Efficient, Robust and Constant-Round Distributed RSA Key Generation

Ivan Damgård and Gert Læssøe Mikkelsen

Department of Computer Science, Aarhus University

Abstract. We present the first protocol for distributed RSA key generation which is constant round, secure against malicious adversaries and has a negligibly small bound on the error probability, even using only one iteration of the underlying primality test on each candidate number.

1 Introduction

The idea of distributed key generation is to generate a key in secret shared form among a number players such that it is never available in a single location. Together with a protocol for distributed signatures, for instance, one gets a distributed signature scheme that has no single point of attack throughout its lifetime.

Specifically for distributed RSA key generation, the main problem is to generate a modulus such that the prime factors are shared among the players. Two approaches have been suggested in the literature: Boneh and Franklin (BF) [4] suggest to generate a random candidate modulus $N = ab$ where a, b are random and shared among the players. One then runs a so-called biprimality test involving an exponentiation modulo N which is easy to do in a distributed fashion, and will accept an N with more than two prime factors with probability at most $\frac{1}{2}$. An alternative method was suggested by Algesheimer et al.(ACS) [1], where one generates candidate primes separately in shared form and tests each one for primality, by doing a Miller-Rabin primality test securely, i.e., by doing the required exponentiation while base, exponent and modulus are all secret-shared.

We now compare the methods and discuss whether there is room for improvement. Boneh and Franklin's test is very efficient because the modulus N is public. On the other hand, one has to wait until both factors a and b happen to be prime which requires more candidates than the standard method. The error probability is unfortunately very hard to bound: using only the worst-case result of $\frac{1}{2}$ leads to a very poor result that would seem to require many iterations of the biprimality test to bring the error down. For the Miller-Rabin test, it was shown by Damgård et al.[11] that the average case behavior is much better than the worst case, most composites pass the test with probability much smaller than the worst case, and hence for large numbers, only one iteration of the test is necessary for negligible error probability. One might hope for a similar result for the biprimality test, but this turns out to be unclear. The method from [11] relies heavily on the fact that for any prime factor p in a number N to be tested,

p does not divide $N-1$. To argue in a similar way for the biprimality test, one would need that if $N=ab$ and p divides a, p does not divide $(a-1)(b-1)$, but this is clearly not true in general.

Algesheimer et al. test each candidate prime individually and so need fewer candidates, and one can use [11] to estimate the error. On the other hand, all exponentiations must be done with a secret modulus which makes them much slower. According to [1], for a small number of players (the interesting case in practice) the computational complexity of BF is somewhat larger than ACS while the communication is smaller. The big difference, however, is that BF is constant-round for checking a candidate while ACS require $\theta(n)$ rounds for checking an n-bit prime. While Algesheimer et al. claim that this is not important, we disagree, and believe the issue is very significant both from a theoretic and a practical point of view (see discussion at the end of the introduction).

Our conclusion is that in many, if not most cases BF is the more attractive approach. It is therefore of interest to construct a protocol with a good bound on the error probability which is as efficient as BF. In this paper we do this by combining the two methods from [1,4] to get, in a sense, the best of both worlds. We are going to compute a public candidate modulus $N=ab$ like Boneh and Franklin, but we are going to test a and b for primality separately, as follows: if we choose $a=b=3 \bmod 4$, then doing the Miller-Rabin test say on a reduces to testing if $r^{(a-1)/2} \bmod a = \pm 1$ for a random base r. Now, because N is public, we can very efficiently choose a random $g \in Z_n^*$ and compute $y = g^{a-1} \bmod N$ in secret shared form using essentially Boneh and Franklin's protocol. Now we just need to reduce y modulo the secret a and test against $1, -1$. This can be done efficiently and in constant-round using a subprotocol from ACS. Note that there is no need for an exponentiation mod a, we just need a reduction, which is something ACS must do for every secure multiplication.

In this way, we get a protocol that is essentially as fast as Boneh and Franklin's, but where we can directly use [11] to estimate the error. Note that when testing a candidate, we can run the (simpler) biprimality test first without affecting the error probability since it never rejects a good modulus. This way, even if we cannot prove how well it does in the average case, we still get maximal mileage from it.

A second contribution of the paper is an efficient way to get a protocol secure against active (malicious) adversaries. Both BF and ACS were described for passive adversaries. Frankel et al. [14] suggested a way to get active security for the Boneh-Franklin protocol, but estimated themselves that the cost of this would be prohibitive in practice.

We suggest an alternative method where our secure computation is based on replicated integer secret sharing suggested by Damgård and Thorbek [13]. Here, the secret is shared additively over the integers, but each player gets several shares. Because of this replication, the scheme allows for secure multiplication in much the same way as Shamir's scheme. The observation is that because the scheme is also additive over the integers, we can use Algesheimer et al.'s protocol for modular reduction which exactly requires such a sharing scheme. This is in contrast to their original protocol where one uses Shamir's scheme for multiplication and so one must convert back and forth between the schemes throughout the protocol using interactive procedures. Finally, we make the scheme verifiable by keeping players committed to their shares.

The price we pay for the simplifying the protocol is that computational complexity of the scheme does not scale well with the number of players, but we believe this is not a serious issue: in contrast to earlier proposals the protocol is genuinely practical for less than, say, 10 players, and in threshold cryptography, one usually thinks of the number of players as a small constant. For instance, in the framework for distributed RSA signatures suggested by the authors[12], it is natural to run a 3-party protocol where a PC, a server and a mobile device held by the user execute the protocol.

Our protocol is secure against an active and static adversary corrupting any minority of the players, and the cost of going from passive security to active security is a constant factor, both regarding computations, network traffic and the number of rounds. In practice the constant is fairly low, covering committing including local exponentiations, which are already done in the passive protocol, and broadcast of commitments.

We close the introduction by discussing the claim by Algesheimer et al. that the difference in communication and round complexities between BF and ACS do not matter because one can test many candidates in parallel. We disagree with this: It is true that on a network with large round trip time, one can make the average cost of a protocol go down if many instances are to be done in parallel. Each player sends the next message in an instance as soon as he is ready to do so, and if we have enough instances, each player has enough local computation to keep him busy until the other players respond. Ideally, this means that the amortized time per instance can be almost as if there were no network delays. This is the basis of the ACS claim that their large round complexity is not a problem, since of course we can test many candidate primes in parallel. However, on the other hand, the *real time* elapsed from we start until we are done can of course never be smaller than the time it takes to do a single instance stand-alone.

The ACS protocol has some constant number of rounds for every bit in a candidate prime number so for 1000-2000 bit RSA, it will have something like 5000- 10.000 rounds pr. test (as opposed to our protocol with less than 100 rounds pr. test.) If we further assume a malicious adversary and that we are running on a network as the Internet that is basically asynchronous, rounds will tend to take a long time: a corrupt player may not send anything, so to distinguish this from a delay of an honest player's message, one has to wait long enough in each round so that the chance of an honest player's message failing to arrive is negligible. Otherwise, we may exclude an honest player as being corrupt, and then the protocol is no longer secure. If, for instance, we need to set a time-out of 1 second to be sure to avoid mistakes, 5000-10.000 rounds will take between 1 and 3 hours to execute. The conclusion is that when the number of rounds is very large, as for ACS, the parallellization paradigm only make sense on a fast network with very strong guarantees on delivery time.

2 Security Model

The protocols described in this paper are all three player protocols, where we assume that at most one of the players are corrupted. The protocols can be generalized to n players, however, the underlying secret sharing scheme does not scale well in the case of many players. Corruption of players is either considered to be passive, where a corrupt player still follows the protocol, or active where a corrupted player can misbehave arbitrarily.

Our security model only ensure that misbehavior can be detected, it might not be the case that the honest players can tell which player misbehaved. Furthermore, the protocols does not guarantee termination, in case of dishonest behavior. This simplifies the protocols and security proofs. Both detecting which player misbehaved and guaranteed termination can, however, easily be ensured by applying digital signatures such that all messages are signed.

We assume point to point secure communication channels, meaning authenticated and only the length of messages are leaked to the adversary. We also use a broadcast channel, however, since the channel does not have to ensure synchronous broadcast and we allow abort, this can easily be implemented on top of the point to point channels. The player that broadcast a message sends the message to the two other players, they send what they have received to each other to check if they agree. If they do not agree on what they have received they tell the other players and stop the protocol.

Universal Composability and Common Reference String We use the Universal Composability (UC) framework [7,8] to specify the security of our protocols. The active secure versions of our protocols assumes the *(chosen) common reference string model* (CRS), where all players have access to a common string, which can contain key material used to implement, in our case, commitments. The CRS model is used to improve the power of the simulator. Concretely in our case the reference string contains among others an RSA key N used to implement commitments. By giving the simulator additional information on some elements in $\mathbb{Z}_N$, the commitment scheme is not binding for the simulator, which is needed for our proofs. The CRS model might be circumvented by letting each player choosing its own N that will be used by the other players, when committing values, however, this is conceptually more complicated, and less efficient. Therefore it has been left out. On the other hand, the CRS model might be justifiable in the case of a PKI based on a CA. The CA' public key might be used as N, and as long as the CA does not actively cheats e.g., corrupts one of the players in the protocol, during key generation the protocol remains secure.

Definition 1. *Let $\mathcal{A}_{Pass}$ be the class of passive static adversaries corrupting at most one of the three players; and let $\mathcal{A}_{Act}$ be the class of active static adversaries corrupting at most one of the three players.*

3 Probabilistic Primality Test

In this section we present a probabilistic primality test based on the Miller-Rabin test [19]. The advantage of the test described here is, as we will see later, that it can be very efficiently implemented as a distributed protocol. By requiring that the candidate a being tested fulfills $a \equiv 3 \pmod 4$ the Miller-Rabin test on a reduces to testing if $v^{(a-1)/2} \pmod a \equiv \pm 1$ for a random base v.

ProbPrime Takes input a and N s.t. $a|N$. We assume $a \in [2^{n-1}, 2^n]$ and $N \in [2^{2n-2}, 2^{2n}]$.

1. $v \in_{\mathrm{R}} \mathbb{Z}_N$.
2. $\gamma_a \leftarrow v^{(a-1)/2} \bmod N$
3. If $\pm 1 \equiv \gamma_a \pmod a$, then output *Probably prime*, else output *Composite.*

The correctness and the error probability are stated in the following theorems. Since the protocol is based on the Miller-Rabin test these theorems are likewise based on error estimates of this test.

Theorem 1. ***ProbPrime*** *is a* Monte Carlo *algorithm with random input v. A correctly formed prime $a \equiv 3 \pmod 4$ is always accepted, and for* worst case *input it accepts a composite with probability $< \frac{1}{4}$.*

Proof. **ProbPrime** is essentially the Miller-Rabin test restricted to numbers $a \equiv 3 \pmod 4$ and therefore always accepts correctly formed primes, and has the same worst case error estimate: $\frac{1}{4}$ ([19]).

Theorem 2. *Let* ***ProbPrime*** *be utilized to generate probable primes by inputting randomly chosen n bit integers $a_i \equiv 3 \pmod 4$, running the test t independent times on each a_i and outputting the first number passing all test. Let $P_{n,t}$ denote the probability that a composite number is output. Assuming the Extended Riemann Hypothesis and $a_i > 2.3 \times 10^{10}$ then $P_{n,1} < n^2 4^{3-\sqrt{n}}$ and $P_{n,t} < n^{3/2} 2^t t^{-1/2} 4^{3-\sqrt{tk}}$ for $2 \leq t \leq n/9$.*

Proof. Damgård et al. [11] estimates $P_{n,1} < n^2 4^{2-\sqrt{n}}$ and $P_{n,2 \leq t \leq n/9} < n^{3/2} 2^t t^{-1/2} 4^{2-\sqrt{tk}}$ for the Miller-Rabin test for input chosen uniformly random in the set $I_{odd}(n)$ of n bit odd positive integers. We restrict the set we choose candidates from to $I_{3 \bmod 4}(n)$ the set of n bit positive integers $a \equiv 3 \pmod 4$.

Let $S_{odd}(n)$ and $S_{3 \bmod 4}(n)$ denote the density of the false positives, composite numbers accepted with high probability in $I_{odd}(n)$ and $I_{3 \bmod 4}(n)$ respectively. Since $I_{3 \bmod 4}(n)$ is half the size of $I_{odd}(n)$, $S_{3 \bmod 4}(n)$ is at most the double of $S_{odd}(n)$, which therefore at most doubles the average error probability.

We then consider the density of primes in $I_{3 \pmod 4}(n)$ compared with $I_{odd}(n)$. Heuristically the density of primes in $I_{odd}(n)$ and $I_{3 \bmod 4}(n)$ are asymptotically the same. However, by assuming the Extended Riemann Hypothesis and that $a_i > 2.3 \times 10^{10}$ the concrete bound [1]: $|\pi(x,4,3) - \frac{x}{2\log x}| < \frac{x}{\log 2x} = \frac{x}{2\log x}\frac{2}{\log x}$

[1] This bound follows from [2][Theorem 8.8.18]

can be found. This means that the difference between the density of primes in $I_{odd}(n)$ and $I_{3 \pmod 4}(n)$ is at most: $\frac{2}{\ln x}$ which for $2^n = x > 2.3 \times 10^{10}$ means $\frac{2}{\ln(x)} < 2$. This gives us another doubling of the average error probability and our average error probability is therefore 4 times higher than the one from [11] on the original Miller-Rabin test.

Two examples of concrete bounds of the average error probability on $n = 1024$ are $P_{1024,1} < 2^{-38}$ and $P_{1024,4} < 2^{-105}$. This means that running the test only one or a few times on each number is sufficient in practise.

4 Replicated Integer Secret Sharing

Secret sharing [20] is a known primitive used in many cryptographic protocols. This section describes an additive secret sharing scheme over the integers that enables *multi party computation* (MPC) over integers in a given interval. Additive secret sharing over the integers makes it possible to share an integer in some public known interval $[-T, T]$, by choosing the shares $s_0, \ldots, s_n \in_{\mathrm{R}} [-2^\kappa T, 2^\kappa T]$, where κ is the security parameter, such that $s = \sum s_i$. The shares are chosen in the lager interval to make a sharing of s statistically close to a sharing of zero, and therefore an adversary only gains negligible information of s even if all except one share is known to the adversary. Addition of two additive secret shared integers is done by locally adding the shares, however multiplication is not strait forward.

Replicated Integer Secret Sharing (RISS) is a revised variant of additive integer secret sharing, where multiplication and other calculations are made possible by replicating the shares, s.t. each player holds multiple shares, in case of three players they each holds two shares. The product of two secrets s and t can be rewritten as $st = (s_1, s_2, s_3) \times (t_1, t_2, t_3) = s_1t_1 + s_1t_2 + \cdots s_3t_2 + s_3t_3$, and therefore replicating the shares enables multiplication, because each product on the right hand side are known to at least one player.

When a dealer wants to share a secret s it is done as in the nonreplicated case: The dealer generates three uniform random numbers s_1, s_2 and $s_3 \in [-2^\kappa T, 2^\kappa T]$ s.t. $s = s_1 + s_2 + s_3$ then the dealer distributes these shares such that player 1 gets s_2 and s_3, player 2 gets s_1 and s_3 and player 3 gets s_1 and s_2.

In the rest of this section we will see how to implement multi party computation based on RISS, and *Verifiable Replicated Integer Secret Sharing* (VRISS) an active secure version of RISS. The specification of MPC using (V)RISS is defined as the ideal functionality $\mathcal{F}_{\mathrm{RISS}}$ in figure 1. The intuition of $\mathcal{F}_{\mathrm{RISS}}$ is a black box where the players can input values, associated with an index, then the players can do some computations on the values and the box can output results to one or more players. The simulator ideal-world adversary is allowed to delay output from $\mathcal{F}_{\mathrm{RISS}}$ maybe for infinitely long time, however, not to change values inside the functionality nor input or output from honest players.

Ideal functionality $\mathcal{F}_{\mathbf{RISS}}$

When started $\mathcal{F}_{\text{RISS}}$ initializes an empty list $\mathcal{L}$, let $\mathcal{L}(i)$ denote either the value or the memory that can store a value at index i. All output from $\mathcal{F}_{\text{RISS}}$ can be delayed (maybe infinitely) by the adversary.
Input: Upon receiving $(\texttt{Input}, pid, i, x)$ from player pid and $(\texttt{Input}, pid, i)$ from all other players store x at $\mathcal{L}(i)$.
Output: Upon receiving $(\texttt{Output}, pid, i)$ from all players; send $(\texttt{Value}, \mathcal{L}(i))$ to player pid.
Publish: Upon receiving $(\texttt{Publish}, i)$ from all players; send $(\texttt{Value}, \mathcal{L}(i))$ to the adversary and afterward to all players. (Maybe output to some or all players is delayed by the adversary)
Addition and Multiplication: Upon receiving $(\texttt{ADD}, i, j, k)$ or $(\texttt{MUL}, i, j, k)$ from all players store at $\mathcal{L}(i)$ the sum or the product of $\mathcal{L}(j)$ and $\mathcal{L}(k)$.
Constant Addition and Multiplication: Upon receiving $(\texttt{C-ADD}, i, j, x)$ or $(\texttt{C-MUL}, i, j, x)$ from all players store at $\mathcal{L}(i)$ the sum or product of $\mathcal{L}(j)$ and x.
Detected Misbehavior: Upon receiving $(\texttt{Misbehavior})$ from the adversary, at any point in the protocol. Then output $(\texttt{Misbehavior})$ to all players and halt (Note: This part is only necessary for active secure protocols)

Fig. 1. Ideal functionality defining the security of RISS and VRISS (See section 4.2)

4.1 Passive Secure Protocol Realizing $\mathcal{F}_{\mathbf{RISS}}$.

We will here describe some protocols based on RISS, which together implements the functionality $\mathcal{F}_{\text{RISS}}$. In this section we prove the security of the passive secure protocols, therefore we only consider the security when the players follow the protocols as described. In section 4.2 active secure protocols are described.

It is easy to see that if the shares of $s \in [-T, T]$ has been chosen uniformly s.t: $s_1, s_2 \in_{\text{R}} [-2^\kappa T, 2^\kappa T]$ and $s_3 = s - s_1 - s_2$ and s.t $s_3 \in [-2^\kappa T, 2^\kappa T]$ then two shares of s are indistinguishable from two shares of a sharing of zero, because the distributions are statistically close, with security parameter κ

Lemma 1. *Generating and distributing shares in RISS UC-realizes* Input *in $\mathcal{F}_{RISS}$ with respect to all $\mathcal{A}_{Pass}$ adversaries.*

Addition and Constant Multiplication To add shared numbers, each player locally adds the shares. Multiplication by a public known constant is done in the same way by locally multiplying the constant with the shares.

Lemma 2. *Since* addition *and* constant multiplication *in RISS only involves local computations it UC-realizes* Addition*,* Constant Addition *and* Constant Multiplication *in $\mathcal{F}_{RISS}$ with respect to all $\mathcal{A}_{Pass}$ adversaries.*

Jointly Generating (Pseudo) Random Sharing of Zero The multiplication protocol has to generate a random nonreplicated integer secret sharing of zero, such that no player know the complete sharing. By using the technique *pseudo random secret sharing* (PRSS) [9] in a novel way, this can be implemented as a noninteractive protocol. If the players pairwise share a secret key for a *pseudo random function* (PRF) in the same way they would share a RISS share, they can use this PRF and the keys to generate three numbers $r_1, r_2, r_3 \in_{\text{R}} [-2^n, 2^n]$ this is a replicated integer secret sharing of $r = r_1 + r_2 + r_3$. 0 can be written as $0 = r - r = (r_1 + r_2 + r_3) - (r_1 + r_2 + r_3) = (r_1 - r_2) + (r_2 - r_3) + (r_3 - r_1)$, and each of the three summands can be calculated by one of the players. The shares of zero have size $n + 2$. If one of the players is corrupt there are $n + 1$ bit uncertainty for the adversary of the two shares the adversary does not know; due to the fact that there are $n+1$ different equally possible values for the share r_x unknown to the adversary. We will later use PRSS to generate random secret shared values, and publicly known random values. Given point to point secure channels between the players the shared keys can easily be set up beforehand.

Multiplication To multiply two RISS shared numbers $\langle a \rangle^R$ and $\langle b \rangle^R$, such that $\langle c \rangle^R = \langle a \rangle^R \times \langle b \rangle^R$, each player i locally multiplies the local shares a_{i-1} and a_{i+1} of a and b_{i-1} and $b_{i+1}$of b. Now each player holds some shares of $\langle ab \rangle^R$, however, not all of these shares are replicated, to solve this and to bring the number of shares at each player down to two again, each player sum shares of ab and replicates the shares again. To rerandomize the shares a nonreplicated integer sharing of zero is added to the result before replication.
MUL($\langle a \rangle^R$, $\langle b \rangle^R$) Player i holds a_{i-1}, a_{i+1}, b_{i-1} and b_{i-1}. S.t. $-2^n < ab < 2^n$.

- Calculate $\langle ab \rangle^I_i \leftarrow (a_{i-1} \times b_{i-1}) + (a_{i-1} \times b_{i+1}) + (a_{i+1} \times b_{i-1})$
  (Note that $\langle ab \rangle^I$ is a nonreplicated integer secret sharing of $a \times b$)
- Jointly generate a $\kappa + n$ bit integer secret sharing of zero $\langle 0 \rangle^I$.
- $\langle c \rangle^R_{i-1} \leftarrow \langle ab \rangle^I_i + \langle 0 \rangle^I_i$
- Send $\langle c \rangle^R_{i-1}$ to player P_{i+1}, and wait for $\langle c \rangle^R_{i+1}$ from player P_{i-1}.

Lemma 3. Multiplication *in RISS UC-realizes* Multiplication *in $\mathcal{F}_{RISS}$ with respect to all $\mathcal{A}_{Pass}$ adversaries.*

Proof. We simulate multiplication by using sharings of zero instead of the real values. This is statistically close to the real values because the interval of the shares are κ bit greater than the values. Afterward the simulator can simulate any result by adjusting the share not known to the adversary. This will result in a share in the correct interval, except with negligible probability.

Theorem 3. *MPC with RISS UC-realizes $\mathcal{F}_{RISS}$ with respect to all $\mathcal{A}_{Pass}$ adversaries.*

Proof. lemma 1 - 3

4.2 Verifiable Replicated Integer Secret Sharing

The previous section described how RISS can be used to do multiparty computation securely against a passive adversary. To extend the security to active security, and ensure that secret values are not leaked, and that the adversary cannot influence the output of a protocol except by changing the input of a corrupted machine, we need to force the players to follow the protocol. We note that the protocol we describe in this section does not guarantee termination, and we cannot always determine who has been misbehaving if dishonest behavior is detected. However, to resolve conflicts of which player cheated, digital signatures might be utilized.

To obtain an active secure version of RISS, we need to ensure that, when a player shares a value the player is committed to this value, and that a receiver of a share is committed to the received value. We also need to enforce that each player proves to the others that calculations has been done correctly. To achieve these goals we need to assume the *chosen common reference string* (CRS) model for our commitments.

Commitment Scheme Fujisaki and Okamoto [15](see Damgård and Fujisaki [10] for a revised version) describes an integer commitment scheme, which is additively homomorph and can be simulated in UC in the CRS model. The common reference string used consists of an RSA modulus N_{CRS} and two elements $g, h \in \mathbb{Z}_{N_{CRS}}$, where the discrete log between g and h is unknown to the players, however, not to the simulator, which allows simulation. When a player wants to commit to a value s a uniform random value $r \in_{\text{R}} \mathbb{Z}_{N_{CRS}}$ is chosen and the commitment of s is: $\text{com}(s, r) \mapsto g^s h^r \mod N_{CRS}$, this scheme is additive homomorphic because: $\text{com}(s + t, r_s + r_t) = (g^s h^{r_s}) \times (g^t h^{r_t}) \mod N_{CRS}$ To open a commitment s and r are revealed. These commitments are statistically hiding and computationally binding, assuming the strong RSA assumption. If the discrete log between g and h is known the commitments are no longer binding.

We also need an other primitive from the commitment scheme, which is the ability to prove that two commitments c_1 and c_2 are commitments of the same value. In the case where the same base (g and h) is used this is an easy task. To show that $c_1 = com(s, r_1)$ and $c_2 = com(s, r_2)$ the prover shows that $c_1 \times c_2^{-1} = com(s - s, r_1 - r_2)$ can be opened to zero. In the case where different bases are used the problem is more difficult, however, in our case with three players and at most one corrupted player, there exists an easy solution. The prover just need to prove to the two others that he is committing correctly if none of the two verifiers are corrupted, because if the prover is corrupted then both verifiers are honest and can thus trust each other, if one of the verifiers is corrupt, then the prover is honest and by assumption committed to the correct value. The actual protocol proving $s = \hat{s}$ for $(g^s h^r)$ and $(\hat{g}^{\hat{s}} \hat{h}^{\hat{r}})$ is the following:

1. Generate:
   s_1, s_2 s.t. $s_1 + s_2 = s = \hat{s}$; $\quad r_1, r_2$ s.t. $r_1 + r_2 = r$ and $\hat{r}_1, \hat{r}_2$ s.t. $\hat{r}_1 + \hat{r}_2 = \hat{r}$
2. Publish:
   $c_1 = g^{s_1} h^{r_1} \mod N$, $c_2 = g^{s_2} h^{r_2} \mod N$, $\hat{c}_1 = \hat{g}^{s_1} \hat{h}^{\hat{r}_1} \mod N$ and $\hat{c}_2 = \hat{g}^{s_2} \hat{h}^{\hat{r}_1} \mod N$

3. Open c_1 and $\hat{c}_1$ to verifier 1 and c_2 and $\hat{c}_2$ verifier 2. Both accept if c_x and $\hat{c}_x$ opens to the same value.

Generating Shares When player i wants to share a secret s it is done as in the passive case, with the exception that before the shares are distributed the player broadcasts a commitment of each share, this is also a commitment to s due to the additive homomorphic property of com(). When the shares are distributed as in RISS, player i opens the commitment to each share to the receivers of the share.

Addition, Constant multiplication As in the passive case *addition* and *constant multiplication* can be computed without communication between the players. This can be done because when a player adds shares of two secrets a and b locally, the other players can calculate $\text{com}(a_i + b_i)$ due to the additive homomorphic property of com(). Analogously with constant multiplication because: $\text{com}(ca_i, cr_i) = g^{ca_i}h^{cr_i} = (g^{a_i}h^{r_i})^c$

Joint Generation of Shares Utilizing *pseudo random secret sharing* enables active secure generation of a random secret shared number s, by only one broadcast message pr. player. This is done by generating shares s_1, s_2 and s_3 as in the passive case, and in addition generate the randomness r_1, r_2 and r_3 for the commitments by the PRF and the shared keys. Now each player calculates the commitments $c_x = \text{com}(s_x, r_x)$ to the two shares and broadcasts the result. All three players can check if the two commitments to the same share are equal, if not, one of the players misbehaved. Because of the additive homomorphism of com() the joint sharing of zero can also be done in one round with only one broadcast message pr. player.

VRISS Multiplication Enabling multiplication in VRISS requires that one player can prove to the others that he have multiplied two committed values correctly. This can be done if the prover is committed to a and b with $c_a = \text{com}(a, r_a)$ and $c_b = \text{com}(b, r_y)$ and proves that $c_{ab} = \text{com}(ab, r_{ab})$. First let:

$$c_{ab} \leftarrow (c_a)^b h^r \mod N_{crs} \equiv g^{ab}h^{r_a b + r} \mod N_{crs}$$

This is indeed a commitment to ab with the base g and h. To prove that it is correct the prover proves that c_{ab} base c_a and h is a commitment to the same value as c_b base g and h using the algorithm described earlier. Now the passive protocol is executed with each player committing and proving to the others that the commitments are well formed and that the steps of the protocol has been followed.

Theorem 4. *Assuming the strong RSA assumption and the existence of PRF, then MPC with VRISS UC-realizes $\mathcal{F}_{RISS}$ with respect to all $\mathcal{A}_{Act}$ adversaries.*

Proof. Theorem 3 proves that we can simulate the protocol, if all players follow the protocol. Adding the commitment scheme and the checks of the commitments, forces a corrupt player to follow the protocol, or the other players will detect the misbehavior. On the other hand, simulation is still possible because we assume the discrete log between g and h is know to the simulator, and it can therefore circumvent the binding property of the commitment scheme.

4.3 Distributed Primality Testing

The ideal functionality $\mathcal{F}_{\text{RISS}}$ and the protocols of RISS and VRISS describes a general secret sharing scheme. However, in addition to this our protocol for RSA key generation needs a protocol implementing a distributed version of the primality test described in section 3. This extended RISS is described as an ideal functionality $\mathcal{F}_{\text{EXT-RISS}}$ (see Fig. 2), which is an extension of $\mathcal{F}_{\text{RISS}}$. The protocols implementing $\mathcal{F}_{\text{EXT-RISS}}$ has additional requirements on the integers used as input and on how these integers are shared. We call these *special form integers.*

Definition 2 (Special Form Integer). *An integer a is a* special form integer *if it has been generated as such and fulfills:* $2^{n-1} < a < 2^n$ *and* $a \equiv 3 \pmod 4$.

A resharing of a shared integer does not preserve *special form* of an integer, therefore only integers generated as *special form* can be on *special form.* This is because in the realization of $\mathcal{F}_{\text{EXT-RISS}}$ requires that the sharing of the integer is on the following form:

Definition 3 (Special Form Integer Sharing). *A* special form integer sharing *is a sharing of a* special form integer *fulfilling:* $a_1 \equiv 3 \pmod 4$ *and* $a_2 \equiv a_3 \equiv 0 \pmod 4$.

Ideal functionality $\mathcal{F}_{\text{EXT-RISS}}$

$\mathcal{F}_{\text{EXT-RISS}}$ is identical to $\mathcal{F}_{\text{RISS}}$ except it is extended with the following:
Randomly Generate Special Form Integer Upon receiving $(\texttt{GenSFI}, pid, i)$ from all players generate a uniform random *special form integer* a. Store a at $\mathcal{L}(i)$, with a flag specifying that $\mathcal{L}(i)$ holds a *special form integer.*
Trial Division Upon receiving $(\texttt{Div?}, pid, B, i)$ from all players, if $\exists \ell < B$ s.t. $\ell | \mathcal{L}(i)$ then output $(\texttt{Fail})$ to all players, otherwise output $(\texttt{Success})$ to all players.
Probabilistic Prime Test Upon receiving $(\texttt{ProbPrime?}, pid, i, j, N)$ from all players and if $\mathcal{L}(i)$ is a *special form integer*, and $N = \mathcal{L}(i) \times \mathcal{L}(j)$, then let $a \leftarrow \mathcal{L}(i)$, choose $v \in_{\text{R}} \mathbb{Z}_N$ and calculate $\gamma = v^{\frac{a-1}{2}} \bmod a$. If $\gamma = \pm 1$ output $(\texttt{ProbPrime}, v)$ to all players, else send $\{\mathcal{L}(i), \mathcal{L}(j), v\}$ to the adversary and $(\texttt{Composite}, v)$ to all players.

Fig. 2. Ideal functionality defining the security of Extended RISS

Randomly Generate Special Form Integer Player 1 and 2 each pick a random integer $a^{(i)} \in_{\mathrm{R}} [2^{n-2}, 2^{n-1}]$, s.t. $a^{(1)} \equiv 3 \pmod 4$ and $a^{(2)} \equiv 0 \pmod 4$. This ensures that $a = a^{(1)} + a^{(2)} \in [2^{n-1}, 2^n]$ and $a \equiv 3 \pmod 4$. Both players share them s.t $a_1^{(1)} \equiv 3 \pmod 4$ and for all other shares: $a_j^{(i)} \equiv 0 \pmod 4$. The shares are distributed and added, which ensures that the shares fulfills the congruence requirement for a *special form integer sharing.* The special requirement of the congruency of the shares only leaks the value of $a \bmod 4$. The security follows from the security of input and addition of RISS shares.

Lemma 4. Randomly Generate Special Form Integer *UC-realizes this part of $\mathcal{F}_{EXT\text{-}RISS}$ with respect to all $\mathcal{A}_{Pass}$ adversaries.*

Trial Division To do trial division up to a bound B on a shared number a, the players test if a is divisible by a small prime ℓ by randomly choosing an $n + \kappa$-bit secret shared number r using PRSS as described in section 4.1. $\langle ra \rangle^R$ is calculated by the multiplication protocol and all shares of $\langle ra \rangle^R$ are locally reduced modulo ℓ and afterward broadcast to open $\alpha = (ra \mod \ell) + \beta\ell$, where $0 \leq \beta < 3$. If $\alpha \not\equiv 0 \pmod \ell$ then $\ell \not| a$, however, if $\alpha \equiv 0 \pmod \ell$ then either $\ell | a$ or $\ell | r$. To prevent the protocol from rejecting a when $\ell | r$ the protocol is executed a number of times with new random values r. For optimization reasons local reductions modulo ℓ should be done before and during the calculation of $\langle ra \rangle^R$, and the trial divisions should be executed in parallel.

Lemma 5. Trial Division *UC-realizes this part of $\mathcal{F}_{EXT\text{-}RISS}$ with respect to all $\mathcal{A}_{Pass}$ adversaries.*

Proof. This can be simulated by simulating the result of $\langle ra \rangle^R$. The leaked value $\alpha = (ra \mod \ell) + \beta\ell$ does not leak information because it can be perfectly simulated, by choosing r and its shares appropriately.

Probabilistic Primality Test Here we present a distributed version of the primality test described in section 3.

ProbPrime The players holds special form integer shares of the value being tested $\langle a \rangle^R$. The value $\langle b \rangle^R$ is secret shared among the players, and $N = ab$ is publicly known. We assume $a, b \in [2^{n-1}, 2^n]$ and $N \in [2^{2n-2}, 2^{2n}]$.

1. Distributed generate a public known value $v \in_{\mathrm{R}} \mathbb{Z}_N$.
2. The players locally calculates γ_{a_i} s.t.:
   $\gamma_{a_1} = v^{(a_1-1)/2} \bmod N$ and
   $\gamma_{a_2} = v^{(a_2)/2} \bmod N$
   $\gamma_{a_3} = v^{(a_3)/2} \bmod N$
3. The players share the values γ_{a_i} and calculate $\langle \gamma_a \rangle^R = \prod \gamma_{a_i} \equiv v^{(a-1)/2} \pmod N$
4. Distributed check if $\pm 1 \equiv \langle \gamma_a \rangle^R \pmod{\langle a \rangle^R}$, if so output *Probably prime*, otherwise output *Composite.*

Generating v can efficiently be done if all players uses the same key for the PRF. To check if $\pm 1 \equiv \gamma_a \pmod{a}$ we use a technique based on the ACS protocols [1]. However, due to a different setting, where we assume $N = ab$ is publicly known, we can improve the protocols from $O(log(n))$ rounds to $O(1)$ rounds, n being the bit length of a. First note that $(\gamma_a \bmod a) = \gamma_a - \left\lfloor \frac{\gamma_a}{a} \right\rfloor a$. If we assume the following $2^{n-1} < a, b < 2^n$, $2^{n-2} < N = ab < 2^{2n}$ and $\gamma_a < 2^{2n+2}$ we can approximate $\frac{\gamma_a}{a}$ in the following way [2]:

$$\widetilde{N} = \left\lceil \frac{2^{5n+2}}{N} \right\rfloor \Rightarrow \left| \frac{1}{N} - \frac{\widetilde{N}}{2^{5n+2}} \right| < 2^{-3n+2} \quad \text{and} \quad \widetilde{N} < 2^{3n+4} \tag{1}$$

$$\widetilde{a} = b \times \widetilde{N} \Rightarrow \left| \frac{1}{a} - \frac{\widetilde{a}}{2^{5n+2}} \right| < 2^{-2n+2} \quad \text{and} \quad \widetilde{a} < 2^{4n+4} \tag{2}$$

$$\Rightarrow \left| \frac{\gamma_a}{a} - \frac{\gamma_a \widetilde{a}}{2^{5n+2}} \right| < 1 \quad \text{and} \quad \gamma_a \widetilde{a} < 2^{6n+6} \tag{3}$$

$$\Rightarrow \gamma_a - \left\lfloor \gamma_a \widetilde{a} 2^{-5n+2} \right\rfloor a = (\gamma_a \bmod a) + \delta a, \quad -1 \leq \delta \leq 1 \tag{4}$$

In the above equations $\widetilde{N}$ is calculated locally by each player, $\langle \widetilde{a} \rangle^R$ is calculated by the distributed constant multiplication protocol. The value $\langle \gamma_a \widetilde{a} \rangle^R$ is calculated by the multiplication protocol. Calculating $\left\lfloor \gamma_a \widetilde{a} 2^{-5n+2} \right\rfloor$ is done by each player locally dividing the shares of $\gamma_a \widetilde{a}$ by 2^{5n+2} and rounding the result downwards: $\lfloor c \rfloor \approx c' = \lfloor c_1 \rfloor + \lfloor c_2 \rfloor + \lfloor c_3 \rfloor$, $|c' - c| \leq 3$. Therefore we can calculate $y = \gamma_a - \left\lfloor \gamma_a \widetilde{a} 2^{-5n+2} \right\rfloor = (\gamma_a \bmod a) + \delta a$ s.t. $-4 \leq \delta \leq 4$ and $y < 2^{2n+3}$.

The last step in the protocol to test if $\pm 1 \equiv \gamma_a \pmod{a}$ is to calculate:

$$z = \left(\prod_{i=-4}^{4} ((y + ia) + 1 \bmod Q) \right) \left(\prod_{i=-4}^{4} ((y + ia) - 1 \bmod Q) \right) \bmod Q \tag{5}$$

The number Q is a publicly known prime s.t. $Q > 2^{2n+3} > (y + ia) + 1, |i| \leq 4$. The multiplications in (5) are done modulo Q to limit the size of the numbers we are calculating on to $2n + 3$ bit numbers. Multiplications modulo Q can be done by doing local modulo reduction on the shares before and after the multiplication protocol. Now the players opens z and if $z = 0$ they output *success*, otherwise they output *failure*. The last step of the protocol is correct because if $\pm 1 \equiv \gamma_a \bmod a$ then $((y + ia) \pm 1)$ is zero when $i = \delta$, and since the numbers we calculate on are less than Q, and thereby relatively prime to Q, then $z \neq 0$ is always the case if $\pm 1 \not\equiv \gamma_a \bmod a$

Lemma 6. *Assuming the existence of PRF's, then on well formed input (special form integer sharing)* ***ProbPrime*** *UC-realizes this part of $\mathcal{F}_{EXT\text{-}RISS}$ with respect to all $\mathcal{A}_{Pass}$ adversaries.*

Proof. We assume the existence of PRF's, therefore the value v in the ideal and real world cannot be distinguished efficiently. If $\gamma \neq \pm 1$ in $\mathcal{F}_{\text{EXT-RISS}}$ the

[2] $\lceil x \rfloor$ meaning rounding x to nearest integer ($\lceil x \rfloor = \lfloor x + \frac{1}{2} \rfloor$)

simulator gets knowledge of all private input and can therefore easily simulate the protocol. If $\gamma = \pm 1$ the protocol can be simulated as follows. Step two only contains local calculations and does therefore not leak any information. Step three can be simulated, see lemma 3. The last step can in the same way be simulated to output $z = 0$, because of lemma 3 (it is easy to see that the lemma still holds modulo Q).

4.4 Active Security Distributed Primality Testing

For active security we need to ensure that the players follow the protocol. This means player 1 and 2 have to prove that there random input during generation of a special form integer sharing is well formed. The correct congruence modulo 4 can easily be tested, because each share will be send to two players, and therefore at least one honest. To prove that $a^{(1)}$ or $a^{(2)}$ is in the correct range we use a technique, from [6], we note that the solution described here is less efficient than [6], however, it is conceptually simpler. Proving that a number $a \in [2^{n-1}, 2^n]$ can be done by proving that $a - 2^{n-1} \geq 0$ and that $2^n - a \geq 0$. Proving that $x \geq 0$ is done by writing x as a sum of squares. Any positive number can be written as the sum of four squares which efficiently can be calculated [17].

A protocol for player i to prove $a \in [2^{n-1}, 2^n]$ is: Player i calculates $\alpha_1, \ldots, \alpha_4$ and $\beta_1, \ldots \beta_4$ s.t. $\sum \alpha_i^2 = a - 2^{n-1}$ and $\sum \beta_i^2 = 2^n - a$. Player i shares the numbers using VRISS and the three players calculates $\tilde{\alpha} = (a - 2^{n-1}) - \prod \alpha_i^2$ and $\tilde{\beta} = (2^n - a) - \prod \beta_i^2$. The values $\tilde{\alpha}$ and $\tilde{\beta}$ is opened and if they are opened to zero then $a \in [2^{n-1}, 2^n]$ is true, otherwise player i is deviating from the protocol.

The protocols also includes local computations on the shares, these do not leak information, and is therefore passively secure. They can be made active secure with one broadcast message pr. player: The players use the PRF to generate three random and replicated values. Now the players uses these random values to commit to the result of the local computation such that each share of the result is committed with the same randomness by the two players calculating the same share. The commitments are broadcast, and if the two commitments of the same value are not equal, one of the players misbehaves.

Theorem 5. *Assuming the strong RSA assumption and existence of PRF, then the above protocols UC-realizes $\mathcal{F}_{EXT\text{-}RISS}$ with respect to all $\mathcal{A}_{Act}$ adversaries.*

Proof. This follows from theorem 4, lemma 4 - 6, and the above description.

5 Distributed RSA Moduli Generation

The security of our RSA moduli generating protocol is given by the ideal functionality $\mathcal{F}_{\mathrm{KeyGen}}$ (Fig. 3). The intuition is that if the players follow the protocol then the factorization of N is secret, however, if misbehavior is detected by all players then N should not be used, and it is secure to reveal p and q.

The protocol Π_{KeyGen} implementing $\mathcal{F}_{\mathrm{KeyGen}}$ is described in Fig. 4. The protocol is based on the BF protocol [4], with an other probabilistic primality

Ideal functionality $\mathcal{F}_{\mathbf{KeyGen}}$

Key Generation: Upon receiving ($\mathtt{KeyGen}, sid, n$) from all players; generate two n-bit primes p and q, s.t. $p \equiv q \equiv 3 \mod 4$ and let $N = pq$.
Send N to the adversary. When the adversary replies with (`Deliver`) then send N to all players and halt.
Detected Misbehavior: Upon receiving (`Misbehavior`) from the adversary, at any point in the protocol. Then send p and q to the adversary, output (`Misbehavior`) to all players and halt

Fig. 3. Ideal functionality for generating an RSA modulus.

test. We start by describing a passive secure protocol, and afterward we extend it to active security. Π_{KeyGen} assume an MPC scheme realizing $\mathcal{F}_{\text{EXT-RISS}}$.

RSA Moduli Generation Protocol: $\Pi_{\mathbf{KeyGen}}$

1. **Pick candidates:** Secretly pick random numbers a and b s.t. $a \equiv b \equiv 3 \mod 4$.
2. **Trial division:** Distributed trial divide a and b up to a bound B.
   Repeat step 1 and 2 until two candidates a and b passes the trial division.
3. **Compute N:** The players calculate and publish $N = ab$.
4. **Primality test:** Run primality test to check a. If a is accepted, b is tested.
   If either a or b was rejected, the protocol is restarted, otherwise output N.
5. **Proof honest behavior** The players prove that they in the earlier steps of the protocol followed the protocol honestly.

Fig. 4. Protocol for distributed generation of RSA moduli.

Picking candidates: By the protocol for randomly generating special form integers, the players jointly generates two prime candidates a and b s.t $2^{n-1} < a, b < 2^n$ and $a \equiv b \equiv 3 \pmod 4$

Trial Division: Trial division up to a bound B is performed on a and b. Instead of trial division distributed sieving, which is more efficient, can be utilized, see section 6.

Computing N: To compute N the parties use the multiplication protocol and make the result N public. When N is public the players might do more local trial division before continuing.

Primality Test: To test whether a and b, both having survived trial division, are indeed primes, or at least with overwhelming probability are primes, **ProbPrime** is used to test a and b one or a few times. If a or b is rejected the protocol is restarted, otherwise in the passive case N is output as the RSA modulus. In the active secure case the players need to prove honest behavior before N is output.

Active Security Extending the protocol to active security, can be done using VRISS instead of RISS. However, a more efficient solution exists. When the players choose the input they commit and broadcasts the commitments. The rest of the protocol is run using the RISS protocol for distributed calculations. If either a or b at some point is rejected as primes the players opens the commitments of a and b publicly and each player can locally test if a or b should have been rejected or if a player is misbehaving. When a modulus N is accepted the players calculates and broadcasts all the proofs of well formed input and of having executed the protocol correctly. If a player cannot broadcast correct proofs, the other players reports that misbehavior is detected.

Lemma 7. *The probability of generating a modulus N which is* not *the product of two primes is the same as in the generic RSA key generation using the Miller-Rabin test to generate Blum integers.*

Proof. In the last round of the protocol, where both a and b passes the test the value a has been chosen completely independent of b and vice versa. Because we choose a and b at the same time there are rounds before the last one where b is rejected and we have to sample a new a. However, since we sample independent in each round these rounds can just be seen as a delay of randomly chosen time inserted in the protocol.

Theorem 6. *Assuming the Extended Riemann Hypothesis, existence of Pseudo Random Functions and the Strong RSA assumption, then Π_{KeyGen} UC-realizes $\mathcal{F}_{KeyGen}$ with respect to all $\mathcal{A}_{Act}$ adversaries.*

Proof. From lemma 7 it follows that the output of the protocol and the ideal functionality is indistinguishable. We also need simulate the transcript of the protocol given N from $\mathcal{F}_{\text{KeyGen}}$. First we assume that the adversary follows the protocol as described. In that case simulating the number of rounds of Π_{KeyGen} where candidates are rejected either by trial division or by the primality test can be done as: The simulator run the real protocols on input a and b not being two primes, such that they are rejected, with the same distribution as in the real world.

The last round, where N is accepted and output from Π_{KeyGen}, can be simulated in the following way: The simulator can simulate trial division and the primality test without knowing the input, this means it can simulate acceptance of the two protocols, without knowing the factorization of N.

If the corrupted player does not follow the protocol there are the following two cases: In one of the rounds where a and b should be rejected, but are not, the adversary cannot present proofs of following the protocol. Therefore in the real world the honest players will detect misbehavior and in the ideal world the simulator will report misbehavior. In the last round where N is supposed to be output, but is rejected, then in the real world the honest players will detect misbehavior when the adversary cannot present shares of a and b making the test fail. In the ideal world the simulator reports misbehavior and is given the factorization of N and can therefor show shares of $a = p$, $b = q$ to the adversary such that the test should have passed.

6 Optimizations

Parallelization If the bottleneck of the protocol is network latency, then testing many candidates in parallel will decrease effect of the latency.

Distributed Sieving Instead of first pick candidates to be primes, and thereafter perform trial division. It is possible to do distributed sieving for candidates relatively prime to all small primes less that some bound B. This technique is due to Malkin et al. [18], and in their implementation distributed sieving gave a 10 fold speedup when generating 1024 bit keys. Distributed sieving is done by letting $M = \prod_{\ell \in PRIMES}^{B}(\ell)$ and let the players pick random values $a_i \in \mathbb{Z}_M^*$ and letting the candidate $a = (\prod a_i) + rM$ for a random value r in an appropriate interval. This makes a relatively prime to M and thereby relatively prime to all small primes less than B. After converting a into additive shares the players must initiate a protocol that ensures the additive shares has the right properties (congruence modulo four), this applies [4] and to our protocol, however, not to [18] due to their simpler (only heuristically secure) primality test.

Using Multi Prime RSA Modulus As mentioned in [4] it is possible to avoid the quadratic slowdown of testing two candidates at the same time instead of testing two candidates independently as done in [1] and when generating RSA keys locally by [19]. The trick is to use a modulus which is a product of three primes, known as multi prime RSA. $N = p_1 p_2 (a_1 + a_2)$ where p_1 is a prime known to player 1 and p_2 is a prime known to player 2 and $a = a_1 + a_2$ is a candidate for a third prime. Unlike [3] that need a special tri-primality test like [5] our protocol can easily be extended to test multi prime moduli due to primality test. It should be noted that the latest PKCS #1 version (v2.1) [16] includes the use of multi prime RSA, although the motivation there is improved speedup when utilizing the Chinese remainder theorem technique.

7 Conclusion and Acknowledgment

We have presented a novel approach to do distributed generation of RSA moduli, with an active secure constant round primality test with a good bound on the average error probability. By using parallelization the complete generation of RSA moduli can made constant round, even when guarantying active security. An second contribution is a novel way to do multi party computations with replicated integer secret sharing. An open question remains, whether a better average case analysis of the Boneh and Franklins biprimality test is possible.

We thank Arjen Lenstra for some useful pointers.

References

1. Joy Algesheimer, Jan Camenisch, and Victor Shoup. Efficient computation modulo a shared secret with application to the generation of shared safe-prime products. In

Moti Yung, editor, *CRYPTO*, volume 2442 of *Lecture Notes in Computer Science*, pages 417–432. Springer, 2002.

2. Eric Bach and Jeffrey Shallit. *Algorithmic Number Theory: Efficient algorithms*. MIT Press, 1996.
3. Dan Boneh and Matthew K. Franklin. Efficient generation of shared RSA keys (extended abstract). In Burton S. Kaliski Jr., editor, *CRYPTO*, volume 1294 of *Lecture Notes in Computer Science*, pages 425–439. Springer, 1997.
4. Dan Boneh and Matthew K. Franklin. Efficient generation of shared RSA keys. *J. ACM*, 48(4):702–722, 2001.
5. Dan Boneh and Jeremy Horwitz. Generating a product of three primes with an unknown factorization. In Joe Buhler, editor, *ANTS*, volume 1423 of *Lecture Notes in Computer Science*, pages 237–251. Springer, 1998.
6. Fabrice Boudot. Efficient proofs that a committed number lies in an interval. In Bart Preneel, editor, *EUROCRYPT*, volume 1807 of *Lecture Notes in Computer Science*, pages 431–444. Springer, 2000.
7. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000. (2005 version).
8. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.
9. Ronald Cramer, Ivan Damgård, and Yuval Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In Joe Kilian, editor, *TCC*, volume 3378 of *Lecture Notes in Computer Science*, pages 342–362. Springer, 2005.
10. Ivan Damgård and Eiichiro Fujisaki. A statistically-hiding integer commitment scheme based on groups with hidden order. In Yuliang Zheng, editor, *ASIACRYPT*, volume 2501 of *Lecture Notes in Computer Science*, pages 125–142. Springer, 2002.
11. Ivan Damgård, Peter Landrock, and Carl Pomerance. Average case error estimates for the strong probable prime test. *Mathematics of Computation*, 61(203):177–194, July 1993.
12. Ivan Damgård and Gert Læssøe Mikkelsen. On the theory and practice of personal digital signatures. In Stanislaw Jarecki and Gene Tsudik, editors, *Public Key Cryptography*, volume 5443 of *Lecture Notes in Computer Science*, pages 277–296. Springer, 2009.
13. Ivan Damgård and Rune Thorbek. Linear integer secret sharing and distributed exponentiation. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *Public Key Cryptography*, volume 3958 of *Lecture Notes in Computer Science*, pages 75–90. Springer, 2006.
14. Yair Frankel, Philip D. MacKenzie, and Moti Yung. Robust efficient distributed RSA-key generation. In *STOC*, pages 663–672, 1998.
15. Eiichiro Fujisaki and Tatsuaki Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In Burton S. Kaliski Jr., editor, *CRYPTO*, volume 1294 of *Lecture Notes in Computer Science*, pages 16–30. Springer, 1997.
16. RSA Laboratories. PKCS #1 v2.1: RSA cryptography standard. Technical report, 2002.
17. Helger Lipmaa. On diophantine complexity and statistical zero-knowledge arguments. In Chi-Sung Laih, editor, *ASIACRYPT*, volume 2894 of *Lecture Notes in Computer Science*, pages 398–415. Springer, 2003.
18. Michael Malkin, Thomas D. Wu, and Dan Boneh. Experimenting with shared generation of RSA keys. In *NDSS*. The Internet Society, 1999.
19. Michael O. Rabin. Probabilistic algorithm for testing primality. *Journal of Number Theory*, 12, issue 1:128–138, February 1980.
20. Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.