mathematics

Article

On Efficient Parallel Secure Outsourcing of Modular Exponentiation to Cloud for IoT Applications

Satyabrat Rath [1], Jothi Ramalingam [1] and Cheng-Chi Lee [2,3,*]

1 Department of Mathematical and Computational Sciences, NITK Surathkal, Mangaluru 575025, Karnataka, India; satyarath.207ma006@nitk.edu.in (S.R.); jothiram@nitk.edu.in (J.R.)
2 Department of Library and Information Science, Fu Jen Catholic University, New Taipei City 24205, Taiwan
3 Department of Computer Science and Information Engineering, Fintech and Blockchain Research Center, Asia University, Taichung City 41354, Taiwan
* Correspondence: cclee@blue.lins.fju.edu.tw or cclee@mail.fju.edu.tw

Abstract: Modular exponentiation is crucial for secure data exchange in cryptography, especially for resource-constrained Internet of Things (IoT) devices. These devices often rely on third-party servers to handle computationally intensive tasks like modular exponentiation. However, existing outsourcing solutions for the RSA algorithm may have security vulnerabilities. This work identifies a critical flaw in a recent outsourcing protocol for RSA proposed by Hu et al. We demonstrate how this flaw compromises the security of the entire RSA system. Subsequently, we propose a robust solution that strengthens the RSA algorithm and mitigates the identified vulnerability. Furthermore, our solution remains resilient against existing lattice-based attacks. The proposed fix offers a more secure and efficient way for IoT devices to leverage the power of third-party servers while maintaining data integrity and confidentiality. An extensive performance evaluation confirms that our solution offers comparable efficiency while significantly enhancing security compared to existing approaches.

Keywords: modular exponentiation; outsourcing; RSA algorithm; secure message communication; cloud computing

MSC: 11T71; 11T06

Citation: Rath, S.; Ramalingam, J.; Lee, C.-C. On Efficient Parallel Secure Outsourcing of Modular Exponentiation to Cloud for IoT Applications. *Mathematics* **2024**, *12*, 713. https://doi.org/10.3390/math12050713

Academic Editor: Daniel-Ioan Curiac

Received: 19 January 2024
Revised: 23 February 2024
Accepted: 25 February 2024
Published: 28 February 2024

1. Introduction

Within the rapidly evolving Internet of Things (IoT) ecosystem, where interconnected devices generate and exchange sensitive data, ensuring robust security and privacy is paramount. Modular exponentiation, a fundamental operation underpinning public-key cryptosystems like RSA, plays a pivotal role in safeguarding data during transmission and storage.

The Rivest–Shamir–Adleman (RSA) algorithm, a cornerstone of public-key cryptography, leverages the computational difficulty of factoring large prime numbers. This intrinsic complexity forms the basis for its robust security, enabling secure key exchange and encrypted communication across diverse applications.

However, within the realm of IoT devices, limited processing power and resource constraints pose a significant challenge. Efficiently executing computationally intensive operations like modular exponentiation, crucial for utilizing RSA's full potential, often proves difficult for these resource-constrained devices.

Cloud computing emerges as a viable solution, offering on-demand access to vast computational resources through a pay-per-use model. By outsourcing computationally demanding tasks like modular exponentiation to a third-party cloud server, even resource-scarce IoT devices can leverage the power of RSA for robust data security. This cost-effective approach ensures their uninterrupted participation in secure communication protocols without compromising performance or energy efficiency.

However, outsourcing sensitive data to a cloud service introduces a new layer of security considerations. Maintaining data integrity and confidentiality while utilizing cloud resources necessitates the implementation of robust security measures, such as a "secure outsourcing" of computation.

By effectively navigating the interactions between IoT devices and the cloud, we can leverage the combined strengths of both to achieve a future where secure and efficient communication becomes the cornerstone of a truly connected world.

1.1. Related Works

The exploration of outsourcing computationally expensive tasks has been a primary focus for researchers over an extended period. Broadly, research on outsourcing computation is progressing along two distinct trajectories.

One outsourcing model, proposed by [1,2], aims to evaluate any computational function and resolve all outsourcing computations concurrently. However, this model relies on complex cryptographic tools, rendering it time-consuming and costly.

In contrast, another model targets specific outsourcing tasks. For instance, refs. [3,4] proposed task-specific outsourcing for polynomial evaluation, while refs. [5–7] focused on modular exponentiation. This model concentrates on particular computational tasks.

Modular exponentiation has consistently been at the forefront of cryptographic tools, primarily utilized for ensuring secure communication, key exchange protocols, message authentication, and more. Within cryptographic algorithms based on the discrete logarithm, modular exponentiation emerges as a complex and widely employed operation. Numerous studies have focused on securely outsourcing modular exponentiation. Hohenberger and Lysyanskaya [5] introduced the pioneering solution under the assumption of one malicious entity in a two-server model. Chen et al. [6] proposed a more efficient algorithm under the same assumption, including an initial algorithm targeting the outsourcing of simultaneous modular exponentiations. Ding et al. [8] proposed an outsourcing algorithm aiming to achieve a high level of checkability of the cloud's output. Zhou et al. [9] presented a secure outsourcing algorithm for exponentiation under a single untrusted program model, incorporating a secure verification scheme with elevated checkability. However, Rangasamy and Kuppusamy [10] identified a vulnerability in the ExpSOS algorithm, where its security could be compromised if two modular exponentiations with the same exponent were delegated to the server. Ren et al. [11] proposed two algorithms for outsourcing modular exponentiation capable of detecting malicious behaviors of servers. Subsequently, Su et al. [12] introduced two new algorithms for outsourcing single as well as composite modular exponentiation using a single untrusted cloud. Nevertheless, Bouillaguet et al. [13] conducted a cryptanalysis on all existing outsourcing schemes (including Su et al. [12]) and proposed lattice-based attacks to breach security in all the existing schemes of outsourcing modular exponentiation.

Additionally, research has explored outsourcing applications of modular exponentiation. For instance, Zhang et al. [14] devised a secure outsourcing algorithm for RSA decryption, involving modular exponentiation. Furthermore, Hu et al. [15] proposed a modification to the cloud computing phase of the ExpSOS algorithm [9], employing Parallel Binary Modular Exponentiation (PBME) for cloud-side computation. Hu et al. [15] also applied their algorithm to outsource the computation of modular exponentiation necessary for secure message communication between two users, U_1 and U_2, using the RSA algorithm.

1.2. Our Contributions

In this work, we first take a critical view of the most well-known outsourcing protocol, Hu et al. [15] outsourcing protocol for the RSA algorithm, and then address the security issues related to their outsourcing modular exponentiation algorithm. The work can be summarized as follows:

- **Overview:** Provides a comprehensive overview of Hu et al.'s outsourcing protocol for the RSA algorithm [15].

- **Flaws and Deficiencies:** Identify the flaws and security deficiencies in Hu et al.'s protocol.
- **Proposed Solution:** Introduces a novel solution to rectify these deficiencies, supported by an in-depth security and performance analysis.
- **Verification in Malicious Cloud:** Incorporates the omitted verification steps in Hu et al's protocol, considering a fully malicious cloud environment [16].

We have assumed a security and system model similar to Hu et al. ([15], Section II). These definitions are widely used and were proposed by Hohenberger and Lysyanskaya [5].

2. Description of Hu et al. [15] Protocol

To start with, we investigate Hu et al. 's outsourcing algorithm, which involves a resource-limited user U delegating the task of computing modular exponentiation

$$b^e \mod N$$

to a trustworthy yet curious cloud C, where b is the base and the exponent e and the modulus $N = pq$ with p and q being two distinct large primes.

The cloud computing approach suggested by Hu et al. [15] relies on the Binary Modular Exponentiation (BME) algorithm, which represents the exponent e in binary form. Algorithm 1 contains the pseudocode for the BME algorithm. In Algorithm 1, the base is taken as B_{base}, the exponent $E_{exp} = (e_{n-1}e_{n-2}\cdots e_1e_0)_2$ is considered in binary form (i.e., $e_i = 0$ or 1), and the modulus is chosen as N. Algorithm 1, proposed by Hu et al., provides the result $R_{result} = B_{base}^{E_{exp}} \mod N$ for the inputs B_{base}, E_{exp} and modulus N.

Algorithm 1 The algorithm BME [15]

Input: A base B_{base}, the exponent $E_{exp} = (e_{n-1}e_{n-2}\cdots e_1e_0)_2$ in binary representation, the modulus N
Output: $R_{result} = B_{base}^{E_{exp}} \mod N$

```
1.  $R_{result} \leftarrow 1$                                    // Initialize result to 1
2.  for $i = n-1$ to 0 do              // Loop through each bit of the exponent (reverse order)
    $R_{result} \leftarrow R_{result}^2 \mod N$                  // Square the result and take modulo $N$
    if $e_i = 1$ then
    $R_{result} \leftarrow R_{result} \cdot B_{base} \mod N$   //Multiply by base $B_{base}$ if $e_i$ is 1, and take modulo $N$
    end
3.  return $R_{result}$                                   // Return the final result
```

Subsequently, Hu et al. [15] provided a parallel computing algorithm for cloud-side computation. This algorithm, Parallel Binary Modular Exponentiation (PBME), described in Algorithm 2, is a parallel computing algorithm extension of Algorithm 1 with each component of the output $X_{result} = (x_0, x_1, \cdots, x_{n-1})$ computed parallelly.

As follows from Algorithm 2, for the exponent $E_{exp} = (e_{n-1}e_{n-2}\cdots e_1e_0)_2$, if $e_i = 1$, then x_i takes the value B_{base} and squares it i times, or else if $e_i = 0$, then x_i takes the value 1. Here, x_i's are the results of each parallel computation and collectively

$$\prod_{i=0}^{n-1} x_i = \prod_{i \text{ s.t. } e_i=1} B_{base}^{2i} = B_{base}^{E_{exp}} \mod N.$$

Hu et al. introduced Algorithm 2 for achieving secure parallel outsourcing of modular exponentiation. This algorithm allows a user U to delegate the task of computing modular exponentiation of the base b, exponent e, and modulus N to an honest-but-curious cloud C to compute the result $R = b^e \mod N$. The secure outsourcing of modular exponentiation is illustrated in Algorithm 3.

Algorithm 2 The algorithm PBME [15]

Input: A base B_{base}, the binary representation of the exponent $E_{exp} = (e_{n-1}e_{n-2}\cdots e_1e_0)_2$, the modulus N

Output: $X_{result} = (x_0, x_1, \cdots, x_{n-1})$

```
1.  for $i = 0$ to $n-1$ do                      // Loop through each bit of the exponent
2.  if $e_i \neq 0$ then                         // Check if the current bit is non-zero
3.  $x_i \leftarrow B_{base}$;                   // Initialize $x_i$ with base $B_{base}$
4.  for $j = 1$ to $i$ do                        // Square $x_i$, $i$ times
5.  $x_i \leftarrow x_i^2 \mod N$;               // Square $x_i$ and take modulo $N$
6.  end for                                      // End of inner loop
7.  else                                         // If the current bit is zero
8.  $x_i \leftarrow 1$;                          // Set $x_i$ to 1
9.  end if                                       // End of if statement
10. end for                                      // End of outer loop
11. return $X_{result}$                          // Return the resulting array $X_{result}$
```

To provide an overview, Algorithm 3 entails the following steps:

- User U generates a secret key to conceal the modulus N.
- Encrypting the base b and exponent e as B and E, respectively.
- Outsourcing the computation of modular exponentiation to the cloud C for the encrypted system.
- Eventually, retrieving the original problem's result from the encrypted result.

Algorithm 3 Hu et al.'s outsourcing model [15]

Input: The base and exponent $b, e \in \mathbb{R}_N$, the modulus N
Output: The result $R = b^e \mod N$
Key Generation: $KeyGen(1^k, N) \rightarrow (p, L)$
U selects a large prime p at random and computes $L \leftarrow pN$, then keeps the secret key $SK = \{p, N\}$ and public key $PK = \{L\}$.
Encryption: $Enc(b, e) \rightarrow (B, E)$

1. U selects random integers $r, k \in \mathbb{R}_N$.
2. U computes $B \leftarrow (b + rN) \mod L$, $E \leftarrow e + k\phi(N)$.
3. U outsources (B, E, PK) to the cloud.

Computing Outsourcing: $C(B, E, PK) \rightarrow R_1$

1. C computes $R_1 \leftarrow C(B, E, PK) = B^E \pmod L$
2. Exponent E is presented in binary form, and each part is computed in parallel using Algorithm 2.
3. C gets the product of all parts and returns the result R_1 to U.

Decryption: $Dec(R_1, SK) \rightarrow R$
U computes $R \leftarrow Dec(R_1, SK) = R_1 \mod N$

As RSA is an application of modular exponentiation, Hu et al. utilized Algorithm 3 to delegate the computation of the RSA cryptosystem for transmitting a plaintext T between two IoT devices, U_1 and U_2, which have limited resources.

The parallel secure outsourcing protocol for the RSA algorithm developed by Hu et al. ([15], Protcol 2) involves taking the **plaintext message** T, which U_1 wants to send to U_2, as input. To accomplish this, U_1 generates a key-pair (e, d) for a modulus $n = pq$, where p and q are two large primes such that $ed \equiv 1 \mod \phi(n)$. Subsequently, U_1 delegates the computation of the RSA encryption to a cloud server, receives the result from the cloud, and forwards the encrypted message, the decryption key d, and modulus n to U_2. The detailed procedure of the algorithm is outlined in Algorithm 4 (exactly as mentioned in ([15], Protocol 2)).

In an overview, Algorithm 4 illustrates the following:

- U_1 generates keys, i.e., a decryption key $KP_2 = \{d, n\}$, secret key $SK = \{p', T\}$, and public key $PK = \{L\}$
- U_1 encrypts the plaintext message T and exponent e as B and E, respectively.
- U_1 outsources the computation of $B^E \pmod{L}$ to cloud C.
- U_1 forwards the decryption key KP_2 along with encrypted result F_1 to U_2.
- U_2 retrieves the original message F by decrypting the ecnrypted result F_1 using decryption key KP_2

Algorithm 4 Hu et al.s outsourcing for RSA algorithm [15]

```
Input: The plaintext message $T$
Output: The plaintext message $F$
Key Generation: $KeyGen(1^k, T) \to (e, d, p', L)$
1.  End-user $U_1$ generates two large primes $p$ and $q$ and calculates $n \leftarrow pq$, $\phi(n) \leftarrow (p-1)(q-1)$.
2.  $U_1$ chooses a random integer $e$ as encryption key such that $gcd(e, \phi(n)) = 1$ and $2 \le e \le \phi(n) - 1$.
3.  $U_1$ computes $d$ as decryption key such that $ed \equiv 1 \pmod{\phi(n)}$
4.  $U_1$ generates another large prime $p'$ and computes $L \leftarrow p'T$, then gets the decryption key $KP_2 = \{d, n\}$, secret key $SK = \{p', T\}$, and public key $PK = \{L\}$
Encryption: $Enc(T, e) \to (B, E)$
1.  $U$ selects two random integers $r, k \in \mathbb{R}_N$
2.  $U$ computes $B \leftarrow (b + rN)$ and $E \leftarrow e + k\phi(N)$
3.  $U$ outsource $(B, E, PK)$ to the cloud.
Computing Outsourcing: $C(B, E, PK) \leftarrow F_1$
1.  Cloud $C$ computes $F_1 \leftarrow C(B, E, PK) = B^E \pmod{L}$.
2.  Exponent $E$ is presented in a binary way, each part computed in parallel.
3.  $C$ gets the product of all parts and sends the result $F_1$ to $U_1$.
Message Forwarding: $\mathbb{F}(F_1, KP_2)$
1.  $U_1$ forwards key pair $KP_2$ and encrypted message $F_1$ to another user $U_2$.
Decryption: $Dec(F_1, KP_2) \to F$
1.  $U_2$ calculates $F \leftarrow Dec(F_1, KP_2) = F_1^d \pmod{N}$ to get the plaintext.
```

2.1. Drawbacks

Our analysis of Algorithms 3 and 4 reveals several security and usability drawbacks.

2.1.1. Algorithm 3

This algorithm closely resembles the ExpSOS algorithm by Zhou et al. [9], differing only in cloud computation methods. However, Rangasamy and Kuppusamy [10] presented a polynomial-time attack on ExpSOS that recovers the modulus N when two modular exponentiations with the same exponent are delegated. This attack applies here as well, allowing any probabilistic polynomial-time (PPT) adversary to retrieve N.

2.1.2. Algorithm 4

Several errors exist in Hu et al.'s RSA implementation:

- **Misinterpretation of RSA for message communication:**
  - The algorithm misinterprets the intended use of RSA for secure message communication.
- **Incorrect outsourcing protocol:**
  - The RSA outsourcing protocol is presented incorrectly. When using RSA for message transfer from sender U_1 to receiver U_2, U_2 (not U_1) generates the key pair (e, d) and modulus $N = pq$ (p and q are large primes).

- **Public key misuse:**
  - The public key (e, N) allows any user, including U_1, to send an encrypted message to U_2, who then uses the private key d for decryption. However, the algorithm claims U_1 generates (e, d, N), requiring them to create a new key pair for each message, rendering it impractical and inefficient.
- **Insecure key transmission:**
  - U_1 transmits the key pair (secret key) $KP_2 = \{d, N\}$ along with the encrypted message F_1 through an insecure channel. This exposes the original message T to potential adversaries on the channel.
- **Computationally infeasible encryption:**
  - In the Encryption phase, U_1 computes $E \leftarrow e + k\phi(N)$. However, if the RSA algorithm is to be correctly followed, U_1 lacks access to $\phi(N)$, and computing it for any given N is infeasible for a resource-constrained user.

Note 1. *Deviations from Secure RSA Communication:*

We highlight two key issues with Algorithm 4 alongside the drawbacks mentioned above regarding its adherence to the secure RSA message communication scheme:

1. ***Incorrect Key Distribution:*** *Algorithm 4 assumes user U_1 has access to both the public key (e, N) and the private key d. In RSA, only the receiver (U_2 in this case) holds the private key, while the public key is widely distributed. Allowing U_1 access to d contradicts this fundamental principle and compromises message security.*

2. ***Inconsistent Plaintext Usage:*** *The algorithm exhibits inconsistencies in its usage of the plaintext message T. Initially, T is correctly identified as the user's plaintext. However, two errors emerge in subsequent stages:*

- *Stage 4 of* ***Key Generation*** *utilizes the secret modulus L as $p'T$, a meaningless expression as T is considered as plaintext, which further deviates from expected operations.*
- *The* ***Encryption*** *phase employs b instead of T as the plaintext, further adding to the confusion and potentially leading to incorrect operations.*

These inconsistencies, combined with the incorrect key distribution, render Algorithm 4 unsuitable for secure message communication and highlight the need for careful adherence to standard cryptographic protocols.

3. Reproducing Hu et al.'s Protocol for RSA

Considering the misinterpretation and typing mistakes in ([15], Section VI, Protocol 2), we reproduce the protocol for secure message communication using the RSA algorithm while considering the following points:

- In the scenario of secret message communication between two end-users U_1 and U_2, if U_1 wants to send a plaintext T securely to U_2, U_1 should use the public key (e, N) generated by U_2.
- U_2 generates the key pair (e, d) for a given modulus only once for the entire session of message transfer as e and N are made public for any user to send a message. Thus, the values of e, d, and N are fixed.
- In practice, during message transfer, the value e is chosen to be a very small integer compared to the security parameter, whereas the size of d is almost as large as the modulus N. Thus, RSA encryption is computationally less challenging for a user than RSA decryption. As Vergnaud [17] mentioned, the encryption key is often a small key of a 16-bit integer, and e can take values up to $2^{16} + 1 = 65{,}537$.
- First, we reproduce Hu et al.'s protocol for secure message communication using the RSA algorithm, where outsourcing the RSA decryption computation is performed with the help of an honest-but-curious cloud server C.

Note 2. *We noticed that the outsourcing algorithm proposed by Hu et al. [15] assumes the cloud to be an honest-but-curious model. Therefore, the* ***Verification*** *step is missing in both the algorithms (Algorithms 3 and 4).*

Algorithm 5 sketches the Hu et al. algorithm properly in terms of how it is supposed to be with proper interpretation of RSA communication and without any typographical errors.

Algorithm 5 Reproduced Hu et al. [15] outsourcing for RSA decryption

```
Input: The plaintext $T$ to be sent by $U_1$
Output: The plaintext $T$ to be received by $U_2$
Key Generation: $KeyGen(1^k, N) \rightarrow (e, d)$
1. $U_2$ generates two large primes $p$ and $q$ ($p \neq q$) and computes $N = pq$ and $\phi(N) = (p-1)(q-1)$. $U_2$ keeps $p$ and $q$ as secret.
2. $U_2$ chooses a random integer $e$ as public encryption key such that $gcd(e, \phi(N)) = 1$ and $2 \leq e \leq \phi(N) - 1$.
3. $U_2$ computes $d$ as private decryption key such that $ed \equiv 1 \pmod{\phi(N)}$
4. $U_2$ makes the pair $(e, N)$ public, and keep $d$ as secret.
Encryption: $Enc(T, e) \rightarrow T^e \mod L$
1. $U_1$ generates a large prime $p'$ and computes $L = p'N$ where secret key $SK = \{p', N\}$, and public key $PK = \{L\}$.
2. $U_1$ encrypt the plaintext $T$ to $F = T^e \mod L$.
Message Forwarding: $U_1 \rightarrow (F, L) \rightarrow U_2$
1. $U_1$ forwards the encrypted message $F$ along with the modulus $L$ to another user $U_2$.
Secure Outsourcing:
1. $U_2$ selects two random integers $r, k \in \mathbb{R}_N$
2. $U_2$ computes $B \leftarrow (F + rN)$ and $D \leftarrow d + k\phi(N)$
3. $U_2$ outsource $(B, D, L)$ to the cloud.
Computing Outsourcing: $C(B, D, L) \leftarrow F_1$
1. Cloud $C$ computes $F_1 \leftarrow C(B, D, L) = B^D \pmod{L}$.
2. $C$ sends the result $F_1$ to $U_2$.
Decryption: $Dec(F_1) \rightarrow T$
1. $U_2$ calculates $T = F_1 \pmod{N}$ to get the plaintext $T$.
```

Now, we present an attack on this reproduced Algorithm 5 of Hu et al. and claim that an adversarial cloud can learn the value $\phi(N)$ (and hence the prime factors of N) while remaining an honest party.

3.1. Attack against Algorithm 5

In this section, we present an attack on the reproduced Hu et al. outsourcing protocol for secure message communication using the RSA algorithm. Our attack is based on multiple delegations and is similar to the attack by Rangasamy and Kuppusamy [10] on the Zhou et al. [9] protocol. We take into consideration the fact that both end-users, U_1 and U_2, have limited resources. For U_2, the values ofe, d, and N are fixed because (e, N) is the public key that can be used by anyone to send a secret message to U_2, which also means that the value of d remains the same throughout the communication. (d is the modular inverse of e modulo $\phi(N)$, which is unique. Since (e, N) is fixed, so is d).

Now, we consider the user U_1 communicating two encrypted texts (T_1 encrypted to F_1 and T_2 encrypted to F_2 using the public exponent e and the modulus $L_1 = p_1^{'}N$ and $L_2 = p_2^{'}N$) to U_2. Upon **Decryption**, U_2 selects random integers $r_1, k_1, r_2, k_2 \in \mathbb{R}_N$ and computes

$$B_1 \leftarrow (F_1 + r_1 N)$$

$$D_1 \leftarrow d + k_1\phi(N)$$

$$B_2 \leftarrow (F_2 + r_2 N)$$
$$D_2 \leftarrow d + k_2\phi(N)$$

and outsources (B_1, D_1, L_1) and (B_2, D_2, L_2) to the cloud.

An adversarial cloud C' performs the operation

$$D_1 - D_2 = (k_1 - k_2)\phi(N)$$

to obtain a multiple of $\phi(N)$. Given an RSA modulus $N = pq$ and a multiple of its Euler's totient function $\phi(N)$, Rabin's PPT algorithm (Rabin et al. [18]) produces the factorization (p, q) in anticipated polynomial time $\mathcal{O}(\log(N)^3)$.

3.2. *The Fix for Algorithm 5 against Our Attack*

Here, we propose a modification to the outsourcing model proposed by Hu et al. [15] for secure message communication using the RSA algorithm. To address the security issues in Algorithm 5, we introduce our fix to Algorithm 5 with proper interpretation of RSA communication.

Additionally, we consider a scenario where an end-user U_1 needs to send a secure message to another user U_2 with the help of an **untrustworthy fully malicious cloud/server**. Thus, we introduce the verification step that has been missing in Algorithms 4 and 5. We begin by considering the scenario where the end users U_1 and U_2 are resource-constrained. User U_1 wants to send the message T to U_2.

3.2.1. Correctness of Algorithm 6

We demonstrate the validity of our solution for Hu et al.'s outsourcing model concerning the RSA algorithm. This clarification confirms that our algorithm adheres to the RSA communication assumptions and indeed provides accurate results when outsourced.

1. U_2 computes $L = p'N$ for some random large prime p' and makes (e, L) public and keeps d, p' and N as secret, where $ed \equiv 1 \mod \phi(N)$.
2. U_1 computes $F = T^e \mod L$ and sends it to U_2.
3. U_2 computes $B = F + rN \mod L$ and $R = 2^t$ and $D \equiv d - R \mod \phi(N)$ (i.e., $D = d - R + k\phi(N)$ for some integer k).
4. U_2 sends (B, D, L) and (B, t, L) to the cloud C.
5. C computes $F_1 = B^D \mod L$ and $R_1 = B^R \mod L$
6. U_2 computes

$$\begin{aligned}
&F_1.R_1 \mod N \\
\implies &(B^D \mod L).(B^R \mod L) \mod N \\
\implies &((F + rN)^{(d-R)+k\phi(N)} \mod L). \\
&((F + rN)^R \mod L) \mod N \\
\implies &(F^{d-R+k\phi(N)}).(F^R) \mod N \\
\implies &F^d \mod N = T^{ed} \mod N = T \mod N
\end{aligned}$$

This shows the correctness of our algorithm.

Algorithm 6 Our fix for Hu et al.'s Outsourcing Model for RSA algorithm

```
Input: The plaintext $T$ to be sent by $U_1$
Output: The plaintext $T$ to be received by $U_2$
Key Generation: $KeyGen(1^k, N) \to (e, d)$
1. $U_2$ generates two large primes $p$ and $q$ ($p \neq q$) and computes $N = pq$ and $\phi(N) = (p-1)(q-1)$. $U_2$ keeps $p$ and $q$ as secret.
2. $U_2$ chooses a random integer $e$ as public encryption key such that $gcd(e, \phi(N)) = 1$ and $2 \leq e \leq \phi(N) - 1$.
3. $U_2$ computes $d$ as private decryption key such that $ed \equiv 1 \pmod{\phi(N)}$
4. $U_2$ computes $L = p'N$ where $p' \in \mathbb{N}$ is a large prime.
5. $U_2$ makes the pair $(e, L)$ public, and keep $d$ as secret.
Encryption: $Enc(T, e) \to T^e \mod L$
1. $U_1$ encrypt the plaintext $T$ to $F = T^e \mod L$.
Message Forwarding: $U_1 \to F \to U_2$
1. $U_1$ forwards the encrypted message $F$ to intended recipient $U_2$.
Secure Outsourcing:
1. $U_2$ selects a random integer $r \in \mathbb{Z}$.
2. $U_2$ computes $B \leftarrow (F + rN) \mod L$
3. $U_2$ generates $R \leftarrow 2^t$ for some fixed integer $t$ and $D \leftarrow d - R \mod \phi(N)$.
4. $U_2$ outsource the tuples $(B, D, L)$ and $(B, R, L)$ to the cloud.
Computing Outsourcing: $C(B, D, L) \leftarrow F_1$, $C(B, R, L) \leftarrow R_1$
1. Cloud $C$ computes $F_1 \leftarrow C(B, D, L) = B^D \pmod{L}$ and $R_1 \leftarrow C(B, R, L) = B^R \pmod{L}$.
2. $C$ sends the result $F_1$ and $R_1$ to $U_2$.
Verification: $U_2(F_1, R_1, F) \to 0 \cup 1$
Upon receiving $F_1$ and $R_1$ from cloud $C$, $U_2$ checks whether the results satisfy
    $$(F_1.R_1)^e \equiv F \mod N.$$
Decryption: $Dec(F_1, R_1, N) \to T$
1. If the results $F_1$ and $R_1$ passes the verification step, $U_2$ proceeds for the decryption.
2. $U_2$ calculates $T = F_1.R_1 \pmod{N}$ to get the plaintext $T$.
```

3.2.2. Security Analysis of Algorithm 6

We discuss the security of our proposed fix as outlined in Algorithm 6. We begin by listing the entities that are made public and can be accessed either by other users or the cloud.

- The public exponent e and public modulus L are shared by U_2 publicly, where $L = p'N$.
- It is assumed that the cloud knows all the abovementioned entities.

Privacy of $\phi(N)$

As evident from Algorithm 6, the equation representing the decryption process is given by:

$$d \equiv D + R \pmod{\phi(N)}$$
$$D \equiv d - R \pmod{\phi(N)}$$

where D and R have specific relations with $\phi(N)$ and t, respectively.

- The equation $D + R \equiv d + k_1\phi(N) \pmod{\phi(N)}$ implies the existence of an integer k_1 such that the sum of D and R equals the sum of d and k_1 multiplied by Euler's totient function of N ($\phi(N)$).

- If the cloud C has access to the value of $\phi(N)$, it can compute the secret key of U_2 and breach its security.
- However, our previously described attack in Section 3.1 will not work since D remains unchanged for multiple delegations (ref. Note 1).
- Instead, we now discuss a potential passive attack that can be used to compute the value of $\phi(N)$.

We follow the attack for exponent splitting as proposed by Mefenza and Vergnaud ([19], Section 3). Using the values e, D, and R, we obtain:

$$\begin{aligned} e(D+R) &= e(d-R+k_1\phi(N)+R) \\ &= ed+k_1\phi(N) \\ &= 1+(k_1+k_2)\phi(N) \\ &= 1+k^*\phi(N) \end{aligned}$$

Thus, the polynomial $f(x,y) = 1+x(N+y)$ has a root $(k^*, 1-p-q)$ modulo e' where $e' = e(D+R)$. However, the value $N = pq$ is not made public by U_2 and guessing the value N has probability $\frac{1}{\phi(N)}$.

Another possible way to approach the attack is to take the polynomial $f(x,y) = 1+xy$ that has a root $(k^*, \phi(N))$ modulo e' (ref. Mefenza and Vergnaud [19]). Here, $y = \phi(N)$ has the size $y \simeq N$. Considering the sizes of $e \simeq N^{\alpha}$, $d \simeq N^{\beta}$, $D \simeq N^{\delta}$, $k^* \simeq N^{\alpha+\beta_1+\delta}$, and $e' \simeq N^{1+\alpha+\beta_1}$ ($\beta = 1+\beta_1$) and following the lattice-based attack of Mefenza and Vergnaud [19], we evaluate the condition for the existence of a solution to the equation $f(x,y) = 1+xy = 0 \mod e'$ as:

$$\det(\mathcal{L}) < ((e')^m)^w$$

where $\mathcal{L}$ is the lattice of dimension w that is generated using the polynomial $f(x,y)$. However, since now $y \simeq N$, we obtain the condition for having a solution to $f(x,y) = 0 \mod e'$ as:

$$(\frac{1}{3}+\frac{\tau}{3})(\alpha+\beta_1+\delta)+1(\frac{1}{6}+\frac{\tau}{3}+\frac{\tau^2}{6}) < (\frac{1}{6}+\frac{\tau}{3})(1+\alpha+\beta_1)$$

which simplifies to:

$$(\frac{1}{3}+\frac{\tau}{3})(\alpha+\beta_1+\delta)+\frac{\tau^2}{6} < (\frac{1}{6}+\frac{\tau}{3})(\alpha+\beta_1)$$

Maximizing the value of δ by taking $\tau_{\max} = 1-2\delta$, the inequality transforms to:

$$(1+\alpha+\beta_1) < 0$$

This indicates that this attack is only possible when $\alpha+\beta_1 < -1$. Since α, β_1 are chosen to be positive numbers in our scheme, this attack will fail. This demonstrates that our scheme defeats the attack proposed by Mefenza and Vergnaud [19], and the value $\phi(N)$ cannot be computed by any curious cloud.

Privacy of N and p'

The public modulus is given as $L = p'N$, where both p' and N are kept secret by the user U_2. Thus, for two consecutive instances, the adversary will have $L_1 = p_1^{'}N$ and $L_2 = p_2^{'}N$, and with high probability $\gcd(L_1, L_2) = N$. Thus, to avoid such a scenario, U_2 can fix the public modulus as $L = p'N$ for some fixed large prime p'.

Note 3. *The computation of $R = 2^t$ must be performed by the user U_2 [20]. There exist various methods to efficiently compute R. For example, the binary representation of 2^t is $(100\cdots000)_2$ of bit-length $t+1$. Therefore, the value R (and subsequently D) can be readily delegated to the cloud C in its binary form. Cloud C can execute the delegated task in a parallel environment using Algorithm 2. However, the computation of D might raise concerns for a resource-constrained device like U_2, as $D = d - R \mod \phi(N)$ necessitates modular operations. To alleviate the computation burden on U_2, t can be chosen as an integer with a smaller bit-size (e.g., a 20-bit integer). Furthermore, the value R is computed only once for multiple delegations. It is permissible to select t as a random 20-bit integer for each delegation, although this may impact the algorithm's efficiency.*

3.3. Verification Analysis of Algorithm 6

In our scheme, user U_2 receives F_1 and R_1 from the cloud and verifies their correctness using the equation:

$$(F_1 \cdot R_1)^e \equiv F \pmod N,$$

since U_2 already received F from U_1. While Section 3.2.1 ensures this equation validates accurate results, we now explore its robustness against a malicious cloud attempting to bypass verification.

We consider a scenario where the cloud sends forged results $\overline{F_1}$ and $\overline{R_1}$, which still satisfy the verification equation:

$$(\overline{F_1} \cdot \overline{R_1})^e \equiv F \pmod N.$$

This implies

$$(\overline{F_1} \cdot \overline{R_1})^e = F + mN.$$

for some integer m. However, due to the modular operation with modulus N, the verification still yields the correct result $F \pmod N$:

$$(F + mN) \pmod N = F \pmod N.$$

This demonstrates the inherent robustness of our verification scheme, even against manipulation attempts. The resemblance of our verification scheme to RSA signature authentication further strengthens its security foundation.

4. Performance Analysis

In this section, we detail the experimental application of our proposed algorithm. Our algorithm is coded in Python (version 3.9.12) within the Jupyter Notebook environment, utilizing the NumPy package for computational tasks. The computational operations on the users' side U_1 and U_2 are executed on a system equipped with an Intel(R) Core(TM) i7-3770 processor running at 3.40 GHz with 8 GB RAM. On the cloud side, we leverage a GPU server labeled **Tesla V100-PCIE**, equipped with NVIDIA-SMI and CUDA version 11.5, 32 GB RAM.

For clarity and research focus, communication time between the cloud server and the local device is excluded from the results. Two reasons justify this:

1. **Computational Dominance**: Computational tasks significantly outweigh communication in terms of time expenditure.
2. **Prediction Challenges**: Accurately predicting communication time is difficult due to experimental setup limitations and research objectives.

We conduct experiments by executing the scheme across various moduli, ranging from 1024 to 4096 bits, with the base as the "plaintext" and exponent (e) set to 65,537 (a commonly used public key for RSA). To ensure more accurate computational time measurements, we carried out 20 trials for modulus size with specific bit size and subsequently computed their average.

Note 4. *As the Hu et al. scheme is proven insecure and impossible to follow, we have not shown the comparison analysis with Hu et al.'s scheme. Additionally, Hu et al.'s protocol assumed the cloud server to be honest-but-curious and hence avoided the verification step. We have assumed the cloud to be untrustworthy and proposed a new verification scheme. Therefore, we have skipped the comparison of performance with Hu et al.s protocol. Instead, we provide evidence that our outsourcing scheme is more efficient in executing RSA communication in comparison to local execution.*

Following Algorithm 6, we begin our experimental setup from the end-user U_2. U_2 generates the key pairs, i.e., private key $(d, N, \phi(N))$ and public key (e, L). The experimental setup considers four different bit sizes (1024, 2048, 3072, and 4096 b) for the modulus size. We list out the computation times for the **Key Generation** (Table 1).

Table 1. Computational time for **Key Generation**.

Bit Size	Computation Time (in s)
1024	0.53481
2048	6.93154
3072	23.44951
4096	93.92856

Using the public key (e, L), user U_1 encrypts the plaintext T to $F = T^e \mod L$. For experimental purposes, T is always taken as a 1000-bit number (randomly generated) for all different modulus sizes. The computation times for user U_1 are listed below Table 2.

Table 2. Computational time for **Encryption** by U_1.

Bit Size	Computation Time (in s)
1024	0.001822
2048	0.004237
3072	0.006481
4096	0.009796

Upon receipt of the transmitted ciphertext F, user U_2 chooses to securely delegate the decryption of F to an untrustworthy cloud. We document the computational time for U_2 in the phases of **Secure Outsourcing**, **Verification**, and **Decryption**. Additionally, we record the computation time for U_2 for the same input when the decryption of ciphertext F is conducted locally. Time units are measured in seconds.

As shown in Table 3, our secure outsourcing scheme demonstrates efficient performance, accelerating the decryption process as the modulus size increases compared to local execution. For instance, U_2 requires only 0.005318 s to securely outsource the decryption task for a 3072-bit modulus, whereas executing the decryption task itself takes 0.548388 s. This observation serves as evidence bolstering the efficiency of our algorithm. The acceleration in processing speed is evident from the "Speed Up" column in Table 3, demonstrating the performance enhancement achieved through our algorithm.

Table 3. Computational time comparison using outsourcing vs. performing locally by U_2.

Bit Size	Secure Outsourcing	Verification	Decryption	Total Time	Locally Executed	Speed up
1024	0.001291	0.000955	0.000269	0.002515	0.054834	21.8×
2048	0.001643	0.001792	0.00042	0.003855	0.256242	66.47×
3072	0.002058	0.002679	0.000581	0.005318	0.548388	103.119×
4096	0.002563	0.003817	0.000873	0.007253	1.24538	171.7055×

In local execution, the decryption process occurs entirely within U_2's system, without relying on the outsourcing services. This eliminates the need for U_2 to engage in cloud-based encryption, verification, and decryption steps. Instead, U_2 solely utilizes its computational resources to retrieve the message originally sent by U_1.

Figure 1 illustrates the graphical comparison between the 'Total Time' required for outsourcing and the time needed for decryption when executed 'locally'. The data utilized for this comparison are sourced from Table 3 (columns 'Total Time' and 'Locally Executed') and visualized in Figure 1.

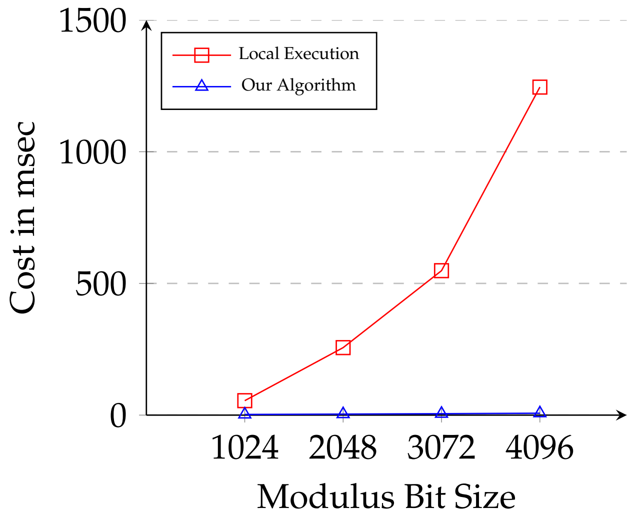

Figure 1. Local execution (without outsourcing) vs. using our algorithm (with outsourcing).

5. Conclusions and Future Scope

The RSA cryptosystem is a widely recognized and extensively employed cryptographic algorithm for secure data transmission. We conducted a comprehensive investigation of Hu et al.'s outsourcing algorithm for modular exponentiation and its usage in a message communication application using the RSA algorithm. Our analysis demonstrated the inadequacy and insecurity of Hu et al.'s algorithm. Moreover, we proposed a new modified algorithm for outsourcing RSA computations and provided a detailed security analysis of it. Additionally, we have also included the verification step that was missing in Hu et al.'s algorithm as we chose an untrustworthy malicious cloud.

While our work focused on secure message transmission through an insecure channel, future studies could explore real-life scenarios where encrypted messages are transmitted through noisy channels. Such channels introduce the risk of errors during transmission, necessitating additional measures like bit error correction. This area presents promising avenues for further research to ensure robust communication even in challenging environments.

Author Contributions: Writing—original draft, S.R.; Writing—review & editing, J.R. and C.-C.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Applebaum, B.; Ishai, Y.; Kushilevitz, E. From secrecy to soundness: Efficient verification via secure computation. In *International Colloquium on Automata, Languages, and Programming*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 152–163.
2. Gennaro, R.; Gentry, C.; Parno, B. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Advances in Cryptology—CRYPTO 2010: 30th Annual Cryptology Conference, Santa Barbara, CA, USA, 15–19 August 2010*; Proceedings 30; Springer: Berlin/Heidelberg, Germany, 2010; pp. 465–482.

3. Benabbas, S.; Gennaro, R.; Vahlis, Y. Verifiable delegation of computation over large datasets. In *Annual Cryptology Conference*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 111–131.
4. Papamanthou, C.; Shi, E.; Tamassia, R. Publicly Verifiable Delegation of Computation. *IACR Cryptol. ePrint Arch.* **2011**, *2011*, 587.
5. Hohenberger, S.; Lysyanskaya, A. How to securely outsource cryptographic computations. In *Theory of Cryptography Conference*; Springer: Berlin/Heidelberg, Germany, 2005; pp. 264–282.
6. Chen, X.; Li, J.; Ma, J.; Tang, Q.; Lou, W. New algorithms for secure outsourcing of modular exponentiations. *IEEE Trans. Parallel Distrib. Syst.* **2013**, *25*, 2386–2396. [CrossRef]
7. Wang, Y.; Wu, Q.; Wong, D.; Qin, B.; Chow, S.; Liu, Z.; Tan, X. Securely outsourcing exponentiations with single untrusted program for cloud storage. In *Computer Security-ESORICS 2014: 19th European Symposium on Research in Computer Security, Wroclaw, Poland, 7–11 September 2014*; Proceedings, Part I 19; Springer International Publishing: Berlin/Heidelberg, Germany, 2014; pp. 326–343.
8. Ding, Y.; Xu, Z.; Ye, J.; Choo, K. Secure outsourcing of modular exponentiations under single untrusted programme model. *J. Comput. Syst. Sci.* **2017**, *90*, 1–13. [CrossRef]
9. Zhou, K.; Afifi, M.; Ren, J. ExpSOS: Secure and verifiable outsourcing of exponentiation operations for mobile cloud computing. *IEEE Trans. Inf. Forensics Secur.* **2017**, *12*, 2518–2531. [CrossRef]
10. Rangasamy, J.; Kuppusamy, L. Revisiting single-server algorithms for outsourcing modular exponentiation. In *International Conference on Cryptology in India*; Springer International Publishing: Cham, Switzerland, 2018; pp. 3–20.
11. Ren, Y.; Dong, M.; Qian, Z.; Zhang, X.; Feng, G. Efficient algorithm for secure outsourcing of modular exponentiation with single server. *IEEE Trans. Cloud Comput.* **2018**, *9*, 145–154. [CrossRef]
12. Su, Q.; Zhang, R.; Xue, R. Secure outsourcing algorithms for composite modular exponentiation based on single untrusted cloud. *Comput. J.* **2020**, *63*, 1271. [CrossRef]
13. Bouillaguet, C.; Martinez, F.; Vergnaud, D. Cryptanalysis of modular exponentiation outsourcing protocols. *Comput. J.* **2022**, *65*, 2299–2314. [CrossRef]
14. Zhang, H.; Yu, J.; Tian, C.; Tong, L.; Lin, J.; Ge, L.; Wang, H. Efficient and secure outsourcing scheme for RSA decryption in Internet of Things. *IEEE Internet Things J.* **2020**, *7*, 6868–6881. [CrossRef]
15. Hu, Q.; Duan, M.; Yang, Z.; Yu, S.; Xiao, B. Efficient parallel secure outsourcing of modular exponentiation to cloud for IoT applications. *IEEE Internet Things J.* **2020**, *8*, 12782–12791. [CrossRef]
16. Rath, S.; Rangasamy, J. Privacy-Preserving Outsourcing Algorithm for Solving Large Systems of Linear Equations. *SN Comput. Sci.* **2023**, *4*, 656. [CrossRef]
17. Vergnaud, D. Comment on "Efficient and Secure Outsourcing Scheme for RSA Decryption in Internet of Things". *IEEE Internet Things J.* **2020**, *7*, 11327–11329. [CrossRef]
18. Rabin, M. *Digitalized Signatures and Public-Key Functions as Intractable as Factorization*; Massachusetts Inst of Tech Cambridge Lab for Computer Science: Cambridge, MA, USA, 1979.
19. Mefenza, T.; Vergnaud, D. Cryptanalysis of server-aided RSA protocols with private-key splitting. *Comput. J.* **2019**, *62*, 1194–1213. [CrossRef]
20. Kuppusamy, L.; Rangasamy, J. Improved Cryptographic Puzzle Based on Modular Exponentiation. In Proceedings of the Mathematics and Computing: ICMC, Haldia, India, 5–10 January 2015; pp. 107–121.