Multiparty Generation of an RSA Modulus[*]

Megan Chen, Ran Cohen, Jack Doerner, Yashvanth Kondi,
Eysa Lee, Schuyler Rosefield, and abhi shelat

Northeastern University, Boston, MA, USA

Abstract. We present a new multiparty protocol for the distributed generation of biprime RSA moduli, with security against any subset of maliciously colluding parties assuming oblivious transfer and the hardness of factoring.

Our protocol is highly modular, and its uppermost layer can be viewed as a template that generalizes the structure of prior works and leads to a simpler security proof. We introduce a combined sampling-and-sieving technique that eliminates both the inherent leakage in the approach of Frederiksen et al. (Crypto'18), and the dependence upon additively homomorphic encryption in the approach of Hazay et al. (JCrypt'19). We combine this technique with an efficient, privacy-free check to detect malicious behavior retroactively when a sampled candidate is not a biprime, and thereby overcome covert rejection-sampling attacks and achieve both asymptotic and concrete efficiency improvements over the previous state of the art.

1 Introduction

A *biprime* is a number N of the form $N = p \cdot q$ where p and q are primes. Such numbers are used as a component of the public key (i.e., the *modulus*) in the RSA cryptosystem [33], with the factorization being a component of the secret key. A long line of research has studied methods for sampling biprimes efficiently; in the early days, the task required specialized hardware and was not considered generally practical [31, 32]. In subsequent years, advances in computational power brought RSA into the realm of practicality, and then ubiquity. Given a security parameter κ, the de facto standard method for sampling RSA biprimes involves choosing random κ-bit numbers and subjecting them to the Miller-Rabin primality test [27, 30] until two primes are found; these primes are then multiplied to form a 2κ-bit modulus. This method suffices when a single party wishes to generate a modulus, and is permitted to know the associated factorization.

Boneh and Franklin [3, 4] initiated the study of *distributed* RSA modulus generation.[1] This problem involves a set of parties who wish to jointly sample a

[*] The full version [7] of this work is available at `http://ia.cr/2020/370`

[1] Prior works generally consider RSA *key generation* and include steps for generating shares of e and d such that $e \cdot d \equiv 1 \pmod{\varphi(N)}$. This work focuses only on

biprime in such a way that no corrupt and colluding subset (below some defined threshold size) can learn the biprime's factorization.

It is clear that applying generic multiparty computation (MPC) techniques to the standard sampling algorithm yields an impractical solution: implementing the Miller-Rabin primality test requires repeatedly computing $a^{p-1} \bmod p$, where p is (in this case) secret, and so such an approach would require the generic protocol to evaluate a circuit containing many modular exponentiations over κ bits each. Instead, Boneh and Franklin [3, 4] constructed a new biprimality test that generalizes Miller-Rabin and avoids computing modular exponentiations with secret moduli. Their test carries out all exponentiations modulo the public biprime N, and this allows the exponentiations to be performed locally by the parties. Furthermore, they introduced a three-phase structure for the overall sampling protocol, which subsequent works have embraced:

1. **Prime Candidate Sieving**: candidate values for p and q are sampled jointly in secret-shared form, and a weak-but-cheap form of trial division sieves them, culling candidates with small factors.
2. **Modulus Reconstruction**: $N \coloneqq p \cdot q$ is securely computed and revealed.
3. **Biprimality Testing**: using a distributed protocol, N is tested for biprimality. If N is not a biprime, then the process is repeated.

The seminal work of Boneh and Franklin considered the semi-honest n-party setting with an honest majority of participants. Many extensions and improvements followed (as detailed in Section 1.3), the most notable of which (for our purposes) are two recent works that achieve malicious security against a dishonest majority. In the first, Hazay et al. [19, 20] proposed an n-party protocol in which both sieving and modulus reconstruction are achieved via additively homomorphic encryption. Specifically, they rely upon both ElGamal and Paillier encryption, and in order to achieve malicious security, they use zero-knowledge proofs for a variety of relations over the ciphertexts. Thus, their protocol represents a substantial advancement in terms of its security guarantee, but this comes at the cost of additional complexity assumptions and an intricate proof, and also at substantial concrete cost, due to the use of many custom zero-knowledge proofs.

The subsequent protocol of Frederiksen et al. [16] (the second recent work of note) relies mainly on oblivious transfer (OT), which they use to perform both sieving and, via Gilboa's classic multiplication protocol [17], modulus reconstruction. They achieved malicious security using the folklore technique in which a "Proof of Honesty" is evaluated as the last step and demonstrated practicality by implementing their protocol; however, it is not clear how to extend their approach to more than two parties in a straightforward way. Moreover, their approach to sieving admits selective-failure attacks, for which they account by

the task of sampling the RSA modulus N. Prior techniques can be applied to sample (e, d) after sampling N, and the distributed generation of an RSA modulus has standalone applications, such as for generating the trusted setup required by verifiable delay functions [28, 35]; consequently, we omit further discussion of e and d.

including some leakage in the functionality. It also permits a malicious adversary to selectively and *covertly* induce false negatives (i.e., force the rejection of true biprimes after the sieving stage), a property that is again modeled in their functionality. In conjunction, these attributes degrade *security*, because the adversary can rejection-sample biprimes based on the additional leaked information, and *efficiency*, because ruling out malicious false-negatives involves running sufficiently many instances to make the probability of statistical failure in all instances negligible.

Thus, given the current state of the art, it remains unclear whether one can sample an RSA modulus among two parties (one being malicious) without leaking additional information or permitting covert rejection sampling, or whether one can sample an RSA modulus among many parties (all but one being malicious) without involving heavy cryptographic primitives such as additively homomorphic encryption, and their associated performance penalties. In this work, we present a protocol which efficiently achieves both tasks.

1.1 Results and Contributions

A Clean Functionality. We define $\mathcal{F}_{\mathsf{RSAGen}}$, a simple, natural functionality for sampling biprimes from the same well-known distribution used by prior works [4, 20, 16], with no leakage or conflation of sampling failures with adversarial behavior.

A Modular Protocol, with Natural Assumptions. We present a protocol π_{RSAGen} in the $(\mathcal{F}_{\mathsf{AugMul}}, \mathcal{F}_{\mathsf{Biprime}})$-hybrid model, where $\mathcal{F}_{\mathsf{AugMul}}$ is an augmented multiplier functionality and $\mathcal{F}_{\mathsf{Biprime}}$ is a biprimality-testing functionality, and prove that it UC-realizes $\mathcal{F}_{\mathsf{RSAGen}}$ in the malicious setting, assuming the hardness of factoring. More specifically, we prove:

Theorem 1.1. (Main Security Theorem, Informal)**.** In the presence of a PPT malicious adversary corrupting any subset of parties, $\mathcal{F}_{\mathsf{RSAGen}}$ can be securely computed with abort in the $(\mathcal{F}_{\mathsf{AugMul}}, \mathcal{F}_{\mathsf{Biprime}})$-hybrid model, assuming the hardness of factoring.

Additionally, because our security proof relies upon the hardness of factoring only when the adversary cheats, we find to our surprise that our protocol achieves *perfect* security against semi-honest adversaries.

Theorem 1.2. (Semi-Honest Security Theorem, Informal)**.** In the presence of a computationally unbounded semi-honest adversary corrupting any subset of parties, $\mathcal{F}_{\mathsf{RSAGen}}$ can be computed with perfect security in the $(\mathcal{F}_{\mathsf{AugMul}}, \mathcal{F}_{\mathsf{Biprime}})$-hybrid model.

Supporting Functionalities and Protocols. We define $\mathcal{F}_{\mathsf{Biprime}}$, a simple, natural functionality for biprimality testing, and show that it is UC-realized in the semi-honest setting by a well known protocol of Boneh and Franklin [4], and in the malicious setting by a derivative of the protocol of Frederiksen et al. [16]. We believe this dramatically simplifies the composition of these two protocols, and as a consequence, leads to a simpler analysis. Either protocol can be based exclusively upon oblivious transfer.

We also define $\mathcal{F}_{\mathsf{AugMul}}$, a functionality for sampling and multiplying secret-shared values in a special form derived from the Chinese Remainder Theorem. In the context of π_{RSAGen}, this functionality allows us to efficiently sample numbers in a specific range, with no small factors, and then compute their product. We prove that it can be UC-realized exclusively from oblivious transfer, using derivatives of well-known multiplication protocols [13, 14].

Asymptotic Efficiency. We perform an asymptotic analysis of our composed protocols and find that our semi-honest protocol is a factor of $\kappa/\log\kappa$ more bandwidth-efficient than that of Frederiksen et al. [16]. Our malicious protocol is a factor of κ/s more efficient than theirs in the optimistic case (when parties follow the protocol), and a factor of κ more efficient when parties deviate from the protocol. Recall that κ is the bit-length of the primes p and q, and s is a statistical security parameter. Frederiksen et al. claim in turn that their protocol is strictly superior to the protocol of Hazay et al. [20] with respect to asymptotic bandwidth performance.

Concrete Efficiency. We perform a closed-form concrete analysis of our protocol (with some optimizations, including the use of random oracles), and find that in terms of communication, it outperforms the protocol of Frederiksen et al. (the most efficient prior work) by a factor of roughly five in the presence of worst-case malicious adversaries, and by a factor of eighty or more in the semi-honest setting.

1.2 Overview of Techniques

Constructive Sampling and Efficient Modulus Reconstruction. Most prior works use rejection sampling to generate a pair of candidate primes, and then multiply those primes together in a separate step. Specifically, they sample a shared value $p \leftarrow [0, 2^\kappa)$ uniformly, and then run a trial-division protocol repeatedly, discarding both the value and the work that has gone into testing it if trial division fails. This represents a substantial amount of wasted work in expectation. Furthermore, Frederiksen et al. [16] report that multiplication of candidates after sieving accounts for two thirds of their concrete cost.

We propose a different approach that leverages the Chinese Remainder Theorem (CRT) to *constructively* sample a pair of candidate primes and multiply them together efficiently. A similar sieving approach (in spirit) was initially formulated as an optimization in a different setting by Malkin et al. [26]. The CRT implies an isomorphism between a set of values, each in a field modulo a distinct prime, and a single value in a ring modulo the product of those primes (i.e., $\mathbb{Z}_{m_1} \times \ldots \times \mathbb{Z}_{m_\ell} \simeq \mathbb{Z}_{m_1 \cdot \ldots \cdot m_\ell}$). We refer to the set of values as the *CRT form* or *CRT representation* of the single value to which they are isomorphic. We formulate a sampling mechanism based on this isomorphism as follows: for each of the first $O(\kappa/\log\kappa)$ odd primes, the parties jointly (and efficiently) sample shares of a value that is nonzero modulo that prime. These values are the shared CRT form of a single κ-bit value that is guaranteed to be indivisible by any prime in the set sampled against. For technical reasons, we sample two such candidates simultaneously.

Rather than converting pairs of candidate primes from CRT form to standard form, and then multiplying them, we instead multiply them component-wise in CRT form, and then convert the product to standard form to complete the protocol. This effectively replaces a single "full-width" multiplication of size κ with $O(\kappa/\log\kappa)$ individual multiplications, each of size $O(\log\kappa)$. We intend to perform multiplication via an OT-based protocol, and the computation and communication complexity of such protocols grows at least with the square of their input length, even in the semi-honest case [17]. Thus in the semi-honest case, our approach yields an overall complexity of $O(\kappa\log\kappa)$, as compared to $O(\kappa^2)$ for a single full-width multiplication. In the malicious case, combining the best known multiplier construction [13, 14] with the most efficient known OT extension scheme [5] yields a complexity that also grows with the product of the input length and a statistical parameter s, and so our approach achieves an overall complexity of $O(\kappa\log\kappa+\kappa\cdot s)$, as compared to $O(\kappa^2+\kappa\cdot s)$ for a single full-width malicious multiplication. Via closed-form analysis, we show that this asymptotic improvement is also reflected concretely.

Achieving Security with Abort Efficiently. The fact that we sample primes in CRT form also plays a crucial role in our security analysis. Unlike the work of Frederiksen et al. [16], our protocol achieves the *standard*, intuitive notion of security with abort: the adversary can instruct the functionality to abort regardless of whether a biprime is successfully sampled, and the honest parties are always made aware of such adversarial aborts. There is, in other words, absolutely no conflation of sampling failures with adversarial behavior. For the sake of efficiency, our protocol permits the adversary to cheat prior to biprimality testing, and then rules out such cheats retroactively using one of two strategies. In the case that a biprime is successfully sampled, adversarial behavior is ruled out retroactively in a privacy-preserving fashion using well-known but moderately expensive techniques, which is tolerable only because it need not be done more than once. In the case that a sampled value is not a biprime, however, the inputs to the sampling protocol are revealed to all parties, and the retroactive check is carried out in the clear. Proving the latter approach secure turns out to be surprisingly subtle.

The challenge arises from the fact that the simulator must simulate the protocol transcript for the OT-multipliers on behalf of the honest parties without knowing their inputs. Later, if the sampling-protocol inputs are revealed, the simulator must "explain" how the simulated transcript is consistent with the true inputs of the honest parties. Specifically, in maliciously secure OT-multipliers of the sort we use [13, 14], the OT receiver (Bob) uses a high-entropy encoding of his input, and the sender (Alice) can, by cheating, learn a one-bit predicate of this encoding. Before Bob's true input is known to the simulator, it must pick an encoding at random. When Bob's input is revealed, the simulator must find an encoding of his input which is consistent with the predicate on the random encoding that Alice has learned. This task closely resembles solving a random instance of subset sum.

We are able to overcome this difficulty because our multiplications are performed component-wise over CRT-form representations of their operands. Because each component is of size $O(\log \kappa)$ bits, the simulator can simply guess random encodings until it finds one that matches the required constraints. We show that this strategy succeeds in strict polynomial time, and that it induces a distribution statistically close to that of the real execution.

This form of "privacy-free" malicious security (wherein honest behavior is verified at the cost of sacrificing privacy) leads to considerable efficiency gains in our case: it is up to a multiplicative factor of s (the statistical parameter) cheaper than the privacy-preserving check used in the case that a candidate passes the biprimality test (and the one used in prior OT-multipliers [13, 14]). Since most candidates fail the biprimality test, using the privacy-free check to verify that they were generated honestly results in substantial savings.

Biprimality Testing as a Black Box. We specify a functionality for biprimality testing, and prove that it can be realized by a maliciously secure version of the Boneh-Franklin biprimality test. Our functionality has a clean interface and does not, for example, require its inputs to be authenticated to ensure that they were actually generated by the sampling phase of the protocol. The key insight that allows us to achieve this level of modularity is a reduction to factoring: if an adversary is able to cheat by supplying incorrect inputs to the biprimality test, relative to a candidate biprime N, and the biprimality test succeeds, then we show that the adversary can be used to factor biprimes. We are careful to rely on this reduction only in the case that N is actually a biprime, and to prevent the adversary from influencing the distribution of candidates.

The Benefits of Modularity. We claim as a contribution the fact that modularity has yielded both a simpler protocol description and a reasonably simple proof of security. We believe that this approach will lead to derivatives of our work with stronger security properties or with security against stronger adversaries. As a first example, we prove that a semi-honest version of our protocol (differing only in that it omits the retroactive consistency check in the protocol's final step) achieves *perfect* security. We furthermore observe that in the malicious setting, instantiating $\mathcal{F}_{\mathsf{Biprime}}$ and $\mathcal{F}_{\mathsf{AugMul}}$ with security against *adaptive* adversaries yields an RSA modulus sampling protocol that is adaptively secure.

Similarly, only minor adjustments to the main protocol are required to achieve security with *identifiable abort* [22, 11]. If we assume that the underlying functionalities $\mathcal{F}_{\mathsf{AugMul}}$ and $\mathcal{F}_{\mathsf{Biprime}}$ are instantiated with identifiable abort, then it remains only to ensure the use of consistent inputs across these functionalities, and to detect which party has provided inconsistent inputs if an abort occurs. This can be accomplished by augmenting $\mathcal{F}_{\mathsf{Biprime}}$ with an additional interface for revealing the input values provided by all the parties upon global request (e.g., when the candidate N is not a biprime). Given identifiable abort, it is possible to guarantee output delivery in the presence of up to $n-1$ corruptions via standard techniques, although the functionality must be weakened to allow the adversary to reject one biprime per corrupt party.[2] A proof of this extension is beyond the scope of this work; we focus instead on the advancements our framework yields in the setting of security with abort.

1.3 Additional Related Work

Frankel, MacKenzie, and Yung [15] adjusted the protocol of Boneh and Franklin [3] to achieve security against malicious adversaries in the honest-majority setting. Their main contribution was the introduction of a method for robust distributed multiplication over the integers. Cocks [8] proposed a method for multiparty RSA key generation under heuristic assumptions, and later attacks by Coppersmith (see [9]) and Joye and Pinch [23] suggest this method may be insecure. Poupard and Stern [29] presented a maliciously secure two-party protocol based on oblivious transfer. Gilboa [17] achieved improved efficiency in the semi-honest two-party model, and introduced a novel method for multiplication from oblivious transfer, from which our own multipliers ultimately derive.

Malkin, Wu, and Boneh [26] implemented the protocol of Boneh and Franklin and introduced an optimized sieving method similar in spirit to ours. In particular, their protocol generates sharings of random values in $\mathbb{Z}_M^*$ (where M is a primorial modulus) during the sieving phase, instead of naïve random candidates for primes p and q. However, their method produces *multiplicative* sharings of p and q, which are converted into additive sharings for biprimality testing via an honest-majority, semi-honest protocol. This conversion requires rounds linear in the party count, and it is unclear how to adapt it to tolerate a malicious majority of parties without a significant performance penalty.

Algesheimer, Camenish, and Shoup [1] described a method to compute a distributed version of the Miller-Rabin test: they used secret-sharing conversion techniques reliant on approximations of $1/p$ to compute exponentiations modulo a shared p. However, each invocation of their Miller-Rabin test still has complexity in $O(\kappa^3)$ per party, and their overall protocol has communication complexity in $O(\kappa^5/\log^2\kappa)$, with $\Theta(\kappa)$ rounds of interaction. Concretely, Damgård and Mikkelsen [12] estimate that 10000 rounds are required to sample a 2000-bit biprime using this method. Damgård and Mikkelsen also extended their work to improve both its communication and round complexity by several orders of magnitude, and to achieve malicious security in the honest-majority setting. Their protocol is at least a factor of $O(\kappa)$ better than that of Algesheimer, Camenish, and Shoup, but it still requires *hundreds* of rounds. We were not able to compute an explicit complexity analysis of their approach.

[2] The folklore technique involves invoking the protocol iteratively, each iteration eliminating one corrupt party until a success occurs. For a constant fraction of corruptions, the implied linear round complexity overhead can be reduced to super-constant (e.g., $\log^* n$) [10].

1.4 Organization

Basic notation and background information are given in Section 2. Our ideal biprime-sampling functionality is defined in Section 3, and we give a protocol that realizes it in Section 4. In Section 5, we present our biprimality-testing protocol. In the full version [7] of this work, we give an efficiency analysis, full proofs of security, and the details of our multiplication protocol.

2 Preliminaries

Notation. We use $=$ for equality, $:=$ for assignment, $\leftarrow$ for sampling from a distribution, $\equiv$ for congruence, $\approx_{\mathrm{c}}$ for computational indistinguishability, and $\approx_{\mathrm{s}}$ for statistical indistinguishability. In general, single-letter variables are set in *italic* font, multi-letter variables and function names are set in sans-serif font, and string literals are set in `slab-serif` font. We use mod to indicate the modulus operator, while $\pmod{m}$ at the end of a line indicates that all equivalence relations on that line are to be taken over the integers modulo m. By convention, we parameterize computational security by the bit-length of each prime in an RSA biprime; we denote this length by κ throughout. We use s to represent the statistical parameter. Where concrete efficiency is concerned, we introduce a second computational security parameter, λ, which represents the length of a symmetric key of equivalent strength to a biprime of length 2κ.[3] κ and λ must vary together, and a recommendation for the relationship between them has been laid down by NIST [2].

Vectors and arrays are given in bold and indexed by subscripts; thus $\mathbf{x}_i$ is the i^{th} element of the vector $\mathbf{x}$, which is distinct from the scalar variable x. When we wish to select a row or column from a two-dimensional array, we place a $*$ in the dimension along which we are not selecting. Thus $\mathbf{y}_{*,j}$ is the j^{th} column of matrix $\mathbf{y}$, and $\mathbf{y}_{j,*}$ is the j^{th} row. We use $\mathcal{P}_i$ to denote the party with index i, and when only two parties are present, we refer to them as Alice and Bob. Variables may often be subscripted with an index to indicate that they belong to a particular party. When arrays are owned by a party, the party index always comes first. We use $|x|$ to denote the bit-length of x, and $|\mathbf{y}|$ to denote the number of elements in the vector $\mathbf{y}$.

Universal Composability. We prove our protocols secure in the Universal Composability (UC) framework, and use standard UC notation. We refer the reader to Canetti [6] for further details. In functionality descriptions, we leave some standard bookkeeping elements implicit. For example, we assume that the functionality aborts if a party tries to reuse a session identifier inappropriately, send messages out of order, etc. For convenience, we provide a function GenSID, which takes *any* number of arguments and deterministically derives a unique Session ID from those arguments.

[3] In other words, a biprime of length 2κ provides λ bits of security.

Chinese Remainder Theorem. The Chinese Remainder Theorem (CRT) defines an isomorphism between a set of residues modulo a set of respective coprime values and a single value modulo the product of the same set of coprime values. This forms the basis of our sampling procedure.

Theorem 2.1. (CRT)**.** Let $\mathbf{m}$ be a vector of coprime positive integers and let $\mathbf{x}$ be a vector of numbers such that $|\mathbf{m}| = |\mathbf{x}| = \ell$ and $0 \leq \mathbf{x}_j < \mathbf{m}_j$ for all $j \in [\ell]$, and finally let $M \coloneqq \prod_{j\in[\ell]} \mathbf{m}_j$. Under these conditions there exists a unique value y such that $0 \leq y < M$ and $y \equiv \mathbf{x}_j \pmod{\mathbf{m}_j}$ for every $j \in [\ell]$.

We refer to $\mathbf{x}$ as the *CRT form* of y with respect to $\mathbf{m}$. For completeness, we give the CRTRecon algorithm, which finds the unique y given $\mathbf{m}$ and $\mathbf{x}$.

Algorithm 2.2. $\mathsf{CRTRecon}(\mathbf{m}, \mathbf{x})$

1. With $\ell \coloneqq |\mathbf{m}|$, compute $M = \prod_{j\in[\ell]} \mathbf{m}_j$.
2. For $j \in [\ell]$, compute $\mathbf{a}_j \coloneqq M/\mathbf{m}_j$ and find $\mathbf{b}_j$ satisfying $\mathbf{a}_j \cdot \mathbf{b}_j \equiv 1 \pmod{\mathbf{m}_j}$ using the Extended Euclidean Algorithm (see Knuth [25]).
3. Output $y \coloneqq \sum_{j\in[\ell]} \mathbf{a}_j \cdot \mathbf{b}_j \cdot \mathbf{x}_j \bmod M$.

3 Assumptions and Ideal Functionality

We begin this section by discussing the distribution of biprimes from which we sample, and thus the precise factoring assumption that we make, and then we give an efficient sampling algorithm and an ideal functionality that computes it.

3.1 Factoring Assumptions

The standard factoring experiment (Experiment 3.1) as formalized by Katz and Lindell [24] is parametrized by an adversary $\mathcal{A}$ and a biprime-sampling algorithm GenModulus. On input 1^κ, this algorithm returns (N, p, q), where $N = p \cdot q$, and p and q are κ-bit primes.[4]

Experiment 3.1. $\mathsf{Factor}_{\mathcal{A},\mathsf{GenModulus}}(\kappa)$

1. Run $(N, p, q) \leftarrow \mathsf{GenModulus}(1^\kappa)$.
2. Send N to $\mathcal{A}$, and receive $p', q' > 1$ in return.
3. Output 1 if and only if $p' \cdot q' = N$.

In many cryptographic applications, $\mathsf{GenModulus}(1^\kappa)$ is defined to sample p and q *uniformly* from the set of primes in the range $[2^{\kappa-1}, 2^\kappa)$ [18], and the

[4] Technically, Katz and Lindell specify that sampling failures are permitted with negligible probability, and require GenModulus to run in strict polynomial time. We elide this detail.

factoring assumption with respect to this common $\mathsf{GenModulus}$ function states that for every PPT adversary $\mathcal{A}$ there exists a negligible function negl such that

$$\Pr\left[\mathsf{Factor}_{\mathcal{A},\mathsf{GenModulus}}(\kappa) = 1\right] \leq \mathsf{negl}(\kappa).$$

Because *efficiently* sampling according to this uniform biprime distribution is difficult in a multiparty context, most prior works sample according to a different distribution, and thus using the moduli they produce requires a slightly different factoring assumption than the traditional one. In particular, several recent works use a distribution originally proposed by Boneh and Franklin [4], which is well-adapted to multiparty sampling. Our work follows this pattern.

Boneh and Franklin's distribution is defined by the sampling algorithm BFGM, which takes as an additional parameter the number of parties n. The algorithm samples n integer shares, each in the range $[0, 2^{\kappa-\log n})$, and sums these shares to arrive at a candidate prime. This does *not* induce a uniform distribution on the set of κ-bit primes. Furthermore, BFGM only samples individual primes p or q that have $p \equiv q \equiv 3 \pmod 4$, in order to facilitate efficient distributed primality testing, and it filters out the subset of otherwise-valid moduli $N = p \cdot q$ that have $p \equiv 1 \pmod q$ or $q \equiv 1 \pmod p$.[5]

Algorithm 3.2. $\mathsf{BFGM}(\kappa, n)$

1. For $i \in [n]$, sample $p_i \leftarrow \left[0, 2^{\kappa-\log n}\right)$ and $q_i \leftarrow \left[0, 2^{\kappa-\log n}\right)$ subject to $p_1 \equiv q_1 \equiv 3 \pmod 4$ and $p_j \equiv q_j \equiv 0 \pmod 4$ for $j \in [2, n]$.
2. Compute

$$p := \sum_{i\in[n]} p_i \qquad \text{and} \qquad q := \sum_{i\in[n]} q_i \qquad \text{and} \qquad N := p \cdot q$$

3. If $\gcd(N, p + q - 1) = 1$, and both p and q are primes, then output $(N, \{(p_i, q_i)\}_{i\in[n]})$. Otherwise, repeat this procedure from Step 1.

Any protocol whose security depends upon the hardness of factoring moduli output by our protocol (including our protocol itself) must rely upon the assumption that for every PPT adversary $\mathcal{A}$,

$$\Pr\left[\mathsf{Factor}_{\mathcal{A},\mathsf{BFGM}}(\kappa, n) = 1\right] \leq \mathsf{negl}(\kappa)$$

3.2 The Distributed Biprime-Sampling Functionality

Unfortunately, our ideal modulus-sampling functionality cannot merely call BFGM; we wish our functionality to run in *strict* polynomial time, whereas the running time of BFGM is only *expected* polynomial. Thus, we define a new sampling algorithm, $\mathsf{CRTSample}$, which might fail, but conditioned on success

[5] Boneh and Franklin actually propose two variations, one of which has no false negatives; we choose the other variation, as it leads to a more efficient sampling protocol.

outputs samples statistically close to BFGM.[6] Furthermore, we give CRTSample a specific distribution of failures that is tied to the design of our protocol. As a second concession to our protocol design (and following Hazay et al. [20]), CRTSample takes as input up to $n-1$ integer shares of p and q, arbitrarily determined by the adversary, while the remaining shares are sampled randomly. We begin with a few useful notions.

Definition 3.3. (Primorial Number)**.** The i^{th} *primorial number* is defined to be the product of the first i prime numbers.

Definition 3.4. ((κ, n)-Near-Primorial Vector)**.** Let ℓ be the largest number such that the ℓ^{th} primorial number is less than $2^{\kappa - \log n - 1}$, and let $\mathbf{m}$ be a vector of length ℓ such that $\mathbf{m}_1 = 4$ and $\mathbf{m}_2, \ldots, \mathbf{m}_\ell$ are the odd factors of the ℓ^{th} primorial number, in ascending order. $\mathbf{m}$ is the unique (κ, n)*-near-primorial vector*.

Definition 3.5. ($\mathbf{m}$-Coprimality)**.** Let $\mathbf{m}$ be a vector of integers. An integer x is $\mathbf{m}$*-coprime* if and only if it is not divisible by any $\mathbf{m}_i$ for $i \in [|\mathbf{m}|]$.

Algorithm 3.6. CRTSample$(\kappa, n, \{(p_i, q_i)\}_{i \in \mathbf{P}^*})$

```
1. Let $\mathbf{m}$ be the $(\kappa, n)$-near-primorial vector, with length $\ell$, and let $M$ be the product of $\mathbf{m}$.
2. For $i \in [n] \setminus \mathbf{P}^*$, sample $p_i \leftarrow [0, M)$ and $q_i \leftarrow [0, M)$ subject to
       $$p_i \equiv q_i \equiv \begin{cases} 3 \pmod 4 & \text{if } i = 1 \\ 0 \pmod 4 & \text{if } i \neq 1 \end{cases}$$
   and subject to $p$ and $q$ being $\mathbf{m}$-coprime, where
       $$p := \sum_{i \in [n]} p_i \qquad \text{and} \qquad q := \sum_{i \in [n]} q_i$$
   are computed over the integers.
3. If $\gcd(p \cdot q, p + q - 1) = 1$, and if both $p$ and $q$ are primes, and if $p \equiv q \equiv 3 \pmod 4$, then output $(\texttt{success}, p, q)$; otherwise, output $(\texttt{failure}, p, q)$.
```

Boneh and Franklin [4, Lemma 2.1] showed that knowledge of $n-1$ integer shares of the factors p and q does not give the adversary any meaningful advantage in factoring biprimes from the distribution produced by BFGM and, by extension, CRTSample. Hazay et al. [20, Lemma 4.1] extended this argument to the malicious setting, wherein the adversary is allowed to choose its own shares.

[6] CRTSample never outputs biprimes with factors smaller than κ, whereas BFGM outputs such biprimes with negligible probability. The discrepancy of share ranges can be remedied by using non-integer values of κ with BFGM.

Lemma 3.7. ([4, 20]). Let $n < \kappa$ and let $(\mathcal{A}_1, \mathcal{A}_2)$ be a pair of PPT algorithms. For $(\mathsf{state}, \{(p_i, q_i)\}_{i \in [n-1]}) \leftarrow \mathcal{A}_1(1^\kappa, 1^n)$, let N be a biprime sampled by running $\mathsf{CRTSample}(\kappa, n, \{(p_i, q_i)\}_{i \in [n-1]})$. If $\mathcal{A}_2(\mathsf{state}, N)$ outputs the factors of N with probability at least $1/\kappa^d$, then there exists an expected-polynomial-time algorithm $\mathcal{B}$ that succeeds with probability $1/2^4 n^3 \kappa^d$ in the experiment $\mathsf{Factor}_{\mathcal{B},\mathsf{BFGM}}(\kappa, n)$.

Multiparty functionality. Our ideal functionality $\mathcal{F}_{\mathsf{RSAGen}}$ is a natural embedding of $\mathsf{CRTSample}$ in a multiparty functionality: it receives inputs $\{(p_i, q_i)\}_{i \in \mathbf{P}^*}$ from the adversary and runs a single iteration of $\mathsf{CRTSample}$ with these inputs when invoked. It either outputs the corresponding modulus $N := p \cdot q$ if it is valid, or indicates that a sampling failure has occurred. Running a single iteration of $\mathsf{CRTSample}$ per invocation of $\mathcal{F}_{\mathsf{RSAGen}}$ enables significant freedom in the use of $\mathcal{F}_{\mathsf{RSAGen}}$, because it can be composed in different ways to tune the trade-off between resource usage and execution time. It also simplifies the analysis of the protocol π_{RSAGen} that realizes $\mathcal{F}_{\mathsf{RSAGen}}$, because the analysis is made independent of the success rate of the sampling procedure.

The functionality may not deliver N to the honest parties for one of two reasons: either $\mathsf{CRTSample}$ failed to sample a biprime, or the adversary caused the computation to abort. In either case, the honest parties are informed of the cause of the failure, and consequently the adversary is unable to conflate the two cases. This is essentially the standard notion of security with abort, applied to the multiparty computation of the $\mathsf{CRTSample}$ algorithm. In both cases, the p and q output by $\mathsf{CRTSample}$ are given to the adversary. This leakage simplifies our proof considerably, and we consider it benign, since the honest parties never receive (and therefore cannot possibly use) N.

Functionality 3.8. $\mathcal{F}_{\mathsf{RSAGen}}(\kappa, n)$**. Distributed Biprime Sampling**

This n-party functionality attempts to sample an RSA modulus with prime length κ, and interacts directly with an ideal adversary $\mathcal{S}$ who corrupts the parties indexed by $\mathbf{P}^*$. Let M be the largest number such that $M/2$ is a primorial number and $M < 2^{\kappa - \log n}$.

Sampling: On receiving $(\texttt{sample}, \mathsf{sid})$ from each party $\mathcal{P}_i$ for $i \in [n] \setminus \mathbf{P}^*$ and $(\texttt{adv-sample}, \mathsf{sid}, i, p_i, q_i)$ from $\mathcal{S}$ for $i \in \mathbf{P}^*$, if $0 \leq p_i < M$ and $0 \leq q_i < M$ for all $i \in \mathbf{P}^*$, then run $\mathsf{CRTSample}(\kappa, n, \{(p_i, q_i)\}_{i \in \mathbf{P}^*})$, and receive as a result either $(\texttt{success}, p, q)$ or $(\texttt{failure}, p, q)$.

- If $p \not\equiv 3 \pmod 4$ or $q \not\equiv 3 \pmod 4$, then send $(\texttt{factors}, \mathsf{sid}, p, q)$ to $\mathcal{S}$ and abort, informing all parties in an adversarially delayed fashion.
- If $p \equiv q \equiv 3 \pmod 4$, and the result was $\texttt{failure}$, then store $(\texttt{non-biprime}, \mathsf{sid}, p, q)$ in memory and send $(\texttt{factors}, \mathsf{sid}, p, q)$ to $\mathcal{S}$.
- If $p \equiv q \equiv 3 \pmod 4$, and the result was $\texttt{success}$, then compute $N := p \cdot q$, store $(\texttt{biprime}, \mathsf{sid}, N, p, q)$ in memory, and send $(\texttt{biprime}, \mathsf{sid}, N)$ to $\mathcal{S}$.

Output: On receiving either $(\texttt{proceed}, \mathsf{sid})$ or $(\texttt{cheat}, \mathsf{sid})$ from $\mathcal{S}$, if $(\texttt{biprime}, \mathsf{sid}, N, p, q)$ or $(\texttt{non-biprime}, \mathsf{sid}, p, q)$ exists in memory,

- If $\texttt{proceed}$ was received, then send either $(\texttt{biprime}, \mathsf{sid}, N)$ or $(\texttt{non-biprime}, \mathsf{sid})$ to all parties as adversarially delayed output, as appropriate. Terminate successfully.
- If $\texttt{cheat}$ was received, then abort, notifying all parties in an adversarially delayed fashion, and send $(\texttt{factors}, \mathsf{sid}, p, q)$ directly to $\mathcal{S}$.

Regardless, ignore all further instructions with this sid.

4 The Distributed Biprime-Sampling Protocol

In this section, we present the distributed biprime-sampling protocol π_{RSAGen}, with which we realize $\mathcal{F}_{\mathsf{RSAGen}}$. We begin with a high-level overview, and then in Section 4.2, we formally define the two ideal functionalities on which our protocol relies, after which in Section 4.3 we give the protocol itself. In Section 4.4, we present proof sketches of semi-honest and malicious security.

4.1 High-Level Overview

As described in the Introduction, our protocol derives from that of Boneh and Franklin [4], the main technical differences relative to other recent Boneh-Franklin derivatives [20, 16] being the modularity with which it is described and proven, and the use of CRT-based sampling. Our protocol has three main phases, which we now describe in sequence.

Candidate Sieving. In the first phase of our protocol, the parties jointly sample two κ-bit candidate primes p and q without any small factors, and multiply them to learn their product N. Our protocol achieves these tasks in a unified, integrated way, thanks to the Chinese Remainder Theorem.

Consider a prime m and a set of shares x_i for $i \in [n]$ over the field $\mathbb{Z}_m$. As in the description of $\mathsf{CRTRecon}$, let a and b be defined such that $a \cdot b \equiv 1 \pmod{m}$, and let M be an integer. Observe that if m divides M, then

$$\sum_{i\in[n]} x_i \not\equiv 0 \pmod{m} \quad \implies \quad \sum_{i\in[n]} a \cdot b \cdot x_i \bmod M \not\equiv 0 \pmod{m} \qquad (1)$$

Now consider a vector of coprime integers $\mathbf{m}$ of length ℓ, and let M be their product. Let $\mathbf{x}$ be a vector, each element secret shared over the fields defined by the corresponding element of $\mathbf{m}$, and let $\mathbf{a}$ and $\mathbf{b}$ be defined as in $\mathsf{CRTRecon}$ (i.e., $\mathbf{a}_j := M/\mathbf{m}_j$ and $\mathbf{a}_j \cdot \mathbf{b}_j \equiv 1 \pmod{\mathbf{m}_j}$). We can see that for any $k, j \in [\ell]$ such that $k \neq j$,

$$\mathbf{a}_j \equiv 0 \pmod{\mathbf{m}_k} \quad \implies \quad \sum_{i\in[n]} \mathbf{a}_j \cdot \mathbf{b}_j \cdot \mathbf{x}_{i,j} \bmod M \equiv 0 \pmod{\mathbf{m}_k} \qquad (2)$$

and the conjunction of Equations 1 and 2 gives us

$$\sum_{j\in[\ell]}\sum_{i\in[n]} \mathbf{a}_j \cdot \mathbf{b}_j \cdot \mathbf{x}_{i,j} \bmod M \equiv \sum_{i\in[n]} \mathbf{x}_{i,k} \pmod{\mathbf{m}_k}$$

for all $k \in [\ell]$. Observe that this holds regardless of which order we perform the sums in, and regardless of whether the mod M operation is done at the end, or between the two sums, or not at all.

It follows then that we can sample n shares for an additive secret sharing over the integers of a κ-bit value x (distributed between 0 and $n \cdot M$) by choosing $\mathbf{m}$ to be the (κ, n)-near-primorial vector (per Definition 3.4), instructing each party $\mathcal{P}_i$ for $i \in [n]$ to pick $\mathbf{x}_{i,j}$ locally for $j \in [\ell]$ such that $0 \leq \mathbf{x}_{i,j} < \mathbf{m}_j$, and then instructing each party to *locally* reconstruct $x_i := \mathsf{CRTRecon}(\mathbf{m}, \mathbf{x}_{i,*})$, its share of x. It furthermore follows that if the parties can contrive to ensure that

$$\sum_{i\in[n]} \mathbf{x}_{i,j} \not\equiv 0 \pmod{\mathbf{m}_j} \tag{3}$$

for $j \in [\ell]$, then x will not be divisible by any prime in $\mathbf{m}$.

Observe next that if the parties sample two shared vectors $\mathbf{p}$ and $\mathbf{q}$ as above (corresponding to the candidate primes p and q) and compute a shared vector $\mathbf{N}$ of identical dimension such that

$$\sum_{i\in[n]} \mathbf{p}_{i,j} \cdot \sum_{i\in[n]} \mathbf{q}_{i,j} \equiv \sum_{i\in[n]} \mathbf{N}_{i,j} \pmod{\mathbf{m}_j} \tag{4}$$

for all $j \in [\ell]$, then it follows that

$$\sum_{i\in[n]} \mathsf{CRTRecon}(\mathbf{m}, \mathbf{p}_{i,*}) \cdot \sum_{i\in[n]} \mathsf{CRTRecon}(\mathbf{m}, \mathbf{q}_{i,*}) = \sum_{i\in[n]} \mathsf{CRTRecon}(\mathbf{m}, \mathbf{N}_{i,*})$$

and from this it follows that the parties can calculate integer shares of $N = p \cdot q$ by multiplying $\mathbf{p}$ and $\mathbf{q}$ together element-wise using a modular-multiplication protocol for linear secret shares, and then locally running $\mathsf{CRTRecon}$ on the output to reconstruct N. In fact, our sampling protocol makes use of a special functionality $\mathcal{F}_{\mathsf{AugMul}}$, which samples $\mathbf{p}$, $\mathbf{q}$, and $\mathbf{N}$ simultaneously such that the conditions in Equations 3 and 4 hold.

There remains one problem: our vector $\mathbf{m}$ was chosen for sampling integer-shared values between 0 and $n \cdot M$ (with each share no larger than M), but N might be as large as $n^2 \cdot M^2$. In order to avoid wrapping during reconstruction of N, we must reconstruct with respect to a *larger* vector of primes (while continuing to sample with respect to a smaller one). Let $\mathbf{m}$ now be of length ℓ', and let ℓ continue to denote the length of the prefix of $\mathbf{m}$ with respect to which sampling is performed. After sampling the initial vectors $\mathbf{p}$, $\mathbf{q}$, and $\mathbf{N}$, each party $\mathcal{P}_i$ for $i \in [n]$ must extend $\mathbf{p}_{i,*}$ locally to ℓ' elements, by computing

$$\mathbf{p}_{i,j} := \mathsf{CRTRecon}\left(\{\mathbf{m}_{j'}\}_{j'\in[\ell]}, \{\mathbf{p}_{j'}\}_{j'\in[\ell]}\right) \bmod \mathbf{m}_j$$

for $j \in [\ell+1, \ell']$, and then likewise for $\mathbf{q}_{i,*}$. Finally, the parties must use a modular-multiplication protocol to compute the appropriate extension of $\mathbf{N}$; from this extended $\mathbf{N}$, they can reconstruct shares of $N = p \cdot q$. They swap these shares, and thus each party ends the Sieving phase of our protocol with a candidate biprime N and an integer share of each of its factors, p_i and q_i.

Each party completes the first phase by performing a local trial division to check if N is divisible by any prime smaller than some bound B (which is a parameter of the protocol). The purpose of this step is to reduce the number of calls to $\mathcal{F}_{\mathsf{Biprime}}$ and thus improve efficiency.

Biprimality Test. The parties jointly execute a biprimality test, where every party inputs the candidate N and its shares p_i and q_i, and receives back a biprimality indicator. This phase essentially comprises a single call to a functionality $\mathcal{F}_{\mathsf{Biprime}}$, which allows an adversary to force spurious negative results, but never returns false positive results. Though this phase is simple, much of the subtlety of our proof concentrates here: we show via a reduction to factoring that cheating parties have a negligible chance to pass the biprimality test if they provide wrong inputs. This eliminates the need to authenticate the inputs in any way.

Consistency Check. To achieve malicious security, the parties must ensure that none among them cheated during the previous stages in a way that might influence the result of the computation. This is what we have previously termed the retroactive consistency check. If the biprimality test indicated that N is *not* a biprime, then the parties use a special interface of $\mathcal{F}_{\mathsf{AugMul}}$ to reveal the shares they used during the protocol, and then they verify locally and independently that p and q are not both primes. If the biprimality test indicated that N is a biprime, then the parties run a secure test (again via a special interface of $\mathcal{F}_{\mathsf{AugMul}}$) to ensure that length extensions of $\mathbf{p}$ and $\mathbf{q}$ were performed honestly. To achieve semi-honest security, this phase is unnecessary, and the protocol can end with the biprimality test.

4.2 Ideal Functionalities Used in the Protocol

Augmented Multiparty Multiplier. The augmented multiplier functionality $\mathcal{F}_{\mathsf{AugMul}}$ (Functionality 4.1) is a reactive functionality that operates in multiple phases and stores an internal state across calls. It is meant to help in manipulating CRT-form secret shares. It contains five basic interfaces.

- The `sample` interface allows the parties to sample shares of non-zero multiplication triplets over small primes. That is, given a prime m, the functionality receives a triplet (x_i, y_i, z_i) from every corrupted party $\mathcal{P}_i$, and then samples a triplet $(x_j, y_j, z_j) \leftarrow \mathbb{Z}_m^3$ for every honest $\mathcal{P}_j$ conditioned on

$$\sum_{i\in[n]} z_i \equiv \sum_{i\in[n]} x_i \cdot \sum_{i\in[n]} y_i \not\equiv 0 \pmod{m}$$

  In the context of π_{RSAGen}, this is used to sample CRT-shares of p and q.

- The `input` and `multiply` interfaces, taken together, allow the parties to load shares (with respect to some small prime modulus m) into the functionality's memory, and later perform modular multiplication on two sets of shares that are associated with the same modulus. That is, given a prime m, each party $\mathcal{P}_i$ inputs x_i and, independently, y_i, and when the parties request a product, with each corrupt party $\mathcal{P}_j$ also supplying its own an output share z_j, the functionality samples a share of z from $\mathbb{Z}_m$ for each honest party subject to

  $$\sum_{i\in[n]} z_i \equiv \sum_{i\in[n]} x_i \cdot \sum_{i\in[n]} y_i \pmod{m}$$

  In the context of π_{RSAGen}, this interface is used to perform length-extension on CRT-shares of p and q.
- The `check` interface allows the parties to securely compute a predicate over the set of stored values. In the context of π_{RSAGen}, this is used to check that the CRT-share extension of p and q has been performed correctly, when N is a biprime.
- The `open` interface allows the parties to retroactively reveal their inputs to one another. In the context of π_{RSAGen}, this is used to verify the sampling procedure and biprimality test when N is not a biprime.

These five interfaces suffice for the malicious version of the protocol, and the first three alone suffice for the semi-honest version. We make a final adjustment, which leads to a substantial efficiency improvement in the protocol with which we realize $\mathcal{F}_{\mathsf{AugMul}}$ (which we describe in the full version of this paper [7]). Specifically, we give the adversary an interface by which it can request that any stored value be leaked to itself, and by which it can (arbitrarily) determine the output of any call to the `sample` or `multiply` interfaces. However, if the adversary uses this interface, the functionality remembers, and informs the honest parties by aborting when the `check` or `open` interfaces is used.

Functionality 4.1. $\mathcal{F}_{\mathsf{AugMul}}(n)$. Augmented n-Party Multiplication

This functionality is parametrized by the party count n. In addition to the parties it interacts with an ideal adversary $\mathcal{S}$ who corrupts the parties indexed by $\mathbf{P}^*$. The remaining honest parties are indexed by $\overline{\mathbf{P}^*} := [n] \setminus \mathbf{P}^*$.

Cheater Activation: Upon receiving $(\texttt{cheat}, \mathsf{sid})$ from $\mathcal{S}$, store $(\texttt{cheater}, \mathsf{sid})$ in memory and send every record of the form $(\texttt{value}, \mathsf{sid}, i, x_i, m)$ to $\mathcal{S}$. For the purposes of this functionality, we will consider session IDs to be fresh even when a `cheater` record already exists in memory.

Sampling: Upon receiving $(\texttt{sample}, \mathsf{sid}_1, \mathsf{sid}_2, m)$ from each party $\mathcal{P}_i$ for $i \in \overline{\mathbf{P}^*}$ and $(\texttt{adv-sample}, \mathsf{sid}_1, \mathsf{sid}_2, x_i, y_i, z_i, m)$ from $\mathcal{S}$ for $i \in \mathbf{P}^*$,[a] if sid_1 and sid_2 are fresh, agreed-upon values and if m is an agreed-upon prime, and if neither $(\texttt{cheater}, \mathsf{sid}_1)$ nor $(\texttt{cheater}, \mathsf{sid}_2)$ exists in memory, then sample $(x_i, y_i, z_i) \leftarrow \mathbb{Z}_m^3$ uniformly for each $i \in \overline{\mathbf{P}^*}$ subject to

$$\sum_{i\in[n]} z_i \equiv \sum_{i\in[n]} x_i \cdot \sum_{i\in[n]} y_i \not\equiv 0 \pmod{m}$$

If the previous conditions hold, but $(\texttt{cheater}, \mathsf{sid}_1)$ or $(\texttt{cheater}, \mathsf{sid}_2)$ exists in memory, then send $(\texttt{cheat-sample}, \mathsf{sid}_1, \mathsf{sid}_2)$ to $\mathcal{S}$ and in response receive $(\texttt{cheat-samples}, \mathsf{sid}_1, \mathsf{sid}_2, \{(x_i, y_i, z_i)\}_{i\in\overline{\mathbf{P}^*}})$ where $0 \le x_i, y_i, z_i < m$ for all i and where

$$\sum_{i\in[n]} z_i \not\equiv 0 \pmod{m}$$

(if these conditions are violated, then ignore the response from $\mathcal{S}$). Regardless, store $(\texttt{value}, \mathsf{sid}_1, i, x_i, m)$ and $(\texttt{value}, \mathsf{sid}_2, i, y_i, m)$ in memory for $i \in [n]$, and then send $(\texttt{sampled-product}, \mathsf{sid}_1, \mathsf{sid}_2, x_i, y_i, z_i)$ to each party $\mathcal{P}_i$ as adversarially delayed private output.

Input: Upon receiving $(\texttt{input}, \mathsf{sid}, x_i, m)$ from each party $\mathcal{P}_i$, where $i \in [n]$: if sid is a fresh, agreed-upon value and if m is an agreed-upon prime, and if $0 \le x_i < m$ for all $i \in [n]$, then store $(\texttt{value}, \mathsf{sid}, i, x_i, m)$ in memory for each $i \in [n]$ and send $(\texttt{value-loaded}, \mathsf{sid})$ to all parties. If $(\texttt{cheater}, \mathsf{sid})$ exists in memory, then send $(\texttt{value}, \mathsf{sid}, i, x_i, m)$ to $\mathcal{S}$ for each $i \in [n]$.

Multiplication: Upon receiving $(\texttt{multiply}, \mathsf{sid}_1, \mathsf{sid}_2, \mathsf{sid}_3)$ from each party $\mathcal{P}_i$ for $i \in \overline{\mathbf{P}^*}$ and $(\texttt{adv-multiply}, \mathsf{sid}_1, \mathsf{sid}_2, \mathsf{sid}_3, i, z_i)$ from $\mathcal{S}$ for each $i \in \mathbf{P}^*$,[a] if all three session IDs are agreed upon and sid_3 is fresh, and if no record of the form $(\texttt{cheater}, \mathsf{sid}_1)$ or $(\texttt{cheater}, \mathsf{sid}_2)$ exists in memory, and if records of the form $(\texttt{value}, \mathsf{sid}_1, i, x_i, m_1)$ and $(\texttt{value}, \mathsf{sid}_2, i, y_i, m_2)$ exist in memory for all $i \in [n]$ such that $m_1 = m_2$, then sample $z_i \leftarrow \mathbb{Z}_{m_1}$ for $i \in \overline{\mathbf{P}^*}$ subject to

$$\sum_{i\in[n]} z_i \equiv \sum_{i\in[n]} x_i \cdot \sum_{i\in[n]} y_i \pmod{m_1}$$

If the previous conditions hold, but $(\texttt{cheater}, \mathsf{sid}_1)$ or $(\texttt{cheater}, \mathsf{sid}_2)$ exists in memory, then send $(\texttt{cheat-multiply}, \mathsf{sid}_1, \mathsf{sid}_2, \mathsf{sid}_3)$ to $\mathcal{S}$ and in response receive $(\texttt{cheat-product}, \mathsf{sid}_3, \{z_i\}_{i\in\overline{\mathbf{P}^*}})$ where $0 \le z_i < m_1$ for all i. Regardless, send $(\texttt{product}, \mathsf{sid}_3, z_i)$ to each party $\mathcal{P}_i$ for $i \in [n]$ as adversarially delayed private output. Note that this procedure only permits multiplications of values associated with the *same* modulus.

Predicate Cheater Check: Upon receiving $(\texttt{check}, \mathsf{sids}, f)$ from all parties, where f is the description of a predicate over the set of stored values associated with the vector of session IDs sids, if f is not agreed upon, or if any record $(\texttt{cheater}, \mathsf{sid})$ exists in memory such that $\mathsf{sid} \in \mathsf{sids}$, then abort, informing all parties in an adversarially delayed fashion. Otherwise, let $\mathbf{x}$ be the vector of stored values associated with sids, or in other words, let it be a vector such that for all $j \in [|\mathbf{x}|]$ and $i \in [n]$, records of the form $(\texttt{value}, \mathsf{sids}_j, i, y_i, m)$ exist in memory such that

$$0 \leq \mathbf{x}_j < m \qquad \text{and} \qquad \mathbf{x}_j \equiv \sum_{i \in [n]} y_i \pmod{m}$$

Send $(\texttt{predicate-result}, \mathsf{sids}, f(\mathbf{x}))$ to all parties as adversarially delayed private output, and refuse all future messages with any session ID in sids.

Input Revelation: Upon receiving $(\texttt{open}, \mathsf{sid})$ from all parties, if a record of the form $(\texttt{cheater}, \mathsf{sid})$ exists in memory, then abort, informing all parties in an adversarially delayed fashion. Otherwise, for each record of the form $(\texttt{value}, \mathsf{sid}, i, x_i)$ in memory, send $(\texttt{opening}, \mathsf{sid}, i, x_i)$ to all parties as adversarially delayed output. Refuse all future messages with this sid.

[a] In the semi-honest setting, the adversary does not send these values to the functionality; instead the functionality samples the shares for corrupt parties just as it does for honest parties.

Biprimality Test. The biprimality-test functionality $\mathcal{F}_{\mathsf{Biprime}}$ (Functionality 4.2) abstracts the behavior of the biprimality test of Boneh and Franklin [4]. The functionality receives from each party a candidate biprime N, along with shares of its factors p and q. It checks whether p and q are primes and whether $N = p \cdot q$. The adversary is given an additional interface, by which it can ask the functionality to leak the honest parties' inputs, but when this interface is used then the functionality reports to the honest parties that N is not a biprime, even if it is one.

Functionality 4.2. $\mathcal{F}_{\mathsf{Biprime}}(M, n)$. Distributed Biprimality Test

This functionality is parametrized by the integer M and the party-count n. In addition to the parties it interacts with an ideal adversary $\mathcal{S}$.

Biprimality Test:

1. Wait to receive $(\texttt{check-biprimality}, \mathsf{sid}, N, p_i, q_i)$ from each party $\mathcal{P}_i$ for $i \in [n]$, where sid is a fresh, agreed-upon value.
2. Over the integers, compute

$$p := \sum_{i \in [n]} p_i \qquad \text{and} \qquad q := \sum_{i \in [n]} q_i \qquad \text{and} \qquad N' := p \cdot q$$

3. If all parties agreed on the value of N in Step 1, and $N = N'$, and both p and q are primes, and $p \not\equiv 1 \pmod{q}$, and $q \not\equiv 1 \pmod{p}$, and $0 \leq p < M$ and $0 \leq q < M$, then send a message $(\texttt{biprime}, \mathsf{sid})$ to $\mathcal{S}$. If $\mathcal{S}$ responds with $(\texttt{proceed}, \mathsf{sid})$, then output $(\texttt{biprime}, \mathsf{sid})$ to all parties as adversarially delayed output. If $\mathcal{S}$ responds with $(\texttt{cheat}, \mathsf{sid})$[a], or if any of the previous predicates is false, then output $(\texttt{leaked-shares}, \mathsf{sid}, \{(p_i, q_i)\}_{i\in[n]})$ directly to $\mathcal{S}$, and output $(\texttt{not-biprime}, \mathsf{sid})$ to all parties as adversarially delayed output.

[a] Semi-honest adversaries are forbidden to send the cheat instruction.

Realizations In the full version of this paper [7], we discuss a protocol to realize $\mathcal{F}_{\mathsf{AugMul}}$, and in Section 5, we propose a protocol to realize $\mathcal{F}_{\mathsf{Biprime}}$. Both make use of generic MPC, but in such a way that no generic MPC is required unless N is a biprime.

4.3 The Protocol Itself

We refer the reader back to Section 4.1 for an overview of our protocol. We have mentioned that it requires a vector of coprime values, which is prefixed by the (κ, n)-near-primorial vector. We now give this vector a precise definition. Note that the efficiency of our protocol relies upon this vector, because we use its contents to sieve candidate primes. Since smaller numbers are more likely to be factors for the candidate primes, we choose the largest allowable set of the smallest sequential primes.

Definition 4.3. ((κ, n)-Compatible Parameter Set)**.** Let ℓ' be the smallest number such that the ℓ'^{th} primorial number is greater than $2^{2\kappa-1}$, and let $\mathbf{m}$ be a vector of length ℓ' such that $\mathbf{m}_1 = 4$ and $\mathbf{m}_2, \ldots, \mathbf{m}_{\ell'}$ are the odd factors of the ℓ'^{th} primorial number, in ascending order. $(\mathbf{m}, \ell', \ell, M)$ is the (κ, n)*-compatible parameter set* if $\ell < \ell'$ and the prefix of $\mathbf{m}$ of length ℓ is the (κ, n)-near-primorial vector per Definition 3.4, and if M is the product of this prefix.

Protocol 4.4. $\pi_{\mathsf{RSAGen}}(\kappa, n, B)$**. Distributed Biprime Sampling**

This protocol is parametrized by the RSA prime length κ, the number of parties n, and the trial-division bound B. Let $(\mathbf{m}, \ell', \ell, M)$ be the (κ, n)-compatible parameter set, per Definition 4.3. In this protocol the parties have access to the functionalities $\mathcal{F}_{\mathsf{AugMul}}$ and $\mathcal{F}_{\mathsf{Biprime}}$.

Candidate Sieving:

1. Upon receiving input $(\texttt{sample}, \mathsf{sid})$ from the environment, the parties begin the protocol. Every party $\mathcal{P}_i$ for $i \in [n]$ computes three vectors of session IDs
$$\begin{aligned}\mathsf{psids} &:= \{\mathsf{GenSID}(\mathsf{sid}, j, \mathtt{p})\}_{j\in[\ell']}\\ \mathsf{qsids} &:= \{\mathsf{GenSID}(\mathsf{sid}, j, \mathtt{q})\}_{j\in[\ell']}\\ \mathsf{Nsids} &:= \{\mathsf{GenSID}(\mathsf{sid}, j, \mathtt{N})\}_{j\in[\ell']}\end{aligned}$$
and sends $(\texttt{sample}, \mathsf{psids}_j, \mathsf{qsids}_j, \mathbf{m}_j)$ to $\mathcal{F}_{\mathsf{AugMul}}(n)$ for every $j \in [2, \ell]$, and receives $(\texttt{sampled-product}, \mathsf{psids}_j, \mathsf{qsids}_j, \mathbf{p}_{i,j}, \mathbf{q}_{i,j}, \mathbf{N}_{i,j})$ in response. The parties also set $\mathbf{p}_{1,1} := \mathbf{q}_{1,1} := 3$ and $\mathbf{p}_{i',1} := \mathbf{q}_{i',1} := 0$ for $i' \in [2, n]$.
2. Each party $\mathcal{P}_i$ for $i \in [n]$ computes
$$\begin{aligned}p_i &:= \mathsf{CRTRecon}\left(\{\mathbf{m}_j\}_{j\in[\ell]}, \{\mathbf{p}_{i,j}\}_{j\in[\ell]}\right)\\ q_i &:= \mathsf{CRTRecon}\left(\{\mathbf{m}_j\}_{j\in[\ell]}, \{\mathbf{q}_{i,j}\}_{j\in[\ell]}\right)\end{aligned}$$
and then, for $j \in [\ell + 1, \ell']$, $\mathcal{P}_i$ computes
$$\mathbf{p}_{i,j} := p_i \bmod \mathbf{m}_j \qquad \text{and} \qquad \mathbf{q}_{i,j} := q_i \bmod \mathbf{m}_j$$
Note that each party $\mathcal{P}_i$ is now in possession of a pair of vectors
$$\mathbf{p}_{i,*} \in \mathbb{Z}_{\mathbf{m}_1} \times \ldots \times \mathbb{Z}_{\mathbf{m}_{\ell'}} \qquad \text{and} \qquad \mathbf{q}_{i,*} \in \mathbb{Z}_{\mathbf{m}_1} \times \ldots \times \mathbb{Z}_{\mathbf{m}_{\ell'}}$$
3. For $j \in [\ell + 1, \ell']$, every party $\mathcal{P}_i$ for $i \in [n]$ sends the following sequence of messages to $\mathcal{F}_{\mathsf{AugMul}}(n)$, waiting for confirmation after each:
   (a) $(\texttt{input}, \mathsf{psids}_j, \mathbf{p}_{i,j}, \mathbf{m}_j)$
   (b) $(\texttt{input}, \mathsf{qsids}_j, \mathbf{q}_{i,j}, \mathbf{m}_j)$
   (c) $(\texttt{multiply}, \mathsf{psids}_j, \mathsf{qsids}_j, \mathsf{Nsids}_j)$

   and at the end of this sequence, each party $\mathcal{P}_i$ receives $(\texttt{product}, \mathsf{Nsids}_j, \mathbf{N}_{i,j})$ from $\mathcal{F}_{\mathsf{AugMul}}(n)$ in response. Note that each party $\mathcal{P}_i$ is now in possession of a vector $\mathbf{N}_{i,*} \in \mathbb{Z}_{\mathbf{m}_1} \times \ldots \times \mathbb{Z}_{\mathbf{m}_{\ell'}}$.
4. For $j \in [2, \ell']$, each party $\mathcal{P}_i$ for $i \in [n]$ broadcasts $\mathbf{N}_{i,j}$. Once all parties have received shares from all other parties, they compute
$$N := \mathsf{CRTRecon}\left(\mathbf{m}, \left\{\sum_{i'\in[n]} \mathbf{N}_{i',j} \bmod \mathbf{m}_j\right\}_{j\in[\ell']}\right)$$

5. Each party $\mathcal{P}_i$ performs a local trial division on N by all primes less than B. If N is divisible by some prime, then the parties skip directly to Step 7, and take the privacy-free branch.

Biprimality Test:

6. Each party $\mathcal{P}_i$ for $i \in [n]$ sends $(\texttt{check-biprimality}, \mathsf{sid}, N, p_i, q_i)$ to $\mathcal{F}_{\mathsf{Biprime}}(M, n)$ and waits for either $(\texttt{biprime}, \mathsf{sid})$ or $(\texttt{not-biprime}, \mathsf{sid})$ in response.

Consistency Check: [a]

7. Let f be the predicate that is defined to compute

$$p_{i'} := \mathsf{CRTRecon}\left(\mathbf{m}, \mathbf{p}_{i',*}\right) \qquad \text{and} \qquad q_{i'} := \mathsf{CRTRecon}\left(\mathbf{m}, \mathbf{q}_{i',*}\right)$$

for all $i' \in [n]$ and to return 1 if and only if

$$\begin{aligned} N = & \sum_{i' \in [n]} p_{i'} \cdot \sum_{i' \in [n]} q_{i'} \\ & \wedge \quad 0 \leq p_{i'} < M \quad \wedge \quad 0 \leq q_{i'} < M \qquad \text{for all } i' \in [n] \end{aligned}$$

where the sums and product are taken over the integers.

- If $\texttt{biprime}$ is received from $\mathcal{F}_{\mathsf{Biprime}}(M, n)$, then N is a biprime, and a privacy-preserving check must be performed. Each party sends $(\texttt{check}, \mathbf{psids} \| \mathbf{qsids}, f)$ to $\mathcal{F}_{\mathsf{AugMul}}(n)$. If $\mathcal{F}_{\mathsf{AugMul}}$ returns $(\texttt{predicate-result}, \mathbf{psids} \| \mathbf{qsids}, 1)$ then the parties halt successfully and output $(\texttt{biprime}, \mathsf{sid}, N)$ to the environment; otherwise, they abort.
- If $\texttt{not-biprime}$ is received from $\mathcal{F}_{\mathsf{Biprime}}(M, n)$, then either N is not a biprime or some party has cheated; consequently, a privacy-free check is performed.
  (a) For $j \in [2, \ell']$, each party $\mathcal{P}_i$ for $i \in [n]$ sends $(\texttt{open}, \mathbf{psids}_j)$ and $(\texttt{open}, \mathbf{qsids}_j)$ to $\mathcal{F}_{\mathsf{AugMul}}(n)$. If $\mathcal{P}_i$ observes $\mathcal{F}_{\mathsf{AugMul}}(n)$ to abort in response to any of these queries, then $\mathcal{P}_i$ itself aborts. Otherwise, $\mathcal{P}_i$ receives $(\texttt{opening}, \mathbf{psids}_j, \mathbf{p}_{i',j})$ and $(\texttt{opening}, \mathbf{qsids}_j, \mathbf{q}_{i',j})$ for each $i' \in [n]$ and $j \in [2, \ell']$.
  (b) The parties individually check that the predicate f holds over the vectors of shares which they now all possess. If this predicate holds and p and q are not both prime, then all parties halt successfully and output $(\texttt{non-biprime}, \mathsf{sid})$ to the environment. Otherwise, a party has cheated, and they abort.

[a] If only security against semi-honest adversaries is required, the protocol can terminate after the Biprimality-Test phase, and these checks are unnecessary.

4.4 Security Sketches

We now informally argue that π_{RSAGen} realizes $\mathcal{F}_{\mathsf{RSAGen}}$ in the semi-honest and malicious settings. We give a full proof for the malicious setting in the full version of this paper [7].

Theorem 4.5. π_{RSAGen} UC-realizes $\mathcal{F}_{\mathsf{RSAGen}}$ with perfect security in the $(\mathcal{F}_{\mathsf{AugMul}}, \mathcal{F}_{\mathsf{Biprime}})$-hybrid model against a static, semi-honest adversary that corrupts up to $n-1$ parties.

Proof Sketch. In lieu of arguing for the correctness of our protocol, we refer the reader to the explanation in Section 4.1, and focus here on the strategy of a simulator $\mathcal{S}$ against a semi-honest adversary $\mathcal{A}$ who corrupts the parties indexed by $\mathbf{P}^*$. $\mathcal{S}$ forwards all messages between $\mathcal{A}$ and the environment faithfully.

In Step 1 of π_{RSAGen}, for each $j \in [2, \ell]$, $\mathcal{S}$ receives the `sample` instruction with modulus $\mathbf{m}_j$ on behalf of $\mathcal{F}_{\mathsf{AugMul}}$ from all parties indexed by $\mathbf{P}^*$. For each j it then samples $(\mathbf{p}_{i,j}, \mathbf{q}_{i,j}, \mathbf{N}_{i,j}) \leftarrow \mathbb{Z}^3_{\mathbf{m}_j}$ uniformly for $i \in \mathbf{P}^*$, and returns each triple to the appropriate party.

Step 2 involves no interaction on the part of the parties, but it is at this point that $\mathcal{S}$ computes p_i and q_i for $i \in \mathbf{P}^*$, in the same way that the parties themselves do. Note that since $\mathbf{p}_{*,1}$ and $\mathbf{q}_{*,1}$ are deterministically chosen, they are known to $\mathcal{S}$. The simulator then sends these shares to $\mathcal{F}_{\mathsf{RSAGen}}$ via the functionality's `adv-input` interface, and receives in return either a biprime N, or two factors p and q such that $N := p \cdot q$ is not a biprime. Regardless, it instructs $\mathcal{F}_{\mathsf{RSAGen}}$ to `proceed`.

In Step 3 of π_{RSAGen}, $\mathcal{S}$ receives two `input` instructions from each corrupted party for each $j \in [\ell+1, \ell']$ on behalf of $\mathcal{F}_{\mathsf{AugMul}}$, and confirms receipt as $\mathcal{F}_{\mathsf{AugMul}}$ would. Subsequently, for each $j \in [\ell+1, \ell']$, the corrupt parties all send a `multiply` instruction, and then $\mathcal{S}$ samples $\mathbf{N}_{i,j} \leftarrow \mathbb{Z}_{\mathbf{m}_j}$ for $i \in [n]$ subject to

$$\sum_{i \in [n]} \mathbf{N}_{i,j} \equiv N \pmod{\mathbf{m}_j}$$

and returns each share to the matching corrupt party.

In Step 4 of π_{RSAGen}, for every $j \in [\ell']$, every corrupt party $\mathcal{P}_{i'}$ for $i' \in \mathbf{P}^*$, and every honest party $\mathcal{P}_i$ for $i \in [n] \setminus \mathbf{P}^*$, $\mathcal{S}$ sends $\mathbf{N}_{i,j}$ to $\mathcal{P}_{i'}$ on behalf of $\mathcal{P}_i$, and receives $\mathbf{N}_{i',j}$ (which it already knows) in reply.

To simulate the final steps of π_{RSAGen}, $\mathcal{S}$ tries to divide N by all primes smaller than B. If it succeeds, then the protocol is complete. Otherwise, it receives `check-biprimality` from all of the corrupt parties on behalf of $\mathcal{F}_{\mathsf{Biprime}}$, and replies with `biprime` or `not-biprime` as appropriate. It can be verified by inspection that the view of the environment is identically distributed in the ideal-world experiment containing $\mathcal{S}$ and honest parties that interact with $\mathcal{F}_{\mathsf{RSAGen}}$, and the real-world experiment containing $\mathcal{A}$ and parties running π_{RSAGen}. □

Theorem 4.6. If factoring biprimes sampled by BFGM is hard, then π_{RSAGen} UC-realizes $\mathcal{F}_{\mathsf{RSAGen}}$ in the $(\mathcal{F}_{\mathsf{AugMul}}, \mathcal{F}_{\mathsf{Biprime}})$-hybrid model against a static, malicious PPT adversary that corrupts up to $n-1$ parties.

Proof Sketch. We observe that if the adversary simply follows the specification of the protocol and does not cheat in its inputs to $\mathcal{F}_{\mathsf{AugMul}}$ or $\mathcal{F}_{\mathsf{Biprime}}$, then the simulator can follow the same strategy as in the semi-honest case. At any point if the adversary deviates from the protocol, the simulator requests $\mathcal{F}_{\mathsf{RSAGen}}$ to reveal all honest parties' shares, and thereafter the simulator uses them by effectively running the code of the honest parties. This matches the adversary's view in the real protocol as far as the distribution of the honest parties' shares is concerned.

It remains to be argued that any deviation from the protocol specification will also result in an abort in the real world with honest parties, and will additionally be recognized by the honest parties as an adversarially induced cheat (as opposed to a statistical sampling failure). Note that the honest parties must only detect cheating when N is truly a biprime and the adversary has sabotaged a successful candidate; if N is not a biprime and would have been rejected anyway, then cheat-detection is unimportant. We analyze all possible cases where the adversary deviates from the protocol below. Let N be defined as the value implied by parties' sampled shares in Step 1 of π_{RSAGen}.

Case 1: N is a non-biprime and reconstructed correctly. In this case, $\mathcal{F}_{\mathsf{Biprime}}$ will always reject N as there exist no satisfying inputs (i.e., there are no two prime factors p, q such that $p \cdot q = N$).

Case 2: N is a non-biprime and reconstructed incorrectly as N'. If by fluke N' happens to be a biprime then the incorrect reconstruction will be caught by the explicit secure predicate check during the consistency-check phase. If N' is a non-biprime then the argument from the previous case applies.

Case 3: N is a biprime and reconstructed correctly. If consistent inputs are used for the biprimality test and nobody cheats, the candidate N is successfully accepted (this case essentially corresponds to the semi-honest case). Otherwise, if inconsistent inputs are used for the biprimality test, one of the following events will occur:

- $\mathcal{F}_{\mathsf{Biprime}}$ rejects this candidate. In this case, all parties reveal their shares of p and q to one another (with guaranteed correctness via $\mathcal{F}_{\mathsf{AugMul}}$) and locally test their primality. This will reveal that N was a biprime, and that $\mathcal{F}_{\mathsf{Biprime}}$ must have been supplied with inconsistent inputs, implying that some party has cheated.
- $\mathcal{F}_{\mathsf{Biprime}}$ accepts this candidate. This case occurs with negligible probability (assuming factoring is hard). Because N only has two factors, there is exactly one pair of inputs that the adversary can supply to $\mathcal{F}_{\mathsf{Biprime}}$ to induce this scenario, apart from the pair specified by the protocol. In our full proof (see the full version [7] of this paper) we show that finding this alternative pair of satisfying inputs implies factoring N. We are careful to rely on the hardness of factoring only in this case, where by premise N is a biprime with κ-bit factors (i.e., an instance of the factoring problem).

Case 4: N is a biprime and reconstructed incorrectly as N'. If N' is a biprime then the incorrect reconstruction will be caught during the consistency-check phase, just as when N is a biprime. If N' is a non-biprime then it will by rejected by $\mathcal{F}_{\mathsf{Biprime}}$, inducing all parties to reveal their shares and find that their shares do not in fact reconstruct to N', with the implication that some party has cheated.

Thus the adversary is always caught when trying to sabotage a true biprime, and it can never sneak a non-biprime past the consistency check. Because the real-world protocol always aborts in the case of cheating, it is indistinguishable from the simulation described above, assuming that factoring is hard. □

5 Distributed Biprimality Testing

In the semi-honest setting, $\mathcal{F}_{\mathsf{Biprime}}$ can be realized by the biprimality-testing protocol of Boneh and Franklin [4]. We discuss this in the full version [7] of this paper. The following lemma follows immediately from their work.

Lemma 5.1. The biprimality-testing protocol described by Boneh and Franklin [4] UC-realizes $\mathcal{F}_{\mathsf{Biprime}}$ with statistical security in the $\mathcal{F}_{\mathsf{ComCompute}}$-hybrid model against a static, semi-honest adversary who corrupts up to $n-1$ parties.

5.1 The Malicious Setting

Unlike a semi-honest adversary, we permit a malicious adversary to force a true biprime to fail our biprimality test, and detect such behavior using independent mechanisms in the π_{RSAGen} protocol. However, we must ensure that a non-biprime can never pass the test with more than negligible probability. To achieve this, we use a derivative of the biprimality-testing protocol of Frederiksen et al. [16]; relative to their protocol, ours is simpler, and we prove that it UC-realizes $\mathcal{F}_{\mathsf{Biprime}}$.

The protocol essentially comprises a randomized version of the semi-honest Boneh-Franklin test described previously, followed by a Schnorr-like protocol to verify that the test was performed correctly. The soundness error of the underlying biprimality test is compounded by the Schnorr-like protocol's soundness error to yield a combined error of $3/4$; this necessitates an increase in the number of iterations by a factor of $\log_{4/3}(2) < 2.5$. While this is sufficient to ensure the test itself is carried out honestly, it does not ensure the correct inputs are used. Consequently, generic MPC is used to verify the relationship between the messages involved in the Schnorr-like protocol and the true candidate given by N and shares of its factors. As a side effect, this generic computation samples $r \leftarrow \mathbb{Z}_N$ and outputs $z = r\cdot(p+q-1) \bmod N$ so that the GCD test can afterward be run locally by each party.

Our protocol makes use of a number of subfunctionalities, all of which are standard and described in the full version of this paper [7]. Namely, we use a coin-tossing functionality $\mathcal{F}_{\mathsf{CT}}$ to uniformly sample an element from some set, the one-to-many commitment functionality $\mathcal{F}_{\mathsf{Com}}$, the generic MPC functionality over committed inputs $\mathcal{F}_{\mathsf{ComCompute}}$, and the integer-sharing-of-zero functionality $\mathcal{F}_{\mathsf{Zero}}$. In addition, the protocol uses the algorithm VerifyBiprime (Algorithm 5.3).

Protocol 5.2. $\pi_{\mathsf{Biprime}}(M, n)$**. Distributed Biprimality Testing**

This protocol is parametrized by an integer M and the number of parties n. In addition, there is a statistical parameter s. The parties have access to the $\mathcal{F}_{\mathsf{CT}}$, $\mathcal{F}_{\mathsf{Com}}$, $\mathcal{F}_{\mathsf{ComCompute}}$, and $\mathcal{F}_{\mathsf{Zero}}$ functionalities.

Input Commitment:

1. Upon receiving input $(\texttt{check-biprimality}, \mathsf{sid}, N, p_i, q_i)$ from the environment, each party $\mathcal{P}_i$ for $i \in [n]$ samples $\boldsymbol{\tau}_{i,j} \leftarrow \mathbb{Z}_{M \cdot 2^{s+1}}$ for $j \in [2.5s]$ and commits to these values, along with its shares of p and q, by sending $(\texttt{commit}, \mathsf{GenSID}(\mathsf{sid}, i), (p_i, q_i, \boldsymbol{\tau}_{i,*}))$ to $\mathcal{F}_{\mathsf{ComCompute}}(n)$.

Boneh-Franklin Test:

2. Each party $\mathcal{P}_i$ for $i \in [n]$ sends $(\texttt{sample}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{Zero}}(n, 2^{2\kappa+s})$ and receives $(\texttt{zero-share}, \mathsf{sid}, r_i)$ in response.
3. For $j \in [2.5s]$, the parties invoke $\mathcal{F}_{\mathsf{CT}}(n, \mathbb{J}_N)$, where $\mathbb{J}_N$ is the subdomain of $\mathbb{Z}_N^*$ that contains only values with Jacobi symbol 1. The parties define vector $\boldsymbol{\gamma}$ that contains the $2.5s$ sampled values.
4. For every $j \in [2.5s]$, party $\mathcal{P}_1$ computes[a]
$$\boldsymbol{\chi}_{1,j} := \boldsymbol{\gamma}_j^{r_1 - (p_1+q_1-6)/4} \mod N$$
and every other party $\mathcal{P}_i$ for $i \in [2, n]$ computes
$$\boldsymbol{\chi}_{i,j} := \boldsymbol{\gamma}_j^{r_i - (p_i+q_i)/4} \mod N$$
5. Every $\mathcal{P}_i$ for $i \in [n]$ sends $(\texttt{commit}, \mathsf{GenSID}(\mathsf{sid}, i), \boldsymbol{\chi}_{i,*}, [n])$ to $\mathcal{F}_{\mathsf{Com}}(n)$.
6. After being notified that all other parties are committed, each party $\mathcal{P}_i$ for $i \in [n]$ sends $(\texttt{decommit}, \mathsf{GenSID}(\mathsf{sid}, i))$ to $\mathcal{F}_{\mathsf{Com}}(n)$, and in response receives $\boldsymbol{\chi}_{i',*}$ from $\mathcal{F}_{\mathsf{Com}}(n)$ for $i' \in [n] \setminus \{i\}$.
7. The parties output $(\texttt{not-biprime}, \mathsf{sid})$ to the environment and halt if there exists $j \in [2.5s]$ such that
$$\boldsymbol{\gamma}_j^{(N-5)/4} \cdot \prod_{i \in [n]} \boldsymbol{\chi}_{i,j} \not\equiv \pm 1 \pmod N$$

Consistency Check and GCD Test:

8. For $j \in [2.5s]$, each party $\mathcal{P}_i$ for $i \in [n]$ computes $\boldsymbol{\alpha}_{i,j} := \boldsymbol{\gamma}_j^{\boldsymbol{\tau}_{i,j}} \mod N$. The parties all broadcast the values they have computed to one another.
9. The parties all send $(\texttt{flip}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{CT}}(n, \{0,1\}^{2.5s})$ to obtain an agreed-upon random bit vector $\mathbf{c}$ of length $2.5s$.

10. For $j \in [2.5s]$, party $\mathcal{P}_1$ computes $\boldsymbol{\zeta}_{1,j} \coloneqq \boldsymbol{\tau}_{1,j} - \mathbf{c}_j \cdot (p_1 + q_1)/4$, and every other party $\mathcal{P}_i$ for $i \in [2, n]$ computes $\boldsymbol{\zeta}_{i,j} \coloneqq \boldsymbol{\tau}_{i,j} - \mathbf{c}_j \cdot (p_i + q_i - 6)/4$. They all broadcast the values they have computed to one another.
11. The parties halt and output $(\texttt{not-biprime}, \mathsf{sid})$ if there exists any $j \in [2.5s]$ such that
$$\prod_{i\in[n]} \boldsymbol{\gamma}_j^{\boldsymbol{\zeta}_{i,j}} \not\equiv \prod_{i\in[n]} \boldsymbol{\alpha}_{i,j} \cdot \boldsymbol{\chi}_{i,j}^{\mathbf{c}_j} \pmod{N}$$
12. Let C be a circuit computing $\mathsf{VerifyBiprime}(N, M, \mathbf{c}, \{\cdot, \cdot, \cdot, \boldsymbol{\zeta}_{i,*}\}_{i\in[n]})$; that is, let it be a circuit representation of Algorithm 5.3 with the public values N, M, $\mathbf{c}$, and $\boldsymbol{\zeta}$ hardcoded. The parties send $(\texttt{compute}, \mathsf{sid}, \{\mathsf{GenSID}(\mathsf{sid}, i)\}_{i\in[n]}, C)$ to $\mathcal{F}_{\mathsf{ComCompute}}(n)$, and in response they all receive $(\texttt{result}, \mathsf{sid}, z)$. If $z = \bot$, or if $\mathcal{F}_{\mathsf{ComCompute}}(n)$ aborts, then the parties halt and output $(\texttt{not-biprime}, \mathsf{sid})$.
13. The parties halt and output $(\texttt{biprime}, \mathsf{sid})$ to the environment if $\gcd(z, N) = 1$, or halt and output $(\texttt{not-biprime}, \mathsf{sid})$ otherwise.

[a] Recall that $p_1 \equiv q_1 \equiv 3 \pmod{4}$, and so subtracting 6 from their sum ensures that division by 4 can be performed without computing a modular multiplicative inverse in $\mathbb{Z}_N^*$. We compensate for this offset using another offset in Step 7.

Below we present the algorithm $\mathsf{VerifyBiprime}$ that is used for the GCD test. The inputs are the candidate biprime N, an integer M (the bound on the shares' size), a bit-vector $\mathbf{c}$ of length $2.5s$, and for each $i \in [n]$ a tuple consisting of the shares p_i and q_i with the Schnorr-like messages $\boldsymbol{\tau}_{i,*}$ and $\boldsymbol{\zeta}_{i,*}$ generated by $\mathcal{P}_i$. The algorithm verifies that all input values are compatible, and returns $z = r \cdot (p + q - 1) \bmod N$ for a random r.

Algorithm 5.3. $\mathsf{VerifyBiprime}(N, M, \mathbf{c}, \{(p_i, q_i, \boldsymbol{\tau}_{i,*}, \boldsymbol{\zeta}_{i,*})\}_{i\in[n]})$

1. Sample $r \leftarrow \mathbb{Z}_N$ and compute
$$z \coloneqq r \cdot \Big(-1 + \sum_{i\in[n]} (p_i + q_i) \Big) \bmod N$$
2. Return z if and only if it holds that
$$\begin{aligned} N = \sum_{i\in[n]} p_i \cdot \sum_{i\in[n]} q_i & \\ \wedge \quad 0 \le p_i < M \quad \wedge \quad 0 \le q_i < M & \qquad \text{for all } i \in [n] \\ \wedge \quad \boldsymbol{\tau}_{1,j} = \boldsymbol{\zeta}_{1,j} + \mathbf{c}_j \cdot (p_1 + q_1 - 6)/4 & \qquad \text{for all } j \in [2.5s] \\ \wedge \quad \boldsymbol{\tau}_{i,j} = \boldsymbol{\zeta}_{i,j} + \mathbf{c}_j \cdot (p_i + q_i)/4 & \qquad \text{for all } i \in [2, n] \text{ and } j \in [2.5s] \end{aligned}$$
If any part of the above predicate does not hold, output $\bot$.

Theorem 5.4. π_{Biprime} UC-realizes $\mathcal{F}_{\mathsf{Biprime}}$ in the $(\mathcal{F}_{\mathsf{Com}}, \mathcal{F}_{\mathsf{ComCompute}}, \mathcal{F}_{\mathsf{CT}}, \mathcal{F}_{\mathsf{Zero}})$-hybrid model with statistical security against a static, malicious adversary that corrupts up to $n-1$ parties.

Proof Sketch. Our simulator $\mathcal{S}$ for $\mathcal{F}_{\mathsf{Biprime}}$ receives N as common input. Let $\mathbf{P}^*$ and $\overline{\mathbf{P}^*}$ be vectors indexing the corrupt and honest parties, respectively. To simulate Steps 1 through 3 of π_{Biprime}, $\mathcal{S}$ simply behaves as $\mathcal{F}_{\mathsf{CT}}$, $\mathcal{F}_{\mathsf{Zero}}$, and $\mathcal{F}_{\mathsf{ComCompute}}$ would in its interactions with the corrupt parties on their behalf, remembering the values received and transmitted. Before continuing, $\mathcal{S}$ submits the corrupted parties' shares of p and q to $\mathcal{F}_{\mathsf{Biprime}}$ on their behalf. In response, $\mathcal{F}_{\mathsf{Biprime}}$ either informs $\mathcal{S}$ that N is a biprime, or leaks the honest parties' shares. In Step 4, $\mathcal{S}$ again behaves exactly as $\mathcal{F}_{\mathsf{Com}}$ would. During the remainder of the protocol, the simulator must follow one of two different strategies, conditioned on whether or not N is a biprime. We will show that both strategies lead to a simulation that is statistically indistinguishable from the real-world experiment.

– If $\mathcal{F}_{\mathsf{Biprime}}$ reported that N is a biprime, then we know by the specification of $\mathcal{F}_{\mathsf{Biprime}}$ that the corrupt parties committed to correct shares of p and q in Step 1 of π_{Biprime}. Boneh and Franklin [4] showed that the value (i.e., sign) of the right-hand side of the equality in Step 7 is predictable and related to the value of $\boldsymbol{\gamma}_j$. We refer to them for a precise description and proof. If without loss of generality we take that value to be 1, then $\mathcal{S}$ can simulate iteration j of Steps 6 and 7 as follows. First, $\mathcal{S}$ computes $\hat{\boldsymbol{\chi}}_{i,j}$ for $i \in \mathbf{P}^*$ to be the corrupt parties' ideal values of $\boldsymbol{\chi}_{i,j}$ as defined in Step 4 of π_{Biprime}. Then, $\mathcal{S}$ samples $\boldsymbol{\chi}_{i,j} \leftarrow \mathbb{Z}_N^*$ uniformly for $i \in \overline{\mathbf{P}^*}$ subject to

$$\prod_{i \in \overline{\mathbf{P}^*}} \boldsymbol{\chi}_{i,j} \equiv \frac{\boldsymbol{\gamma}_j^{(5-N)/4}}{\prod\limits_{i \in \mathbf{P}^*} \hat{\boldsymbol{\chi}}_{i,j}} \pmod{N}$$

and simulates Step 6 by releasing $\boldsymbol{\chi}_{i,j}$ for $i \in \overline{\mathbf{P}^*}$ to the corrupt parties on behalf of $\mathcal{F}_{\mathsf{Com}}$. These values are statistically close to their counterparts in the real protocol. Finally, $\mathcal{S}$ simulates Step 7 by running the test for itself and sending the `cheat` command to $\mathcal{F}_{\mathsf{Biprime}}$ on failure.
Given the information now known to $\mathcal{S}$, Steps 8 through 11 of π_{Biprime} can be simulated in a manner similar to the simulation of a common Schnorr protocol: $\mathcal{S}$ simply chooses $\boldsymbol{\zeta}_{i,*} \leftarrow \mathbb{Z}_{M \cdot 2^{s+1}}^{2.5s}$ uniformly for $i \in \overline{\mathbf{P}^*}$, fixes $\mathbf{c} \leftarrow \{0,1\}^{2.5s}$ ahead of time, and then works backwards via the equation in Step 11 to compute the values of $\boldsymbol{\alpha}_{i,*}$ for $i \in \overline{\mathbf{P}^*}$ that it must send on behalf of the honest parties in Step 8. These values are statistically close to their counterparts in the real protocol.
$\mathcal{S}$ finally simulates the remaining steps of π_{Biprime} by checking the VerifyBiprime predicate itself (since the final GCD test is purely local, no action need be taken by $\mathcal{S}$). If at any point after Step 4 the corrupt parties have cheated (i.e., sent an unexpected value or violated the VerifyBiprime predicate), then $\mathcal{S}$ sends the `cheat` command to $\mathcal{F}_{\mathsf{Biprime}}$. Otherwise, it sends the `proceed` command to $\mathcal{F}_{\mathsf{Biprime}}$, completing the simulation.

– If $\mathcal{F}_{\mathsf{Biprime}}$ reported that N is *not* a biprime (which may indicate that the corrupt parties supplied incorrect shares of p or q), then it also leaked the honest parties' shares of p and q to $\mathcal{S}$. Thus, $\mathcal{S}$ can simulate Steps 4 through 13 of π_{Biprime} by running the honest parties' code on their behalf. In all instances of the ideal-world experiment, the honest parties report to the environment that N is a non-biprime. Thus, we need only prove that there is no strategy by which the corrupt parties can successfully convince the honest parties that N is a biprime in the real world.

In order to get away with such a real-world cheat, the adversary must cheat in every iteration j of Steps 4 through 6 for which

$$\boldsymbol{\gamma}_j^{(N-p-q)/4} \not\equiv \pm 1 \pmod{N}$$

Specifically, in every such iteration j, the corrupt parties must contrive to send values $\boldsymbol{\chi}_{i,j}$ for $i \in \mathbf{P}^*$ such that

$$\boldsymbol{\gamma}_j^{(N-5)/4} \cdot \prod_{i\in[n]} \boldsymbol{\chi}_{i,j} \equiv \boldsymbol{\gamma}_j^{(N-p-q)/4+\boldsymbol{\Delta}_{1,j}} \equiv \pm 1 \pmod{N}$$

for some nonzero offset value $\boldsymbol{\Delta}_{1,j}$. We can define a similar offset $\boldsymbol{\Delta}_{2,j}$ for the corrupt parties' transmitted values of $\boldsymbol{\alpha}_{i,j}$, relative to the values of $\boldsymbol{\tau}_{i,j}$ committed in Step 1:

$$\boldsymbol{\gamma}_j^{\boldsymbol{\Delta}_{2,j}} \cdot \prod_{i\in[n]} \boldsymbol{\alpha}_{i,j} \equiv \prod_{i\in[n]} \boldsymbol{\gamma}_j^{\boldsymbol{\tau}_{i,j}} \pmod{N}$$

Since we have presupposed that the protocol outputs `biprime`, we know that the corrupt parties *must* transmit correctly calculated values of $\boldsymbol{\zeta}_{i,*}$ in Step 10 of π_{Biprime}, or else Step 12 would output `non-biprime` when these values are checked by the VerifyBiprime predicate. It follows from this fact and from the equation in Step 11 that $\boldsymbol{\Delta}_{2,j} \equiv \mathbf{c}_j \cdot \boldsymbol{\Delta}_{1,j} \pmod{\varphi(N)}$, where $\varphi(\cdot)$ is Euler's totient function. However, both $\boldsymbol{\Delta}_{1,*}$ and $\boldsymbol{\Delta}_{2,*}$ are fixed before $\mathbf{c}$ is revealed to the corrupt parties, and so the adversary can succeed in this cheat with probability at most $1/2$ for any individual iteration j.

Per Boneh and Franklin [4, Lemma 4.1], a particular iteration j of Steps 4 through 6 of π_{Biprime} produces a false positive result with probability at most $1/2$ if the adversary behaves honestly. If we assume that the adversary cheats always and only when a false positive would not have been produced by honest behavior, then the total probability of an adversary producing a positive outcome in the j^{th} iteration of Steps 4 through 6 is upper-bounded by $3/4$. The probability that an adversary succeeds over all $2.5s$ iterations is therefore at most $(3/4)^{2.5s} < 2^{-s}$. Thus, the adversary has a negligible chance to force the acceptance of a non-biprime in the real world, and the distribution of outcomes produced by $\mathcal{S}$ is statistically indistinguishable from the real-world distribution. □

Acknowledgements

The authors thank Muthuramakrishnan Venkitasubramaniam for the useful conversations and insights he provided, Tore Frederiksen for reviewing and confirming our cost analysis of his protocol [16], and Xiao Wang and Peter Scholl for providing detailed cost analyses of their respective protocols [34, 21].

This research was supported in part by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Project Activity (IARPA) under contract number 2019-19-020700009 (ACHILLES).

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of ODNI, IARPA, DoI/NBC, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.

Bibliography

[1] Algesheimer, J., Camenisch, J., Shoup, V.: Efficient computation modulo a shared secret with application to the generation of shared safe-prime products. In: CRYPTO. pp. 417–432 (2002)

[2] Barker, E.: Nist special publication 800-57, part 1, revision 4. http://dx.doi.org/10.6028/NIST.SP.800-57pt1r4 (2016)

[3] Boneh, D., Franklin, M.K.: Efficient generation of shared RSA keys. In: CRYPTO. pp. 425–439 (1997)

[4] Boneh, D., Franklin, M.K.: Efficient generation of shared RSA keys. J. ACM **48**(4), 702–722 (2001)

[5] Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Rindal, P., Scholl, P.: Efficient two-round OT extension and silent non-interactive secure computation. In: ACM CCS. pp. 291–308 (2019)

[6] Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: FOCS. pp. 136–145 (2001)

[7] Chen, M., Cohen, R., Doerner, J., Kondi, Y., Lee, E., Rosefield, S., abhi shelat: Muliparty generation of an RSA modulus. http://eprint.iacr.org/2020/370 (2020)

[8] Cocks, C.: Split knowledge generation of RSA parameters. In: Cryptography and Coding. pp. 89–95 (1997)

[9] Cocks, C.: Split generation of RSA parameters with multiple participants. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.177.2600 (1998)

[10] Cohen, R., Haitner, I., Omri, E., Rotem, L.: From fairness to full security in multiparty computation. pp. 216–234 (2018)

[11] Cohen, R., Lindell, Y.: Fairness versus guaranteed output delivery in secure multiparty computation. JCRYPT **30**(4), 1157–1186 (2017)

[12] Damgård, I., Mikkelsen, G.L.: Efficient, robust and constant-round distributed RSA key generation. In: TCC. pp. 183–200 (2010)

[13] Doerner, J., Kondi, Y., Lee, E., Shelat, A.: Secure two-party threshold ECDSA from ECDSA assumptions. In: S&P. pp. 980–997 (2018)

[14] Doerner, J., Kondi, Y., Lee, E., Shelat, A.: Threshold ECDSA from ECDSA assumptions: The multiparty case. In: S&P (2019)

[15] Frankel, Y., MacKenzie, P.D., Yung, M.: Robust efficient distributed RSA-key generation. In: PODC. p. 320 (1998)

[16] Frederiksen, T.K., Lindell, Y., Osheter, V., Pinkas, B.: Fast distributed RSA key generation for semi-honest and malicious adversaries. In: CRYPTO. pp. 331–361 (2018)

[17] Gilboa, N.: Two party RSA key generation. In: CRYPTO. pp. 116–129 (1999)

[18] Goldreich, O.: The Foundations of Cryptography - Volume 1: Basic Techniques. Cambridge University Press (2001)

[19] Hazay, C., Mikkelsen, G.L., Rabin, T., Toft, T.: Efficient RSA key generation and threshold Paillier in the two-party setting. In: CT-RSA. pp. 313–331 (2012)

[20] Hazay, C., Mikkelsen, G.L., Rabin, T., Toft, T., Nicolosi, A.A.: Efficient RSA key generation and threshold paillier in the two-party setting. JCRYPT **32**(2), 265–323 (2019)

[21] Hazay, C., Scholl, P., Soria-Vazquez, E.: Low cost constant round MPC combining BMR and oblivious transfer. In: ASIACRYPT. pp. 598–628 (2017)

[22] Ishai, Y., Ostrovsky, R., Zikas, V.: Secure multi-party computation with identifiable abort. In: CRYPTO. pp. 369–386 (2014)

[23] Joye, M., Pinch, R.: Cheating in split-knowledge rsa parameter generation. In: Workshop on Coding and Cryptography. pp. 157–163 (1999)

[24] Katz, J., Lindell, Y.: Introduction to Modern Cryptography, Second Edition, chap. Digital Signature Schemes, pp. 443–486. Chapman & Hall/CRC (2015)

[25] Knuth, D.E.: The Art of Computer Programming, Volume II: Seminumerical Algorithms (1969)

[26] Malkin, M., Wu, T., Boneh, D.: Experimenting with shared RSA key generation. In: NDSS. pp. 43–56 (1999)

[27] Miller, G.L.: Riemann's hypothesis and tests for primality. J. Comput. Syst. Sci. **13**(3), 300–317 (1976)

[28] Pietrzak, K.: Simple verifiable delay functions. In: ITCS. pp. 60:1–60:15 (2019)

[29] Poupard, G., Stern, J.: Generation of shared RSA keys by two parties. In: ASIACRYPT. pp. 11–24 (1998)

[30] Rabin, M.O.: Probabilistic algorithm for testing primality. Journal of Number Theory **12**(1), 128–138 (1980)

[31] Rivest, R.L.: A description of a single-chip implementation of the RSA cipher (1980)

[32] Rivest, R.L.: RSA chips (past/present/future). In: Theory and Application of Cryptographic Techniques. pp. 159–165. Springer (1984)

[33] Rivest, R.L., Shamir, A., Adleman, L.M.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM **21**(2), 120–126 (1978)

[34] Wang, X., Ranellucci, S., Katz, J.: Global-scale secure multiparty computation. In: ACM CCS. pp. 39–56 (2017)

[35] Wesolowski, B.: Efficient verifiable delay functions. In: EUROCRYPT. pp. 379–407 (2019)