

USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Secure Distributed RSA Modulus Generation Revisited: A Faster and More Communication-Efficient Construction

Shaojing Zhang, *College of Computer Science and Technology, Qingdao University;* Chengliang Tian, *College of Computer Science and Technology, Qingdao University, and State Key Laboratory of Cryptography and Digital Economy Security, Shandong University;* Ruixue Wang and Hequn Xian, *College of Computer Science and Technology, Qingdao University;* Guangwu Xu, *State Key Laboratory of Cryptography and Digital Economy Security and School of Cyber Science and Technology, Shandong University*

https://www.usenix.org/conference/usenixsecurity26/presentation/zhang-shaojing

This paper is included in the Proceedings of the 35th USENIX Security Symposium.

August 12–14, 2026 • Baltimore, MD, USA

ISBN 978-1-939133-58-8

Open access to the Proceedings of the 35th USENIX Security Symposium is sponsored by



جامعة الملك عبدالله للعلوم والتقنية

King Abdullah University of Science and Technology

ARTIFACT EVALUATED
USENIX
AVAILABLE

Secure Distributed RSA Modulus Generation Revisited: A Faster and More Communication-Efficient Construction

Shaojing Zhang[1], Chengliang Tian[1,2]*, Ruixue Wang[1], Hequn Xian[1], Guangwu Xu[2,3]
[1]*College of Computer Science and Technology, Qingdao University,*
[2]*State Key Laboratory of Cryptography and Digital Economy Security, Shandong University,*
[3]*School of Cyber Science and Technology, Shandong University*

Abstract

Securely generating an RSA modulus $N = pq$ in a distributed n-party setting while keeping its factors private is a core challenge in threshold cryptography. This paper revisits this problem and improves the state-of-the-art distributed Miller–Rabin primality test by replacing its core component—the divisibility test—with a novel zero-testing protocol.

Our key innovation is an elegant distributed design of Montgomery reduction, which transforms checking $\delta \bmod f = 0$ (given shares of δ and f) into testing whether a product equals zero. This approach enables efficient distributed evaluation and offers: (1) *Perfect correctness.* Unlike prior approaches that are probabilistic or require impractically large shares, our zero test decides $\delta \bmod f = 0$ deterministically. (2) *Improved efficiency.* Our zero test runs in only 7 rounds and uses 5 secure multiplications, improving over previous divisibility tests that require either $3 + \lceil \log_2 n \rceil$ rounds with $n+3$ multiplications or 8 rounds with $5n-1$ multiplications. Extensive experiments under honest-majority settings show increasing speedups for $n \geq 9$, with the efficiency advantage growing as the number of parties increases. Moreover, our semi-honest protocol supports both honest- and dishonest-majority settings via appropriate secure multiplication primitives, and can be upgraded to malicious security with standard compilers.

1 Introduction

Securely generating an RSA modulus $N = pq$—the product of two secret primes—in a distributed setting without leaking the factors p and q is a foundational task in threshold cryptography, e.g., for threshold RSA or threshold Paillier systems [17, 19, 24]. It also serves as a crucial building block in numerous modern cryptographic primitives and decentralized protocols, such as verifiable delay functions (VDFs) used in consensus mechanisms [2, 25, 28]. As a result, research on threshold-based distributed RSA modulus generation has transitioned from academia to industry, with organizations such as Unbound, the VDF Alliance, and various Ethereum-related initiatives now deploying services that leverage these technologies.

Boneh and Franklin [3], whose original motivation was to avoid the single-point-of-attack risk in classical RSA, pioneered the study of distributed RSA modulus generation. Their blueprint, followed by most subsequent works, consists of three main steps: (1) generating shares of candidate primes, (2) securely multiplying them to obtain a public candidate modulus $N = pq$, and (3) testing whether N is a biprime. The first two steps aim to produce candidates that are likely prime, thereby reducing the cost of the third step—the biprimality test, which is typically the most expensive part of the procedure. Unlike the plaintext setting, where a candidate is tested directly with a probabilistic Miller–Rabin test, the distributed setting requires operating on secret shares of the candidates. To avoid expensive modular exponentiation with a secret modulus, Boneh and Franklin introduced a *biprimality test* that exploits the fact that N is public while p and q remain secret-shared. Their test always accepts when p and q are prime, but may accept a non-RSA modulus with probability at most $1/2$ in the worst case. A recent refinement by Chuang et al. [8] reduced this worst-case error probability to $1/4$. Nevertheless, no stronger average-case guarantees are currently known. Consequently, to achieve a negligible soundness error, the biprimality test must still be repeated a number of times proportional to the security parameter. As a result, the cost of repeated executions becomes a major performance bottleneck in distributed RSA modulus generation. Moreover, biprimality testing requires both candidates to be sampled simultaneously, increasing the number of protocol invocations.

In contrast, the Miller–Rabin primality test exhibits significantly better average-case performance. Damgård et al. [9] proved that for sufficiently large integers, a single iteration already yields a negligible error probability, provided the input is chosen at random. The first distributed version of the Miller–Rabin test was introduced by Algesheimer, Camenisch and Shoup [1] for generating safe primes. Their protocol re-

*Corresponding author

quired distributed modular exponentiation with a secret modulus, resulting in high round and communication complexity. Damgård and Mikkelsen [10] later observed that when the candidate prime satisfies $f \equiv 3 \pmod 4$ and the composite N is public, the distributed Miller–Rabin test reduces to securely deciding whether $\delta \bmod f = 0$ given shares of δ and f, where $\delta = \gamma^{(f-1)/2} \pm 1$ and $\gamma \in \mathbb{Z}_N$ is a public random element. Leveraging the secure computation techniques from [1] and the fact that N is public, they designed a distributed variant of Barrett reduction that first approximates $\delta \bmod f$ and then transforms the problem into checking whether a product of several possible values equals zero. This reduced the round complexity from $O(n)$ to constant and—under a setup assumption that provides a commitment scheme—upgraded security from passive to active, representing a substantial advance. Nevertheless, their construction is inherently limited to a small number of parties (specifically three), and their divisibility test requires an auxiliary prime whose bit-length is $\Omega(n\kappa)$, where κ is the bit-length of the prime candidate. This makes the protocol inefficient for larger n.

A recent breakthrough by Burkhardt et al. [4] answered the open question of how to use the Miller–Rabin test for efficient distributed RSA key generation with more than three parties and without setup assumptions. Their core innovation is a novel divisibility test that transforms the problem of deciding $\delta \bmod f = 0$ into checking whether f divides δ. The test uses two public auxiliary primes P, Q (with $N < nP < Q$) and verifies that $\delta \cdot f^{-1} \bmod P$ equals $\delta \cdot f^{-1} \bmod Q$ when $f \mid \delta$; otherwise, equality holds only with negligible probability. The equality check is performed via a secure membership test. While this yields a constant-round protocol with a negligible error probability in the security parameter, two limitations remain: (1) achieving *perfect* correctness requires Q to be roughly twice as large as N, harming efficiency; (2) the protocol still requires $O(n)$ online secure multiplications.

This state of affairs motivates the following question:

> *Can we design a distributed Miller–Rabin test that employs a more efficient divisibility construction with perfect correctness, achieving both $O(1)$ online multiplications and $O(1)$ round complexity?*

1.1 Our Contribution

In this work, we build upon the line of research established by Algesheimer et al. [1] (CRYPTO'02), Damgård and Mikkelsen [10] (TCC'10), and more recently Burkhardt et al. [4] (CCS'23). We affirmatively answer the above-mentioned question by revisiting the core cryptographic task of secure modular reduction with a shared secret modulus: computing $\delta \bmod f$ given secret shares of δ and f.

Unlike the secure approximate division technique used by Algesheimer et al. and followed by Damgård and Mikkelsen, or the (probabilistic) divisibility test introduced by Burkhardt et al., we take inspiration from the fast Montgomery reduction algorithm [23] for modular arithmetic on large integers. A straightforward distributed simulation of Montgomery reduction could, in principle, handle modular reduction with a hidden modulus. However, a literal implementation would require secure truncation and expensive secure comparison protocols. We observe that deciding whether $\delta \bmod f = 0$ is fundamentally easier than computing the exact residue $\delta \bmod f$. Leveraging this insight, we refine the distributed Montgomery reduction approach and construct a novel zero-test protocol that achieves improved round and computational complexity for this specific decision problem.

Integrating this new building block, we present an optimized distributed construction of the Miller–Rabin primality test for generating an RSA modulus. Overall, our contributions are summarized as follows:

(1) **Perfect correctness.** The current state-of-the-art method for distributively deciding $\delta \bmod f = 0$ is the divisibility test of Burkhardt et al. [4], which is inherently probabilistic. Its core equivalence, $\delta \bmod f = 0 \iff \delta \cdot f^{-1} \bmod P = \delta \cdot f^{-1} \bmod Q$, can be made deterministic only by choosing an impractically large auxiliary primes P, Q, at a significant performance cost. Our zero-test protocol, in contrast, is deterministic by design. It builds on a distributed adaptation of Montgomery reduction and eliminates randomness entirely.

(2) **Theoretical efficiency.** Burkhardt et al. [4] suggest two implementation variants of their divisibility test: when instantiated with additive secret sharing, one requiring $3 + \lceil \log_2 n \rceil$ rounds and $n + 3$ secure multiplications, and a constant-round variant using 8 rounds and $5n - 1$ secure multiplications. Although the latter achieves constant round complexity, both variants scale linearly in the number of parties n in terms of secure multiplication count. In contrast, our new zero test completes in only 7 rounds and uses merely 5 secure multiplications, achieving complexity entirely independent of n. This yields a growing theoretical advantage as the system scales.

(3) **Practical performance.** Our experiments demonstrate that the performance advantage of the proposed protocol grows with the number of parties. In the honest-majority setting under the semi-honest security model, for RSA moduli of various bit lengths, our protocol consistently outperforms the state-of-the-art divisibility test of Burkhardt et al. once the number of parties reaches $n \geq 9$. Moreover, the performance gap widens as n grows. In particular, for a 4096-bit RSA modulus and $n = 105$ parties, Burkhardt et al.'s protocol incurs approximately $10\times$ higher total runtime than our new zero test protocol. These results confirm the superior scalability of our construction and demonstrate its practical benefits for large-scale distributed RSA modulus generation. A

comprehensive experimental evaluation is presented in Section 7.

(4) **Robust security.** Our construction treats secure multiplication as a black-box primitive, making it secure against semi-honest adversaries in both honest-majority setting and dishonest-majority setting (by instantiating the underlying multiplication appropriately). Moreover, since our protocol relies on the same family of cryptographic primitives, it can be upgraded to active security using established techniques analogous to those in [4], such as existing compilers (e.g., Damgård et al. [11]) or zero-knowledge proof frameworks.

1.2 Technique Overview

We are given shares $[p]_{\mathbb{Z}}$ and $[q]_{\mathbb{Z}}$ of two random numbers $p \equiv q \equiv 3 \bmod 4$, and a public composite $N = pq$ which can be obtained by employing a secure distributed multiplication protocol followed by a reconstruction step. Our main objective is to determine whether N is a product of two safe primes. Following the line of work in [1, 4, 10] that builds on the efficient Miller–Rabin primality test, we aim to redesign a more efficient secure distributed Miller–Rabin test based on the shares $[f]_{\mathbb{Z}}$ of a prime candidate f (where $f = p$ or q). Recall that in the standard Miller–Rabin test, for $f \equiv 3 \pmod 4$, we need to randomly and uniformly select $\gamma \in \mathbb{Z}_N$, compute $\delta = \gamma^{\frac{f-1}{2}} \pm 1 \bmod f$, and check whether $\delta = 0$. Hence, the key challenge in the distributed setting is to securely determine whether $\delta \bmod f = 0$, given the public modulus N, shares $[f]_{\mathbb{Z}}$, and shares of δ.

Our Approach. Unlike the roadmap of the recent work [4], which transforms the problem into a divisibility test, we directly design a distributed modular reduction algorithm based on Montgomery reduction. As is well known, for an odd integer f and two integers T, R satisfying $0 \le T < fR$ and $f < R = 2^l$, the Montgomery reduction algorithm computes

$$\frac{T + ((T \bmod R) f' \bmod R) f}{R} = TR^{-1} \bmod f + bf, \quad (1)$$

where $b \in \{0,1\}$ and $f' = -f^{-1} \bmod R$. Hence, by choosing a suitable R and setting $T = \delta \bmod N$, we obtain $TR^{-1} \bmod f = \delta R^{-1} \bmod f$. Building on the above observation and setting

$$d = T + ((T \bmod R) f' \bmod R) f, \quad (2)$$

we deduce

$$d \bmod f = T \bmod f = (\delta \bmod N) \bmod f = \delta \bmod f. \quad (3)$$

Thus, checking whether $\delta \bmod f = 0$ is equivalent to checking whether $d \bmod f = 0$. Moreover, since $R \mid d, 0 \le d < 2Rf$, and $\gcd(R, f) = 1$, the condition $d \bmod f = 0$ ultimately reduces to checking whether $d(d - Rf) = 0$.

Reduction to Secure Primitives. Thus the problem reduces to the following distributed computation: given N, $R = 2^l$, shares $[f]_{\mathbb{Z}}$ and $[\delta]_{\mathbb{Z}}$, compute securely

(1) $f' = -f^{-1} \bmod R$,

(2) d defined in equation (2), and

(3) $d\,(d - fR)$ (or its variant as shown below).

The first step (computing f') can be realized via an off-the-shelf secure distributed inversion protocol. The second step requires computing d, which involves modular reduction modulo $R = 2^l$ (a power of two, achievable via secure truncation) and secure multiplications. Specifically, we obtain d by interleaving secure truncation and secure multiplication. Finally, the last step computes $d\,(d - fR)$ using another secure multiplication. However, exact secure truncation in step 2 typically incurs the high cost of secure comparison protocols. To avoid this overhead, we further refine the approach by employing a probabilistic truncation method: the resulting error is absorbed into an additional possible value $d = -fR$, and we ultimately in step 3 compute the product $d\,(d - fR)(d + fR)$ in step 3 using secure multiplication.

Overall, within the above-mentioned framework and with the proposed optimization, we achieve an efficient primality testing using only a constant number of online secure multiplications.

2 Related Work

The problem of securely generating an RSA modulus in a distributed setting has attracted significant attention since the seminal work of Boneh and Franklin in 1997 [3], which provided security against passive (honest-but-curious) adversaries. In recent years, driven by emerging applications, the topic has regained widespread interest. As outlined in Section 1, most follow-up works adhere to the three-phase framework established by Boneh and Franklin. The most computationally intensive component is invariably the distributed biprimality test. Apart from the notable exceptions of Algesheimer et al. [1], Damgård and Mikkelsen [10], and Burkhardt et al. [4]— discussed in Section 1—the majority of subsequent research builds upon the Boneh-Franklin biprimality test. These works have introduced improvements from various angles, including communication and computation efficiency, semi-honest versus malicious threat models, and the number of participating parties.

While the [3] protocol established a comprehensive framework for distributed RSA key generation, its security was confined to the honest-but-curious setting. To overcome this limitation, Frankel et al. [14] made a key advancement by extending the security to the malicious model. Subsequently, Malkin et al. [21] systematically enhanced the efficiency of the Boneh-Franklin protocol through a series of optimization

techniques. In particular, for the key generation phase, critical methods such as distributed sieving and parallel candidate testing were introduced, thereby comprehensively optimizing the overall performance of the protocol. Poupard and Stern [26] addressed another critical drawback of [3]—its reliance on a trusted third party in the two-party scenario—by proposing, for the first time, a method for two-party RSA key generation that eliminates the need for any trusted third party. Similarly, Gilboa [16] is also a protocol that requires only two parties, but places greater emphasis on efficiency improvements. This semi-honest secure protocol provides three different methods for two-party computation of $N = pq$: protocols based on Oblivious Transfer (OT), Oblivious Polynomial Evaluation (OPE), and homomorphic encryption. Among them, the OT-based protocol outperforms [26] in terms of efficiency. Hazay et al. [18] introduced a secure trial division protocol tailored for pre-screening candidate primes between two parties. This effectively upgraded the pre-screening efficiency while elevating the security guarantee against active, malicious adversaries. In contrast, Frederiksen et al. [15] pursued a different optimization strategy focused on reducing verification overhead in the malicious model. Its core approach is to run the main protocol in a "strong" semi-honest mode, followed only by a lightweight zero-knowledge argument that attests to the correctness of the execution, leading to significant overall performance improvement.

In contrast to the conventional rejection sampling approach, Chen et al. [6] introduces an integrated sampling-and-sieving method that leverages the Chinese Remainder Theorem (CRT) to constructively sample candidate primes. This approach not only addresses the inherent information leakage present in the [15] protocol but also achieves a significant improvement in efficiency. In subsequent work, Chen et al. [7] implemented the first MPC protocol capable of scaling beyond 1,000 parties, devised a protocol resilient to passive corruption at this scale, and demonstrated the practical viability of large-scale distributed generation of RSA moduli. Then, Delpech et al. [12] significantly improved the communication efficiency compared to [6] by semi-honest multiplication and postponing the even costlier consistency check on the shares contributed to the Jacobi test.

In Table 1, we provide a detailed comparison of the aforementioned distributed RSA protocols, contrasting their respective primality testing methods, security models, and supported number of parties.

3 Preliminaries

To facilitate readability and comparison, we follow the blueprint and use the same notations as in [4]. Let $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ be the set of n parties and $[n] = \{1, 2, \ldots, n\}$. The bit-length of an RSA modulus N is denoted by $|N| = \lfloor \log_2 N \rfloor + 1$. For a secret value x, we write $[x]_\alpha$ for its additive sharing over $\mathbb{Z}_\alpha$, where each party P_i holds a share $[x]_\alpha^{(i)} \in \mathbb{Z}_\alpha$ such that $\sum_{i\in[n]} [x]_\alpha^{(i)} \equiv x \pmod{\alpha}$. Similarly, $\langle x \rangle_\alpha$ denotes a multiplicative sharing of x modulo α, with each P_i holding a share $\langle x \rangle_\alpha^{(i)}$ satisfying $\prod_{i\in[n]} \langle x \rangle_\alpha^{(i)} \equiv x \pmod{\alpha}$.

Given a constant $c \in \mathbb{Z}_\alpha$ and two secret values $x, y \in \mathbb{Z}_\alpha$, the operation $c[x]_\alpha + [y]_\alpha$ is performed locally: each party computes $c[x]_\alpha^{(i)} + [y]_\alpha^{(i)} \bmod \alpha$. The operation $[x]_\alpha + c$ only requires one party (without loss of generality, P_1) to compute $[x]_\alpha^{(1)} + c \bmod \alpha$, while the other parties take no action. Similarly, for multiplicative sharing, $\langle xy \rangle_\alpha$ can be computed locally: each party multiplies its own shares $\langle x \rangle_\alpha^{(i)}$ and $\langle y \rangle_\alpha^{(i)}$. Throughout this work, we focus on the setting with static corruptions and synchronous networks.

3.1 Miller-Rabin Primality Test

The Miller–Rabin primality test is a probabilistic algorithm widely employed to determine whether a given integer f is prime. Owing to its efficiency and practical reliability, it is extensively used in cryptographic applications, particularly for testing large integers. The algorithm was originally introduced by Miller [22] in 1976 and later refined by Rabin [27] in 1980, resulting in a provably effective method that does not rely on the Riemann hypothesis. The test is based on two fundamental facts: (1) Fermat's little theorem, which asserts that $a^{f-1} \equiv 1 \pmod{f}$ for any $a \in \mathbb{Z}_f$ if f is prime, and (2) the property that, modulo a prime f, the only square roots of 1 are 1 and -1.

In our distributed RSA setting, the candidate prime f is further restricted to satisfy $f \equiv 3 \pmod 4$, which leads to a more concise description of the test, as shown in **Algorithm 1**. The correctness of the algorithm is formally stated by the following theorem.

Algorithm 1 MillerRabin(f,t)

Input: A large odd number $f \equiv 3 \bmod 4$ and the number of iterations t.
Output: "probably prime" or "composite"

1: Calculate the odd number $e = \frac{f-1}{2}$.
2: **for** $i = 1$ to t **do**
3: Select a random integer $\gamma_i \in [2, f-2]$.
4: $x \leftarrow \gamma_i^e \bmod f$.
5: $\delta \leftarrow x \pm 1 \bmod f$.
6: **if** $\delta \neq 0$ **then**
7: **return** ('composite')
8: **end if**
9: **end for**
10: **return** ('probably prime').

Theorem 1 ([27]). *If f is prime, the Miller–Rabin test always outputs "probably prime". If f is composite, the probability that the test outputs "probably prime" is about 4^{-t} for the worst case.*

Table 1: Comparison of existing distributed RSA modulus generation protocols

Scheme	Primality Test	Security Model	Parties
Boneh and Franklin [3]	Boneh-Franklin	Semi-honest	≥ 3
Frankel et al. [14]	Boneh-Franklin	Malicious	≥ 3
Poupard and Stern [26]	Boneh-Franklin	Malicious	2
Gilboa [16]	Boneh-Franklin	Semi-honest	2
Malkin et al. [21]	Boneh-Franklin	Semi-honest	≥ 3
Algesheimer et al. [1]	Miller-Rabin	Semi-honest	≥ 3
Damgård and Mikkelsen [10]	Miller-Rabin	Malicious	3
Hazay et al. [18]	Boneh-Franklin	Malicious	≥ 2
Frederiksen et al. [15]	Boneh-Franklin	Malicious	2
Delpech de Saint Guilhem et al. [12]	Boneh-Franklin	Malicious	≥ 2
Chen et al. [7]	Boneh-Franklin	Malicious	≥ 2
Chen et al. [6]	Boneh-Franklin	Malicious	≥ 2
Burkhardt et al. [4]	Miller-Rabin	Malicious	≥ 2

We remark that although a single iteration of the Miller–Rabin test may, in the worst case, accept a composite number with probability up to $1/4$, the analysis of Damgård et al. [9] and the subsequent refinement for random integers satisfying $f \equiv 3 \pmod 4$ by Damgård and Mikkelsen [10] show that the error probability decreases rapidly for sufficiently large inputs. More precisely, for a uniformly random κ-bit integer $f \equiv 3 \bmod 4$, the average-case error probability after one iteration is bounded by $\kappa^2 \cdot 4^{3-\sqrt{\kappa}}$. More generally, for $t (2 \leq t \leq 9)$, the error probability after t independent iterations is bounded by $\kappa^{3/2} 2^{t-1/2} 4^{3-\sqrt{t\kappa}}$. Table 2 reports numerical estimates of these bounds for commonly used RSA parameter sizes. As the table shows, for RSA moduli of practical interest ($\kappa \geq 1024$), the error probability becomes negligible after only a small number of iterations. Consequently, setting $t \leq 3$ in **Algorithm 1** is sufficient in practice, depending on the candidate size and the desired security level.

Table 2: Estimated average-case error bounds after t Miller–Rabin iterations for a random κ-bit candidate satisfying $f \equiv 3 \pmod 4$.

κ	1024	1536	2048
$t = 1$	2^{-38}	2^{-51}	2^{-62}
$t = 2$	2^{-68}	2^{-87}	2^{-104}
$t = 3$	2^{-87}	2^{-111}	2^{-132}

3.2 Montgomery Reduction

Given an odd integer f and a modulus $R = 2^l$, let $0 \leq T < fR$. The well-known Montgomery reduction algorithm, first introduced in 1985 [23], is designed to compute $TR^{-1} \bmod f$ efficiently using only multiplication and shift operations, which is typically more efficient than direct division. This is achieved by adding a carefully chosen small multiple of f to cancel the residue modulo R. The correctness of the algorithm can be interpreted via the Chinese Remainder Theorem [29]. The detailed description can be found in **Algorithm 2**.

Algorithm 2 $\mathsf{MoRe}(R, f', T)$

Input: Integers $R, f' = -f^{-1} \bmod R$ and $T \in [0, fR)$
Output: $TR^{-1} \bmod f$

```
m = ((T mod R) · f') mod R
res = (T + m · f)/R
if res ≥ f then
    res = res − f
end if
return res
```

3.3 Secure Distributed Computing Primitives

In this section, we introduce the fundamental secure distributed computing protocols necessary for our construction. We specify the inputs and outputs of these primitives and employ them as modular components. Details are provided in Table 3.

As is shown in Table 3, our new construction requires some basic primitives from Burkhardt et al. [4] and Catrina et al. [5] together with the secure multiplication and secure comparison operation, which are black-box components. For the complexity analysis that follows, we provide the communication cost of each primitive; detailed descriptions of these primitives can be found in the respective references. As in Burkhardt et al. [4], our construction only needs black-box secure multiplication between two secret values, which typically requires one communication round. Importantly, thanks to the optimizations presented in Section 4.2, our construction

Table 3: Secure distributed computing primitives

Ref	Primitive	Functionality	Rounds
[4]	Int-To-Mod$(\mathbb{Z}_\beta,[x]_\alpha)$	Translate the additive shares over $\mathbb{Z}$ into the additive shares over $\mathbb{Z}_\beta$	0
[4]	Larger-Domain$(\mathbb{Z}_\beta,[x]_\alpha)$	Translate the additive shares over $\mathbb{Z}_\alpha$ into the additive shares over $\mathbb{Z}_\beta$	0
[4]	random-sample$(\mathbb{Z}_A,l)$	Generate shares of a uniformly random value with $0 \le x < 2^l < A$.	0
black-box	SecMult$([x]_\beta,[y]_\beta)$	Compute the shares over $\mathbb{Z}_\beta$ of the product of two secret values x,y	1
black-box	SecCmp$([x]_\alpha,y)$	Compute the shares of the result of the comparison $x > y$	r_c
[4]	Invert$([x]_\alpha)$	For shares of $x \in \mathbb{Z}_\alpha^*$, compute the shares over $\mathbb{Z}_\alpha$ of x^{-1} mod α	2
[5]	SecTrunc$([x]_\alpha,R)$	For public $R = 2^l$, compute the shares of x mod R over $\mathbb{Z}_\alpha$	$1+r_c$

does not actually require any secure comparison on secret values. Therefore, we treat the comparison protocol SecCmp as a black box that takes r_c rounds, regardless of whether r_c is a constant or a function of other parameters.

Moreover, our construction needs one invocation of Invert with the modulus $R = 2^l (l > \kappa)$, where κ refers to the bit length of the random candidate primes p,q. Although Burkhardt et al. [4] requires the modulus of the primitive Invert to be a prime to guarantee the security, it suffices to select a multiplicative cyclic group of sufficiently large order. Therefore, the security of our invocation with modulus R still holds.

To simplify the theoretical analysis, we assume that all the randomness has been generated in the preprocessing phase and will be given to the parties before running the protocols. Thus, the protocol random-sample$(\mathbb{Z}_A,l)$ does not require any communication in online phase.

4 Our New Construction

Building on the computational number-theoretic and secure distributed computing primitives introduced earlier, we now present our new protocol. We begin by restating the problem: in a distributed system of n parties $P_1,\cdots,P_n$, the goal is to jointly and securely generate shares of RSA key pair (p,q) such that p,q are prime numbers with $p,q \equiv 3 \bmod 4$. An efficient way to generate candidates, suggested by prior works [1, 3], is as follows: every party except P_1 selects a random $(\kappa - \lfloor \log_2 n \rfloor - 1)$-bit share $[x]_\mathbb{Z}^{(i)} \equiv 0 \pmod 4$, and P_1 selects a random $(\kappa - \lfloor \log_2 n \rfloor - 1)$-bit number $\widetilde{[x]}_\mathbb{Z}^{(1)} \equiv 3 \pmod 4$ and sets $[x]_\mathbb{Z}^{(1)} = 2^{\kappa-1} + \widetilde{[x]}_\mathbb{Z}^{(1)}$. Thus $x = \sum_{i=1}^n [x]_\mathbb{Z}^{(i)}$ will be a κ-bit number. Although the distribution of x is not uniform, one can show that it has at least $(\kappa - \lfloor \log_2 n \rfloor - 1)$ bits of entropy [3]. After that, they run a secure multiplication protocol with input the two prime candidates to open a public modulus $N = pq$. The core problem thus reduces to the following: given shares $[f]_\mathbb{Z}$ of a candidate f (where $f = p$ or q) with $f \equiv 3 \pmod 4$, and given $N = pq$, how can the primality of f be decided efficiently in a distributed manner? To solve this, we adopt the distributed Miller–Rabin framework as in [1, 4, 10] (see the schematic description in **Algorithm 3**). The central task is to securely and efficiently determine whether $\delta \bmod f = 0$ given shares of δ and f.

4.1 A Warm-Up Protocol

Algorithm 3 SecMiRa$([f]_\mathbb{Z},N,R,M)$

Input: $[f]_\mathbb{Z}$ where $f \in \{p,q\}$ satisfies $f \in [2^{\kappa-1},2^\kappa] \wedge f \equiv 3 \pmod 4$. Here, the share $[f]_\mathbb{Z}^{(1)}$ is odd and the other shares are even. Public parameter $N = pq, R = 2^{\kappa+\log_2 n}$ and a public prime $M > 2^{3(\kappa+\log_2 n)+\sigma}$, where n refers to the numebr of the parties and σ is the securtiy parameter.

Output: The test result y, where $y = 0$ indicates f passes the primality test

1: Locally compute $[e]_\mathbb{Z}^{(1)} = \frac{[f]_\mathbb{Z}^{(1)}-1}{2}$ and $[e]_\mathbb{Z}^{(i)} = \frac{[f]_\mathbb{Z}^{(i)}}{2} (i \neq 1)$
2: Sample a random integer $\gamma \in \mathbb{Z}_N$
3: Locally compute $\langle x \rangle_N \leftarrow \gamma^{[e]_\mathbb{Z}} \bmod N$
4: Compute $[x]_N \leftarrow$ Mul-To-Add$(\langle x \rangle_N)$
5: Compute $[y_1]_M \leftarrow$ SecMoRe$([x+1]_N,[f]_\mathbb{Z},N,R,M)$
6: Compute $[y_2]_M \leftarrow$ SecMoRe$([x-1]_N,[f]_\mathbb{Z},N,R,M)$
7: Compute $[y]_M \leftarrow$ SecMult$([y_1]_M,[y_2]_M)$
8: $y \leftarrow$ OpenAll$([y]_M)$
9: **return** y

In **Algorithm 3**, Steps 1–3 involve only local computations, while the multiplication-to-additive-share conversion in Step 4 can be efficiently realized using existing secure multiplication protocols. Therefore, the main challenge lies in Steps 5–6, which require the distributed computation of $\delta \bmod f$, where $\delta \in \{x+1, x-1\}$.

To address this challenge, we first present a reduction based on a direct distributed realization of the Montgomery reduction algorithm. Recall that, given an integer $T \in [0, Rf)$, Montgomery reduction (see **Algorithm 2**) outputs $TR^{-1} \bmod f$ without performing an explicit reduction modulo f. Therefore, by choosing a sufficiently large power of two R and setting $T = \delta R \bmod N$, the output becomes $TR^{-1} \bmod f = (\delta R \bmod N) \cdot R^{-1} \bmod f = \delta \bmod f$. Consequently, computing $\delta \bmod f$ can be reduced to securely evaluating the Montgomery reduction procedure in a distributed setting.

Algorithm 4 SecMoRe$([\delta]_N, [f]_{\mathbb{Z}}, N, R, M)$

Input: The shares $[\delta]_N, [f]_{\mathbb{Z}}$ and the public numbers $N = pq, R = 2^{\kappa+\log_2 n}, M > 2^{3(\kappa+\log_2 n)+\sigma}$, where $f \in \{p,q\} \wedge f \in [2^{\kappa-1}, 2^{\kappa}]$, n refers to the number of parties and σ is the security parameter.

Output: The shares $[res]_M = [\delta \bmod f]_M$

1: $[T]_N = [\delta]_N \cdot R$
2: $[f]_R = \mathsf{Int\text{-}To\text{-}Mod}(\mathbb{Z}_R, [f]_{\mathbb{Z}})$
3: $[f]_M = \mathsf{Int\text{-}To\text{-}Mod}(\mathbb{Z}_M, [f]_{\mathbb{Z}})$
4: $[f']_R = -\mathsf{Invert}([f]_R)$
5: $[T + c_1 N]_M = \mathsf{Larger\text{-}Domain}(\mathbb{Z}_M, [T]_N)$
6: $[f' + c_2 R]_M = \mathsf{Larger\text{-}Domain}(\mathbb{Z}_M, [f']_R)$
7: $[z]_M = \mathsf{SecMult}([T + c_1 N]_M, [f' + c_2 R]_M)$
8: $[z']_M = \mathsf{SecTrunc}([z]_M, R)$
9: $[z'f]_M = \mathsf{SecMult}([z']_M, [f]_M)$
10: $[res]_M = ([T + c_1 N]_M + [z'f]_M) \cdot (R^{-1} \bmod M)$
11: $[c]_M = \mathsf{SecCmp}([res]_M, R)$
12: $[res]_M = [res]_M - [c]_M \cdot R$
13: **return** $[res]_M$

Following this observation, we construct a step-by-step distributed implementation of Montgomery reduction using a small set of standard secure primitives, namely secure inversion, secure multiplication, secure truncation, and secure comparison. The resulting protocol, denoted by (SecMoRe), is presented in **Algorithm 4** and serves as a warm-up construction for our final protocol.

The correctness of SecMoRe follows directly from the correctness of Montgomery reduction. Moreover, since the protocol is obtained by composing standard secure primitives and introduces no additional information leakage, its security follows immediately from the security of the underlying building blocks and the standard composition theorem. However, this literal distributed simulation of the plaintext Montgomery reduction algorithm inherently involves secure truncation and secure comparison, which typically incurs high communication and computational overhead.

4.2 Our Optimized Design

We observe that deciding whether $\delta \bmod f = 0$ is fundamentally easier than computing the exact residue value. We therefore can further optimize the SecMoRe protocol to avoid invoking expensive sub-protocols altogether.

As shown by equation (3) and the related discussion in Section 1.2,

$$\begin{aligned} &\delta \bmod f = 0 \Leftrightarrow d \bmod f = 0 \\ \Leftrightarrow &d = 0 \ \vee \ d = fR \Leftrightarrow d(d - fR) = 0, \end{aligned} \tag{4}$$

where

$$\begin{aligned} d &= T + \big((T \bmod R)\, f' \bmod R\big) f \\ &= \delta \bmod N + \big(((\delta \bmod N)\, f') \bmod R\big) f. \end{aligned}$$

The derivation above implies that deciding whether $\delta \bmod f = 0$ does not require computing the exact value of $\delta \bmod f$; it suffices to evaluate d. However, computing d directly involves a secure truncation sub-routine to obtain $z = \big((\delta \bmod N)\, f'\big) \bmod R$, which in turn relies on expensive secure comparisons. To bypass the comparison, we employ a probabilistic truncation method that outputs shares of $z \bmod R - bR$ for $b \in \{0,1\}$ using only a single opening and no secure comparison. This modification expands the possible range of d from $0 \le d < 2fR$ to $-fR \le d < 2fR$. Consequently, the equivalence can be refined to

$$\begin{aligned} &\delta \bmod f = 0 \Leftrightarrow d \bmod f = 0 \\ &\Leftrightarrow d(d - fR)(d + fR) = 0. \end{aligned} \tag{5}$$

Building on this observation, we detail our optimization steps below.

Optimization 1: Secure Probabilistic Reduction Modulo R

Our first optimization eliminates the secure comparison from the standard secure truncation protocol (e.g., as in [5]), yielding a probabilistic truncation protocol SecProTrunc (**Algorithm 5**). Instead of returning the exact value $z \bmod R$, the protocol outputs shares of $(z \bmod R) - bR$ where $b \in \{0,1\}$.

Note that the largest integer appearing in **Algorithm 5** is $c = z + r_1 R + r_2$, which satisfies $c < 2^{m+\sigma+1} < 2^{3(k+\lceil \log_2 n \rceil)+\sigma} < M$; hence no overflow occurs modulo M. The bit b originally indicated the result of the secure comparison $(c \bmod R) < r_2$. By removing the comparison entirely while preserving the arithmetic structure, the protocol still correctly outputs a value of the form $(z \bmod R) - bR$ for some $b \in \{0,1\}$, as justified by the discussion above.

Algorithm 5 SecProTrunc$([x]_M, R)$

Input: $[z]_M, R, z < 2^m, R = 2^l, m > l$

Output: $[k]_M = [z \bmod R - bR]_M, b \in \{0,1\}$

1: $[r_1]_M = \textbf{random-sample}(\mathbb{Z}_M, m - l + \sigma)$
2: $[r_2]_M = \textbf{random-sample}(\mathbb{Z}_M, l)$
3: $[c]_M = [z]_M + [r_1]_M \cdot R + [r_2]_M$
4: $c = \textbf{OpenAll}([c]_M)$
5: $[k]_M = (c \bmod R) - [r_2]_M$

For completeness, we provide a simulator for SecProTrunc (see **Algorithm 6**) to formally prove the passive security of the modified construction. Without loss of generality, we assume that party P_1 is honest and the remaining parties $\mathcal{P} \setminus \{P_1\}$ are corrupted; this assumption is used consistently in all subsequent simulators. The simulator takes as input the protocol

Algorithm 6 $\mathcal{S}_{\mathsf{SecProTrunc}}([x]_M, R, \text{state}_0)$

Simulate:

1: $([r_1]_M, \text{view}_1, \text{state}_1) \leftarrow \mathcal{S}_{\mathsf{random\text{-}sample}}(\mathbb{Z}_M, m-l+\sigma, \text{state}_0)$
2: $([r_2]_M, \text{view}_2, \text{state}_2) \leftarrow \mathcal{S}_{\mathsf{random\text{-}sample}}(\mathbb{Z}_M, l, \text{state}_1)$
3: Set $[c]_M = [x]_M + [r_1]_M \cdot R + [r_2]_M$
4: Pick $r \in [0, 2^{m+\sigma})$ uniformly at random
5: Set $c^{(1)} = r - \sum_{i=2}^{n} c^{(i)}$
6: Set $c' = c \bmod R$
7: Set $[k]_M = (c' - r_2^{(1)}, -r_2^{(2)}, \ldots, -r_2^{(n)})$
8: Update $\text{state}_3 = \text{state}_2 \cup \{[k]_M\}$
9: Set $\text{view} = \text{view}_1 \cup \text{view}_2 \cup \{c^{(1)}\}$
10: **return** $([k]_M, \text{view}, \text{state}_4)$

inputs and a state variable that is passed and updated across all sub-simulators to maintain consistency of random values and intermediate results. Each sub-simulator receives the already simulated values and the current state, and outputs new simulated values, an updated state, and a sub-view for the corrupted parties.

The simulator $\mathcal{S}_{\mathsf{SecProTrunc}}$ first executes the same operations as the real protocol. Instead of opening the real value c, it reveals a uniformly random value r and sets the share of P_1 to $c^{(1)}$. Because the statistical distance between r and c is negligible in the security parameter, the adversary's view before Step 5 is statistically indistinguishable from the simulated view, which includes the honest party P_1's share of c. Hence, the protocol SecProTrunc remains secure against semi-honest adversaries.

Optimization 2: *Replacing* SecCmp *with* SecMult

Additionally, we have already discussed that $\delta \bmod f = 0$ if and only if $d(d+fR)(d-fR) = 0$, which reduces the problem of checking whether $\delta \bmod f = 0$ to check whether the product of these three secret values equals 0. Therefore, we can further optimize the secure comparison in step 10-step 12 of **Algorithm 4** with the two secure multiplications. Also, there is no need to compute $R^{-1} \bmod M$ now.

Using the two optimization strategies described above, we transform the expensive protocol SecMoRe (**Algorithm 4**)—which computes the exact value of $\delta \bmod f$— into an efficient zero-test protocol SecZeroTest that only decides whether $\delta \bmod f = 0$. The detailed pseudocode is presented in **Algorithm 7**.

Because our optimizations remove all secure comparison operations, the resulting protocol SecZeroTest requires only a constant number of secure multiplications, making it highly efficient. For completeness, we analyze the correctness and security of SecZeroTest in Section 5, and provide a full complexity analysis in Section 6.

Finally, we replace steps 5 and 6 in **Algorithm 3** with our SecZeroTest protocol, thereby obtaining the optimized secure Miller– Rabin test presented in **Algorithm 8**.

Algorithm 7 $\mathsf{SecZeroTest}([\delta]_N, [f]_Z, N, R, M)$

Input: The shares $[\delta]_N, [f]_{\mathbb{Z}}$ and the public numbers $N = pq, R = 2^{\kappa+\log_2 n}, M > 2^{3(\kappa+\log_2 n)+\sigma}$, where $f \in \{p,q\} \wedge f \in [2^{\kappa-1}, 2^{\kappa}]$, n refers to the number of parties and σ is the security parameter.

Output: The shares of y, which equals 0 if and only if $\delta \bmod f = 0$

1: $[f]_R = \mathsf{Int\text{-}To\text{-}Mod}(\mathbb{Z}_R, [f]_{\mathbb{Z}})$
2: $[f]_M = \mathsf{Int\text{-}To\text{-}Mod}(\mathbb{Z}_M, [f]_{\mathbb{Z}})$
3: $[f']_R = -\mathsf{Invert}([f]_R)$
4: $[\delta + c_1 N]_M = \mathsf{Larger\text{-}Domain}(\mathbb{Z}_M, [\delta]_N)$
5: $[f' + c_2 R]_M = \mathsf{Larger\text{-}Domain}(\mathbb{Z}_M, [f']_R)$
6: $[z]_M = \mathsf{SecMult}([\delta + c_1 N]_M, [f' + c_2 R]_M)$
7: $[z']_M = \mathsf{SecProTrunc}([z]_M, R)$
8: $[d]_M = [\delta + c_1 N]_M + \mathsf{SecMult}([z']_M, [f]_M)$
9: $[y]_M = \mathsf{SecMult}([d]_M, [d]_M + [f]_M \cdot R)$
10: $[y]_M = \mathsf{SecMult}([y]_M, [d]_M - [f]_M \cdot R)$
11: **return** $[y]_M$

Algorithm 8 $\mathsf{OptSecMiRa}([f]_{\mathbb{Z}}, N, R, M)$

Input: $[f]_{\mathbb{Z}}$ where $f \in \{p,q\}$ satisfies $f \in [2^{\kappa-1}, 2^{\kappa}] \wedge f \equiv 3 \pmod 4$. Here, the share $[f]_{\mathbb{Z}}^{(1)}$ is odd and the other shares are even. Public parameter $N = pq, R = 2^{\kappa+\log_2 n}$ and a public prime $M > 2^{3(\kappa+\log_2 n)+\sigma}$, where n refers to the numebr of the parties and σ is the securtiy parameter.

Output: The test result y, where $y = 0$ indicates f passes the primality test.

1: Locally compute $[e]_{\mathbb{Z}}^{(1)} = \frac{[f]_{\mathbb{Z}}^{(1)} - 1}{2}$ and $[e]_{\mathbb{Z}}^{(i)} = \frac{[f]_{\mathbb{Z}}^{(i)}}{2} (i \neq 1)$
2: Sample a random integer $\gamma \in \mathbb{Z}_N$
3: Locally compute $\langle x \rangle_N \leftarrow \gamma^{[e]_{\mathbb{Z}}} \bmod N$
4: Compute $[x]_N \leftarrow \mathsf{Mul\text{-}To\text{-}Add}(\langle x \rangle_N)$
5: Compute $[y_1]_M \leftarrow \mathsf{SecZeroTest}([x+1]_N, [f]_{\mathbb{Z}}, N, R, M)$
6: Compute $[y_2]_M \leftarrow \mathsf{SecZeroTest}([x-1]_N, [f]_{\mathbb{Z}}, N, R, M)$
7: Compute $[y]_M \leftarrow \mathsf{SecMult}([y_1]_M, [y_2]_M)$
8: $y \leftarrow \mathsf{OpenAll}([y]_M)$
9: **return** y

5 Correctness and Security Analysis

In this section, we first establish the perfect correctness of the protocol SecZeroTest, showing that it decides $\delta \bmod f = 0$ exactly. Then, we provide a simulation-based proof of its security.

Lemma 1. *Given the shares $[\delta]_N, [f]_{\mathbb{Z}}$ and the modulus M used for modular operations in the secret-sharing scheme, satisfying $M > 2^{3(k+\log_2 n)+\sigma}$, the protocol* SecZeroTest *returns the share $[y]_M$ satisfying $[y]_M = [0]_M$ if and only if $\delta \bmod f = 0$, which indicates* ***Algorithm*** *7 has perfect correctness.*

Proof. By step 6, the protocol computes the shares of $z = (\delta +$

$c_1N)(f'+c_2R)$, where $0 \leq c_1, c_2 < n$. Since step 7 invokes SecProTrunc, we know that

$$z' = z \bmod R - bR = ((\delta + c_1N)f') \bmod R - bR, b \in \{0,1\},$$
$$d = (\delta + c_1N) + z'f$$
$$= (\delta + c_1N) + (((\delta + c_1N)f') \bmod R)f - bfR, b \in \{0,1\}.$$

From the fact that $f \mid N$ and $f' = -f^{-1} \bmod R$, we can deduce that $d \bmod f = \delta \bmod f \wedge R \mid d$. Also, we can assert that the range of d can be bounded by $d \in [-fR, 2fR)$ since $\delta + c_1N < nN < fR$. Thus, we draw the conclusion that:

$$\delta \bmod f = 0 \Leftrightarrow d \bmod f = 0 \Leftrightarrow d \bmod fR = 0$$
$$\Leftrightarrow d = -fR \vee d = 0 \vee d = fR$$
$$\Leftrightarrow y = (d + fR)d(d - fR) = 0$$

Additionally, since we choose the moduli of the secret sharing scheme M as a prime with $M > 2^{3(k+\log_2 n)+\sigma}$, there is no situation that the product of non-zero values in $\mathbb{Z}_M$ can be 0. Above all, we can deduce that $\delta \bmod f = 0$ if and only if the protocol outputs the shares of 0, i.e. $[y]_M = [0]_M$, which proves that the protocol SecZeroTest has perfect correctness. □

Algorithm 9 $\mathcal{S}_{\mathsf{SecZeroTest}}([\delta]_N, [f]_{\mathbb{Z}}, \text{state}_0)$

Simulate:

1: $([f]_R, \text{view}_1, \text{state}_1) \leftarrow \mathcal{S}_{\mathsf{Int\text{-}To\text{-}Mod}}(\mathbb{Z}_R, [f]_{\mathbb{Z}}, state_0)$
2: $([f]_M, \text{view}_2, \text{state}_2) \leftarrow \mathcal{S}_{\mathsf{Int\text{-}To\text{-}Mod}}(\mathbb{Z}_M, [f]_{\mathbb{Z}}, \text{state}_1)$
3: $([f']_R, \text{view}_3, \text{state}_3) \leftarrow \mathcal{S}_{\mathsf{Invert}'}([f]_R, \text{state}_2)$
4: $([\delta_1]_M, \text{view}_4, \text{state}_4) \leftarrow \mathcal{S}_{\mathsf{Larger\text{-}Domain}}(\mathbb{Z}_M, [\delta]_N, \text{state}_3)$
5: $([f'_1]_M, \text{view}_5, \text{state}_5) \leftarrow \mathcal{S}_{\mathsf{Larger\text{-}Domain}}(\mathbb{Z}_M, [f']_R, \text{state}_4)$
6: $([z]_M, \text{view}_6, \text{state}_6) \leftarrow \mathcal{S}_{\mathsf{SecMult}}([\delta_1]_M, [f'_1]_M, \text{state}_5)$
7: $([z']_M, \text{view}_7, \text{state}_7) \leftarrow \mathcal{S}_{\mathsf{SecProTrunc}}([z]_M, R, \text{state}_6)$
8: $([z'f]_M, \text{view}_8, \text{state}_8) \leftarrow \mathcal{S}_{\mathsf{SecMult}}([z']_M, [f]_M, \text{state}_7)$
9: Set $[d]_M = [\delta_1]_M + [z'f]_M$
10: Update $\text{state}_8 = \text{state}_8 \cup \{[d]_M\}$
11: $([y_1]_M, \text{view}_9, \text{state}_9) \leftarrow \mathcal{S}_{\mathsf{SecMult}}([d]_M, [d]_M + [f]_M \cdot R, \text{state}_8)$
12: $([y]_M, \text{view}_{10}, \text{state}_{10}) \leftarrow \mathcal{S}_{\mathsf{SecMult}}([y_1]_M, [d]_M - [f]_M \cdot R, \text{state}_9)$
13: Set $\text{view} = \bigcup_{i=1}^{10} \text{view}_i$
14: **return** $([y]_M, \text{view}, \text{state}_{10})$

5.1 Passive Security

Now, we show that the view of the adversary in the real protocol execution is computationally indistinguishable from the view generated by the simulator, thereby establishing the security of the protocol in the semi-honest model.

Simulator: The simulator of SecZeroTest, denoted as $\mathcal{S}_{\mathsf{SecZeroTest}}$, is presented in **Algorithm 9**. Note that the simulator $\mathcal{S}_{\mathsf{Invert}'}$ derives from the simulator $\mathcal{S}_{\mathsf{Invert}}$ by adding a multiplication by -1 in the final step of Invert. The simulators $\mathcal{S}_{\mathsf{Int\text{-}To\text{-}Mod}}$, $\mathcal{S}_{\mathsf{Larger\text{-}Domain}}$ and $\mathcal{S}_{\mathsf{SecMult}}$ have been discussed by Burkhardt et al. in [4], we refer the readers to their paper for more details. It can be seen that the simulator simply invokes the sub-simulators of the underlying protocols, whose outputs' views are indistinguishable from the view in the actual protocol. Additionally, all the values that are opened during the protocol are the direct output of a secret value randomly masked by either adding a sufficiently large number or multiplying a random value. Since adding two secret values can be done by local computations, we can thus assert that the security of the protocol SecZeroTest relies on the implementation of the protocl SecMult only, which leads to the following:

Lemma 2. *Assuming a secure implementation of the protocol* SecMult*, the protocol* SecZeroTest *securely implements the ideal functionality* $\mathcal{F}_{\mathsf{SecZeroTest}}$ *against semi-honest adversaries.*

5.2 Upgrade to Active Security

Similar to the approach of Burkhardt et al. [4], our semi-honest protocols can be upgraded to withstand active adversaries at moderate cost using existing compilers. Two main strategies exist. The first applies the compiler from [11], which preserves the protocol structure but reduces corruption tolerance while increasing computational and communication overhead. The second executes the semi-honest protocol and uses zero-knowledge proofs to verify correctness. Although this approach can tolerate a larger number of corruptions, it incurs additional performance overhead.

Here, we focus on upgrading our passively secure protocols using the compiler from Eerikson et al. [20], an optimized version of the compiler introduced by Damgård et al. [11]. These compilers provide generic mechanisms to verify both the consistency of secret shares and the correctness of protocol execution in the presence of malicious adversaries.

For SecZeroTest, the only interactive operations are the opening procedures used in Invert and SecProTrunc, together with secure multiplications. Existing compilers already provide actively secure realizations of these primitives by enforcing share consistency and correctness of execution. Therefore, upgrading SecZeroTest to active security mainly requires replacing each passively secure multiplication and opening operation with its actively secure counterpart, while leaving all local computations unchanged.

For the secure Miller–Rabin protocol OptSecMiRa, an additional modification is required. Besides actively secure multiplications and openings, the protocol invokes share-conversion procedures between different sharing domains. These share-conversion protocols must also be equipped with active-security guarantees. Once the multiplication, opening, and share-conversion subroutines are upgraded, the remaining components of OptSecMiRa inherit active security directly.

As shown in [11, 20], the compiler transforms a protocol that is secure against passive adversaries into one that tolerates active adversaries. To achieve security against up to u

malicious corruptions, the underlying protocol must be secure against at least u^2+u passive corruptions. Therefore, a necessary condition for applying the compiler in our setting is $n \geq u^2+u+1$. Both compilers adopt the virtual-party paradigm. In this approach, each virtual party $\mathbb{P}_i$ is jointly simulated by $u+1$ real parties $P_i, P_{i+1}, \ldots, P_{i+u}$. During input sharing and randomness generation, broadcast functionality ensures that all real parties associated with the same virtual party agree on identical inputs and randomness.

The optimization of Eerikson et al. [20] designates a single party, called the *brain*, to send messages on behalf of each virtual party, while the remaining parties, called *pinkies*, only perform local computations and monitor the execution. Correctness checks are deferred until the end of the protocol through a message-checking functionality. As a result, the communication cost of a virtual message decreases from $O(u^2)$ to $O(u)$, reducing the overall communication complexity from $O(lu^2)$ to $O(lu+u^2)$, where l denotes the number of messages in the underlying protocol. Consequently, active security can be achieved with significantly lower communication overhead.

6 Round and Computational Complexity

6.1 Evaluation Methodology and Benchmark

To the best of our knowledge, the construction by Burkhardt et al. [4] represents the state-of-the-art distributed Miller–Rabin protocol for any number of parties $n \geq 2$. Their key innovation is a novel divisibility test, while our main contribution is the SecZeroTest subprotocol. Therefore, in this section, we only need to compare the round and theoretical computational complexity of these two core components.

6.2 Evaluation Results

Recall that the most expensive component of Burkhardt et al.'s protocol Divisible is the call to the Membership sub-protocol, which securely tests whether a secret-shared value $[x]$ belongs to the set $\{1,\ldots,n\}$ by computing shares of $\prod_{i=1}^{n}(x-i)$. Using a standard binary-tree evaluation strategy, Membership requires $\lceil \log_2 n \rceil$ communication rounds and $n-1$ online secure multiplications. The remaining steps of Divisible incur an additional 3 communication rounds with 4 additional online secure multiplications when the underlying multiplications are implemented for additive shares. In particular, if the underlying multiplications are implemented for Shamir shares, then 3 extra communication rounds are required to convert additive shares into Shamir shares. Consequently, Divisible requires a total of $3+\lceil \log_2 n \rceil$ (resp., $6+\lceil \log_2 n \rceil$) communication rounds and $n+3$ online secure multiplications.

To obtain constant round complexity, Burkhardt et al. also provide a modified version of Membership that uses only 8 rounds under additive secret sharing (ASS), or 11 communication rounds under Shamir secret sharing (SSS), for any $n \geq 2$, at the cost of increasing the online secure multiplications to $5n-1$. In what follows, we refer to the logarithmic-round variant as Div and the constant-round variant as Div′.

Table 4: Theoretical Complexity Comparison

	SecMult	Round (ASS)	Round (SSS)
Div [4]	$n+3$	$3+\lceil \log_2 n \rceil$	$6+\lceil \log_2 n \rceil$
Div′ [4]	$5n-1$	8	11
SecZeroTest	5	7	9

In contrast, our protocol SecZeroTest (**Algorithm 7**) requires only 7 communication rounds when the underlying multiplications are implemented for additive shares, and 9 rounds when they are implemented for Shamir shares, independent of the number of parties n. This bound follows directly from the complexity of the underlying primitives invoked by the protocol, as summarized in Table 3. The theoretical efficiency arises from the fact that SecZeroTest performs only 5 online secure multiplications and 2 opening operations. A detailed side-by-side comparison of the different constructions is provided in Table 4.

7 Experimental Evaluation

In this section, we experimentally validate the theoretical analysis presented in Section 6. Our evaluation proceeds at two levels.

First, we perform a *component-level evaluation* of our main technical contribution, namely the proposed divisibility-testing protocol SecZeroTest, and compare it with the state-of-the-art protocol Divisible proposed by Burkhardt et al. [4]. This component-level evaluation allows us to directly quantify the performance gains introduced by our new Montgomery-based construction. The results in the semi-honest setting are reported in Figures 1 and 2.

Second, we conduct a *system-level evaluation* to investigate how the performance improvements of SecZeroTest translate to end-to-end applications. Specifically, we benchmark a complete iteration of the secure Miller–Rabin primality test, instantiated either with our SecZeroTest protocol or Burkhardt et al.'s Divisible protocol as the underlying divisibility-testing subroutine. This system-level evaluation demonstrates the practical impact of our optimization in the context of distributed RSA modulus generation. The results in the semi-honest setting are shown in Figures 3 and 4, while the corresponding malicious-secure results are presented in Figures 5 and 6.

For the first evaluation, we measure the total time required to test $\delta \bmod f = 0$. Following the description in **Algorithm 7**, we decompose the total runtime into the following

three components:

- **Computational cost:** the total local computation time incurred by all parties, dominated by secure multiplication operations.
- **Communication cost:** the time spent on protocol communication, including both data transmission and network latency.
- **Pure time cost:** the execution time of the protocol logic excluding secure multiplications and communication. This metric is obtained by replacing the multiplication and networking modules with dummy implementations.

This decomposition enables us to identify the main performance bottlenecks and to quantify the contribution of each component to the overall runtime. It is worth noting that our experiments show that, across all parameter settings, the pure time overhead of our new protocol (consistently below 1ms) is indeed lower than that of Burkhardt et al.'s Divisible protocol (below 4ms). However, this component constitutes only a negligible fraction of each protocol's total execution time. Therefore, we omit its detailed discussion in the subsequent analysis. Moreover, our experiments deliberately exclude the setup phase of both protocols—i.e., the time required to load all necessary classes and initialize parameters. In our benchmarks, the auxiliary prime P and Q (required by Burkhardt et al.'s protocol) as well as the parameters R and M (required by our protocol) can be precomputed once the bit-length κ, the number of parties n, and the statistical security parameter σ are fixed. Hence, it is reasonable to omit the time for sampling these parameters. Once the parameters are initialized, an unlimited number of tests can be executed as long as the input shares are sampled independently of the parameters.

Code Design. To ensure a fair benchmarking comparison, our implementation builds upon the modular source code (available at https://github.com/jot2re/rsa) publicly released by Burkhardt et al. [4]. The modular architecture of their codebase allowed us to seamlessly integrate our implementation of the SecProTrunc protocol and subsequently construct the novel SecZeroTest protocol by reusing their existing subroutines—such as those for secure multiplication (SecMult) and share conversions. This approach guarantees that the underlying cryptographic primitives are implemented consistently across both constructions, thereby establishing a fair and reliable basis for performance comparison. Furthermore, we extended their Shamir Engine Class to support settings with more than 15 parties ($n > 15$).

Experiment Environment and Setup Network latency is modeled using a gigabit link with a fixed 10 ms delay. All experiments are executed on a physical workstation equipped with an 11th-gen Intel(R) Core(TM) i7-11700K @ 3.60GHz CPU equipped with 62.5 GB of RAM, running Ubuntu 22.04.5 LTS with OpenJDK 21.0.9. To accelerate large-integer arithmetic, we wrap the GNU Multiple Precision Arithmetic Library (GMP) via the Java Native Interface (JNI) [13] and used it for all computationally intensive operations in the benchmarks.

Table 5: The number of secure multiplications needed for different n

	SecZeroTest	Div [4]	Div′ [4]
$n = 3$	5	6	14
$n = 5$	5	8	24
$n = 7$	5	10	34
$n = 9$	5	12	44
$n = 11$	5	14	54
$n = 13$	5	16	64
$n = 15$	5	18	74
$n = 17$	5	20	84
$n = 19$	5	22	94
$n = 21$	5	24	104
$n = 23$	5	26	114

Secure Multiplications We primarily measure the computational cost and communication overhead associated with secure multiplications during protocol execution. Due to the different secret share modulus sizes required by the correctness and security of each protocol, our evaluation proceeds in two steps: first, we benchmark the per-multiplication computational cost and communication overhead for each protocol; second, we count the total number of secure multiplications each protocol invokes. The overall cost is then obtained by multiplying the per-operation cost by the respective count. To ensure statistical reliability, each per-multiplication benchmark is obtained by running 100 independent multiplications.

All secure multiplications are implemented using Shamir secret sharing. While Shamir secret sharing provides security only in the honest-majority setting, Burkhardt et al. [4] demonstrate that it yields more efficient multiplications than replicated secret sharing or OT-based approaches.

To count the number of secure multiplications actually invoked by each protocol, we rely on an instrumentation counter included in the code of Burkhardt et al. that tracks online secure multiplications. This allows a direct validation of the theoretical complexity analysis presented in Section 6. The results, shown in Table 5, closely match the theoretical predictions, confirming the accuracy of our analysis.

Experimental Result and Analysis For the first *component-level evaluation*, we first benchmark the computational time of our protocol SecZeroTest and Burkhardt et al.'s protocol Divisible. The reported time includes all required online secure multiplications and is measured for various RSA modulus lengths and numbers of parties. The results, presented in **Figure 1a**, show that the computational cost of our new construction begins to demonstrate an advantage when $n = 9$.

This advantage becomes more pronounced as n increases. It is noteworthy that in our experiments, our construction achieves perfect correctness when deciding $\delta \bmod f = 0$ using a secret-sharing modulus M with $\log M = 3(\kappa + \lceil \log_2 n \rceil) + \sigma$. In contrast, Burkhardt et al.'s construction employs a secret-sharing modulus Q with $\log Q = 2\kappa + \lceil \log_2 n \rceil$ to achieve negligible correctness error. To attain perfect correctness, the construction of Burkhardt et al. requires a secret-sharing modulus Q' with $\log Q' = 4\kappa + 2\lceil \log_2 n \rceil$. Under this common requirement for perfect correctness, our protocol achieves an even more significant efficiency advantage.

Figure 1b compares the communication overhead of the two protocols under different parameter settings. The results show that the communication costs of our protocol SecZeroTest and Burkhardt et al.'s Divisible are nearly identical when $n = 7$. However, our protocol becomes more communication-efficient once $n \geq 9$. Moreover, the communication overhead of SecZeroTest grows only gradually as the number of parties increases, resulting in relatively stable performance across different values of n. In contrast, the protocol of Burkhardt et al. exhibits noticeable jumps in communication cost at certain thresholds, most prominently when n increases from 7 to 9 and from 15 to 17. This behavior is a direct consequence of the difference in round complexity between the two constructions. Since the experiments are instantiated using Shamir-sharing-based secure multiplications, our protocol always requires only 9 communication rounds regardless of n, whereas Burkhardt et al.'s protocol requires $6 + \lceil \log_2 n \rceil$ rounds. Consequently, whenever $\lceil \log_2 n \rceil$ increases, the round complexity—and hence the communication overhead—of their protocol increases discretely. Such transitions occur, for example, at $n = 9$ (where $\lceil \log 9 \rceil = 4$) and $n = 17$ (where $\lceil \log 17 \rceil = 5$), which explains the abrupt increases observed in the figure. Since $6 + \lceil \log_2 n \rceil > 9$ for all $n \geq 9$, our protocol achieves both a lower and a more predictable round complexity. As a result, it consistently provides better communication efficiency, with the advantage becoming increasingly pronounced as the number of participating parties grows.

The total runtime comparison, shown in **Figure 1c**, indicates that SecZeroTest consistently outperforms Divisible, the runtime of Divisible is 1.01×-1.77× that of SecZeroTest when n increases from 7 to 23. This efficiency gain exhibits a clear positive correlation with the number of parties n.

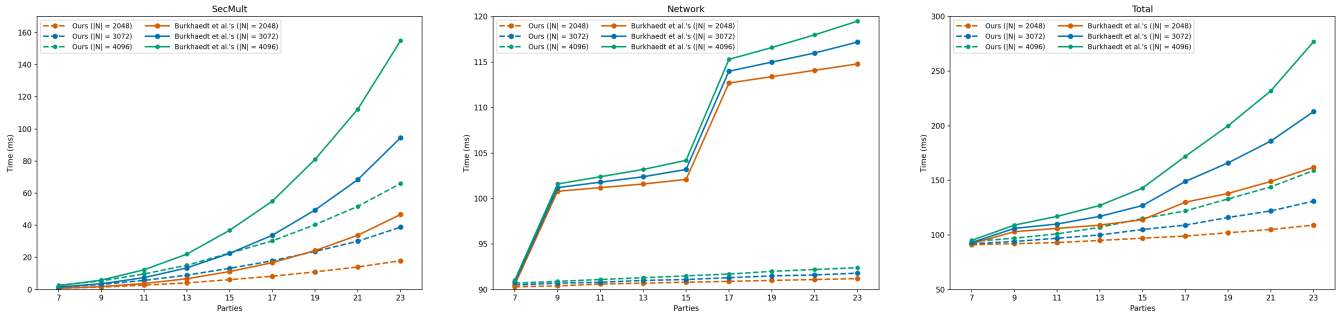
To further illustrate the performance advantage of our SecZeroTest protocol in large-scale distributed settings, we present additional experimental results across varying numbers of participants n. Figures 2a, 2b, and 2c compare the computational overhead, communication overhead, and total execution time, respectively. The results reveal two main findings: (1) for large n, the communication overhead of secure multiplications dominates the total runtime; (2) as n increases from 25 to 105, the total runtime of the Divisible protocol by Burkhardt et al. becomes approximately 2× to 10× longer than that of our SecZeroTest protocol. This demonstrates that the efficiency gain of our scheme grows significantly with the number of participants.

For the second system-level evaluation, we compare the performance of a complete secure Miller–Rabin iteration instantiated with either our SecZeroTest-based implementation, denoted OptSecMiRa, or Burkhardt et al.'s Divisible-based implementation. The experimental results in the semi-honest setting are reported in Figures 3 and 4. Both figures exhibit a consistent trend. When the number of parties is small ($n = 7$), the two implementations achieve comparable performance. However, once $n \geq 9$, our protocol OptSecMiRa becomes consistently faster, and the performance advantage grows steadily as the number of parties increases. More specifically, for $n = 7$–23, the total runtime of the Miller–Rabin test based on Burkhardt et al.'s Divisible protocol is 1.08×–1.30× that of OptSecMiRa. For larger settings with $n = 25$–105, this ratio further increases to 1.38×–6.30×.

Compared with the component-level evaluation of the divisibility-testing subroutine, the observed speedups are somewhat smaller. This is expected because a complete Miller–Rabin iteration contains several additional operations that are identical in both implementations, including modular exponentiation, share-conversion procedures, and other auxiliary computations. These common costs contribute equally to both protocols and therefore dilute the relative impact of optimizing the divisibility-testing component. As a simple analogy, replacing a cost component from 2 to 1 yields a 2× speedup, whereas replacing the same component in a larger computation changes the total cost from $2 + 1$ to $1 + 1$, resulting in a smaller speedup of $3/2$. Nevertheless, the results clearly demonstrate that the efficiency gains achieved by SecZeroTest translate into substantial end-to-end improvements for secure Miller–Rabin testing and become increasingly significant as the number of participating parties grows.

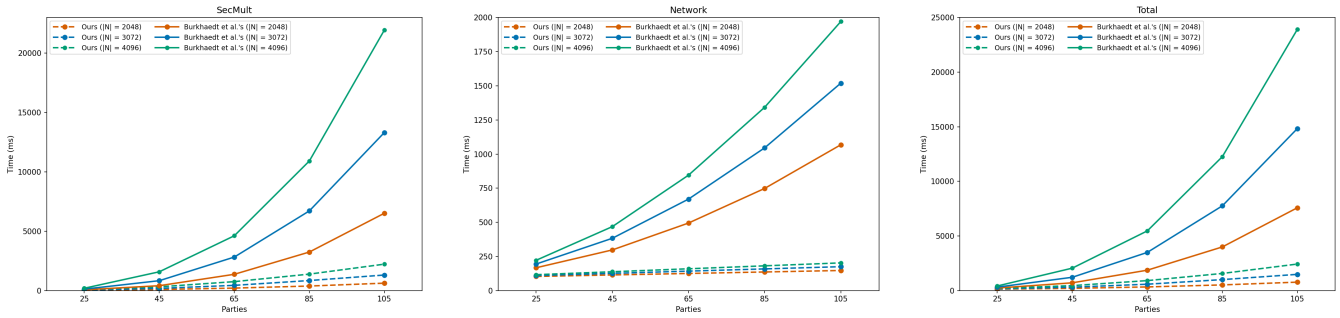
Malicious Setting Since our implementation is built upon the framework of Burkhardt et al. [4], which serves as our primary baseline, we follow their evaluation methodology for the malicious setting. Specifically, because their maliciously secure implementation is intended as a proof-of-concept rather than a fully optimized realization, we likewise do not implement the complete Miller–Rabin protocol under malicious security. Instead, we implement the necessary operations (e.g. underlying secure multiplication) and communication required to transform the semi-honest protocol into a maliciously secure one. Our implementation is based on the compiler of Damgård et al. [11], incorporating several communication optimizations proposed by Eerikson et al. [20]. We intentionally omit JNI optimizations to ensure a fair comparison with the baseline implementation. The results for small-scale settings ($7 \leq n \leq 23$) are presented in Figure 5, while the results for larger-scale settings ($n \in \{25, 45, 65, 85\}$) are shown in Figure 6.

It is well known that secure multiplication dominates the cost of actively secure MPC protocols. In Burkhardt et al.'s



a. Computational cost. b. Communication cost. c. Total runtime.

Figure 1: Performance comparison between our SecZeroTest protocol and Burkhardt et al.'s Divisible protocol in the semi-honest setting, for RSA modulus lengths $|N| = \lfloor \log_2 N \rfloor + 1 \in \{2048, 3072, 4096\}$ and $7 \leq n \leq 23$. (a) Computational cost, (b) communication cost, and (c) total runtime, over a 1Gbps network with 10ms latency.



a. Computational cost. b. Communication cost. c. Total runtime.

Figure 2: Performance comparison between our SecZeroTest protocol and Burkhardt et al.'s Divisible protocol in the semi-honest setting, for RSA modulus lengths $|N| = \lfloor \log_2 N \rfloor + 1 \in \{2048, 3072, 4096\}$ and $n \in \{25, 45, 65, 85, 105\}$. (a) Computational cost, (b) Communication cost, and (c) total runtime, over a 1Gbps network with 10ms latency.

construction, the number of secure multiplications grows linearly with the number of parties n, whereas our protocol requires only a constant number of multiplications independent of n. Consequently, as shown in Figures 5 and 6, a single Miller–Rabin iteration instantiated with our SecZeroTest protocol already outperforms the Divisible-based implementation when $n = 7$, and the advantage becomes increasingly pronounced as n grows. More specifically, across all tested configurations, the total runtime of the Divisible-based implementation is approximately $1.38\times$–$12.40\times$ that of the SecZeroTest-based implementation.

We also compare the overhead introduced by malicious security. As n increases from 7 to 85, the runtime of our maliciously secure implementation is approximately $3.4\times$–$13.7\times$ that of its semi-honest counterpart. In contrast, the corresponding overhead for Burkhardt et al.'s implementation ranges from $4.8\times$ to $24.1\times$. These results indicate that the benefits of our constant-multiplication design become even more significant in the malicious setting, where secure multiplication constitutes the dominant performance bottleneck.

8 Conclusion

This work revisits the problem of securely generating an RSA modulus in a distributed setting. We focus on a core subroutine required in distributed Miller–Rabin primality testing: modular reduction with a secretly shared modulus. Inspired by the Montgomery reduction algorithm, we propose a novel distributed zero-test protocol, SecZeroTest, which follows a fundamentally different approach from the state-of-the-art divisibility test of Burkhardt et al. [4].

Leveraging this improved core component, we obtain what is, to the best of our knowledge, the most efficient distributed Miller–Rabin primality test currently available. We provide a rigorous security proof for the protocol and a detailed analysis of its communication and computational complexity. Experimental benchmarks confirm our theoretical analysis, showing that the performance advantage of our construction increases with the number of participating parties.

Furthermore, the distributed modular reduction technique we introduce is of independent interest as a secure multiparty computation primitive. It may find application in other cryp-

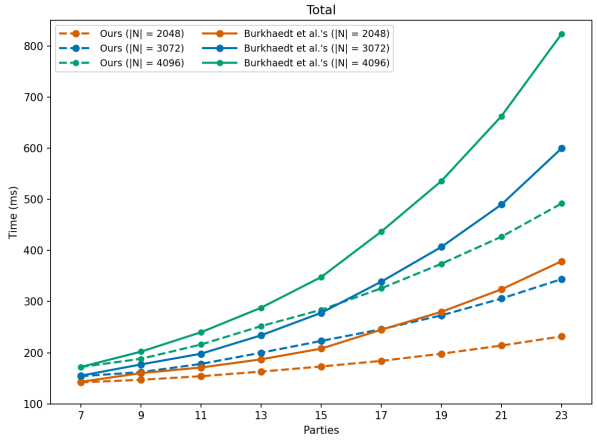


Figure 3: Performance comparison of one complete secure Miller–Rabin test iteration using our SecZeroTest-based implementation OptSecMiRa and Burkhardt et al.'s Divisible-based implementation in the semi-honest setting, for RSA modulus lengths $|N| = \lfloor \log_2 N \rfloor + 1 \in \{2048, 3072, 4096\}$ and $7 \leq n \leq 23$ (total runtime over a 1Gbps network with 10ms latency).

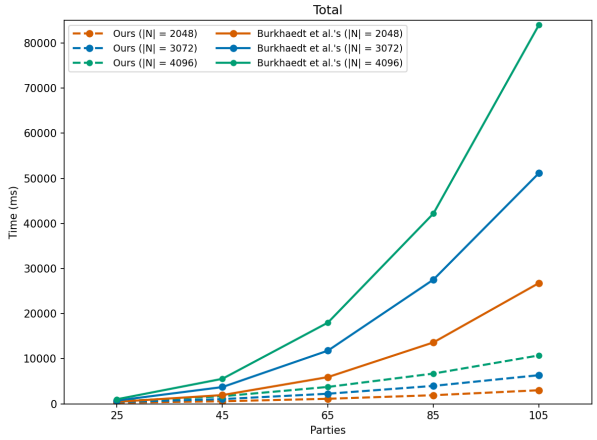


Figure 4: Performance comparison of one complete secure Miller–Rabin test iterationusing our SecZeroTest-based implementation OptSecMiRa and Burkhardt et al.'s Divisible-based implementation in the semi-honest setting, for RSA modulus lengths $|N| = \lfloor \log_2 N \rfloor + 1 \in \{2048, 3072, 4096\}$ and $n \in \{25, 45, 65, 85, 105\}$ (total runtime over a 1Gbps network with 10ms latency).

tographic protocols that require efficient modular arithmetic with a secret, shared modulus.

Acknowledgments

This research is supported by Natural Science Foundation of Shandong Province (No. ZR2025MS1029), National Natural Science Foundation of China (No.12271306), the State Key Laboratory of Cryptography and Digital Economy Security, Shandong University (No.KFYB2505), National Key Research and Development Program of China (Nos. 2024YFB4504700,2022YFB2702804).

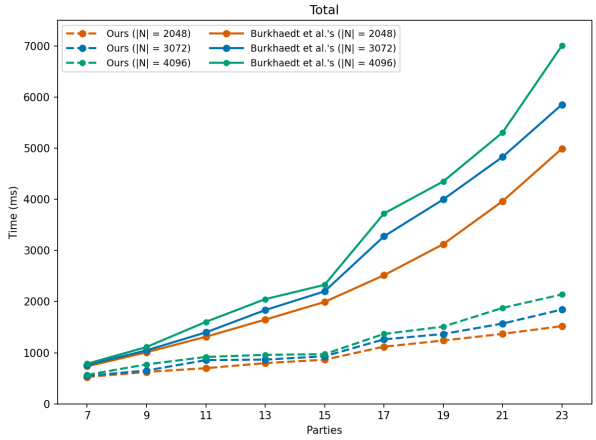


Figure 5: Performance comparison of one complete secure Miller–Rabin iteration using our SecZeroTest-based implementation OptSecMiRa and Burkhardt et al.'s Divisible-based implementation in the malicious setting, for RSA modulus lengths $|N| = \lfloor \log_2 N \rfloor + 1 \in \{2048, 3072, 4096\}$ and $7 \leq n \leq 23$, over a 1Gbps network with 10ms latency.

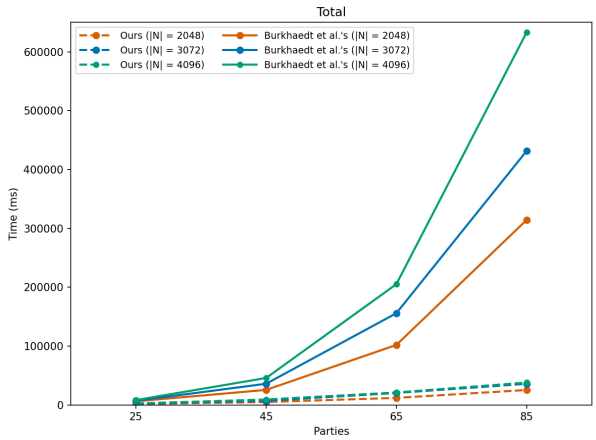


Figure 6: Performance comparison of one complete secure Miller–Rabin iteration using our SecZeroTest-based implementation OptSecMiRa and Burkhardt et al.'s Divisible-based implementation in the malicious setting, for RSA modulus lengths $|N| = \lfloor \log_2 N \rfloor + 1 \in \{2048, 3072, 4096\}$ and $n \in \{25, 45, 65, 85\}$, over a 1Gbps network with 10ms latency.

Ethical Considerations

We follow the ethical guidelines of USENIX Security and reflect on both the research process and the broader implications of this work. Our research involves no human subjects, no physical experimentation, and no collection or analysis of private or sensitive data. All experiments are conducted using publicly available datasets, standard cryptographic benchmarks, and open-source implementations. Therefore, our experimental methodology does not introduce privacy risks or risks to vulnerable populations.

Authorship reflects substantial intellectual contribution to the conception, design, analysis, implementation, and writing of this work. All authors participated meaningfully in the research process, and we acknowledge and respect the contributions of all collaborators. Our work also complies with applicable academic norms and responsible research practices in cryptography and computer security.

This work studies more efficient distributed RSA modulus generation, which is a fundamental building block for threshold RSA systems, hidden-order groups, RSA accumulators, and verifiable delay functions (VDFs). The primary stakeholders include researchers and practitioners developing threshold cryptography systems, operators of distributed trust ceremonies, blockchain and consensus systems relying on VDFs or accumulators, and end users who benefit from stronger decentralization and more distributed trust assumptions. Secondary stakeholders include the broader security community and society at large, which may benefit from more trustworthy and transparent cryptographic infrastructures.

The primary positive impact of this work is improved scalability and efficiency for distributed RSA modulus generation, particularly in settings involving many participants. More efficient protocols may help make large-scale trust ceremonies practical, reduce reliance on a small number of trusted parties, and improve the deployability of threshold RSA systems and hidden-order cryptographic infrastructure. These benefits are especially relevant in applications where decentralization, robustness, and public verifiability are important. For example, reducing the cost of distributed setup ceremonies may enable systems to involve hundreds or thousands of participants rather than relying on a small committee, thereby improving resilience against compromise and concentration of trust.

At the same time, we recognize that cryptographic research is often dual-use. More efficient distributed RSA modulus generation could lower the deployment barrier for hidden-order cryptographic systems at larger scale, including systems that users may not fully understand or independently audit. Similarly, more practical threshold RSA systems could be deployed in sensitive infrastructure where implementation errors, operational misuse, or insufficient governance mechanisms may have significant consequences. In addition, the same techniques that strengthen privacy, decentralization, and robustness for legitimate users could also be adopted by malicious actors to make certain infrastructures more difficult to monitor or disrupt.

However, we believe these risks are limited and outweighed by the benefits of publication. Our work does not weaken existing security assumptions, bypass safeguards, or introduce fundamentally new cryptographic capabilities. Instead, it improves the efficiency and scalability of well-studied cryptographic primitives that are already widely known and actively researched. Moreover, our paper explicitly discusses the distinction between semi-honest and malicious security models, including the additional costs required for secure deployment in adversarial real-world environments.

Our decision to publish this work was based on the belief that improving the scalability and practicality of distributed trust mechanisms is valuable for the broader security community. We believe the positive impact of enabling more decentralized, transparent, and auditable cryptographic systems outweighs the potential risks associated with broader deployment. In future work, we also plan to further consider how large-scale distributed cryptographic protocols interact with governance, accountability, transparency, and operational oversight in real-world trusted setup ceremonies and decentralized infrastructures.

Open Science

We fully embrace the principles of Open Science. Our research artifacts have been made available in a publicly accessible repository at https://doi.org/10.5281/zenodo.20237927. These artifacts have been prepared for review and will remain openly accessible alongside the final version of the paper, ensuring transparency and reproducibility of our work. By adhering to Open Science practices, we aim to promote the broad dissemination of scientific knowledge and facilitate further research in our field. Our approach aligns with the goal of enhancing the reproducibility and reliability of scientific findings, thereby contributing to a more open and collaborative research environment.

References

[1] Joy Algesheimer, Jan Camenisch, and Victor Shoup. Efficient computation modulo a shared secret with application to the generation of shared safe-prime products. In *Annual International Cryptology Conference*, pages 417–432. Springer, 2002.

[2] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In *Annual international cryptology conference*, pages 757–788. Springer, 2018.

[3] Dan Boneh and Matthew Franklin. Efficient generation of shared RSA keys. In *Advances in Cryptology—CRYPTO '97*, pages 425–439. Springer, 1997.

[4] Jakob Burkhardt, Ivan Damgård, Tore Kasper Frederiksen, Satrajit Ghosh, and Claudio Orlandi. Improved distributed RSA key generation using the Miller-Rabin test. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 2501–2515, 2023.

[5] Octavian Catrina and Sebastiaan de Hoogh. Improved primitives for secure multiparty integer computation. In *International Conference on Security and Cryptography for Networks*, 2010. URL: https://api.semanticscholar.org/CorpusID:7104878.

[6] Megan Chen, Jack Doerner, Yashvanth Kondi, Eysa Lee, Schuyler Rosefield, Abhi Shelat, and Ran Cohen. Multiparty generation of an RSA modulus. *Journal of Cryptology*, 35(2):12, 2022.

[7] Megan Chen, Carmit Hazay, Yuval Ishai, Yuriy Kashnikov, Daniele Micciancio, Tarik Riviere, Abhi Shelat, Muthu Venkitasubramaniam, and Ruihan Wang. Diogenes: Lightweight scalable RSA modulus generation with a dishonest majority. In *2021 IEEE Symposium on Security and Privacy*, SP '21, pages 590–607. IEEE, 2021.

[8] ChihYun Chuang, IHung Hsu, and TingFang Lee. Beyond the 1/2 bound: On the theory and practice of biprimality tests. In Shi Bai and Edoardo Persichetti, editors, *Public-Key Cryptography – PKC 2026*, pages 369–400, Cham, 2026. Springer Nature Switzerland.

[9] Ivan Damgård, Peter Landrock, and Carl Pomerance. Average case error estimates for the strong probable prime test. *Mathematics of computation*, 61(203):177–194, 1993.

[10] Ivan Damgård and Gert Læssøe Mikkelsen. Efficient, robust and constant-round distributed RSA key generation. In *Theory of Cryptography Conference*, pages 183–200. Springer, 2010.

[11] Ivan Damgård, Claudio Orlandi, and Mark Simkin. Yet another compiler for active security or: Efficient mpc over arbitrary rings. In *Annual International Cryptology Conference*, pages 799–829. Springer, 2018.

[12] Cyprien Delpech de Saint Guilhem, Eleftheria Makri, Dragos Rotaru, and Titouan Tanguy. The return of eratosthenes: Secure generation of RSA moduli using distributed sieving. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 594–609, 2021.

[13] Didier Deshommes. GMP-java. https://github.com/dfdeshom/GMP-java, 2023. Accessed: 2023-07-27.

[14] Yair Frankel, Philip D. MacKenzie, and Moti Yung. Robust efficient distributed RSA-key generation. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 663–672, New York, NY, 1998. ACM.

[15] Tore Kasper Frederiksen, Yehuda Lindell, Valery Osheter, and Benny Pinkas. Fast distributed RSA key generation for semi-honest and malicious adversaries. In *Annual International Cryptology Conference*, pages 331–361. Springer, 2018.

[16] Niv Gilboa. Two party RSA key generation. In *Annual International Cryptology Conference*, pages 116–129. Springer, 1999.

[17] Carmit Hazay, Gert Læssøe Mikkelsen, Tal Rabin, and Tomas Toft. Efficient rsa key generation and threshold paillier in the two-party setting. In *Cryptographers' Track at the RSA Conference*, pages 313–331. Springer, 2012.

[18] Carmit Hazay, Gert Læssøe Mikkelsen, Tal Rabin, and Tomas Toft. Efficient RSA key generation and threshold paillier in the two-party setting. In *Cryptographers' Track at the RSA Conference*, pages 313–331. Springer, 2012.

[19] Carmit Hazay, Gert Læssøe Mikkelsen, Tal Rabin, Tomas Toft, and Angelo Agatino Nicolosi. Efficient rsa key generation and threshold paillier in the two-party setting. *Journal of Cryptology*, 32(2):265–323, 2019.

[20] Eerikson Hendrik, Keller Marcel, Orlandi Claudio, Pullonen Pille, Puura Joonas, and Simkin Mark. Use your brain! arithmetic 3pc for any modulus with active security. In *1st Conference on Information-Theoretic Cryptography*, 2020.

[21] Michael Malkin, Thomas D. Wu, and Dan Boneh. Experimenting with shared generation of RSA keys. In *Proceedings of the Network and Distributed System Security Symposium*, NDSS '99, 1999.

[22] Gary L Miller. Riemann's hypothesis and tests for primality. *Journal of Computer and System Sciences*, 13(3):300–317, 1976.

[23] Peter L. Montgomery. Modular multiplication without trial division. *Math. Comp.*, 44(170):519–521, 1985.

[24] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *International conference on the theory and applications of cryptographic techniques*, pages 223–238. Springer, 1999.

[25] Krzysztof Pietrzak. Simple verifiable delay functions. In *10th innovations in theoretical computer science conference (itcs 2019)*, pages 60–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2019.

[26] Guillaume Poupard and Jacques Stern. Generation of shared RSA keys by two parties. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 11–24. Springer, 1998.

[27] Michael O Rabin. Probabilistic algorithm for testing primality. *Journal of number theory*, 12(1):128–138, 1980.

[28] Benjamin Wesolowski. Efficient verifiable delay functions. *Journal of Cryptology*, 33(4):2113–2147, 2020.

[29] Guangwu Xu, Yiran Jia, and Yanze Yang. Chinese remainder theorem approach to montgomery-type algorithms, 2025. URL: https://arxiv.org/abs/2402.00675, arXiv:2402.00675.