.com

Dan Boneh
dabo@cs.stanford.edu

Matthew Franklin
franklin@research.att.com

Jeremy Horwitz
horwitz@cs.stanford.edu

1 Executive Summary

The attached report contains a comprehensive survey of our results on distributed generation of shared RSA keys. These results are a central component of our Intrusion Tolerance project.

A private RSA key is often stored (encrypted) at a single location such as a user's machine. The machine then becomes a prime target for a hacker who wishes to obtain the private key. To protect a highly sensitive private key (such as the one belonging to a root Certification Authority) one may wish to split the key into a number of pieces and store each piece at a separate location. Fortunately, for the RSA cryptosystem it is possible to share the private key among several sites without having to reconstruct the key in order to use it — a root CA can issue certificates without reconstructing its private key at a single location. To explain how this is done we briefly recall the details of the RSA system. Let $N = pq$ be a product of two large prime. Let e be a public certificate verification exponent. Let d be a private certificate issuing exponent. We have that $d \cdot e = 1 \bmod \phi(N)$ where $\phi(N) = N - p - q + 1$. A public RSA key is made up of $\langle e, N \rangle$. A private key is made up of $\langle d, N \rangle$ (the private key often includes other information such as p and q for efficient signature generation). To share its private key among four sites, the root CA does the following: it picks four random integers $d_1, \ldots, d_4$ in the range $1, \ldots, N$ such that $d_1 + d_2 + d_3 + d_4 = d$. Each of these shares is then stored (encrypted) in a separate location. To issue a certificate, the CA can create a signature on a message m by having each site compute $s_i = m^{d_i} \bmod N$. Each site then sends s_i to the entity requesting the certificate. The requesting entity can multiply the four s_i's to obtain $s = m^{d_1+d_2+d_3+d_4} = m^d \bmod N$. Thus, a valid signature is obtained. The beauty of this simple procedure is that the private key d is never reconstructed at a single location. If a hacker penetrates even three of the four sites, he obtains no information about the private key. To achieve fault-tolerance the above scheme may be slightly modified so that any 3-out-of-4 sites can issue a certificate.

An important issue left out of the above discussion is key generation. Who generates the modulus N and the private key d? A simple solution is to ask a *trusted dealer* to generate two primes p and q, multiply them to get N and then generate e, d. Finally, the dealer splits d into four pieces and sends each piece to each of the sites. Unfortunately, a trusted dealer introduces a *single point of attack*: the dealer, or anyone who compromises the dealer, knows the private key and can issue false certificates. Our research on distributed generation of shared RSA keys eliminates the need for a trusted dealer. The four sites can jointly generate a shared key such that neither one of them known the private key d or the factorization of N. Our algorithms are useful also in other contexts, such as the Fiat-Shamir authentication scheme where a number of entities use a modulus $N = pq$ for authentication, but none of them should know the factorization of N. The original Fiat-Shamir scheme called for a trusted dealer to generate N. Our research eliminates the need for a trusted dealer.

We briefly outline our approach. Our goal is to have k machines generate a modulus $N = pq$ and exponents e and d. At the end of the computation all parties should be convinced that N is the product of two primes, however none of them should know the factorization. Furthermore, e should be public while d should be shared among the machines in a way that enables t-out-of-k threshold signature generation. That is, any t machines should be able to issue a certificate. At the same time, an attacker who penetrates at most $t-1$ machines should not be able to obtain any information about the private key. Our key generation algorithm proceeds in a number of steps. At a high level these steps are as follows:

(1) pick candidates: Each of the k machines picks two random n-bit integers p_i, q_i and keeps them secret.

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

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

Other than the value of N, this step reveals no information about the secret values $p_1, \ldots, p_k$ and $q_1, \ldots, q_k$ (!) Now that N is public, the k machines apply trial division to ensure that N is not divisible by small primes.

(3) primality test: The k machines 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. As before, this step reveals no information about the private shares.

(4) key generation: Given a public encryption exponent e, the machines use a private distributed computation to generate a *shared* secret decryption exponent d.

Part I of the attached report gives the full details of each of these steps. A number of practical optimizations that we used in the implementation of the algorithm are presented. A description of our implementation including timing measurements is given in [34]. Part II describes an extension of our algorithm enabling k machines to generate a modulus N which is the product of *three* large primes $N = pqr$. Generating an RSA modulus with three primes has some advantages over standard RSA and is occasionally used. The two parts are expanded versions of our papers that recently appeared in technical conferences.

Part I
Algorithms for Generating Shared RSA keys

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.

2 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 [21] where all parties use the same modulus N, but none of them know its factorization. For other examples see [20, 25, 31, 35, 36]. 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. $de = 1 \bmod \phi(N)$. 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 [41], 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 [22]: pick random d_i's satisfying $d = \sum_{i=1}^{d} d_i \bmod \phi(N)$ and give d_i to party i. Abstractly, to sign a message m each party computes $s_i = m^{d_i} \bmod N$ and sends s_i to a combiner (that 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 [17, 14, 23, 40].

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 [37], but not for RSA.

The paper is organized as follows: we give a high level description of our protocol in Section 3. 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 shares of the corresponding private key $d = e^{-1} \bmod \varphi(N)$. To achieve fault tolerance one often shares d using a t-out-of-k scheme. That is, any subset of t users can apply the private key d. We explain how a t-out-of-k sharing of d is obtained in Section 7.3. We discuss various practical improvements of our protocol in Section 8.

We note that generic secure circuit evaluation techniques, e.g. [45, 28, 3, 10] 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.

2.1 Communication and privacy model

The communication and privacy model assumed by our protocol are as follows:

Full connectivity Any party can communicate with any other party. This is a typical setup on a local network or the Internet.

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 model is often called *honest but curious parties*. In Section 9 we discuss some results showing how this assumption can be relaxed. 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 $\lfloor \frac{k}{2} \rfloor$ private. That is, any coalition of size at most $\lfloor \frac{k}{2} \rfloor$ learns no information about the factorization of $N = pq$. However, a coalition of more than $\lfloor \frac{k}{2} \rfloor$ can recover the factorization. The reason for this bound is our reliance on the BGW [3] protocol in one step of our algorithm. The BGW protocol provides information theoretic security and hence limited to achieving $\lfloor \frac{k}{2} \rfloor$ privacy (in the honest parties model). We note that it is possible to achieve $k-1$ privacy by replacing the BGW step with a heuristic protocol due to Cocks [11]. Unfortunately, Cocks' protocol is by far slower than the BGW method. Furthermore, 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.

3 Overview

In this section we give a high level overview of the 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. All parties should be convinced that N is indeed 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.

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) primality test: The k parties engage in a private distributed computation to test that N is indeed the product of two primes. If the test fails, then the protocol is restarted from step 1. We note that the primality 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 and tests that $N = pq$ is a product of two primes. By the prime number theorem the probability that both p and q are prime is asymptotically $1/n^2$. Therefore, naively one has to perform n^2 probes on average until a suitable N

is found. This is somewhat worse than the expected $2n$ probes needed in traditional generation of an RSA modulus (one first generates one prime using n probes and then a second prime using another n probes). This $n/2$ 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 picked a random n bit integer p_i as its secret share. The prime p was 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 can not 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 3.1 *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 $\{p_i\}$ 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 $1/n^{d+2}$ of the integers in $\mathbb{Z}_n^{(2)}$.*

Proof Sketch Given $N \in \mathbb{Z}_n^{(2)}$ algorithm $\mathcal{B}$ works by repeating the following two steps until N is factored: (1) pick random independent n bit integers $p_1, \ldots, p_{k-1}$. (2) Run algorithm $\mathcal{A}$ on the input N and $p_1, \ldots, p_{k-1}$. For $1/n^{d+2}$ of the integers in $\mathbb{Z}_n^{(2)}$, after a polynomial number of iterations algorithm $\mathcal{B}$ will output a factorization of N. $\square$

4 Distributed primality test

We begin the detailed discussion of the protocol with the distributed primality test (step 3). 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 primality test. Our primality test is a probabilistic test [42, 39] 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 twisted group $\mathbb{T}_N = (\mathbb{Z}_N[x]/(x^2+1))^*/\mathbb{Z}_N^*$. Suppose all prime factors of N are 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 $a \in \mathbb{Z}_N^*$. We note that elements of $\mathbb{T}_N$ can also be viewed as points on the projective line over $\mathbb{Z}_N$.

Distributed primality 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 exchange the v_i values with each other and verify that

$$v_1 = \pm \prod_{i=2}^{k} v_i \pmod N$$

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 twisted 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 exchange the u_i values with each other and verify that

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

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 $v_1/\prod v_i = \pm 1 \bmod N$ 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,

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

Since $\left(\frac{g}{p}\right) = \left(\frac{g}{q}\right)$ it follows that $g^{\phi(N)/4} \equiv \pm 1 \bmod N$. Since $\phi(N) = N - p - q + 1$ 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 at least one of p, q is not prime. That is, $N = r_1^{d_1} \cdots r_s^{d_s}$ is a non-trivial factorization of N with $\sum d_i \geq 3$ and $s \geq 1$. 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 $\mathbb{Z}_N^*$:

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

To prove the lemma we show that $|H| \leq \frac{1}{2}|G|$. 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 pigeon hole). 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 $\phi(N)$. 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 both 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 primality 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 fail step (4). 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 primality test protocol reveals no other information about the private shares of the participants.

Lemma 4.2 *Suppose p, q are prime. Then any coalition of $k-1$ parties can simulate their view of the primality testing protocol. Consequently, the protocol is $k-1$ private.*

Proof Since p, q are prime we know that $v_1 = \pm \prod_{i=2}^k v_i \bmod N$ where v_a, v_b are defined as in step (3) of the protocol. Also, $\prod u_i = 1$ where the u_i's are defined in step (4). Let U be a coalition of $k-1$ parties. Say party m is not a member of the coalition. The coalition's view consists of $\langle u_i, v_i, N, g, h\rangle$ for $i = 1, \ldots, k$ and $\langle p_i, q_i\rangle$ for all $i \neq m$.

We construct a simulator of the coalition's view as follows: the simulator is given N, p_i, q_i for $i \neq m$ as input (this is the coalition's input when the protocol is started). To simulate the coalition's view, the simulator first picks a random $g \in \mathbb{Z}_N^*$ with $\left(\frac{g}{N}\right) = 1$ and a random $h \in \mathbb{T}_N$. It then computes the u_i, v_i values for all $i \neq m$ using the p_i, q_i values it is given as input. Next the simulator must generate u_m, v_m. Generating u_m is trivial since $u_m = (\prod_{i \neq m} u_i)^{-1}$. Generating v_m is a bit harder. From the proof of Lemma 4.1 it follows that

$$\begin{cases} v_1 = \prod_{i=2}^k v_i \pmod N & \text{implies} \quad \left(\frac{g}{p}\right) = \left(\frac{g}{q}\right) = 1 \\ v_1 = -\prod_{i=2}^k v_i \pmod N & \text{implies} \quad \left(\frac{g}{p}\right) = \left(\frac{g}{q}\right) = -1 \end{cases}$$

That is, the sign of $v_1 / \prod_{i=2}^k v_i$ leaks whether g is a quadratic residue modulo N. To simulate v_m the simulator computes $\hat{v}_m = v_1 / \prod_{i \neq m} v_i$. It then flips an unbiased coin and sets $v_m = \hat{v}_m$ or $v_m = -\hat{v}_m$ accordingly. The resulting distribution on v_m is computationally indistinguishable from the true distribution, assuming the hardness of quadratic residuosity modulo a Blum integer. To conclude, given that N is the product of two distinct primes, the set of values $\langle u_i, v_i, N, g, h\rangle$ produced by the simulator is computationally 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 primality 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's advantage is that it requires less computation. It's drawback is more communication between the parties.

Observe that N that fall into Case 4 satisfy $\gcd(N, p+q-1) > 1$. 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 that are a product of two distinct primes pass the test with overwhelming probability. We note that this alternate test requires far more communication than the proposed Step (4) in the previous section. In addition, it 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. 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 [3] 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 $P > N$ be some prime. Unless otherwise stated, all arithmetic operations are done modulo P. The protocol works as follows:

Step 1: Let $l = \lfloor\frac{k-1}{2}\rfloor$. For all $i = 1, \dots, k$ 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 $i = 1, \dots, k$ party i computes the $3k$ values:

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

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 $j = 1, \dots, k$ 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 $j = 1, \dots, k$. 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 $i = 1, \dots, k$. Let $\alpha(x)$ be the polynomial

$$\alpha(x) = \left(\sum_j f_j(x) \right) \left(\sum_j g_j(x) \right) + \sum_j h_j(x) \pmod P$$

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 $k \geq 2l+1$. Consequently, since all parties have at least $2l+1$ 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 [3] for the complete details.

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

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 $h_i(0) = 0$. 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 $N_i = (\sum_j p_{i,j})(\sum_j q_{i,j}) + \sum_j h_{i,j}$. 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 truncation 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). We note that communication between the parties can be reduced by a factor of two using a variant of the protocol as described in [7, Section 4].

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 require 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 it's 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 $\lambda_i(x) = \prod_{j \neq i} \frac{(x-j)}{(i-j)}$ is the appropriate Lagrange coefficient. Therefore, rather than broadcast N_i at the end of Step 3, party i simply sets $M_i = \lambda(0)N_i$. 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 3). Let $q = q_1 + \ldots + q_k$ be an integer shared among k parties. Let p be a small prime. To test if p 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 $qr = (\sum q_i)(\sum r_i) \bmod p$. If $qr \neq 0$ then p 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 p does not divide q it is still possible that $qr = 0 \bmod p$. To alleviate this problem one can repeat the above test twice for each small prime p. Then the probability that a good candidate is rejected is at most $1 - \prod_{p<B}(1 - \frac{1}{p^2}) < \frac{1}{2}$.

One caveat in the above approach is that the BGW protocol as described in Section 5.1 cannot be applied to test divisability of q by very small p, namely $p \leq k$. The reason is that for such small p the field $\mathbb{F}_p$ does not contain enough points to do Shamir secret sharing among k parties. For such small p one must run the BGW protocol in an extension field of $\mathbb{F}_p$ 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 $d = e^{-1} \bmod \phi(N)$ for a given encryption exponent e. We have two approaches for computing 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 $\phi = \phi(N)$. Recall that the public modulus $N = (\sum p_i)(\sum q_i)$ satisfies $\phi(N) = \phi = \sum_{i=1}^{k} \phi_i$ where $\phi_1 = N - p_1 - q_1 + 1$ and $\phi_i = -p_i - q_i$ for $i = 2, \ldots, k$. For all $i = 1, \ldots, k$ party i can locally compute ϕ_i.

To compute shares of d the parties must invert e modulo $\sum \phi_i$ without exposing their ϕ_i's. Unfortunately, traditional inversion algorithms, e.g. extended gcd, involve computations modulo $\sum \phi_i$. 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 $e^{-1} \bmod \phi$ with no modulo ϕ reductions. When only a single user is involved the inversion is done in three steps: (1) Compute $\zeta = \phi^{-1} \bmod e$. (2) Set $T = -\zeta \cdot \phi + 1$. Observe that $T \equiv 0 \bmod e$. (3) Set $d = T/e$. Indeed $d = e^{-1} \bmod \phi$ since $d \cdot e \equiv 1 \bmod \phi$. There is no need for modulo ϕ reductions since $e^{-1} \bmod \phi$ can be immediately deduced from $\phi^{-1} \bmod e$. 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 method leaks the value of $\phi(N) \bmod e$ (hence it can only be applied when e is small). On the plus side, it is $k-1$ private.

Step 1: The parties jointly determine the value of $l = \phi \bmod e$. Since $l = \sum \phi_i \bmod e$ it is possible to compute l without revealing any other information about the private shares. To do so we use a protocol due to Benaloh [4] which is $k-1$ private.

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

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

As a result we have $d = \sum d_i + r \bmod \phi(N)$ where $0 \leq r < k$.

Step 3: The above sharing of d enables shared decryption [22] 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 $0 \leq r \leq k$ during a trial decryption.

The above approach leaks $\phi(N) \bmod e$ and r. This is a total of $\log e + \log k$ bits.

7.2 Arbitrary public exponent

Unlike the previous technique, our second method for generating shares of d works for arbitrary public exponent e. It leaks less than $2 \log k$ bits. This information cannot help an opponent since it can be easily guessed. Rather than exposing $\phi \bmod e$ and then inverting it (as we did above) we show how to invert $\phi \bmod e$ while it is shared among the parties. As a result, no information about ϕ is revealed. The protocol is $\frac{k-1}{2}$ private.

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

Step 2. Using the protocol of Section 5.2 they compute $\psi = (\sum r_i) \cdot (\sum \phi_i) \bmod e$. 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 $\sum \zeta_i = (\sum r_i)\psi^{-1} = \phi^{-1} \bmod e$. Hence, the parties are able to share the inverse of $\phi \bmod e$ without revealing any information.

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

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

Each party has a T_i 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

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

We now 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:

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

Therefore, $d = (\sum T_i)/e$. Each party i now sets $d_i = \lfloor T_i/e \rfloor$. As a result we have $d = \sum d_i + r \bmod \phi(N)$ 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. The value of r is found using only one trial decryption: a gateway 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 the gateway. By performing at most k multiplications the gateway finds an $r \in [0, k)$ satisfying $m = (\prod m_i)c^r \bmod N$. It then relates the value of r back to the parties who fix their shares of d accordingly. 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).

Note 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 [23, 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 $g_0 = g(0) = L\phi_i$. 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 $(\sum \zeta_i)(\sum \phi_i)$ 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.

The computation of $\phi^{-1} \bmod e$ (steps 1-3 above) is based on a technique due to Beaver [2].

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 provide fault-tolerance 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 [40], 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 to improve the performance of our distributed protocol. These optimizations are incorporated in an implementation of our protocol [34].

Sieving In Step (1) of our protocol (Section 3) 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 of 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 explained in Section 5.2.

Load balancing In Step (3) of the primality 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 over all 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 primality test (Step 2 in Section 3). 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 large bound B_2, increasing the effectiveness of trial division.

Fermat filter There is no need to run the full primality 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 primes. To carry out this test in zero-knowledge all k parties compute $v_i = g^{p_i+q_i} \bmod N$. They exchange v_i's and verify that $\prod v_i = g^{N+1} \bmod N$. 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 3 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 p_a, r_a and p_a is an n-bit prime. Bob has q_b, r_b and q_b is an n-bit prime. Carol has r_c. 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. Also note that in the example above, the parties must perform a distributed primality 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. We note that one can use recent results of [8] to do just that: test that a shared modulus is the product of three primes without revealing its factorization.

9 Robustness

Throughout the paper we use a model in which parties honestly follow the protocol. This is fine when parties are honestly trying to generate a shared RSA key. For some applications it is desirable to make the protocol robust against active adversaries that cheat during the protocol. Since the RSA function is verifiable (the parties can simply check that they correctly decrypt encrypted messages) active adversaries are limited in the amount of damage they can cause. However, it may still be possible that a party cheat during the protocol and consequently be able to factor the resulting N. Similarly, a party can cheat and cause a non-RSA modulus to be incorrectly accepted.

Recently, Frankel, MacKenzie and Yung [24] showed how our protocol can be made to withstand $\lfloor \frac{k-1}{2} \rfloor$ malicious parties. The protocol enables the parties to detect and exclude the malicious party. In practice, one could run our non-robust protocol until a modulus N is found which is believed to be a product of two primes. Then, the robust Frankel-MacKenzie-Yung protocol can be used to determine that no majority of parties cheated during the non-robust phase. For more results on robust generation of shared RSA keys see [5].

We describe a simple method for making our non-robust protocol robust when the number of participants is small (e.g. less than ten). Consider the case of four parties where at most one of them is malicious. One can run our non-robust protocol until a candidate modulus N is found. At this point the protocol is run four more times, once for each triplet of users. In the first run, party 1 shares her values p_1, q_1 with the other three parties by writing $p_1 = p'_2 + p'_3 + p'_4$ and $q_1 = q'_2 + q'_3 + q'_4$ where p'_j, q'_j are random integers in the range $[0, N]$. Party 1 then sends p'_i, q'_i to party i for $i = 2, 3, 4$. Party i adds these values to its own p_i, q_i. Next, parties $2, 3, 4$ run our non-robust protocol among the three of them (ignoring party 1). If the resulting N does not match the N computed when all four parties were involved, or if N turns out to not be an RSA modulus, the N is rejected and the parties announce that one of them is misbehaving. This experiment is repeated with all four triplets – each time exactly one party is excluded from the computation. Assuming at most one party is malicious, the resulting N must be a product of two large primes. Furthermore, the malicious party cannot know the factorization of N since at no point in the protocol does an honest party reveal any information about it's share to another single party. We note that this approach enables the parties to detect cheating, but it does not help in detecting who the malicious party is.

In general, when k parties are engaged in our non-robust protocol, and c of them are malicious, the protocol can be made robust at the cost of $\binom{k}{c}$ extra invocations. The resulting computation is $\lfloor \frac{k-c-1}{2} \rfloor$ private. Clearly this approach can only be applied as long as both k and c are very small.

10 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 methods 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 10 minutes. See [34] for a description of the implementation as well as detailed timing measurements.

An important open problem is the generation of shared keys of special form. For example, a modulus which is a product of "safe primes" (i.e., where both $\frac{p-1}{2}$ and $\frac{q-1}{2}$ are prime) has been considered for security purposes [29] as well as for technical reasons related to threshold cryptography [17, 27]. Currently, our techniques do not enable shared generation of moduli of special form. Progress in this directions would be very helpful.

A. Cocks' multiplication method

The BGW protocol described in Section 5.1 achieves $\lfloor \frac{k-1}{2} \rfloor$ privacy and is information theoretically secure. Cocks [11] describes a multiplication protocol that heuristically appears to provide $k-1$ privacy. Unfortunately, Cocks' protocol is far slower than the BGW method. Furthermore, it does not seem possible to prove its security using "natural" complexity assumptions. For completeness we describe a generalization of Cocks' protocol. As in the settings of Section 5.1 each of the k parties has a private n-bit p_i, q_i. The parties wish to compute $N = (\sum p_i)(\sum q_i)$ without revealing any other information about their private shares.

Let E, D be a weakly-multiplicative homomorphic public key encryption scheme. $E_p[m]$ encrypts a message m using the public key p, while $D_s(c)$ decrypts a ciphertext c using the secret key s. The homomorphic property implies that given a ciphertext $c_1 = E_p[m_1]$ and a plaintext m_2 there is an efficient algorithm to compute $c = E_p[m_1 \cdot m_2]$. Here we are assuming the plaintexts m_1, m_2 lie in a ring R where $R = \mathbb{Z}/(M\mathbb{Z})$ for some integer M. Examples of multiplicatively homomorphic public key encryption schemes are RSA and ElGamal encryption [18]. In what follows we are assuming each of the participants has its own public/private key pair. We denote by E_i, D_i the pair belonging to party i. All parties can encrypt using E_i while only party i can decrypt using D_i. Also, we assume the plaintext ring R is larger than $2N$.

Step 1: Each party i picks $k-1$ random integers $\langle a_{i,1}, \ldots, a_{i,i-1}, a_{i,i+1}, \ldots, a_k \rangle$ in $[0, |R|/2]$. It sets $b_i = \sum_{j \neq i} a_{i,j}$.

Step 2: Each party i sends $\langle E_i[p_i], E_i[q_i] \rangle$ to all other parties.

Step 3: For all $j \neq i$ party i does the following:

a. Using the homomorphic property it computes the three values

$$\alpha_{i,j} = E_j[p_i q_j]\ , \quad \beta_{i,j} = E_j[q_i p_j]\ , \quad \gamma_{i,j} = E_j[a_{i,j}]$$

b. Each of $\alpha_{i,j}, \beta_{i,j}, \gamma i, j$ is then broken into l shares, where l is a security parameter. The shares of $\alpha_{i,j}$ are constructed by picking random $u_1, \ldots, u_l \in R$ such that $\sum u_i = 1$. Then $\alpha_{i,j,1}, \ldots, \alpha_{i,j,l}$ are defined as: $\alpha_{i,j,m} = E_j[u_m p_i q_j]$. The $\beta_{i,j}$ and $\gamma_{i,j}$ are broken into l pieces in a similar way.

c. Party i sends the $3l$ values $\langle \alpha_{i,j,m}, \beta_{i,j,m}, \gamma_{i,j,m} \rangle$ to party j in a *random order*. The value l should be chosen large enough so that the random order cannot be undone. This is the main reason this protocol is so much slower than the BGW method.

Step 4: For all $j \neq i$ party i receives $3l$ ciphertexts from party j. It decrypts all of them and computes their sum. A moments reflection shows that the sum comes out to be

$$S_{i,j} = p_i q_j + q_j p_i + a_{j,i}$$

Next, party i computes:

$$N_i = 2p_i q_i - b_i + \sum_{j \neq i} S_{i,j}$$

Again, a moments reflection shows that

$$\sum_{i=1}^{k} N_i = 2 \sum_{i,j} p_i q_j = 2 \left(\sum p_i \right) \left(\sum q_i \right) = 2N$$

Hence, an additive sharing of $2N$ is obtained. To complete the protocol, the parties may exchange the values N_i with each other and make the final value of N public.

When generating a 1024 bit modulus, Cocks states that one should use l on the order of 400. This implies that each party must perform $1200 \cdot k$ public key decryptions at Step 4 (!) This is significantly slower than the BGW method (over 1000 times slower) where no encryption is used.

Although the protocol may provide $k-1$ privacy, we cannot give a simulation proof. Note that a coalition of $k-1$ parties learns both the value of N and the encryptions $E_i[p_i], E_i[q_i]$ of the party not in the coalition. Providing a simulation for such a transcript seems unlikely under "natural" assumptions. Indeed, Coppersmith [12] shows that if one uses RSA for E_i (i.e. $E_i(m) = m^{e_i} \bmod M_i$) then there is an attack that factors N in time polynomial in e. Consequently, one cannot use low public exponent RSA as E_i. Similarly, it is not clear how to simulate the random permutation used in Step 3.

Throughout the paper, Cocks' method can be used instead of BGW to compute the function $N = (\sum p_i)(\sum q_i)$. This substitution will make all our protocols heuristically $k-1$ private. However, we stick with the BGW protocol due to its superior performence and rigorous proof of security. Improving the performence of Cocks' method and providing a *rigorous* protocol achieving $k-1$ privacy are excellent open problems.

Part II

Generating a Product of Three Primes With an Unknown Factorization

Abstract

We describe protocols for three or more parties to jointly generate a composite $N = pqr$ which is the product of three primes. After our protocols terminate N is publicly known, but neither party knows the factorization of N. Our protocols require the design of a new type of distributed primality test for testing that a given number is a product of three primes.

11 Introduction

In this paper, we describe how three (or more) parties can jointly generate an integer N which is the product of three prime numbers ($N = pqr$). At the end of our protocol the product N is publicly known, but neither party knows the factorization of N. Our main contribution is a new type of probabilistic primality test that enables the three parties to jointly test that an integer N is the product of three primes without revealing the factorization of N. Our primality test simultaneously uses two groups: the group $\mathbb{Z}_N^*$ and the twisted multiplicative group $\mathbb{T}_N = (\mathbb{Z}_N[x]/(x^2+1))^*/\mathbb{Z}_N^*$.

The main motivation for this problem comes from cryptography, specifically the sharing of an RSA key. Consider classical RSA: $N = pq$ is a public modulus, e is a public exponent, and d is secret where $de = 1 \bmod \varphi(N)$. At a high level, a digital signature of a message M is obtained by computing $M^d \bmod N$. In some cases, the secret key d is highly sensitive (e.g. the secret key of a Certification Authority) and it is desirable to avoid storing it at a single location. Splitting the key d into a number of pieces and storing each piece at a different location avoids this *single point of failure*. One approach (due to Frankel [22]) is to pick three random numbers satisfying $d = d_1 + d_2 + d_3 \bmod \varphi(N)$ and store each of the shares d_1, d_2, d_3 at one of three different sites. To generate a signature of a message M, site i computes $S_i = M^{d_i} \bmod N$ for $i = 1, 2, 3$ and sends the result to a *combiner*. The combiner multiplies the S_i and obtains the signature $S = S_1S_2S_3 = M^d \bmod N$. If one or two of the sites are broken into, no information about the private key is revealed. An important property of this scheme is that it produces standard RSA signatures; the user receiving the signature is totally unaware of the extra precautions taken in protecting the private key. Note that during signature generation the secret key is never reconstructed at a single location.

To provide fault tolerance, one slightly modifies the above technique to enable any two of the three sites to generate a signature. This way if one of the sites is temporarily unavailable the Certification Authority can still generate signatures using the remaining two sites. If the key was only distributed among two sites, the system would be highly vulnerable to faults.

We point out that classic techniques of secret sharing [41] are inadequate in this scenario. Secret sharing requires one to reconstruct the secret at a single location before it can be used, hence introducing a single point of failure. The technique described above of sharing the secret key such that it can be used without reconstruction at a single location is known as *Threshold Cryptography*. See [26] for a succinct survey of these ideas and nontrivial problems associated with them.

An important question left out of the above discussion is key generation. Who generates the RSA modulus N and the shares d_1, d_2, d_3? Previously the answer was a *trusted dealer* would generate N and distribute the shares d_1, d_2, d_3 to the three sites. Clearly this solution is undesirable since it introduces a new single point of failure — the trusted dealer. It knows the factorization of N and the secret key d; if it is compromised, the secret key is revealed. Recently, Boneh and Franklin [7] designed a protocol that enables three (or more) parties to jointly generate an RSA modulus $N = pq$ and shares d_1, d_2, d_3 of a private key. At the end of the protocol the parties are assured that N is indeed the product of two large primes; however, none of them know its factorization. In addition, each party learns exactly one of d_1, d_2, d_3 and has no computational information about the other shares. Thus, there is no need for a trusted dealer. We note that Cocks [11] introduced a heuristic protocol enabling two parties to generate a shared RSA key.

In this paper we design an efficient protocol enabling three (or more) parties to generate a modulus $N = pqr$ such that neither party knows the factorization of N. Once N is generated the same techniques used in [7] can be used to generate shares d_1, d_2, d_3 of a private exponent. For this reason, throughout the paper we focus on the generation of the modulus $N = pqr$ and forgo discussion of the generation of the private key. The methods of [7] do not generalize to generate a modulus with three prime factors and new techniques had to be developed for this purpose.

We remark that techniques of *secure circuit evaluation* [3, 10, 45] can also be used to solve this problem; however, these protocols are mostly theoretical and result in extremely inefficient algorithms.

12 Motivation

The problem discussed in the paper is a natural one and thus our solution is of independent interest. Nonetheless, the problem is well motivated by a method for improving the efficiency of shared generation of RSA keys. To understand this we must briefly recall the method used by Boneh and Franklin [7]. We refer to the three parties involved as Alice, Bob, and Carol. At a high level, to generate a modulus $N = pq$, the protocol works as follows:

Step 1. Alice picks two random n bit integers p_a, q_a, Bob picks two random n bit integers p_b, q_b and Carol picks two random n bit integers p_c, q_c. They keep these values secret.

Step 2. Using a private distributed computation they compute the value

$$N = (p_a + p_b + p_c)(q_a + q_b + q_c).$$

At the end of the computation, N is publicly available; however no other information about the private shares is revealed. This last statement is provable in an information-theoretic sense.

Step 3. The three parties perform a distributed primality test to test that N is the product of exactly two primes. As before, this step provably reveals no information about the private shares.

Step (3), the distributed primality test, is a new type of probabilistic primality test which is one of the main contributions of [7]. Step (2) is achieved using an efficient variation of the BGW [3] protocol.

A drawback of the above approach is that both factors of N are simultaneously tested for primality. Hence, the expected number of times step (3) is executed is $O(n^2)$. This is much worse than single-user generation of N where the two primes are first generated separately by testing $O(n)$ candidates and then multiplied together. When generating a 1024-bit modulus, this results in significant slowdown when compared with single-user generation.

To combat this quadratic slowdown, one may try the following alternate approach.

Step 1. Alice picks a random n-bit prime p and a random n-bit integer r_a. Bob picks a random n-bit prime q and a random n-bit integer r_b. Carol picks a random n-bit integer r_c. They keep these values secret.

Step 2. Using a private distributed computation they compute the value

$$N = pq(r_a + r_b + r_c)$$

At the end of the computation, N is publicly available; however no other information about the private shares is revealed.

Step 3. The three parties use the results of this paper to test that N is the product of exactly three primes. This step provably reveals no information about the private shares.

At the end of the protocol neither party knows the full factorization of N. In addition, this approach does not suffer from the quadratic slowdown observed in the previous method. Consequently, it is faster by roughly a factor of 50 (after taking effects of trial division into account). As before, step (2) is carried out by an efficient variant of the BGW protocol.

Instead of solving the specific problem of testing that $N = pq(r_a + r_b + r_c)$ is a product of three primes we solve the more general problem of testing that

$$N = (p_a + p_b + p_c)(q_a + q_b + q_c)(r_a + r_b + r_c)$$

is a product of three primes without revealing any information about the private shares. This primality test is the main topic of this paper.

For the sake of completeness we point out that in standard single-party cryptography there are several advantages to using an RSA modulus $N = pqr$ rather than the usual $N = pq$ (the size of the modulus is the same in both cases). First, signature generation is much faster using the Chinese Remainder Theorem (CRT). When computing $M^d \bmod N$ one only computes $M^{d \bmod p-1} \bmod p$ for all three factors. Since the numbers (and exponents) are smaller, signature generation is about twice as fast as using CRT with $N = pq$. Another advantage is that an attack on RSA due to Wiener [44] becomes less effective when using $N = pqr$. Wiener showed that for $N = pq$ if $d < N^{1/4}$ one can recover the secret key d from the public key. When $N = pqr$ the attack is reduced to $d < N^{1/6}$ and hence it may be possible to use smaller values of d as the secret key. Finally, we note that the fastest factoring methods [32] cannot take advantage of the fact that the factors of $N = pqr$ are smaller than those of a standard RSA modulus $N = pq$.

13 Preliminaries

In this section, we explain the initial setup for our new probabilistic primality test and how it is obtained. We then explain a basic protocol which we use in the later parts of the paper. At first reading, the reader may wish to skip to Sect. 14 and take on faith that the necessary setup is attainable.

13.1 Communication and Privacy Model

The communication and privacy model assumed by our protocol are as follows:

Full connectivity Any party can communicate with any other party. This is a typical setup on a local network or the Internet.

Private and authenticated channels Messages sent from party A to party B are private and cannot be tampered with en route. This simply states that A and B share a secret key which they can use for encryption and authentications.

Honest parties We assume all parties are honest in their following of the protocol. This is indeed the case when they are truly trying to create a shared key. This assumption is used by both [7] and [11]. We note that some recent work [24] makes the protocol of [7] robust against cheating adversaries at the cost of some slowdown in performance (roughly a factor of 100). These robustness results apply to the protocols described in this paper as well.

Collusion Our protocol is 1-private. That is to say that a single party learns no information about the factorization of $N = pqr$. However, if two of the three parties collude they can recover the factors. For three parties this is fine since our goal is to enable two-out-of-three signature generation. Hence, two parties are always jointly able to recover the secret key. More generally, when k parties participate in our primality test protocol, one can achieve $\lfloor \frac{k-1}{2} \rfloor$-privacy. That is, any minority of parties learns no information about the factors of N.

13.2 Generation of N

In the previous section, we explained that Alice, Bob, and Carol generate N as

$$N = (p_a + p_b + p_c)(q_a + q_b + q_c)(r_a + r_b + r_c),$$

where party i knows p_i, q_i, r_i for $i = a, b, c$ and keeps these shares secret while making N publicly available. To compute N without revealing any other information about the private shares, we use the BGW protocol [3]. For the particular function above, the protocol is quite efficient; it requires three rounds of communication and a total of six messages. The protocol is information-theoretically secure, i.e. other than the value of N, party i has no information about the shares held by other parties. This is to say the protocol is 1-private.

We do not go into the details of how the BGW protocol is used to compute N since it is tangential to the topic of this paper. For our purpose, it suffices to assume that N is public while the private shares are kept secret.

An important point is that our primality test can only be applied when $p_a+p_b+p_c = q_a+q_b+q_c = r_a+r_b+r_c = 3 \bmod 4$. Hence, the parties must co-ordinate the two lower bits of their shares ahead of time so that the sums are indeed 3 modulo 4. Indeed, this means that a priori each party knows the two least significant bits of the other's shares.

13.3 Sharing of $(p-1)(q-1)(r-1)$ and $(p+1)(q+1)(r+1)$

Let $p = p_a + p_b + p_c$, $q = q_a + q_b + q_c$, and $r = r_a + r_b + r_c$. We define $\hat{\varphi} = (p-1)(q-1)(r-1)$. Since p, q, and r are not necessarily prime, $\hat{\varphi}$ may not equal $\varphi(N)$. Our protocol requires that the value $\hat{\varphi}$ be shared additively among the three parties. That is, $\hat{\varphi} = \varphi_a + \varphi_b + \varphi_c$, where only party i knows φ_i for $i = a, b, c$.

An additive sharing of $\hat{\varphi}$ is achieved by observing that $\hat{\varphi} = N - pq - pr - qr + p + q + r - 1$. To share $\hat{\varphi}$, it suffices to represent $pq + pr + qr$ using an additive sharing $A + B + C$ among the three parties. The additive sharing of $\hat{\varphi}$ is then

$$\varphi_a = N - A + p_a + q_a + r_a - 1 \quad ; \quad \varphi_b = -B + p_b + q_b + r_b \quad ; \quad \varphi_c = -C + p_c + q_c + r_c.$$

The conversion of $pq + pr + qr$ into an additive sharing $A + B + C$ is carried out using a simple variant of the BGW protocol used in the computation of N. The BGW protocol can be used to compute the value pq; however, instead of making the final result public the BGW variant shares the result additively among the three parties. The details of this variant can be found in [7, Section 6.2].

As before, we do not give the full details of the protocol for converting $pq + pr + qr$ into an additive sharing. Since we wish to focus on the primality test, will assume that an additive sharing of $\hat{\varphi}$ is available in the form of $\varphi_a + \varphi_b + \varphi_c$.

In addition to a sharing of $\hat{\varphi}$, we also require an additive sharing of $\hat{\psi} = (p+1)(q+1)(r+1)$. Once an additive sharing of $pq + pr + qr$ is available it is trivial to generate an additive sharing of $\hat{\psi}$. Simply set

$$\psi_a = N + A + p_a + q_a + r_a + 1 \quad ; \quad \psi_b = B + p_b + q_b + r_b \quad ; \quad \psi_c = C + p_c + q_c + r_c.$$

13.4 Comparison Protocol

Our primality test makes use of what we call a *comparison protocol*. Let A be a value known to Alice, B a value known to Bob, and C a value known to Carol. We may assume $A, B, C \in \mathbb{Z}_N^*$. The protocol enables the three parties to test that $ABC = 1 \bmod N$ without revealing any other information about the product ABC. We give the full details of the protocol in this section.

Let $P > N$ be some prime known to all parties. The protocol proceeds as follows:

Step 1. Carol picks a random element $C_1 \in \mathbb{Z}_N^*$ and sets $C_2 = CC_1^{-1} \bmod N$. Clearly $C = C_1C_2 \bmod N$. Carol then sends C_1 to Alice and C_2 to Bob.

Step 2. Alice sets $A' = AC_1$ and Bob sets $B' = (BC_2)^{-1} \bmod N$. Both values A' and B' can be viewed as integers in the range $[0, N)$. The problem is now reduced to testing whether $A' = B'$ (as integers) without revealing any other information about A and B.

Step 3. Alice picks a random $c \in \mathbb{Z}_P^*$ and $d \in \mathbb{Z}_P$. She sends c and d to Bob. Alice then computes $h(A') = cA' + d \bmod P$ and sends the result to Carol. Bob computes $h(B') = cB' + d \bmod P$ and sends the result to Carol.

Step 4. Carol tests if $h(A') = h(B') \bmod P$. If so, she announces that $ABC = 1 \bmod N$. Otherwise she announces $ABC \neq 1 \bmod N$.

The correctness and privacy of the protocol are stated in the next two lemmata. Correctness is elementary and is stated without proof.

Lemma 13.1 *Let $A, B, C \in \mathbb{Z}_N^*$. At the end of the protocol, the parties correctly determine if $ABC = 1 \bmod N$ or $ABC \neq 1 \bmod N$.*

Lemma 13.2 *The protocol is 1-private. That is, other than the result of the test, each party learns no new information.*

Proof To prove the protocol is 1-private, we provide a simulation argument for each party's view of the protocol. Alice's view of the protocol is made up of the values A, C_1, c, d, $h(A')$, and the final result of the test. These values can be easily simulated by picking C_1 at random in $\mathbb{Z}_N^*$, picking c at random in $\mathbb{Z}_P^*$ and d at random in $\mathbb{Z}_P$. This is a perfect simulation of Alice's view. A simulation argument for Bob is the same, mutatis mutandis.

Simulating Carol's view is more interesting. Carol's view consists of C, C_1, C_2, $h(A')$, $h(B')$, and the result of the test. The key observation we make is that $h(A')$ and $h(B')$ reveal no information about A and B, since they are either equal or are random independent elements of $\mathbb{Z}_P$. Which of the two actually occurs is determined by the result of the test. The independence follows since the family of hash functions $h(x) = cx + d \bmod P$ is a universal family of hash functions (i.e. knowing neither c nor d, the values $h(x)$ and $h(y)$ are independent for any $x, y \in \mathbb{Z}_P$).

To simulate Carol's view, the simulator picks $C_1, C_2 \in \mathbb{Z}_N^*$ at random so that $C = C_1 C_2 \bmod N$. Then, depending on the results of the test, it either picks the same random element of $\mathbb{Z}_P$ twice or picks two random independent elements of $\mathbb{Z}_P$. This is a perfect simulation of Carol's view. This proves that Carol gains no extra information from the protocol since, given the outcome of the test, she can generate the values sent by Alice and Bob herself. $\square$

14 The Probabilistic Primality Test

We now describe the main primality test. As discussed in the previous section, our primality test applies once the following setup is achieved:

Shares Each party i has three secret n-bit values p_i, q_i, and r_i for $i = a, b, c$.

The modulus $N = (p_a + p_b + p_c)(q_a + q_b + q_c)(r_a + r_b + r_c)$ is public. We set $p = p_a + p_b + p_c$, $q = q_a + q_b + q_c$, and $r = r_a + r_b + r_c$. Throughout this section, we are assuming that $p = q = r = 3 \bmod 4$. Thus, the parties must a priori co-ordinate the two least significant bits of their shares so that this condition holds.

Sharing $\hat{\varphi}, \hat{\psi}$ The parties share $(p-1)(q-1)(r-1)$ as $\varphi_a + \varphi_b + \varphi_c$ and $(p+1)(q+1)(r+1)$ as $\psi_a + \psi_b + \psi_c$.

Given this setup, they wish to test that p, q, and r are distinct primes without revealing any of p, q, and r. At this point, nothing is known about p, q, and r other than that $p = q = r = 3 \bmod 4$.

Throughout the section, we use the following notation:

$$\begin{aligned} \hat{\varphi} &= \varphi_a + \varphi_b + \varphi_c = (p-1)(q-1)(r-1) \\ \hat{\psi} &= \psi_a + \psi_b + \psi_c = (p+1)(q+1)(r+1) \end{aligned}$$

Clearly, if N is a product of three distinct primes, then $\varphi(N) = \hat{\varphi}$ and $\psi(N) = \hat{\psi}$. Otherwise, these equalities may not hold.

Our primality test is made up of four steps. At the beginning of each section, we state what each step tests. In the subsequent subsections, we explain how each step is carried out without revealing any information about the factors of N.

Step 1. The parties pick a random $g \in \mathbb{Z}_N^*$ and jointly test that $g^{\varphi_a+\varphi_b+\varphi_c} = 1 \bmod N$. If the test fails, N is rejected. This step reveals no information other than the outcome of the test. We refer to this step as a Fermat test in $\mathbb{Z}_N^*$ (see Sect. 14.2 for further details).

Step 2. The parties perform a Fermat test in the twisted group $\mathbb{T}_N$ defined as $(\mathbb{Z}_N[x]/(x^2+1))^*/\mathbb{Z}_N^*$. Notice that x^2+1 is irreducible modulo N, since $p = q = r = 3 \bmod 4$.

If N is the product of three distinct primes then the order of $\mathbb{T}_N$ is $\psi(N) = (p+1)(q+1)(r+1)$. To carry out the Fermat test in $\mathbb{T}_N$, the parties pick a random $g \in \mathbb{T}_N$ and jointly test that $g^{\psi_a+\psi_b+\psi_c} = 1$ (see Sect. 14.2 for further details). If the test fails, N is rejected. This step reveals no information other than the outcome of the test.

Step 3. The parties jointly test that N is the product of at most three prime powers. The implementation of this step is explained in Sect. 14.1. If the test fails, N is rejected.

Step 4. The parties jointly test that

$$\gcd(N, p+q+r) = 1.$$

This step reveals no information other than the outcome of the test. The implementation of this step is explained in Sect. 14.3. If the test fails, N is rejected. Otherwise, N is accepted as the product of three primes.

The following fact about the twisted group $\mathbb{T}_N = (\mathbb{Z}_N[x]/(x^2+1))^*/\mathbb{Z}_N^*$ is helpful in the proof of the primality test.

Fact 14.1 *Let N be an integer and k a prime such that $k^2|N$. Then, k divides both $\varphi(N)$ and $\psi(N)$.*

Proof Let $\alpha \geq 2$ be the number of times k divides N, i.e. $N = k^\alpha w$ where $\gcd(k, w) = 1$. Then $\varphi(N) = k^{\alpha-1}(k-1)\varphi(w)$ and, hence, k divides $\varphi(N)$.

To see that k divides $\psi(N) = |\mathbb{T}_N|$, note that $\mathbb{T}_N \cong \mathbb{T}_{k^\alpha} \times \mathbb{T}_w$. When $k = 3 \bmod 4$, we know that x^2+1 is irreducible in $\mathbb{Z}_k$ and, hence, $|\mathbb{T}_{k^\alpha}| = k^{\alpha-1}(k+1)$. It follows that k divides $\psi(N)$. When $k = 1 \bmod 4$, we have $|\mathbb{T}_{k^\alpha}| = k^{\alpha-1}(k-1)$ and, again, k divides $\psi(N)$. □

We can now prove that the aforementioned four steps are indeed a probabilistic test for proving that N is a product of three primes.

Theorem 14.2 *Let $N = pqr = (p_a+p_b+p_c)(q_a+q_b+q_c)(r_a+r_b+r_c)$, where $p = q = r = 3 \bmod 4$ and $\gcd(N, p+q+r) = 1$. If N is a product of three primes, it is always accepted. Otherwise, N is rejected with probability at least half. The probability is over the random choices made in steps 1–4 above.*

Proof Suppose p, q, and r are distinct primes. Then, steps (1), (2), and (3) clearly succeed. Step (4) succeeds by assumption. Hence, in this case, N always passes the test (as required).

Suppose N is *not* the product of three distinct primes. Assume, for the sake of deriving a contradiction, that N passes all four steps with probability greater than 1/2. Since N passes step (3) with probability greater than 1/2 we know that $N = z_1^{\alpha_1} z_2^{\alpha_2} z_3^{\alpha_3}$ for three primes z_1, z_2, and z_3 (not necessarily distinct). Since N passes step (4) we know $\gcd(N, p+q+r) = 1$. Define the following two groups:

$$\begin{aligned} G &= \left\{ g \in \mathbb{Z}_N^* \mid g^{\varphi_a+\varphi_b+\varphi_c} = 1 \right\} \text{ and} \\ H &= \left\{ g \in \mathbb{T}_N \mid g^{\psi_a+\psi_b+\psi_c} = 1 \right\}. \end{aligned}$$

Clearly, G is a subgroup of $\mathbb{Z}_N^*$ and H is a subgroup of the twisted group $\mathbb{T}_N$. By showing that at least one of G or H is a proper subgroup, we will prove that either step (1) or (2) fails with probability at least 1/2. There are two cases to consider:

Case 1: p, q, and r are not pairwise relatively prime. Without loss of generality, we may assume that $\gcd(p, q) > 1$. Let k be a prime factor of $\gcd(p, q)$. Recall that N is odd, so $k > 2$ (since k divides N).

Since $N = pqr$ we know that $k^2 | N$. Hence, by Fact 14.1, $k | \varphi(N)$ and $k | \psi(N)$. We claim that either k does not divide $\hat{\varphi}$ or k does not divide $\hat{\psi}$. To see this, observe that if $k | \hat{\varphi}$ and $k | \hat{\psi}$, then k divides $\hat{\psi} - \hat{\varphi} = p(2q + 2r) + q(2r) + 2$. Since k divides both p and q, we conclude that $k | 2$, which contradicts $k > 2$.

First, we examine when k does not divide $\hat{\varphi}$. Since k is a prime factor of $\varphi(N)$, there exists an element $g \in \mathbb{Z}_N^*$ of order k. However, since k does not divide $\hat{\varphi}$ we know that $g^{\hat{\varphi}} \neq 1$. Hence, $g \notin G$ proving that G is a proper subgroup of $\mathbb{Z}_N^*$. If k does not divide $\hat{\psi}$ a similar argument proves that H is a proper subgroup of the twisted group $\mathbb{T}_N$.

Case 2: p, q, and r are pairwise relatively prime. We can write $p = z_1^{\alpha}$, $q = z_2^{\beta}$, and $r = z_3^{\gamma}$ with z_1, z_2, and z_3 distinct primes. By assumption, we know that one of α, β, or γ is greater than 1. Without loss of generality, we may assume $\alpha > 1$.

We first observe that none of the z_i can divide $\gcd(\hat{\varphi}, \hat{\psi})$. Indeed, if this were not the case, then z_i would divide $\hat{\varphi} + \hat{\psi} = 2(N + p + q + r)$. But then, since z_i divides N, it must also divide $p + q + r$, contradicting that $\gcd(N, p+q+r) = 1$ (as tested in step (4)).

We now know that either z_1 does not divide $\hat{\varphi}$ or it does not divide $\hat{\psi}$. However, since z_1^2 divides N, we obtain, by Fact 14.1, that $z_1 | \varphi(N)$ and $z_1 | \psi(N)$. We can now proceed as in case (1) to prove that either G is a proper subgroup of $\mathbb{Z}_N^*$ or H is a proper subgroup of $\mathbb{T}_N$.

□

Clearly, most integers N that are not a product of three primes will already fail step (1) of the test. Hence, steps (2–4) are most likely executed only once a good candidate N is found.

Notice that the condition $\gcd(N, p+q+r) = 1$ is necessary. Without it, the theorem is false as can be seen from the following simple example: $p = w^3$, $q = aw^2 + 1$, and $r = bw^2 - 1$ where w, q, r are three odd primes with $p = q = r = 3 \bmod 4$. In this case, $N = pqr$ will always pass steps 1–3, even though it is not a product of three distinct primes.

14.1 Step 3: Testing that $N = p^\alpha q^\beta r^\gamma$

Our protocol for testing that N is a product of three prime powers borrows from a result of van de Graaf and Peralta [38]. Our protocol works as follows:

Step 0. From our construction of $\hat{\varphi}$, we know that it is divisible by 8. However, the individual shares φ_a, φ_b, and φ_c which sum to $\hat{\varphi}$ may not all be divisible by 8. To correct this, Alice generates two random numbers $a_1, a_2 \in \mathbb{Z}_8$ such that $a_1 + a_2 = \varphi_a \bmod 8$. She sends a_1 to Bob and a_2 to Carol. Alice sets $\varphi_a \leftarrow \varphi_a - a_1 - a_2$, Bob sets $\varphi_b \leftarrow \varphi_b + a_1$ and Carol set $\varphi_c \leftarrow \varphi_c + a_2$. Observe that, at this point,

$$\frac{\hat{\varphi}}{8} = \frac{\varphi_a}{8} + \left\lfloor \frac{\varphi_b}{8} \right\rfloor + \left\lceil \frac{\varphi_c}{8} \right\rceil .$$

Step 1. The parties first agree on eight random numbers $g_1, g_2, \ldots, g_8$ in $\mathbb{Z}_N^*$, all with Jacobi symbol $+1$.

Step 2. For $i, j \in \{1, 2, \ldots, 8\}$, we say that i is equivalent to j (this defines equivalence classes of $\{1, 2, \ldots, 8\}$) if

$$\left(\frac{g_i}{g_j}\right)^{\frac{\varphi_a + \varphi_b + \varphi_c}{8}} = 1 \pmod{N}.$$

Since all three parties know g_i and g_j, they can test if i is equivalent to j as follows:

1. Alice computes $A = (g_i/g_j)^{\varphi_a/8} \bmod N$,
   Bob computes $B = (g_i/g_j)^{\lfloor \varphi_b/8 \rfloor} \bmod N$, and
   Carol computes $C = (g_i/g_j)^{\lceil \varphi_c/8 \rceil} \bmod N$.

2. Using the comparison protocol of Sect. 13.4, they then test if $ABC = 1 \bmod N$. The comparison protocol reveals no information other than whether or not $ABC = 1 \bmod N$.

Step 3. If the number of equivalence classes is greater than four, N is rejected. Otherwise, N is accepted.

Testing that the number of equivalences classes is at most four requires at most twenty-two invocations of the comparison protocol. Note that we restrict our attention to the elements g_i with Jacobi symbol $+1$ for efficiency's sake. Without this restriction, the number of equivalence classes to check for is eight and, thus, many more applications of the comparison protocol would be necessary.

The following lemma shows that when N is a product of three distinct primes, it is always accepted. When N has more than three prime factors, it is rejected with probability at least 1/2. Note that if N is a product of three prime powers, it will always be accepted by this protocol. We will use the following notation:

$$\begin{aligned} J &= \left\{ g \in \mathbb{Z}_N^* \mid \left(\frac{g}{N}\right) = +1 \right\} \\ Q &= \{ g \in J \mid \ g \text{ is a quadratic residue in } \mathbb{Z}_N^* \} \end{aligned}$$

The index of Q in J is $2^{d(N)-1}$ or $2^{d(N)}$ (exactly when N is a perfect square), where $d(N)$ is the number of distinct prime factors of N.

Lemma 14.3 *Let $N = pqr$ be an integer with $p = q = r = 3 \bmod 4$. If p, q, and r are distinct primes, then N is always accepted. If the number of distinct prime factors of N is greater than three, then N is rejected with probability at least half.*

Proof If N is the product of three distinct primes, then the index of Q in J is four. Two elements $g_1, g_2 \in \mathbb{Z}_N^*$ belong to the same coset of Q in J if and only if g_1/g_2 is a quadratic residue, i.e. if and only if $(g_1/g_2)^{\varphi(N)/8} = 1 \bmod N$. Since, in this case, $\varphi(N) = \hat{\varphi} = \varphi_a + \varphi_b + \varphi_c$, step (2) tests if g_i and g_j are in the same coset of Q. Since the number of cosets is four, there are exactly four equivalence classes and, thus, N is always accepted.

If N contains at least four distinct prime factors, we show that it is rejected with probability at least 1/2. Define

$$\hat{Q} = \left\{ g \in J \mid g^{\hat{\varphi}/8} = 1 \pmod{N} \right\}.$$

Since, in this case, $\hat{\varphi}$ may not equal $\varphi(N)$, the group $\hat{Q}$ need not be the same as the group Q.

We now show that the index of $\hat{Q}$ in J is at least eight. Since $p = q = r = 3 \bmod 4$, we know that $\hat{\varphi}/8$ is odd (since $\hat{\varphi} = (p-1)(q-1)(r-1)$). Notice that f $g \in J$ satisfies $g^x = 1$ for some odd x, g must be a quadratic residue (a root is $g^{(x+1)/2}$). Hence, $\hat{Q} \subseteq Q$ and hence is a subgroup of Q. Since the index of Q in J is at least eight, it follows that the index of $\hat{Q}$ in J is at least eight.

It remains to show that when the index of $\hat{Q}$ in J is at least eight, N is rejected with probability at least 1/2. In step (2), two elements $g_1, g_2 \in J$ are said to be equivalent if they belong to the same coset of $\hat{Q}$ in J. Let R be the event that all 8 elements $g_i \in J$ chosen randomly in step (1) fall into only four of the eight cosets. Then

$$\Pr[R] \leq \binom{8}{4} \cdot \left(\frac{1}{2}\right)^8 \approx 0.27 < \frac{1}{2}.$$

N is accepted only when the event R occurs. Since R occurs with probability less than 1/2, the number N is rejected with probability at least 1/2 as, required. $\square$

Next, we prove that the protocol leaks no information when N is indeed the product of three distinct primes. In case N is not of this form, the protocol may leak some information; however, in this case, N is discarded and is of no interest. To prove that the protocol leaks no information we rely on a classic cryptographic assumption (see [6]) called *Quadratic Residue Indistinguishability* (QRI). This cryptographic assumption states that when $N = pq$ with $p = q = 3 \bmod 4$, no polynomial time algorithm can distinguish between the groups J and Q defined above. In other words, for any polynomial time algorithm $\mathcal{A}$ and any constant $c > 0$,

$$\left| \Pr_{g \in J}[\mathcal{A}(g) = \text{``yes''}] - \Pr_{g \in Q}[\mathcal{A}(g) = \text{``yes''}] \right| < \frac{1}{(\log N)^c}.$$

Lemma 14.4 *If N is a product of three distinct primes, then the protocol is 1-private, assuming QRI.*

Proof Sketch To prove that each party learns no information other than that N is a product of three prime powers, we provide a simulation argument. We show that each party can simulate its view of the protocol; hence, whatever values it receives from its peers, it could have generated itself. By symmetry, we need only consider Alice. Alice's view of the protocol consists of the elements $g_1, g_2, \ldots, g_8$ and bit values $b_{i,j}$ indicating whether $(g_i/g_j)^{\hat{\varphi}} = 1$ (recall that we already gave a simulation algorithm for the comparison protocol in Sect. 13.4). Thus, Alice learns whether or not each g_i/g_j is a quadratic residue. We argue that, under QRI, this provides no computational information (since it can be simulated). To simulate Alice's view, the simulation algorithm works as follows: it picks eight random elements $g_1, g_2, \ldots, g_8 \in J$. It then randomly associates with each g_i a value in the set $\{0, 1, 2, 3\}$. This value represents the coset of Q in which g_i lies. The simulator then says that g_i/g_j is a quadratic residue if and only if the value associated with g_i is equal to that associated with g_j. Under QRI, the resulting distribution on $g_1, g_2, \ldots, g_8, b_{1,1}, b_{1,2}, \ldots, b_{8,8}$ is computationally indistinguishable from Alice's true view of the protocol.

We note that the value $a_1 \in \mathbb{Z}_8$ that Alice sends Bob in step (0) is an element of $\mathbb{Z}_8$ chosen uniformly at random. Hence, Bob can simulate it trivially. Similarly, Carol can trivially simulate $a_2 \in \mathbb{Z}_8$. □

14.2 Implementing a Fermat Test with No Information Leakage

We briefly show how to implement a Fermat test in $\mathbb{Z}_N^*$ without leaking any extra information about the private shares. The exact same method works in the twisted group $\mathbb{T}_N$ as well.

To check that $g \in \mathbb{Z}_N^*$ satisfies $g^{\varphi_a+\varphi_b+\varphi_c} = 1 \bmod N$, we perform the following steps:

Step 1. Each party computes $R_i = g^{\varphi_i} \bmod N$ $(i = a, b, c)$.

Step 2. They test that $R_a R_b R_c = 1 \bmod N$ by simply revealing the values R_1, R_2, and R_3. Accept N if the test succeeds. Otherwise, reject.

Clearly, the protocol succeeds if and only if $g^{\hat{\varphi}} = 1 \bmod N$. We show that it leaks no other information.

Lemma 14.5 *If $N = pqr$ is the product of three distinct primes, then the protocol is 2-private.*

Proof We show that no two parties learn any information about the private share of the third (other than that $g^{\hat{\varphi}} = 1 \bmod N$). By symmetry, we may restrict our attention to Alice and Bob. Since, by assumption, N is the product of three distinct primes, we know that $g^{\hat{\varphi}} = 1 \bmod N$. Hence, $g^{\varphi_a+\varphi_b} = g^{-\varphi_c}$. To simulate the value received from Carol, the simulation algorithm simply computes $R_c = g^{-\varphi_a-\varphi_b}$. Indeed, this is a perfect simulation of Alice and Bob's view. Thus, they learn nothing from Carol's message since they could have generated it themselves. □

14.3 Step 4: Zero-Knowledge Test that $\gcd(N, p+q+r) = 1$

Our protocol for this step is based on a protocol similar to the one used in the computation of N. We proceed as follows:

Step 1. Alice picks a random $y_a \in \mathbb{Z}_N$, Bob picks a random $y_b \in \mathbb{Z}_N$, and Carol picks a random $y_c \in \mathbb{Z}_N$.

Step 2. Using the BGW protocol as in Sect. 13.2, they compute

$$R = (p_a + q_a + r_a + p_b + q_b + r_b + p_c + q_c + r_c)(y_a + y_b + y_c) \bmod N.$$

At the end of the protocol, R is publicly known; however, no other information about the private shares is revealed.

Step 3. Now that R is public, the parties test that $\gcd(R, N) = 1$. If not, N is rejected. Otherwise, N is accepted.

Lemma 14.6 *If N is the product of three distinct n-bit primes p, q, and r, with* $\gcd(N, p+q+r) = 1$*, then N is accepted with probability $1 - \epsilon$ for some $\epsilon < 1/2^{n-3}$. Otherwise, N is always rejected.*

Proof Clearly, if $\gcd(N, p + q + r) > 1$ then $\gcd(R, N) > 1$ and therefore N is always rejected. If $\gcd(N, p + q + r) = 1$, then N is rejected only if $\gcd(N, y_a + y_b + y_c) > 1$. Since $y_a + y_b + y_c$ is a random element of $\mathbb{Z}_N$, this happens with probability less than $(1/2)^{n-3}$. □

Lemma 14.7 *If N is the product of three distinct n-bit primes p, q, and r, with* $\gcd(N, p+q+r) = 1$*, then the protocol is 1-private.*

Proof Note that, since the BGW protocol is 1-private, the above protocol can be at best 1-private. By symmetry, we need only show how to simulate Alice's view. Alice's view consists of her private shares p_a, q_a, y_a and the number R. Since R is independent of her private shares, the simulator can simulate Alice's view by simply picking R in $\mathbb{Z}_N$ at random. This is a perfect simulation. □

15 Extensions

One can naturally extend our protocols in two ways. First, one may allow more than three parties to generate a product of three primes with an unknown factorization. Second, one may wish to design primality tests for testing that N is a product of k primes for some small k. We briefly discuss both extensions below.

Our protocols easily generalize to allow any number of parties. When k parties are involved, the protocols can be made $\lfloor \frac{k-1}{2} \rfloor$-private. This is optimal in the information-theoretic sense and follows from the privacy properties of the BGW protocol. The only complexities in this extension are the comparison protocol of Sect. 13.4 and Step (0) of Sect. 14.1. Both protocols generalize to k parties; however, they require a linear (in k) number of rounds of communication.

Securely testing that N is a product of k primes for some fixed $k > 3$ seems to be harder. Our results apply when $k = 4$ (indeed Theorem 14.2 remains true in this case). For $k > 4$, more complex algorithms are necessary. This extension may not be of significant interest since it is not well motivated and requires complex protocols.

Another natural question is whether only two parties can generate a product of three primes with an unknown factorization. The answer appears to be yes, although the protocols cannot be information-theoretically secure. Essentially, one needs to replace the BGW protocol for computing N with a two-party private multiplication protocol. This appears to be possible using results of [11].

16 Conclusions and Open Problems

Our main contribution is the design of a probabilistic primality test that enables three (or more) parties to generate a number N with an unknown factorization and test that N is the product of three distinct primes. The correctness of our primality test relies on that we simultaneously work in two different subgroups of $\mathbb{Z}_N[x]/(x^2+1)^*$, namely $\mathbb{Z}_N^*$ and the twisted multiplicative group $\mathbb{T}_N$.

Our protocol generalizes to an arbitrary number of parties k and achieves $\lfloor \frac{k-1}{2} \rfloor$-privacy — the best possible in an information-theoretic setting.

Recall that our primality test can be applied to $N = pqr$ whenever $p = q = r = 3 \bmod 4$. We note that simple modifications enable one to apply the test when $p = q = r = 1 \bmod 4$ (essentially, this is done by reversing the roles of $\mathbb{Z}_N^*$ and the twisted group). However, it seems that one of these restrictions is necessary; we do not know how to carry out the test without the assumption that $p = q = r \bmod 4$. The assumption plays a crucial role in the proof of Lemma 14.3.

A natural question to ask is whether more advanced primality testing techniques can be used to improve the efficiency of our test. For instance, recent elegant techniques due to Grantham [30] may be applicable in our scenario as well.

References

[1] N. Alon, Z. Galil and M. Yung, "Dynamic-resharing verifiable secret sharing," ESA 1995.

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

[3] M. Ben-Or, S. Goldwasser, A. Wigderson, "Completeness theorems for non-cryptographic fault tolerant distributed computation", STOC 1988, pp. 1–10.

[4] J. Benaloh (Cohen), "Secret sharing homomorphisms: keeping shares of a secret secret," Crypto '86, 251-260.

[5] S. Blackburn, S. Blake-Wilson, M. Burmester, S. Galbraith, "Secure construction of shared RSA keys", preprint.

[6] M. Blum and S. Goldwasser. An efficient probabilistic public key encryption scheme that hides all partial information. In *Proceedings of Advances in Cryptology: CRYPTO '84*, pages 289–302. Lecture Notes in Computer Science, Springer-Verlag, New York, 1985.

[7] D. Boneh, M. Franklin, "Efficient generation of shared RSA keys", in Proceedings Crypto' 97, pp. 425–439.

[8] D. Boneh, J. Horwitz, "Generating a product of three primes with and unknown factorization", Proc. of the third Algorithmic Number Theory Symposium (ANTS), 1998, pp.

[9] J. Carter and M. Wegman, "Universal classes of hash functions", *J. Comput. Syst. Sci.* 18 (1979), 143–154.

[10] D. Chaum, C. Crépeau, and I. Damgård, "Multiparty unconditionally secure protocols," ACM STOC 1988, 11-19.

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

[12] D. Coppersmith, "Private Communications".

[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", STOC 1994, pp. 522–533.

[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", Asiacrypt '94, pp. 21–32.

[17] Y. Desmedt and Y. Frankel, "Shared generation of authenticators and signatures", Crypto '91, 457–469.

[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] R. Fagin, M. Naor, P. Winkler, "Comparing information without leaking it", CACM, Vol 39, No. 5, May 1996, pp. 77–85.

[20] U. Feige, A. Fiat, and A. Shamir, "Zero-knowledge proofs of identity," Journal of Cryptology 1 (1988), 77-94.

[21] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," Crypto '86, 186-194.

[22] Y. Frankel, "A practical protocol for large group oriented networks", Eurocrypt 89, pp. 56–61.

[23] Y. Frankel, P. MacKenzie, M. Yung, "Optimal resilience proactive proactive public key cryptosystems", FOCS 1997.

[24] Y. Frankel, P. MacKenzie, M. Yung, "Robust efficient distributed RSA key generation", STOC 1998, pp. 663–672.

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

[26] P. Gemmel, "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", Crypto 96, pp. 157–172.

[28] O. Goldreich, S. Micali, A. Wigderson, "How to play any mental game", STOC 1987, pp. 218–229.

[29] J. Gordon, "Strong primes are easy to find", Eurocrypt 84, pp. 216–223.

[30] J. Grantham. A probable prime test with high confidence. Available online (`http://www.clark.net/pub/grantham/pseudo/`).

[31] L. Guillou and J. Quisquater, "A practical zero-knowledge protocol fitted to security microprocessor minimizing both transmission and memory," Eurocrypt '88, 123-128.

[32] A. Lenstra, H. W. Lenstra ed, "The development of the number field sieve", Lecture Notes in Computer Science 1554, Springer-Verlag, 1994.

[33] H. W. Lenstra, "Factoring integers with elliptic curves", *Annals of Mathematics*, 126:649–673, 1987.

[34] M. Malkin, T. Wu, D. Boneh, "Experimenting with shared RSA key generation", preprint.

[35] K. Ohta and T. Okamoto, "A modification of the Fiat-Shamir scheme," Crpto '88, 232-243.

[36] H. Ong and C. Schnorr, "Fast signature generation with a Fiat Shamir-like scheme," Eurocrypt '90, 432-440.

[37] T. Pederson, "A threshold cryptosystem without a trusted party," Proceedings of Eurocrypt 91, pp. 522–526.

[38] R. Peralta and J. van de Graaf. A simple and secure way to show the validity of your public key. In *Proceedings of Advances in Cryptology: CRYPTO '87*, pages 128–134. Lecture Notes in Computer Science, Springer-Verlag, New York, 1988.

[39] M. Rabin, "Probabilistic algorithm for testing primality", J. of Number Theory, vol. 12, pp. 128–138, 1980.

[40] T. Rabin, "A simplified approach to threshold and proactive RSA", Proceedings of Crypto' 98.

[41] A. Shamir, "How to share a secret", Comm. of the ACM, Vol. 22, 1979, pp. 612–613.

[42] R. Solovay, V. Strassen, "A fast monte carlo test for primality", SIAM journal of computing, vol. 6, pp. 84–85, 1977.

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

[44] M. Wiener. Cryptanalysis of short RSA secret exponents. *IEEE Transactions on Information Theory*, 36(3):553-558, 1990.

[45] A. Yao, "How to generate and exchange secrets", FOCS 1986, pp. 162–167.