Efficient Computation Modulo a Shared Secret with Application to the Generation of Shared Safe-Prime Products

Joy Algesheimer Jan Camenisch Victor Shoup

IBM Research
Zurich Research Laboratory
CH–8803 Rüschlikon
{jmu|jca|sho}@zurich.ibm.com

March 6, 2002

Abstract

We present a new protocol for efficient distributed computation modulo a shared secret. We further present a protocol to distributively generate a random shared prime or safe prime that is much more efficient than previously known methods. This allows to distributively compute shared RSA keys, where the modulus is the product of two safe primes, much more efficiently than was previously known.

Keywords. RSA, safe primes, threshold cryptography, distributed primality test.

1 Introduction

Many distributed protocols, e.g., [FFS88, FH94, GJKR96], require that an RSA modulus $N = pq$ is generated during system initialization, together with a public exponent e and shares of the corresponding private exponent. Moreover, many protocols, e.g., [Sho00, CS99, GHR99, ACJT00, CL01], even require that N is the product of "safe" primes, i.e., $p = 2p' + 1$ and $q = 2q' + 1$, where p' and q' are themselves prime. While the requirement for safe primes can sometimes be avoided (e.g., [DK01, FS01]), this typically comes at the cost of extra communication, computation, and/or non-standard intractability assumptions.

While the initialization of the system with an RSA modulus N can be accomplished using a "trusted dealer," it would be preferable not to rely on this.

Given a distributed protocol to generate a random (safe) prime, securely shared among the players, it is not too difficult to solve the above problem. One can of course use general multi-party computation techniques of Ben-Or, Goldwasser and Wigderson [BGW88] to generate a random, shared (safe) prime. Indeed, that would work as follows: one starts with a standard algorithm for generating a random (safe) prime, and converts this algorithm into a corresponding Boolean or arithmetic circuit, and then for each gate in this circuit, the players perform a distributed multiplication modulo a small prime t. This protocol is not very practical, especially as the players need to perform a distributed computation for *every gate* in the circuit, and so unlike in the non-distributed prime generation algorithm, they cannot use much more efficient algorithms for working with large integers.

In this paper, we present new protocols that allow one to perform arithmetic modulo a secret, shared modulus in a way that is much more efficient than can be done using the general techniques of Ben-Or et al. More specifically, we develop a new protocol to efficiently compute shares of c, where $c \equiv ab \pmod{p}$, given shares of a, b, and p. The shares of a, b, c, and p are integers modulo Q, where Q is a prime whose bit-length is roughly twice that of p, and the cost of this protocol is essentially the cost of performing a small, constant number of distributed multiplications modulo Q. Actually, this is the amortized cost of multiplication modulo p assuming many such multiplications are performed for a fixed p. This protocol, together with several other new supporting protocols, gives us a protocol to generate a random, shared prime, or safe prime, that is much more efficient than the generically derived protocol discussed above. In particular, we obtain a protocol for jointly generating an RSA modulus that is the product of *safe* primes that is much more efficient in practice than any generic circuit-based protocol (which are the only previously known protocols for this problem), even using the most efficient circuits for integer multiplication, division, etc.

Our protocols work in the so-called "honest-but-curious" model. That is, we assume that all players follow the protocol honestly, but we guarantee that even if a minority of players "pool" their information they cannot learn anything that they were not "supposed" to. Even though we make this restriction, fairly standard techniques can be used to make our protocols robust, while maintaining their practicality. In fact, using "optimistic" techniques for robustness, we can obtain a fully robust protocol for distributively generating an RSA modulus that is not significantly less efficient than our honest-but-curious solution — this is the subject of on-going work.

Related Work. Boneh and Franklin [BF97] present a protocol for jointly generating an RSA modulus $N = pq$ along with a a public exponent and shares of the corresponding private key. Like us, they also work in the honest-but-curious adversary model. Unlike ours, their protocol is not based on a sub-protocol for generating a random, shared prime. While our protocol for this task is asymptotically more efficient than the protocol of Boneh and Franklin (when the number of players is small), we do not claim that our protocol is in practice more efficient than theirs for typical parameter choices. The relative performance of these protocols in such a practical setting depends on a myriad of implementation details.

Unlike our techniques, those of Boneh and Franklin do not give rise to a protocol for jointly generating an RSA modulus $N = pq$, where p and q are safe primes. Indeed, prior to our work, the only know method for solving this problem was to apply the much less efficient general circuit technique of [BGW88].

As our protocols rely mainly on distributed multiplication over a prime field, rather than over the integers, one can easily make them robust using traditional techniques for verifiable secret sharing modulo a prime, avoiding the somewhat less efficient techniques by Frankel et al. [FMY87] for robust distributed multiplication over the integers. Moreover, using the optimistic approach mentioned above, even further improvements are possible, so that we can get robustness essentially "for free".

2 Model

We consider k players $P_1, \ldots, P_k$ that are mutually connected by secure and authentic channels. Our protocols are secure against a static and honest-but-curious behaving adversary, controlling up to $\tau = \lfloor \frac{k-1}{2} \rfloor$. That is, all players follow the protocol honestly but the dishonest players may pool their data and try to derive additional information. We finally assume that no party stops participating prematurely (we use k-out-of-k secret sharing schemes).

However, these assumptions can be relaxed: First, it's possible to force the participants to behave honestly by having them to commit to their inputs, to generate their individual random strings jointly, and to prove (using zero-knowledge proofs) that they followed the protocols correctly. Second, the k-out-of-k secret sharing schemes can easily be converted into k-out-of-l ones by the 'share back-up' method introduced by Rabin [Rab98]. We do not pursue these possibilities here.

We prove security in the model by Canetti [Can00]. Here, we describe a simplified version of it for a static adversary in the honest-but-curious model. Such an adversary first chooses the players he wants to corrupt and then gets to see their inputs, their internal state and all the messages they receive. A protocol π is proved secure by specifying the functionality f the protocol should provide in an ideal world where all the parties send their inputs to a trusted third party T who then returns to them the outputs they are to obtain according to f. Let $\pi_i(x_1, \ldots, x_k, \rho)$ denote the output of party P_i when running protocol π on input x_i in the presence of adversary $\mathcal{A}$, where ρ is a security parameter. As $\mathcal{A}$ behaves honest-but-curious, the output $\pi_i(x_1, \ldots, x_k, \rho)$ does not depend on $\mathcal{A}$.

Definition 1. *A protocol is said to be statistically secure if for any honest-but-curious behaving adversary $\mathcal{A}$ there exists a probabilistic polynomial-time simulator $\mathcal{S}$ such that the two ensembles of random variables*

$$\{\mathcal{A}(z), \pi_1(x_1, \ldots, x_k, \rho), \ldots, \pi_k(x_1, \ldots, x_k, \rho)\}_{\rho \in \mathbb{N}; z, x_1, \ldots, x_k \in \{0,1\}^*}$$

and

$$\{\mathcal{S}(z), f_1(x_1, \ldots, x_k, \rho), \ldots, f_k(x_1, \ldots, x_k, \rho)\}_{\rho \in \mathbb{N}; z, x_1, \ldots, x_k \in \{0,1\}^*}$$

are statistically indistinguishable.

It can be shown that security is preserved under non-concurrent, modular composition of protocols [Can00].

3 Preliminaries

3.1 Notation

Let a be a real number. We denote by $\lfloor a \rfloor$ the largest integer $b \leq a$, by $\lceil a \rceil$ the smallest integer $b \geq a$, and by $\lceil a \rfloor$ the largest integer $b \leq a + 1/2$. We denote by $\mathsf{trunc}(a)$ the integer b such that $b = \lceil a \rceil$ if $a < 0$ and $b = \lfloor a \rfloor$ if $a \geq 0$; that is, $\mathsf{trunc}(a)$ rounds a towards 0.

Let Q be a positive integer. All modular arithmetic is done centered around 0; to remind the reader of this, we use 'rem' as the operator for modular reduction rather than 'mod', i.e., $c \operatorname{rem} Q$ is $c - \lceil c/Q \rfloor Q$.

Define $\mathcal{Z}_Q$ as the set $\{x \in \mathbb{Z} \mid -Q/2 < x \leq Q/2\}$ (we should emphasize that $\mathcal{Z}_Q$ is properly view as a set of integers rather than a ring). We denote an additive sharing of a value $a \in \mathcal{Z}_Q$ over $\mathcal{Z}_Q$ by $\langle a \rangle_1^Q, \ldots, \langle a \rangle_k^Q \in \mathcal{Z}_Q$, i.e., $a = \sum_{j=1}^{k} \langle a \rangle_j^Q \operatorname{rem} Q$ and by $[a]_1^Q, \ldots, [a]_k^Q \in \mathcal{Z}_Q$ we denote a polynomial sharing (also called Shamir-sharing [Sha79]), i.e., $a = \sum_{i=1}^{\tau} \lambda_j [a]_j^Q \operatorname{rem} Q$, where λ_j are the Lagrange coefficients. The latter only works if $Q > k$ and if Q is prime.

For $a \in \mathcal{Z}$ we denote by $\langle a \rangle_1^I, \ldots, \langle a \rangle_k^I \in \mathcal{Z}$ an additive sharing of a over the integers, i.e., $a = \sum_{j=1}^{k} \langle a \rangle_j^I$.

We denote protocols as follows: the term $a := \mathsf{PROTOCOLNAME}(b)$ means that the player in consideration runs the protocol $\mathsf{PROTOCOLNAME}$ with local input b and gets local output a as the result of the protocol. Finally, $\lg(x)$ denotes the logarithm of x to the base 2.

3.2 Known Primitives

We recall the known secure multi-party protocols for efficient distributed computation with shared secrets that we will use to compose our protocols, and we state the number of bit-operations for which we assume $\lg Q = \Theta(n)$ and that the bit-complexity of a multiplication two n-bit integers is $O(n^2)$ (which is a reasonable estimate for realistic values of n, e.g., $n = 1024$). The round-complexity of all primitives is $O(1)$ and their communication is $O(kn)$ bits (we consider communication complexity to be the number of bits each player sends on average).

Additive sharing over $\mathcal{Z}_Q$: To share a secret $a \in \mathcal{Z}_Q$ player P_j chooses $\langle a\rangle_i^Q \in_R \mathcal{Z}_Q$ for $i \neq j$, sets $\langle a\rangle_j^Q := a - \sum_{i=1,i\neq j}^{k} \langle a\rangle_i^Q \operatorname{rem} Q$, and sends $\langle a\rangle_i^Q$ to player P_i. This takes $O(kn)$ bit operations.

Polynomial sharing over $\mathcal{Z}_Q$: To share a secret $a \in \mathcal{Z}_Q$ player P_j chooses coefficients $a_l \in_R \mathcal{Z}_Q$ for $l = 1, \ldots, \tau$, where $\tau = \lfloor (k-1)/2 \rfloor$, and sets $[a]_i^Q := a + \sum_{l=1}^{\tau} a_l i^l \operatorname{rem} Q$, and sends $[a]_i^Q$ to player P_i. This takes $O(nk^2 \lg k)$ bit operations.

Additive sharing over $\mathbb{Z}$: To share a secret $a \in [-A, A]$ player P_j chooses $\langle a\rangle_i^I \in_R [-A2^\rho, A2^\rho]$ for $i \neq j$, where ρ is a security parameter, and sets $\langle a\rangle_j^I := a - \sum_{i=1,i\neq j}^{k} \langle a\rangle_i^I$, and sends $\langle a\rangle_i^I$ to player P_i. Note that for any set of $k-1$ players, the distribution of shares of different secrets are statistically indistinguishable for suitably large ρ (e.g., $\rho = 128$). This takes $O(k(\rho + \lg A))$ bit operations.

Distributed computation over $\mathcal{Z}_Q$: Addition and multiplication modulo Q of a constant and a polynomially shared secret is done by having all players locally add or multiply the constant to their shares. Hence $[a]_j^Q + c \operatorname{rem} Q$ is a polynomial share of $a + c \operatorname{rem} Q$ and $c \cdot \langle a\rangle_j^Q \operatorname{rem} Q$ is a polynomial share of $ac \operatorname{rem} Q$. These operations take $O(n)$ and $O(n^2)$ bit operations, respectively.
Addition of two shared secrets is achieved by having the players locally add their shares. Thus $[a]_j^Q + [b]_j^Q \operatorname{rem} Q$ is a polynomial share of $a + b \operatorname{rem} Q$ and takes $O(\lg Q)$ bit operations.
Multiplication modulo Q of two polynomially shared secrets is done by jointly executing a multiplication protocol due to Ben-Or, Goldwasser and Wigderson [BGW88] or by a more efficient variant due to Gennaro, Rabin and Rabin [GRR98] which requires $O(n^2 k + nk^2 \lg k)$ bit operations for each player. We denote this protocol by $\mathsf{MUL}([a]_j^Q, [b]_j^Q)$.

Joint random sharing over $\mathcal{Z}_Q$: To generate shares of a secret chosen jointly at random from $\mathcal{Z}_Q$, each player chooses a random number $r_i \in_R \mathcal{Z}_Q$ and shares it according to the required type of secret sharing scheme and sends the shares to the respective players. Each player adds up all the shares gotten to obtain a share of a random value. We denote this protocol by $\mathsf{JRS}(\mathcal{Z}_Q)$ in case the players get additive shares and by $\mathsf{JRP}(\mathcal{Z}_Q)$ if they get polynomial shares. The protocols require $O(nk)$ and $O(nk^2 \lg k)$ bit operations per player, respectively.

Joint random sharing of 0*:* In protocols it is often needed to re-randomized shares obtained from some computation by adding random shares of 0. Such shares can be obtained for any sharing scheme by having each player share 0 according to the required type of secret sharing scheme and sending them to the respective players. Each player adds up all the shares gotten to obtain a share of 0. We denote this protocol by $\mathsf{JRSZ}(\mathcal{Z}_Q)$ in case the players get additive shares over $\mathcal{Z}_Q$ and $\mathsf{JRPZ}(\mathcal{Z}_Q)$ if they get polynomial shares over $\mathcal{Z}_Q$. The protocols require $O(nk)$ and $O(nk^2 \lg k)$ bit operations per player, respectively. In case we want to have additive shares over the integers, it is required to give the range (e.g., $[-A, A]$) from which the players choose the shares they send to the other players. We denote this protocol by $\mathsf{JRIZ}([-A, A])$ and it requires $O(k(\rho + \lg A))$ bit operations per player.

Computing shares of the inverse of a shared secret: This protocol works only for polynomial sharings over $\mathcal{Z}_Q$. Let a be the shared invertible element. Then, a protocol due to Bar-Ilan and

Beaver [BB89] computes shares of $a^{-1} \operatorname{rem} Q$ given shares $[a]_j^Q$. The protocol, denoted by $\mathsf{INV}([a]_j^Q)$, is as follows: first run $[r]_j^Q := \mathsf{JRP}(\mathcal{Z}_Q)$, then compute $[u]_j^Q := \mathsf{MUL}([a]_j^Q, [r]_j^Q)$, reveal $[u]_j^Q$, and reconstruct u. If $u \equiv 0 \pmod{Q}$, the players start over. Otherwise, they each locally compute their share of $a^{-1} \operatorname{rem} Q$ as $(u^{-1} \operatorname{rem} Q) \cdot [r]_j^Q \operatorname{rem} Q$. This protocol requires an expected number of $O(n^2 k + nk^2 \lg k)$ bit operations per player.

Joint random invertible element sharing: This protocol denoted $\mathsf{JRP\text{-}INV}(\mathcal{Z}_Q)$ is due to Bar-Ilan and Beaver [BB89]. The players generate shares of random elements $[r]_j^Q := \mathsf{JRP}(\mathcal{Z}_Q)$ and $[s]_j^Q := \mathsf{JRP}(\mathcal{Z}_Q)$, jointly compute $[u]_j^Q := \mathsf{MUL}([s]_j^Q, [r]_j^Q)$, reveal $[u]_j^Q$ and then reconstruct u. If u is non-zero, they each take $[r]_j^Q$ as their share of a random invertible element. Otherwise, they repeat the protocol. The protocol requires an expected number of $O(nk^2 \lg k + n^2 k)$ bit operations per player.

4 Conversions Between Different Sharings

In our protocols, we work with all three secret sharing schemes introduced in the previous section. For this we need methods to convert shares from one sharing scheme into shares of another one. This section reviews the known methods for such transformations and provides a method to transform additive shares over $\mathcal{Z}_Q$ into additive shares over the integers. The latter is apparently new. The section also provides a method to obtain shares of the bits of a shared secret.

4.1 Converting Between Integer Shares and $\mathcal{Z}_Q$ Shares

It is well known how to convert additive shares modulo Q into polynomial shares modulo Q and vice versa: If the players hold additive (or polynomial) shares of a value a they re-share those with a polynomial (additive) sharing and send the shares to the respective players, which add up (or interpolate) the received shares to obtain a polynomial (or additive) share of a. We denote the first transformation by $\mathsf{SQ2PQ}(\cdot)$ and the latter by $\mathsf{PQ2SQ}(\cdot)$.

Conversions between shares over the integers into shares over $\mathcal{Z}_Q$ naturally requires that $Q/2$ is bigger than the absolute shared value. If this is the case, an additive sharing $\langle c\rangle_1^I, \ldots, \langle c\rangle_k^I$ over the integers of a secret c with $-2^{n-1} < c < 2^{n-1} < Q/2$ can be converted in an additive sharing over $\mathcal{Z}_Q$ (and thus also a polynomial sharing) by reducing the shares modulo Q, i.e., $\langle c\rangle_i^Q := \langle c\rangle_i^I \operatorname{rem} Q$. We denote this transformation by $\mathsf{SI2SQ}(\cdot)$.

Obtaining additive shares over the integers from additive shares over $\mathcal{Z}_Q$ is not so straightforward. The main problem is that if one considers the additive shares over $\mathcal{Z}_Q$ as additive shares over the integers then one is off by an unknown multiple Q, the multiple being the quotient of the sum of these shares and Q. However, if the shared secret is sufficiently smaller than Q (i.e., ρ bits smaller, where ρ is a security parameter), then the players can reveal the high-order bits of their shares without revealing anything about the secret. Knowledge of these high-order bits is sufficient to compute the quotient. This observation leads to the following protocol.

Let $\langle c\rangle_j^Q \in \mathcal{Z}_Q$ be the share of party P_j and let $-2^{n-1} < c = \sum_i \langle c\rangle_i^Q \operatorname{rem} Q < 2^{n-1}$. If $Q > 2^{\rho+n+\lg k+4}$ holds, where ρ is a security parameter, the parties can use the following protocol to securely compute additive shares of c over the integers.

Protocol $\mathsf{SQ2SI}(\langle c\rangle_j^Q)$**:**
Let $t = \rho + n + 2$. Party P_j executes the following steps.

1.Reveal $a_j := \mathsf{trunc}(\frac{\langle c\rangle_j^Q}{2^t})$ to all other parties.

2.Compute $l := \left\lceil \frac{2^t \sum_i a_i}{Q} \right\rfloor$.

3.Run $\langle 0\rangle_j^I := \mathsf{JRIZ}([-Q2^\rho, Q2^\rho])$.

4.If $j \le |l|$ set the output to $\langle c\rangle_j^I := \langle c\rangle_j^Q - Q + \langle 0\rangle_j^I$ if $l > 0$ and to $\langle c\rangle_j^I := \langle c\rangle_j^Q + Q + \langle 0\rangle_j^I$ if $l < 0$.
If $j > |l|$ set the output to $\langle c\rangle_j^I = \langle c\rangle_j^Q + \langle 0\rangle_j^I$.

Theorem 1. *Let $\langle c\rangle_1^Q, \dots, \langle c\rangle_k^Q$ be a random additive sharing of $-2^{n-1} \le c < 2^{n-1}$. If $\lg Q > \rho + n + \lg k + 4$, where ρ is a security parameter, then the protocol $\mathsf{SQ2SI}(\langle c\rangle_j^Q)$ securely computes additive shares of c over the integers.*

Proof. We have to provide a simulator that interacts with the ideal world trusted party T and produces an output indistinguishable from that of the adversary. The trusted party T gets as input the shares $\langle c\rangle_1^Q, \dots, \langle c\rangle_k^Q$, computes c and re-shares c over the integers by choosing integer shares of 0 the same way as it would be done if the parties ran the protocol $\langle 0\rangle_i^I := \mathsf{JRIZ}([-Q2^\rho, Q2^\rho])$. Then T sets $\langle c\rangle_1^I := \langle 0\rangle_1^I + c$ and $\langle c\rangle_i^I := \langle 0\rangle_i^I$ for $i \ne 1$, and then sends $\langle c\rangle_i^I$ to player P_i. Note that the players' outputs are additive shares of c with the right distribution (i.e., the distribution of any subset of $k-1$ shares is statistically close to the distribution of the corresponding subset if another value c' was shared).

A simulator is as follows: it forwards the inputs $\langle c\rangle_i^Q$ of the corrupted players to T and obtains the shares $\langle c\rangle_i^I$ for these players from T. It extends the set of shares $\langle c\rangle_i^Q$ of the corrupted players into a full (and random) sharing of any valid c' (e.g., 0). Let $r_1, \dots, r_n$ be the thereby obtained shares. The simulator then computes $a_i = \mathsf{trunc}(\frac{r_i}{2^t})$ and lets the adversary know the a_i's that the corrupted players would receive in the protocol. Then the simulator computes $l = \left\lceil \frac{2^t \sum_i a_i}{Q} \right\rfloor$ and, for every i where Party P_i is corrupted, it sets

$$\langle 0\rangle_i^I := \begin{cases} \langle c\rangle_i^I - \langle c\rangle_i^Q + Q & \text{if } l > 0, i \le |l| \\ \langle c\rangle_i^I - \langle c\rangle_i^Q - Q & \text{if } l < 0, i \le |l| \\ \langle c\rangle_i^I - \langle c\rangle_i^Q & \text{otherwise.} \end{cases}$$

The simulator finally runs the simulator for $\mathsf{JRIZ}([-Q2^\rho, Q2^\rho])$ such that these shares $\langle 0\rangle_i^I$ are the outputs of the corrupted players. Finally the simulator stops outputting whatever the adversary outputs.

It remains to show that for this simulator the distributions of the players' and the simulators outputs are statistically indistinguishable from the views and outputs of the players and the adversary when running protocol $\mathsf{SQ2SI}(\langle c\rangle_j^Q)$.

Let us first prove that the players' outputs of protocol $\mathsf{SQ2SI}(\langle c\rangle_j^Q)$ are indeed shares of c. Let $\hat{l} = \left\lceil \frac{\sum_i \langle c\rangle_i^Q}{Q} \right\rfloor$. Thus $c = \sum_i \langle c\rangle_i^Q - \hat{l}Q$ fulfills $|c| < 2^{n-1}$ by assumption. Define $b_i = \langle c\rangle_i^Q - a_i 2^t$. Note that $|b_i| < 2^t$. We have to show that $l = \hat{l}$. As $\sum_i a_i 2^t = c + \hat{l}Q - \sum_i b_i$ we have $l = \left\lceil \frac{2^t \sum_i a_i}{Q} \right\rfloor = \left\lceil \frac{c}{Q} + \hat{l} - \frac{\sum_i b_i}{Q} \right\rfloor$. Because $\hat{l}$ is an integer, we have $l = \hat{l}$ if $|\frac{c}{Q}| < 1/4$ and $|\frac{\sum_i b_i}{Q}| < 1/4$, that is, if $n < \lg Q - 2$ and $2 + t + \lg k = \rho + n + \lg k + 4 < \lg Q$ holds. As $\langle c\rangle_i^Q \in \mathcal{Z}_Q$ we have $|l| < k$ and thus $c = \sum_i \langle c\rangle_i^Q - lQ = \sum_i \langle c\rangle_i^I$. Furthermore it is easy to see that the distribution of the shares output is statistically close to the ones produced by T.

Let us now show that the distribution of the a_i's for different shared values c are statistically indistinguishable. We consider the probability that the a_i's take different values if a different value of c was shared. W.l.o.g., we can assume that $\langle c\rangle_1^Q, \ldots, \langle c\rangle_{k-1}^Q$ are random elements from $\mathcal{Z}_Q$ and that $\langle c\rangle_k^Q = c - \sum_{i=1}^{k-1} \langle c\rangle_i^Q \operatorname{rem} Q$. Clearly, the values $a_1 = \mathsf{trunc}(\frac{\langle c\rangle_1^Q}{2^t}), \ldots, a_{k-1} = \mathsf{trunc}(\frac{\langle c\rangle_{k-1}^Q}{2^t})$ do not depend on the shared value. It remains to consider a_k. We have $\langle c\rangle_k^Q \operatorname{rem} Q = a_k 2^t + b_k$ with $b_k < 2^t$. First note that $C = -\sum_{i=1}^{k-1} \langle c\rangle_i \operatorname{rem} Q$ is uniformly distributed over $\mathcal{Z}_Q$ and that $Q > 2^t$. If $C > Q - 2^n$ or if $C \operatorname{rem} 2^t > 2^t - 2^n$ then a_k takes a value that depends on c. These conditions are fulfilled with probability at most $\frac{2^n+2^n}{2^t+2^n} < \frac{2^{n+1}}{2^t} = 2^{-t+n+1}$. Therefore, the statistical difference between the distribution of the a_i's for different shared values must smaller than $2 \cdot 2^{-t+n+1} = 2^{-t+n+2} = 2^{-\rho}$.

As the $\mathsf{JRIZ}([-Q2^\rho, Q2^\rho])$ protocol is secure, the distributions of the outputs in the real world and the outputs of the ideal world with our simulator are statistically indistinguishable. $\square$

Combining the above protocols, we can move from polynomial shares over $\mathcal{Z}_Q$ to additive shares over the integers and vice versa. The bit-complexities for these conversions are $O(nk^2 \lg k + n^2 k)$ and $O(nk^2 \lg k)$, respectively. For both, the communication-complexity is $O(kn)$ bits and the round-complexity is $O(1)$.

Moreover, it follows that we can also move from polynomial shares over $\mathcal{Z}_Q$ to polynomial shares over $\mathcal{Z}_{Q'}$ provided Q and Q' are sufficiently large w.r.t. the security parameter and the shared value.

4.2 Computing Shares of the Binary Representation of a Shared Secret

To do a distributed exponentiation with a shared exponent b it is useful when the players are given shares of the bits b. In the following we assume (w.l.o.g.) that the players hold additive shares of the exponent b over the integers. The idea of the following protocol to obtain shares of the bits is that each player distributes polynomial shares modulo $\widetilde{Q}$ of the bits of her or his additive share. Then the players perform a (general) multi-party computation to add these bits to obtain shares of the bits of b. This multi-party computation, however, is rather simple. In fact, we need to implement a circuit of size $O(kn)$ and depth $O(\lg k + \lg n)$ (c.f., [CLR92]). Each gate in this circuit requires $O(1)$ invocations of the multiplication protocol $\mathsf{MUL}(\cdot,\cdot)$ over $\mathcal{Z}_{\tilde{Q}}$, where $\tilde{Q}$ can be small.

Protocol $\mathsf{I2Q\text{-}BIT}(\langle b\rangle_j^I)$**:**
Let n to be (an upper-bound on) the number of bits of b. Party P_j runs the following steps.

1. Re-share each bit of the share $\langle b\rangle_j^I$ with a polynomial sharing over $\widetilde{Q}$ and send each share to the respective player. Let $[b_{i,l}]_j^{\widetilde{Q}}$ denote the share held by party P_j of the i-th bit of party P_l's additive share of b.

2. The player use the computation techniques of Ben-Or, Goldwasser and Wigderson [BGW88] on a circuit for adding the k n-bit numbers. This takes $O(\lg k + \lg n)$ steps.

   Let $[u_i]_j^{\widetilde{Q}}$, $i = 1, \ldots, n$ be the shares of the bits of the result. (Recall that it is ensured b has n-bits.)

3. For $i = 1, \ldots, n-1$ do (in parallel)

   (a) Execute $[0]_j^{\widetilde{Q}} := \mathsf{JRPZ}(\mathcal{Z}_{\widetilde{Q}})$ and set $[b_i]_j^{\widetilde{Q}} := [0]_j^{\widetilde{Q}} + [u_i]_j^{\widetilde{Q}} \operatorname{rem} \widetilde{Q}$.

4.Output $([b_1]_j^{\widetilde{Q}}, \ldots, [b_n]_j^{\widetilde{Q}})$.

Proving the security of this protocol is straightforward given the security of its sub-protocols and the composition theorem.

Efficiency analysis: computing shares of the bits of b requires $O(nk^3 \lg k \lg \widetilde{Q} + nk^2 (\lg \widetilde{Q})^2)$ bit operations per player. This protocol requires only a relatively small $\widetilde{Q}$, e.g., $\rho + 5 + \lg k$ bits. If shares of the bits modulo a larger prime Q are required, is more efficient to compute shares modulo a small $\widetilde{Q}$ using the above protocol and then convert these shares into ones modulo Q. The number of bit operations for this is $O(\gamma nk^3 \lg k + \gamma^2 nk^2 + n^2k^2 \lg k)$, where $\lg Q = \Theta(n)$ and $\lg \widetilde{Q} = \Theta(\gamma)$, as opposed to $O(n^2k^3 \lg k + n^3k^2)$ when using the bigger Q only. This optimization may be quite important in practice as γ may be much smaller than n (e.g., $\gamma = 100$ and $n = 2000$). The communication-complexity for both variants is $O(n^2k + nk \lg Q)$ bits. and their round-complexity is $O(\lg k + \lg n)$.

4.3 Approximate Truncation

This paragraph presents a truncation protocol, that on input polynomial shares of a and a parameter n outputs polynomial shares of b such that $|b - a/2^n| \leq k + 1$.

Protocol $\mathsf{TRUNC}(a, n)$ **:**
Party P_j executes the following steps.

1.Get additive shares of a over the integers: $\langle a \rangle_j^I := \mathsf{SQ2SI}(\mathsf{PQ2SQ}([a]_j^Q))$.

2.Locally compute $\langle b \rangle_j^I := \mathsf{trunc}(\frac{\langle a \rangle_j^I}{2^n})$

3.Get polynomial shares of b over $\mathcal{Z}_Q$: $[b]_j^Q := \mathsf{SQ2PQ}(\mathsf{SI2SQ}(\langle b \rangle_j^I))$

It is easy to see that the protocol is secure and correct, if $\lg Q > \rho + n + \lg k + 4$ holds, where ρ is a security parameter (c.f. requirements of the $\mathsf{SQ2SI}(\cdot)$ protocol). Its bit-complexity is $O(nk^2 \lg k + n^2 k)$, its communication-complexity is $O(kn)$ bits, and its round-complexity is $O(1)$ rounds.

5 Distributed Computation Modulo a Shared Integer

This section provides efficient protocols for distributed computation modulo a shared, secret modulus p. All computations will be done using shares modulo a prime Q whose bit-length is roughly twice that of p. The main building block is an efficient protocol for reducing a shared secret modulo p. This immediately gives us distributed modular addition and multiplication. The section further provides a protocol for efficient modular exponentiation where the exponent is a shared secret as well. As our modular reduction protocol does not compute the smallest residue in absolute value but only one that is bounded by a small multiple of the modulus, the usual approach for comparing two shared secrets no longer works and therefore a new protocol for comparing such 'almost reduced' shared secrets modulo p is also presented.

The idea of our protocol for modular reduction is based on classical algorithmic techniques (c.f. [AHU74]). Recall that $c \operatorname{rem} p = c - \lceil \frac{c}{p} \rfloor p$. Thus the problem reduces to the problem of distributively computing $\lceil \frac{c}{p} \rfloor$.

By interpreting an integer as the mantissa of a floating point number with a public exponent, we can interpret shares of this integer as shares of the corresponding floating point number. To multiply two such floating point numbers we distributively multiply the mantissas and locally add the exponents. To keep the shared numbers small, we 'round' the product by converting the polynomial shares of the product mantissa modulo Q to additive shares over the integers, by having each party locally right-shift its additive share by ξ bits and add ξ to the exponent, and by converting back to polynomial shares modulo Q. This rounding technique introduces an relative error of $O(k2^{\xi}/m)$.

So we split the problem of distributively computing $\lceil \frac{c}{p} \rfloor$ into the problem of distributively computing a floating point approximation of $1/p$, and of distributively computing $\lceil \frac{c}{p} \rfloor$ using the precomputed shares of $1/p$. The first problem can be solved using Newton iteration and is described in the next subsection. In Section 5.2 we show how to compute a close approximation to $\lceil \frac{c}{p} \rfloor$ if we are given additive shares of a good approximation to $\frac{c}{p}$ over the integers by having each participant locally truncate its share. The resulting (shared) integer s satisfies $|s - \lceil \frac{c}{p} \rfloor| \leq k+1$. It turns out that this is accurate enough to compute a value congruent to c modulo p that is sufficiently small to allow for on-going computations modulo p (Section 5.3).

5.1 Computation of Shares of an Approximation to $1/p$

Assume each party is given polynomial shares $[p]_i^Q$ of p, with $2^{n-1} < p < 2^n$. This section provides a protocol that allows the parties to compute polynomial shares of an integer $0 < \tilde{p} < 2^{t+2}$ such $\tilde{p}\, 2^{-n-t} = 1/p + \epsilon$ where $|\epsilon| < (k+1)2^{-n-t+4}$.

As already mentioned we employ Newton iteration for this task with the function $f(x) = 1/x - p/2^n$ which leads to the iteration formula $x_{i+1} := x_i(2 - x_i p/2^n)$ that has quadratic convergence. Using $3/2$ as a start value gives us an initial error of $|2^n/p - 3/2| < 1/2$ and hence we need to do about $\lg t$ iterations to get a t-bit approximation $\tilde{x}$ to $2^n/p$. We set $\tilde{p} = 2^t \tilde{x}$, which is an integer.

Protocol $\mathsf{APPINV}([p]_j^Q)$ **:**
Party P_j executes the following steps.

1. Set $[u_0]_j^Q := u_0 = 3 \cdot 2^{t-1} \,\mathrm{rem}\, Q$.
2. For $i = 0$ to $\lceil \lg(t - 3 - \lg(k+1)) \rceil - 1$ run
   (a) Distributively compute $[z_{i+1}]_j^Q := \mathsf{MUL}([p]_j^Q, [u_i]_j^Q)$.
   (b) $[w_{i+1}]_j^Q := \mathsf{TRUNC}([z_{i+1}]_j^Q, n)$.
   (c) Compute $[v_{i+1}]_j^Q := 2^{t+1} \cdot [u_i]_j^Q - \mathsf{MUL}([w_{i+1}]_j^Q, [u_i]_j^Q)$.
   (d) $[u_{i+1}]_j^Q := \mathsf{TRUNC}([v_{i+1}]_j^Q, t)$.
3. Run $[0]_j^Q := \mathsf{JRPZ}(\mathcal{Z}_Q)$.
4. Output $[\tilde{p}]_j^Q := [u_{i+1}]_j^Q + [0]_j^Q \,\mathrm{rem}\, Q$.

Theorem 2. *Let ρ be a security parameter and let $Q > 2^{\rho+t+\nu+6+\lg k}$, where $\nu = \max(n,t)$. Then, for any $t > 5 + \lg(k+1)$ and any p satisfying $2^{n-1} < p < 2^n$ for some n, the protocol* $\mathsf{APPINV}([p]_j^Q)$ *securely computes shares of an integer $\tilde{p}$, such that*

$$\left| \frac{2^n}{p} - \frac{\tilde{p}}{2^t} \right| < \frac{k+1}{2^{t-4}} \ ,$$

with $0 < \tilde{p} < 2^{t+2}$. That is, $\tilde{p}/2^{t+n}$ is an approximation to $1/p$ with relative error $\frac{k+1}{2^{t-4}}$.

Proof. We need show that the protocol actually computes an approximation to $1/p$. Then security from the security of the sub-protocols for multiplication and transformation of the shares.

Consider how u_{i+1} is computed from u_i in the protocol. Because of the local truncation, we have $2u_i - pu_i^2 2^{-n-t} - (k+1)(1+u_i/2^t) \leq u_{i+1} \leq 2u_i - pu_i^2 2^{-n-t} + (k+1)(1+u_i/2^t)$. As we will see $u_i/2^t < 3$ holds. Thus $|\frac{2^n}{p} - \frac{u_{i+1}}{2^t}| < \frac{2^n}{p} - 2\frac{u_i}{2^t} + \frac{p}{2^n}(\frac{u_i}{2^t})^2 + \frac{(k+1)}{2^t}(1 + u_i/2^t) = \frac{p}{2^n}(\frac{2^n}{p} - \frac{u_i}{2^t})^2 + \frac{(k+1)}{2^t}(1 + u_i/2^t)$. From this it follows that $\left|\frac{2^n}{p} - \frac{u_{i+1}}{2^t}\right| < \epsilon_i^2 + \frac{k+1}{2^{t-2}} =: \epsilon_{i+1}$. As $2^{n-1} < p < 2^n$ and $u_0 = 2^{t-1}$ we have $\epsilon_0 < 1/2$ and by requiring $k < 2^{t-5} - 1$ we get $e_1 < 1/2$ and $\epsilon_i = 2^{2^{-i}} + \frac{k+1}{2^{t-3}} < 1/2$. In particular, we have $\epsilon_i = \frac{k+1}{2^{t-4}}$ for $i = \lceil \lg(t - 3 - \lg(k+1)) \rceil$.

Consider the size of the integers u_i that are shared during the protocol. As $\epsilon_i < 1/2$ and $1 < 2^n/p < 2$ we have $0 < u_i/2^t < 2 + 1/2$ and hence $0 < u_i < 2^{t+2}$ for all i and hence $0 < z_i < 2^{n+t+2}$. Similarly, one can show that $0 < v_i < 2^{2t+2}$.

The lower-bound on Q follows from the fact that the $\mathsf{SQ2SI}(\cdot)$ algorithm must work on the v_i's and the z_i's. ☐

Let us discuss the choice of t: in order for the b most significant bits of $1/p$ and $\tilde{p}/2^{t+n}$ to be equal, t must be chosen bigger than $b + 5 + \lg(k+1)$. The cost of the protocol is dominated by the $\mathsf{MUL}(\cdot,\cdot)$ protocol and is $O(\lg t(n^2k + nk^2 \lg k))$ bit-operations per player. Its communication-complexity $O(kn \lg t)$ bits and its round-complexity is $O(\lg t)$.

5.2 Reduction of a Shared Integer Modulo a Shared p

Assume the players hold polynomial shares modulo Q of the three integers $-2^w < c < 2^w$, $0 < \tilde{p} < 2^{t+2}$, and $2^{n-1} < p < 2^n$, where $\tilde{p}\,2^{-n-t}$ is an approximation of $1/p$ as computed by the protocol in the previous paragraph. Using the following protocol, the players can compute shares of an integer d such that $d \equiv c \pmod{p}$ and $\lg |d| < \lg(k+1) + w - t + 5$.

As already mentioned this protocol computes d as $c - \lceil c\tilde{p}2^{-n-t} \rfloor p$. For distributively computing the product $c\tilde{p}$ the size of Q would need to be about $w + t$ bits. However, as the $\ell \approx n$ least significant bits of c do not matter in the computation of the quotient, we can first cut these ℓ bits off, obtaining $\tilde{c}$, and then compute d as $c - \lceil \tilde{c}\tilde{p}2^{-n-t+\ell} \rfloor p$ which requires the size of Q to be only about $w + t - \ell$ bits.

Protocol $\mathsf{MOD}([c]_j^Q, [p]_j^Q, [\tilde{p}]_j^Q)$**:**
Player P_j executes the following steps.

1. $[\tilde{c}]_j^Q := \mathsf{TRUNC}([c]_j^Q, \ell)$.
2. Compute $[\hat{q}]_j^Q := \mathsf{MUL}([\tilde{c}]_j^Q, [\tilde{p}]_j^Q)$.
3. $[q]_j^Q := \mathsf{TRUNC}([\hat{q}]_j^Q, n + t - \ell)$.
4. Compute $[d]_j^Q := [c]_j^Q - \mathsf{MUL}([p]_j^Q, [q]_j^Q)$.

Theorem 3. *Given shares of three integers $-2^w < c < 2^w$, $0 < \tilde{p} < 2^{t+2}$, and $0 < p < 2^n$, the above protocol securely computes shares of $d = (c \operatorname{rem} p) + ip$ with $|i| \leq (k+1)(1 + 2^{w+4-n-t} + 2^{\ell-n+2})$, where k is the number of players and given that $Q > \max(2^{\rho+6+w-\ell+t+2\lg(k+1)}, 2^{\rho+w+4+\lg(k+1)})$.*

Proof. Due to the local rounding in the $\mathsf{TRUNC}(\cdot,\cdot)$ protocol in Step 1, we have $c - (k+1)2^\ell \leq \tilde{c}2^\ell \leq c + (k+1)2^\ell$. Due to the local rounding in the $\mathsf{TRUNC}(\cdot,\cdot)$ protocol in Step 3, we have $\mathsf{trunc}(\tilde{c}\tilde{p}2^{-n-t+\ell}) - k \leq q \leq \mathsf{trunc}(\tilde{c}\tilde{p}2^{-n-t+\ell}) + k$. As $\tilde{p}2^{-(n+t)}$ is only an approximation to $1/p$, we have $\mathsf{trunc}(\frac{c}{p} - \frac{c(k+1)}{2^{n-4+t}} - \frac{\tilde{p}(k+1)}{2^{n+t-\ell}}) - k \le q \le \mathsf{trunc}(\frac{c}{p} + \frac{c(k+1)}{2^{n-4+t}} + \frac{\tilde{p}(k+1)}{2^{n+t-\ell}}) + k$ and, as $-2^w < c < 2^w$ and $0 < \tilde{p} < 2^{t+2}$, we get $\lceil \frac{c}{p} \rfloor - (k+1)(1+2^{w+4-n-t}+2^{\ell-n+2}) \le q \le \lceil \frac{c}{p} \rfloor + (k+1)(1+2^{w+4-n-t}+2^{\ell-n+2})$. Thus $d = (c \operatorname{rem} p) + ip$ with $|i| < (k+1)(1+2^{w+4-n-t}+2^{\ell-n+2})$.

The bound on Q follows from the requirements of the $\mathsf{SQ2SI}(\cdot)$ in the $\mathsf{TRUNC}(\cdot,\cdot)$ protocol. $\square$

The cost of the $\mathsf{MOD}(\cdot,\cdot,\cdot)$ protocol is dominated by the $\mathsf{MUL}(\cdot,\cdot)$ protocol and is $O(n^2k + nk^2 \lg k)$ bit operations per players. The communication-complexity of the protocol is $O(kn)$ bits and its round-complexity is $O(1)$.

5.3 Computing with a Shared Modulus p

Now, we are ready to discuss "on-going" distributed computation modulo a shared integer. In particular, we discuss how the parameters for the $\mathsf{MOD}(\cdot,\cdot,\cdot)$ and $\mathsf{APPINV}(\cdot)\cdot$ protocols must be set such that such computation is possible. Assume that the players hold polynomial shares modulo a prime Q of the integers $0 < \tilde{p} < 2^{t+2}$, and $2^{n-1} < p < 2^n$, where $\tilde{p}\, 2^{-t-n}$ is an approximation of $1/p$ as computed above. Let

$$t = \lceil n + 10 + 2\lg(3(k+1)) \rceil, v = n + \lg(3(k+1)) + 1, \text{ and } Q > 2^{\rho+2n+36+6\lg(k+1)}.$$

Then, given polynomial shares modulo a prime Q of an integer $-2^{2v} < c < 2^{2v}$, the players can compute shares of an integer $-2^v < d < 2^v$ as $[d]_j^Q := \mathsf{MOD}([c]_j^Q, [p]_j^Q, [\tilde{p}]_j^Q)$. In particular, given polynomial shares modulo a prime Q of the integers $-2^v < a, b < 2^v$ the players can compute shares of an integer $-2^v < d' < 2^v$ as $[d']_j^Q := \mathsf{MOD}(\mathsf{MUL}([a]_j^Q, [b]_j^Q), [p]_j^Q, [\tilde{p}]_j^Q)$. Thus d and d' can be used as inputs to further modular multiplication computations.

Exponentiation with a Shared Exponent: Assume the players want to compute shares of $c \equiv a^b \pmod p$, where a, b, p, $\tilde{p}$ are shared secrets and $\tilde{p}$ is an approximation to $2^{n+t}/p$. This can be done by distributively running the square and multiply algorithm where the fact that $a^{b_i} = (a-1)b_i + 1$ if $b_i \in \{0,1\}$ comes in handy. We assume that the players hold shares $([b_1]_j^Q, \ldots, [b_n]_j^Q)$ of the bits of b, where b_1 is the low-order bit of b (as computed, say, by protocol $\mathsf{I2Q\text{-}BIT}(\cdot)$).

Assuming that $|a| < 2^v$ then the following protocol securely computes shares of c such $|c| < 2^v$ and $c \equiv a^b \pmod p$.

Protocol $\mathsf{EXPMOD}([a]_j^Q, ([b_1]_j^Q, \ldots, [b_n]_j^Q), [p]_j^Q, [\tilde{p}]_j^Q)$**:**
Player P_j executes the following steps.

1. Compute $[c_n]_j^Q := \mathsf{MUL}([a]_j^Q - 1 \operatorname{rem} Q, [b_n]_j^Q) + 1 \operatorname{rem} Q$.
2. For $i = n-1, \ldots, 1$ do
   (a) $[d_i]_j^Q := \mathsf{MUL}([a]_j^Q - 1 \operatorname{rem} Q, [b_i]_j^Q) + 1 \operatorname{rem} Q$.
   (b) $[c_i]_j^Q := \mathsf{MOD}(\mathsf{MUL}(\mathsf{MOD}(\mathsf{MUL}([c_{i+1}]_j^Q, [c_{i+1}]_j^Q), [p]_j^Q, [\tilde{p}]_j^Q), [d_i]_j^Q), [p]_j^Q, [\tilde{p}]_j^Q)$.
3. Output $[c]_j^Q := [c_1]_j^Q$.

Efficiency analysis: This protocol does about $3n$ invocations of $\mathsf{MUL}(\cdot,\cdot)$ and about $2n$ of $\mathsf{MOD}(\cdot,\cdot,\cdot)$ and hence requires $O(n^3k + n^2k^2 \lg k))$ bit operations per player. The communication complexity $O(n^2k)$ bits and it has $O(n)$ rounds.

Set membership: Assume the players want to establish whether $a \equiv b \pmod{p}$ holds for three shared secrets a, b and p (where p is not necessarily a prime). This can in principle be done by computing shares of $c := a - b \operatorname{rem} p$, (re-)sharing c modulo Q, multiplying it with a jointly generated random invertible element from $\mathcal{Z}_Q$, revealing the result, and see if it is 0 modulo Q (provided $Q > p$). However, because of the properties of $\mathsf{MOD}(\cdot,\cdot,\cdot)$, we can only compute shares of $c = (a - b \operatorname{rem} p) + ip$ with $|i| < 3(k+1)$ and therefore the test does not quite work. But as i is relatively small, it is possible to distributively compute the integer $s := \prod_{l=-3(k+1)+1}^{3(k+1)-1} (c - lp)$ which will be zero if $c \equiv 0 \pmod{p}$ and non-zero otherwise. This also holds for s modulo Q because $Q \nmid s$ if $Q > p6(k+1)$ as then $Q > |(c - ip)|$ holds for all $i \in [-3(k+1), 3(k+1)]$.

The protocol below is a generalization of what we just described in that it allows the players to check whether a equals one of $b_1, \ldots b_m$ modulo p. Here, first an s_i is computed for each b_i similarly as the s above for b and then it is tested whether $\prod_i s_i \equiv 0 \pmod{Q}$.

Assuming that $a, b_1, \ldots, b_m$ are less than 2^v in absolute value, then the following protocol securely tests if $a \equiv b_i \pmod{p}$ for some i.

Protocol $\mathsf{SETMEM}([a]_j^Q, \{[b_1]_j^Q, \ldots, [b_m]_j^Q\}, [p]_j^Q, [\tilde{p}]_j^Q)$**:**
Player P_j runs the following steps.

1. For all $i = 1, \ldots, m$ compute $[c_i]_j^Q := \mathsf{MOD}([a]_j^Q - [b_i]_j^Q \operatorname{rem} Q, [p]_j^Q, [\tilde{p}]_j^Q)$ (in parallel).

2. For all $i = 1, \ldots, m$ do (in parallel)

   (a) Set $[u_{(i,-3(k+1)+1)}]_j^Q := [c_i]_j^Q - (3(k+1) - 1)[p]_j^Q \operatorname{rem} Q$.

   (b) For $l = -3(k+1) + 2, \ldots, 3(k+1) - 1$ do

      i. Compute $[u_{(i,l)}]_j^Q := \mathsf{MUL}([u_{(i,l-1)}]_j^Q, ([c_i]_j^Q - l[p]_j^Q \operatorname{rem} Q))$.

3. Let $[\tilde{u}_1]_j^Q := [u_{(1,3(k+1)+1)}]_j^Q$.

4. For $i = 2, \ldots, m$ do

   (a) Compute $[\tilde{u}_i]_j^Q := \mathsf{MUL}([\tilde{u}_{i-1}]_j^Q, [u_{(i,3(k+1)+1)}]_j^Q)$.

5. Perform $[r]_j^Q := \mathsf{JRP\text{-}INV}(\mathcal{Z}_Q)$, compute $[z]_j^Q := \mathsf{MUL}([\tilde{u}_m]_j^Q, [r]_j^Q)$ and send $[z]_j^Q$ to all other players.

6. Reconstruct z and output $\mathsf{success}$ if $z \equiv 0 \operatorname{rem} Q$ and $\mathsf{failure}$ otherwise.

Security of this protocol follows from the security of its sub-protocols, and the fact that if z is non-zero, then it is a random element from $\mathcal{Z}_Q$ and hence no information about a or any of the b_i's is revealed other than that a is different from all the b_i's modulo p.

Note that this protocol includes as a special case the comparison of two almost reduced residues. It requires $O(mk(n^2k + nk^2 \lg k))$ bit operations per player. The communication-complexity $O(mnk^2)$ bits and it takes is $O(k+n)$ rounds. However, it is trivial to get the number of rounds down to $O(\lg k + \lg n)$ by using a "tree multiplication method" in step 2b and 4.

We note that an alternative to the above protocol would be to use the techniques of Ben-Or et al. [BGW88] on a circuit to fully reduce a and b modulo p. As a and b are "almost reduced" modulo p, this circuit is small.

6 Generation of Shared Random Primes and Safe Primes

In this section we discuss how to use the protocols introduced so far to generate a shared random prime and a random safe prime. Once we know how to do this, we can of course also generate a shared RSA modulus being the product of two primes or of two safe primes. As mentioned earlier, the former protocol may be more efficient than the one of Boneh and Franklin, at least for very large n, and the latter is far more efficient than any previously known protocol for this problem. We conclude the section with an efficiency discussion and a comparison of our protocols and the one by Boneh and Franklin for generating a shared prime product.

Our strategy for generating a random shared prime is the same as the one usually applied in the non-distributed case: choose a random number, do trial division, and then run sufficiently many rounds of some primality test, e.g., the Miller-Rabin test. In the following we describe how each of these steps can be distributed.

6.1 Generating a Shared Candidate p

The first task for the player is to generate a random n-bit number. In principle, this could be done be having each player choose a random n-bit number and then compute shares of the XOR of those strings in a similar way as in the protocol we described in Section 4.2. However, this would mean to already invest significant computation on candidates that with high probability fail the trial division step. A more efficient way to generate the candidates due to Boneh and Franklin [BF97] is as follows. Every party except the first one chooses a random $(n - \lg k - 1)$-bit number $p_i \equiv 0 \pmod 4$; the first one chooses a $(n - \lg k - 1)$-bit number $\tilde{p}_1 \equiv 3 \pmod 4$ and sets $p_1 := 2^{n-1} + \tilde{p}_1$. Thus $p := \sum_i p_i$ will be a n-bit number and $\langle p \rangle_i^I := p_i$. Of course, the distribution of p is not uniform but one can show that the distribution of p has at least $(n - \lg k - 1)$-bits of entropy [BF97]. By restricting $p \equiv 3 \pmod 4$, we loose only about half the primes. This will be sufficient for most applications (otherwise one could still resort to the computationally more involved method sketched before).

We note that the restriction of $p \equiv 3 \pmod 4$ could be dropped when resorting to the Solovay-Strassen test. This, however, requires a protocol to compute shares of the Jacobi symbol of a shared secret; such a protocol is provided in the full version of this paper.

6.2 Trial Division on p

Before doing the costly primality check the players can do a cheaper trial division. For all primes e smaller than some bound B, the players do the following steps (in parallel):

Protocol Trial Division:
Player P_j runs the following steps.

1. Re-share $\langle p \rangle_j^I \operatorname{rem} e$ as polynomial shares over $\mathcal{Z}_e$ and send each share to the respective player.
2. Sum up the shares gotten from the other players and obtain the share $[p \operatorname{rem} e]_j^e$.
3. Run $[r]_j^e := \mathsf{JRP\text{-}INV}(\mathcal{Z}_e)$, then $[z]_j^e := \mathsf{MUL}([r]_j^e, [p \operatorname{rem} e]_j^e)$ and reveal $[z]_j^e$ to all other players.
4. Reconstruct z. If $z \equiv 0 \operatorname{rem} e$ then e divides p.

Note that the above protocol does not work for $e \leq k$, because in such cases the field $\mathbb{F}_e$ does not contain enough points to do Shamir sharing among k players. To overcome this, the player can resort to an extension field of $\mathbb{F}_e$ (c.f. [BF97]). Also note that our proposal for trial division determines exactly whether e divides p or not whereas the proposal by Boneh and Franklin [BF97] has some probability of error which weakens the effect of the trial division somewhat. This trial division costs $O((B/\lg B)(k^2 \lg B + k(\lg B)^2))$ bit-operations and the computation complexity is $O(1)$ rounds and $O(Bk)$ bits..

6.3 Distributed Miller-Rabin Test

As $p \equiv 3 \pmod 4$, the Miller-Rabin test reduces to choosing a random base g from $\mathcal{Z}_p$ and testing whether $g^{\frac{p-1}{2}} \equiv \pm 1 \,\mathrm{rem}\, p$. The following protocol implements this test for a shared secret p. One difficulty here is that the players cannot choose the base randomly from $\mathcal{Z}_p$ directly as p is not known: They have to choose an integer g from an interval that is sufficiently larger than p (e.g., from $\{0,1\}^{2n}$), such that $g \mod p$ will be distributed statistically close to the original distribution.

Protocol Miller-Rabin:
Player P_j runs the following step

1. If $2 \le j \le k$ locally compute $\langle b \rangle_j^I := \langle p \rangle_j^I / 2$. If $j = 1$ locally compute $\langle b \rangle_1^I := (\langle p \rangle_1^I - 1)/2$.
2. Run $([b_1]_j^Q, \ldots, [b_n]_j^Q) := \mathsf{I2Q\text{-}BIT}(\langle b \rangle_j^I)$.
3. Compute $[p]_j^Q := \mathsf{SQ2PQ}(\mathsf{SI2SQ}(\langle p \rangle_j^I))$ and $[\tilde{p}]_j^Q := \mathsf{APPINV}([p]_j^Q)$.
4. Repeat the following step ζ times (in parallel).
   (a) Choose $\langle r \rangle_j^I \in_R \{0,1\}^{2n}$.
   (b) Run $[r]_j^Q := \mathsf{SQ2PQ}(\mathsf{SI2SQ}(\langle r \rangle_j^I))$ and $[g]_j^Q := \mathsf{MOD}([r]_j^Q, [p]_j^Q, [\tilde{p}]_j^Q)$.
   (c) Run $[u]_j^Q := \mathsf{EXPMOD}([g]_j^Q, ([b_1]_j^Q, \ldots, [b_n]_j^Q), [p]_j^Q, [\tilde{p}]_j^Q)$.
   (d) It the result of $\mathsf{SETMEM}([u]_j^Q, \{-1, 1\}, [p]_j^Q, [\tilde{p}]_j^Q)$ is failure then stop and output failure.
5. Output success.

If p is a prime then the parties declare success. Otherwise, they declare that p is a composite with probability at least $1/2$ (over the random choices of g).

Note that in the implementation of $\mathsf{I2Q\text{-}BIT}(\cdot)$ we work with a prime $\widetilde{Q}$ whose bit-length is $\gamma = O(\rho + \lg k)$, where ρ is the security parameter (c.f. Section 4.2). So, the cost of one Miller-Rabin test is $O(nk^3 \lg k\gamma + nk^2\gamma^2 + n^2k^2 \lg k + \zeta(n^3k + n^2k^2 \lg k))$ bit-operation and the communication-complexity $O(n^2k\zeta)$ bits and it takes $O(n + \lg k)$ rounds.

6.4 Generation of a Shared Safe Prime

In this section we recommend a protocol for efficient generation of a safe prime, $p = 2p' + 1$ with p and p' prime. It follows the single party protocol proposed by Cramer and Shoup [CS00].

1. The players generate a random number p' as in Section 6.1.
2. If $j = 1$ compute $\langle p \rangle_j^I := 2\langle p' \rangle_j^I + 1$. If $j \neq 1$ compute $\langle p \rangle_j^I := 2\langle p' \rangle_j^I$.
3. Run the trial division as described in Section 6.2 on p and p'. If either of them appears to be divisible by a small prime, go to step 1.

4. Run the Miller-Rabin test (Section 6.3) on p' with $\zeta = 1$ and $g = 2$. If it fails, go to step 1.

5. Run the Miller-Rabin test (Section 6.3) on p with $\zeta = 1$ and $g = 2$. If it fails, go to step 1.

6. Run the Miller-Rabin test (Section 6.3) on p' with random g and sufficiently large ζ to ensure a small error probability (e.g., 2^{-80}).

As the candidates p' are not random $(n-1)$-bit numbers, some care must be taken in choosing the parameter ζ in step 6. We do not address these details here. Assuming $\lg k \ll n$, $B = O(n)$, and that safe primes are sufficiently dense (as is widely conjectured and supported by empirical evidence), the expected bit-complexity of this protocol is $O(n^3/(\lg n)^2(k^3 \lg k\gamma + k^2\gamma^2 + nk^2 \lg k + n^2 k))$, where $\gamma \approx 128$ is a security parameter smaller than n. Assuming that one tests about $n^2/(\lg n)^2$ candidates in parallel, the round-complexity is $O(n)$, the communication-complexity and $O(n^4/(\lg n)^2 k)$ bits.

6.5 Generation of RSA Moduli, Efficiency Analysis and Comparison

It should now be clear how to generate a modulus N being a prime or a safe prime product. Many applications require also that the players generate shares of the private exponent. This is much less computationally involved than distributively generating the modulus N. In particular, Boneh and Franklin [BF97] as well as Catalano et al. [CGH00] present efficient protocols to accomplish this, given additive shares over the integers of the factors of N. Our techniques can in fact be used to improve the latter protocol as well.

Let us compare the computational cost of our method of generating a shared prime product to the one by Boneh and Franklin. (We do not consider the improvement on the latter protocol described by Malkin, Wu, and Boneh [WB99], as most of them apply to our protocol as well.) We first summarize the latter approach. Boneh and Franklin propose to first choose random n-bit strings and to do a distributed trial division of them. When two strings are found that pass this trial division, they are multiplied to obtain N. Then, local trial division is done on N, and finally a special primality test on N is applied that checks whether N is the product of two primes. Thus, from a bird's eyes view, one finds that with this method, one has the test about $(n/\lg n)^2$ candidates as opposed to about $n/\lg n$ with our method.

A more careful analysis assuming $\lg k \ll n$ shows that the expected bit-complexity of their protocol is $O((n/\lg n)^2(n^3+n^2k+nk^2 \lg k)$ whereas it is $O(n^2/\lg n(k^3 \lg k\gamma + k^2\gamma^2 + nk^2 \lg k + n^2 k))$ for ours, where $\gamma \approx 128$ is a security parameter smaller than n. For this analysis we assumed that the bound B for trial division is about $O(n)$. For small number of players k these figures become $O(n^5/(\lg n)^2)$ and $O(n^4/\lg n)$. Round and communication complexities are $O(1)$ rounds and $O(kn^3/(\lg n)^2)$ bits for theirs and $O(n)$ rounds and $O(kn^3/\lg n)$ bits for ours. We note that, in practice, the round-complexities and communication complexities are not relevant as for this kind of application one would run many instances of the protocol in parallel and thereby keep the party with the least computational power constantly busy.

Acknowledgements

We are grateful to Matt Franklin for enlightening discussions that led to a substantially more efficient test for safe-prime products.

References

[ACJT00] Giuseppe Ateniese, Jan Camenisch, Marc Joye, and Gene Tsudik. A practical and provably secure coalition-resistant group signature scheme. In Mihir Bellare, editor, *Advances in Cryptology — CRYPTO 2000*, volume 1880 of *LNCS*, pages 255–270. Springer Verlag, 2000.

[AHU74] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analyis of Computer Algorithms.* Addision Wesley, 1974.

[BB89] Judit Bar-Ilan and Donald Beaver. Non-cryptographic fault-tolerant computing in a constant number of rounds of interaction. In *8th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, pages 201–209, 1989.

[BF97] Dan Boneh and Matthew Franklin. Efficient generation of shared RSA keys. In Burt Kaliski, editor, *Advances in Cryptology — CRYPTO '97*, volume 1296 of *LNCS*, pages 425–439. Springer Verlag, 1997.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proc. 20th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1–10, 1988.

[Can00] Ran Canetti. Security and composition of multi-party cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[CGH00] Dario Catalano, Rosario Gennaro, and Shai Halevi. Computing inverses over a shared secret modulus. In Bart Preneel, editor, *Advances in Cryptology — EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 190–206. Springer Verlag, 2000.

[CL01] Jan Camenisch and Anna Lysyanskaya. Efficient non-transferable anonymous multi-show credential system with optional anonymity revocation. In Birgit Pfitzmann, editor, *Advances in Cryptology — EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 93–118. Springer Verlag, 2001.

[CLR92] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms.* MIT Press, Cambridge, 1992.

[CS99] Ronald Cramer and Victor Shoup. Signature schemes based on the strong RSA assumption. In *Proc. 6th ACM Conference on Computer and Communications Security*, pages 46–52. ACM press, nov 1999.

[CS00] Ronald Cramer and Victor Shoup. Signature schemes based on the strong RSA assumption. *ACM Transactions on Information and System Security*, 3(3):161–185, 2000.

[DK01] Ivan Damgård and Maciej Koprowski. Practical threshold rsa signatures without a trusted dealer. In Birgit Pfitzmann, editor, *Advances in Cryptology — EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 152–165. Springer Verlag, 2001.

[FFS88] Uriel Feige, Amos Fiat, and Adi Shamir. Zero-knowledge proofs of identity. *Journal of Cryptology*, 1:77–94, 1988.

[FH94] Matthew Franklin and Stuart Haber. Joint encryption and message-efficient secure computation. In Douglas R. Stinson, editor, *Advances in Cryptology — CRYPTO '93*, volume 773 of *LNCS*, pages 266–277. Springer, 1994.

[FMY87] Yair Frankel, Phil MacKenzie, and Moti Yung. Robust efficient distributed rsa key generation. In *Proc. 30th Annual ACM Symposium on Theory of Computing (STOC)*, pages 663–672, 1987.

[FS01] Pierre-Alain Fouque and Jacques Stern. Fully distributed threshold rsa under standard assumptions. In Colin Boyd, editor, *Advances in Cryptology — ASIACRYPT 2001*, volume ??? of *LNCS*, pages ?–? Springer Verlag, 2001.

[GHR99] Rosario Gennaro, Shai Halevi, and Tal Rabin. Secure hash-and-sign signatures without the random oracle. In Jacques Stern, editor, *Advances in Cryptology — EUROCRYPT '99*, volume 1592 of *LNCS*, pages 123–139. Springer Verlag, 1999.

[GJKR96] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Robust and efficient sharing of RSA functions. In Neal Koblitz, editor, *Advances in Cryptology — CRYPT0 '96*, volume 1109 of *LNCS*, pages 157–172, Berlin, 1996. IACR, Springer Verlag.

[GRR98] Rosario Gennaro, Michael O. Rabin, and Tal Rabin. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In *Proc. 17th ACM Symposium on Principles of Distributed Computing (PODC)*, 1998.

[Rab98] Tal Rabin. A simplified approach to threshold and proactive RSA. In Hugo Krawczyk, editor, *Advances in Cryptology — CRYPTO '98*, volume 1642 of *LNCS*, pages 89–104, Berlin, 1998. Springer Verlag.

[Sha79] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, November 1979.

[Sho00] Victor Shoup. Practical threshold signatures. In Bart Preneel, editor, *Advances in Cryptology: EUROCRYPT 2000*, volume 1087 of *LNCS*, pages 207–220. Springer, 2000.

[WB99] Michale Malkin Thomas Wu and Dan Boneh. Experimenting with shared generation of rsa keys. In *Proceedings of the Internet Society's 1999 Symposium on Network and Distributed System Security (SNDSS)*, pages 43–56, 1999.