Eliminating Quadratic Slowdown in Two-Prime RSA Function Sharing

Maged Hamada Ibrahim
Department of Electronics, Communications and Computers, Faculty of Engineering, Helwan University
Sherif St., Helwan, Cairo, Egypt (Email: mhii72@hotmail.com)

(Received Aug. 23, 2006; revised and accepted Oct. 6, 2006)

Abstract

The nature of the RSA public modulus N as a composite of at least two secret large primes was always considered as a major obstacle facing the RSA function sharing without the help of a trusted dealer. The incorporated parties must agree on a suitable RSA modulus with no information revealed to them about its prime factors. Enormous number of trials must be performed before a suitable modulus is established. According to the number theory, for two ℓ-bit primes modulus, the number of trials is in the order of $\mathcal{O}(\ell^2)$. Efforts have been made to reduce the quadratic slowdown in the generation process, however, most of these protocols allow the joint generation of a multi-prime RSA modulus (an RSA modulus with at least three prime factors), which is a drift from standard. Other protocols require distributed primality tests over a shared secret modulus which is an extensive task. In this paper, we introduce a simple yet an efficient idea to allow two parties to jointly generate a two-prime RSA modulus with a running time complexity $\mathcal{O}(\ell)$. In our protocol, the distributed primality test is performed over a public modulus. Consequently, the expected running time will be reduced from several days to only few minutes. The protocol can be extended to the multiparty case. However, for clarity, in this paper, we focus on the two-party case.

Keywords: Digital signature, homomorphic encryption, Primality tests, secret sharing, standard RSA sharing, two-party computations

1 Introduction

A valid RSA public modulus N is a product of distinct odd primes, $N = \sum_{i=1}^{n} q_i$, $n \geq 2$. In case $n = 2$, the cryptosystem is spoken off as standard RSA or two-prime RSA, otherwise, it is a multi-prime RSA. e is the public exponent while d is the private exponent satisfying $ed = 1 \bmod \phi(N)$, where $\phi(N)$ is the RSA secret Euler totient. For distributed trust purposes, the private exponent is to be shared among two or more parties, a straight forward way to do so is to additively share $d = (d_1+d_2) \bmod \phi(N)$ among two parties for example. In order to sign the hash h of a message, each party generates her partial signature as $S_i = h^{d_i} \bmod N$. The final signature is $S = S_1 S_2 \bmod N$. For fault tolerance and availability purposes, the private exponent is to be shared among three or more parties using the techniques of threshold cryptography [3, 12, 13, 14].

The problem with RSA is that the RSA public modulus N is a composite of at least two large primes, these primes must be kept secret from the players. The players need to agree on a modulus N and be convinced that N is a product of two large primes with no information revealed to them about its factorization. The nature of the modulus N of the RSA function increased the difficulties to share the RSA keys without the help of a trusted dealer over other signature schemes which only require large public primes such as DSS [15, 17, 18, 25].

2 Previous Work

Over the past decade, the RSA function sharing problem attracted many researchers in the field of cryptography to reach an efficient and secure solution. Boneh and Franklin [5] showed how to generate the RSA keys without the help of the dealer, several phases of their protocols utilize reduced versions of information theoretic private multiparty computations. Clifford Cocks [10] has proposed another but unproven solution for the two party RSA function sharing, the protocol was extended for the multiparty case in [11], the computational intractability of his problem is weaker than RSA. Blackburn et al [4] have investigated Cocks protocol by adding verifiability to his scheme to face malicious behavior of the two parties. Frankel, Mackenzie and Yung [16] have improved the security of the Boneh-Franklin protocol. Later, Poupard and Stern [28] showed a different protocol for two Parties to jointly generate an RSA key. Niv Gilboa [20] constructed three protocols for the two-party RSA key generation, the first is based on the (1-out-of-2) - oblivious transfer of strings, the second is based on an efficient polynomial evaluation technique, the third uses special type of homomorphic encryption function.

In the above protocols, due to the way the modulus is generated –as a product of two ℓ-bit random numbers chosen simultaneously– the probability that such generated modulus is a product of exactly two primes is $(\ln 2.\ell)^{-2}$ according to the prime number theorem requiring a number of trials in the order of $O(\ell^2)$. Since ℓ ranges from 512 to 1024 bits depending on the security level and policy, the running time to reach a suitable modulus is several days using average processing speed which is quite a burden.

The method of Boneh and Horwitz [6] – Confining itself to the three-party case – achieves an $O(\ell)$ running time. Straub [31] took up ideas of Boneh-Horwitz and Gilboa to obtain an efficient algorithm tailored to the two-party scenario. The methods introduced in [6, 31] allow the generation of a three-prime RSA modulus of length 3ℓ in an expected running time of $O(\ell)$. Although the running time drops efficiently, the modulus is a three-prime modulus not a standard one.

In the above protocols, if trial division test (spoken off as *trivial division test*) is performed to test if the picked random strings are not divisible by small primes, the number of trials required to find a suitable modulus drop by a factor of $\lg \ell$ [4, 10, 11].

The recent three-party protocol of [23] was to completely eliminate the need for distributed primality tests in the three-prime RSA shared generation process, since the three parties select their random parameters originally as primes, the protocol is a one trial protocol and hence it is extremely fast. This protocol was extended for the multiparty threshold case in [22]. The robustness property to tolerate malicious behavior of the parties during the RSA function sharing protocol of [22] was discussed in [21].

Different from the above protocols, efforts have been made in [1] to eliminate the quadratic slowdown in the shared generation of the RSA modulus. However, the protocol requires that the incorporated parties perform distributed primality tests over a shared secret modulus resulting in an extensive computation and communication complexities on per trial basis.

3 Motivations and Contributions

The work in this paper is motivated by the observation that existing protocols that contribute in eliminating the quadratic slowdown [6, 21, 22, 23, 31] are multi-prime RSA protocols and fail to consider the standard RSA. Another major efficiency drawback in the three-prime RSA function sharing protocols of [6, 23, 31] is that the generated RSA modulus is of bit-length 3ℓ while the actual security is only 2ℓ since each party knows one prime factor. Each party is faced with the problem of factorizing the other two primes. The protocol of [1] performs distributed primality tests over a shared secret modulus. Although the protocol of [1] contributes in reducing the quadratic slowdown, the computation and communication complexities are extensively high on per trial basis.

The contributions of this paper is to introduce a simple yet an efficient idea to eliminate this annoying slowdown in the shared generation of a two-prime RSA modulus. The number of trials in our protocol is reduced to $\mathcal{O}(\ell)$ requiring several minutes to reach a suitable modulus and consequently, the expected running time is significantly improved. In our protocol, the distributed primality test is to be performed over a public modulus not a shared secret one, which significantly contributes in improving efficiency.

4 Related Protocols

We review several protocols which are closely related to the protocol presented in this paper. An approach proposed by Boneh and Horwitz [6] to combat the quadratic slowdown is as follows: Alice picks a random ℓ-bit prime p and a random ℓ-bit integer r_a, Bob picks a random ℓ-bit prime q and a random ℓ-bit integer r_b and Carol picks a random ℓ-bit integer r_c. Using a private distributed computation they compute $N = pq(r_a + r_b + r_c)$ with no information revealed about the full factorization of N. The three parties run a distributed primality test to test that $r_a + r_b + r_c$ is exactly a prime.

In the two-party protocol of Straub [31], the two parties Alice and Bob construct a 3ℓ-bit modulus of the form $(r_a + r_b)q_a q_b$ where r_a, r_b are arbitrary ℓ-bit random numbers and q_a, q_b are ℓ-bit primes. Alice holds r_a, q_a while Bob holds r_b, q_b. A suitable modulus is found after an expected time of $O(\ell)$.

In the recent protocol of [23], Alice picks a random ℓ-bit prime q_a, Bob picks a random ℓ-bit prime q_b and Carol picks a random ℓ-bit prime q_c. They share the computation of the RSA modulus $N = q_a q_b q_c$ with no information revealed to any of them about the full factorization of N. The protocol ends with Alice only knows q_a, Bob only knows q_b and Carol only knows q_c, in addition to the published modulus N. Their technique is as follows: Bob picks two ℓ-bit random numbers r_a and r_c such that $q_b = r_a + r_c$. Bob secretly delivers r_a to Alice and r_c to Carol. Alice and Carol run a private distributed computation to compute $N = (r_a + r_c)q_a q_c$.

Although the above protocols contribute in speeding up the shared generation of an RSA modulus, the modulus is a three-prime modulus which is a drift from the standard settings.

5 The Model

In the communication model, the two parties, Alice and Bob are connected such that any of them can communicate with the other through a private and authenticated channel.

In the adversary model, we assume a *passive adversary*, which means that this adversary can see and learn all information sent to or from the corrupted party without compromising the correct behavior of this party. The parties follow the execution steps of the protocol word for word but they are willing to learn any information leaked during execution. This commonly used security model is well-known as the *honest-but-curious* scenario.

6 Our Basic Idea

The two-prime RSA function sharing protocols [4, 5, 10, 11, 16, 20, 28] follow a common strategy. Consider two parties Alice and Bob. Alice picks two ℓ-bit random secret integers a_1, a_2 and Bob picks two ℓ-bit random secret integers b_1, b_2. Using private distributed computations, they jointly compute $N = (a_1 + b_1)(a_2 + b_2)$, then, they employ a distributed primality test to test whether or not N is a composite of exactly two primes. This *simultaneous* and *joint* testing of the two factors of N (biprimality test) is inefficient and is the main reason for the quadratic slowdown.

The idea is to avoid such simultaneous testing and to find a way to share and test each prime factor *individually* and *independently* without compromising their secrecy. If we are able to share and test $a_1 + b_1$, share and test $a_2 + b_2$ independently over a public modulus (unlike the protocol in [1]), then to jointly compute N, the quadratic slowdown is eliminated and the computation complexity is significantly improved.

7 Two-Party Private Computations

In this section we describe the building block used in our protocol.

7.1 Secret Sharing Notion

Let $\mathcal{R}$ be a ring and let $s \in \mathcal{R}$ be a secret. Assume that Alice holds the pair $x, a \in \mathcal{R}$ while Bob holds the pair $y, b \in \mathcal{R}$ where

$$s = x + y = ab.$$

The pair (x, y) is called an additive sharing of s while the pair (a, b) is called a multiplicative sharing of s.

The protocol described in this paper requires a subroutine for two parties to switch from multiplicative sharing of a secret value to additive sharing of this value. Namely, Alice holds a while Bob holds b such that $ab = s$, Alice and Bob run a subroutine which we will call it `mult-to-sum`, at the end of this subroutine Alice holds x and Bob holds y such that $x + y = s$, with no information leaked to any of them about s or the multiplicative shares.

The `mult-to-sum` subroutine can be implemented by different techniques, it may be implemented by Homomorphic encryption which is essentially a public key cryptosystem with a useful homomorphic property [31]. It can also be implemented via oblivious transfer of strings [2, 7, 9, 19, 24, 26, 29, 30]. Different techniques for the `mult-to-sum` and its inverse, `sum-to-mult` have been used for the efficient sharing of the RSA function in [6, 20, 23, 31].

7.2 The Underlying Primitive: Oblivious Transfer

Rabin [29] proposed the concept of oblivious transfer (OT) in the cryptographic scenario. In this case the sender has only one secret bit m and would like to have the receiver to get it with probability 1/2, on the other hand, the receiver does not want the sender to know whether it gets m or not. For OT_2^1, the sender has two secrets m_1 and m_2, the receiver will get one of them at the receiver's choice. The receiver does not want the sender to know which bit he chooses and the receiver must not know any information other than what he has chosen.

OT_n^1 is a natural extension of the OT_2^1 to the case of n secrets. However, constructing OT_n^1 from OT_2^1 is not a trivial problem. OT_n^1 is also known as "All or nothing disclosure of secrets (ANDOS)" [7, 19, 24, 30]. Oblivious transfer is a fundamental primitive in many cryptographic applications and secure distributed computations and has many applications such as private information retrieval (PIR), fair electronic contract signing, oblivious secure computation, etc. [2, 7, 9, 26, 27].

The main objective of the oblivious transfer protocols in [27] by Noar and Pinkas was to improve the efficiency and security of the protocols in [2]. Through out the work in this paper, we will consider the protocols of [27] due to several reasons. First, they prove efficiency over previous protocols, second, there are no number theoretic constraints on the strings to be obliviously transferred, third, the protocols have bandwidth-computation trade-offs which make them suitable for variety of applications.

The underlying OT. The OT protocols of [27] operate over a group Z_q of prime order, more precisely, G_q is a subgroup of order q of Z_p^* where p is prime and $q|p-1$. Let g be a generator group and assume that the Diffie-Hellman assumption holds. In their OT_2^1: The sender owns two strings M_0 and M_1. He chooses a random element $C \in Z_q$ and publishes it. The chooser picks a random $1 \leq k \leq q$ and sets $pk_\sigma = g^k$ where $\sigma \in \{0, 1\}$ is the chooser's choice. The chooser also computes $pk_{1-\sigma} = C/pk_\sigma$ and sends pk_0 to the sender. The sender picks a random R and computes g^R and C^R, he also computes pk_0^R and $pk_1^R = C^R/pk_0^R$. The sender sends g^R as well as the two encryptions, $H(pk_0^R, 0) \oplus M_0$ and $H(pk_1^R, 1) \oplus M_1$ to the chooser, where H is a random oracle modelled by a suitable hash function. The chooser is able to decrypt his choice using pk_σ.

In their OT_n^1: The sender owns n strings, $M_0, \cdots, M_{n-1}$. He picks $n-1$ random values $C_1, \cdots, C_{n-1}$ and publishes them, he also picks a random R and sends g^R to the chooser. The chooser selects a random k and sets $pk_\sigma = g^k$ where $\sigma \in \{0, \cdots, n-1\}$ is his choice, it holds that $pk_i = C_i/pk_0\ \forall i = (1, \cdots, n-1)$. The chooser sends pk_0 to the sender. the sender computes pk_0^R as well as $pk_i^R = C_i^R/pk_0^R\ \forall i = (1, \cdots, n-1)$. The sender sends g^R to the chooser as well as the encryption of each M_i, $H(pk_i^R, w, i) \oplus M_i$ where w is a random string known to both parties. Finally, the chooser is able to decrypt his choice using pk_σ.

7.3 The Mult-to-Sum Subroutine

Now, we describe the subroutine to convert from multiplicative sharing of a secret to additive sharing of the same secret.This subroutine will be called frequently in our protocol. Let $\mathcal{R}$ be a publicly known ring and let $\rho = \log|\mathcal{R}|$. Let $\ell \leq \rho$. Alice holds an ℓ-bit secret value $a \in \mathcal{R}$ and Bob holds an ℓ-bit secret value $b \in \mathcal{R}$. Alice and Bob want to additively share ab with no information revealed about a or b. The protocol is as follows:

- Bob selects uniformly at random ℓ ring elements $c_0, \cdots, c_{\ell-1}$ and defines ℓ pairs of ring elements $(t_0^{(0)}, t_0^{(1)}), \cdots, (t_{\ell-1}^{(0)}, t_{\ell-1}^{(1)})$. He sets $t_i^{(0)} = c_i$ and $t_i^{(1)} = 2^i b + c_i\ \forall i = (0, \cdots \ell - 1)$.
- Let the binary representation of a be $a_{\ell-1}, \cdots, a_0$, Alice and Bob performs ℓ invocations of OT_2^1. In the i-th invocation, Alice chooses $t_i^{(a_i)}$ from the pair $(t_i^{(0)}, t_i^{(1)})$.
- Alice sets $x = \sum_{i=0}^{\ell-1} t_i^{(a_i)}$ while Bob sets $y = -\sum_{i=0}^{\ell-1} c_i$.

Correctness. In the above subroutine, $x = \sum_{i=0}^{\ell-1} t_i^{(a_i)} = \sum_{i=0}^{\ell-1} a_i 2^i b + c_i$ and consequently, $x + y = ab$ over $\mathcal{R}$. Our protocol requires that $x + y = ab$ over the integers. This is easily attained if we choose $\rho > 2\ell$ where ℓ is the bit-length of a and b. This is also the reason why we did not use homomorphic encryption methods, since they do not perform over the integers. They require a prime field to perform. The `mult-to-sum` subroutine is able to compute additive shares of $a^i b^j$ for small integers i, j by setting $\rho > (i + j)\ell$.

8 Our Protocol

In this section we give the complete description of our protocol. Alice and Bob want to agree on a two-prime RSA modulus N with no information revealed to any of them about the factorization of N. They also want to share the private exponent d. The protocol is as follows.

8.1 Sharing and Testing the Prime Factors

- Alice picks an ℓ-bit random secret integer a_1 and an ℓ-bit random secret prime p_a.
- Bob picks an ℓ-bit random secret integer b_1 and an ℓ-bit random secret prime p_b.

The task now is to check whether or not $(a_1 + b_1)$ is a prime, the reader must notice that p_a and p_b are not factors of the final RSA modulus N, they are here to help testing $(a_1 + b_1)$ and to preserve the privacy of a_1 and b_1. Alice and Bob securely compute $N_1 = (a_1 + b_1)p_a p_b$ as follows:

- Alice locally computes $A = a_1 p_a$ while Bob locally computes $B = b_1 p_b$.
- Alice and Bob run the `mult-to-sum` subroutine to compute additive shares of Ap_b. At the end, Alice holds x_a while Bob holds x_b such that $Ap_b = x_a + x_b$.
- Alice and Bob run the `mult-to-sum` subroutine to compute additive shares of Bp_a. At the end, Alice holds y_a while Bob holds y_b such that $Bp_a = y_a + y_b$.
- Alice computes and sends $x_a + y_a$ to Bob while Bob computes and sends $x_b + y_b$ to Alice.
- Both parties are able to compute, $N_1 = x_a + x_b + y_a + y_b$.

Now, Alice and Bob are ready to perform the distributed primality test (Distributed Fermat's test) to check the primality of $(a_1 + b_1)$. Assuming for an instant that $\phi(N_1) = (a_1 + b_1 - 1)(p_a - 1)(p_b - 1)$. Of course, this is not true unless $(a_1 + b_1)$ is also a prime. They both agree on a random $g \in Z_{N_1}^*$ and proceed:

- Alice computes and sends $G_a = g^{(a_1-1)(p_a-1)} \mathrm{mod} N_1$ to Bob while Bob computes and sends $G_b = g^{b_1(p_b-1)} \mathrm{mod} N_1$ to Alice.
- Alice computes and sends $G'_b = G_b^{p_a-1} \mathrm{mod} N_1$ to Bob while Bob computes and sends $G'_a = G_a^{p_b-1} \mathrm{mod} N_1$ to Alice.
- Both parties are able to compute, $G = G'_a G'_b = g^{\phi(N_1)} \mathrm{mod} N_1$. They check G for unity.

The above computations is repeated for fresh quantities a_1, b_1, p_a, p_b until $G = 1$. Since p_a and p_b are originally picked as primes, according to number theory, Alice and Bob will reach a suitable prime $(a_1 + b_1)$ in an expected number of trials of $\mathcal{O}(\ell)$. Once a suitable prime is reached, they repeat the above protocol to share another prime $(a_2 + b_2)$ in exactly the same way. Each prime requires a number of trials of $\mathcal{O}(\ell)$. It is also nice to notice that the independent sharing and testing of each prime factor allows parallel computations. Hence, the running time to share two primes is in the order of $\mathcal{O}(\ell)$. If trivial primality test is performed on the picked random integers, the complexity improves to $\mathcal{O}(\ell/\mathrm{lg}\ell)$.

8.2 Joint Computation of the Modulus N

Let $N_1 = (a_1 + b_1)p_a p_b$ and let $N_2 = (a_2 + b_2)q_a q_b$. Alice holds a_1, a_2, p_a, q_a while Bob holds b_1, b_2, p_b, q_b where $(a_1 + b_1), (a_2 + b_2), p_a, p_b, q_a, q_b$ are all primes. Alice and Bob securely compute the RSA modulus $N = (a_1 + b_1)(a_2 + b_2)$ as follows:

- Alice computes and sends $N_a = (N_1 N_2)/(p_a q_a)$ to Bob.
- Bob computes and sends $N_b = (N_1 N_2)/(p_b q_b)$ to Alice.
- It is obvious that both Alice and Bob can compute N.

Lemma. *Under the assumption that, 1) The mult-to-sum subroutine is secure, 2) The factorization of a composite of two or more large primes is infeasible, 3) The RSA assumption holds, the privacy of Alice and Bob is preserved.*

8.3 Sharing the RSA Euler Totient $\phi(N)$

Alice and Bob want to compute additive shares of $\phi(N) = (a_1+b_1-1)(a_2+b_2-1)$. This can be done noninteractively as follows:

- Alice computes $\phi_a = N - a_1 - a_2 + 1$.
- Bob computes $\phi_b = -b_1 - b_2$.

It is clear that $\phi_a + \phi_b = \phi(N)$.

8.4 Sharing the Private Exponent

Alice and Bob agree on a public key, e. They want to compute additive shares of the private key, d. We recall the efficient GCD algorithm of [8] to compute inverses over the shared secret $\phi(N)$. Alice picks two random secret numbers λ_a, R_a and Bob picks two random secret numbers λ_b, R_b. Following the recommendations in [8], the secrets λ_a, λ_b are much greater than $\phi(N)$ (i.e. in the order of $O(N^2)$) while R_a, R_b are in the order of $O(N^3)$. Alice and Bob want to jointly compute the quantity γ where

$$\gamma = \lambda\phi(N) + Re = (\lambda_a + \lambda_b)(\phi_a + \phi_b) + (R_a + R_b)e.$$

- Alice and Bob run the `mult-to-sum` subroutine twice. At the end of the first run, Alice holds x_1 while Bob holds y_1 such that $\lambda_a\phi_b = x_1 + y_1$. At the end of the second run, Alice holds y_2 while Bob holds x_2 such that $\lambda_b\phi_a = x_2 + y_2$.
- Alice computes and sends $\gamma_a = x_1 + y_2 + \lambda_a\phi_a + R_a e$ to Bob while Bob computes and sends $\gamma_b = x_2 + y_1 + \lambda_b\phi_b + R_b e$ to Alice.
- Both parties are able to compute $\gamma = \gamma_a + \gamma_b$.

Assuming that $\gcd(\gamma, e) = 1$, the parties run the *Euclidean algorithm* to find the pair (x, y) such that $x\gamma + ye = 1$ which must exist. Since $xR + y = e^{-1} \bmod \phi(N)$, one may set $d = xR+y$. Additive shares of d can be computed easily, Alice sets $d_a = xR_a + y$, Bob sets $d_b = xR_b$. Clearly, $d = d_a + d_b$.

9 Improving the Mult-to-Sum Subroutine

One may argue that the computation complexity of our protocol is hidden in the number of oblivious transfers invoked when executing the `mult-to-sum` subroutine. In this section we introduce an efficiency improvement to the `mult-to-sum` subroutine. Since this subroutine is used frequently in our protocol, in this section, we aim to speedup the computation in order to improve the computation complexity of our protocol. We also need to dive into the details of the OT_2^1 invocation.

9.1 The Subroutine using OT_2^1

As a warmup and to declare our idea, in this subsection we give a complete description of the `mult-to-sum` subroutine using the efficient OT_2^1 from [27]. Let $\mathcal{R}$ be a public ring and let $\rho_2 = \log_2|\mathcal{R}|$, each element in $\mathcal{R}$ can be represented by ρ_2 bits. Let the binary representation of $a \in \mathcal{R}$ be $a_{\rho_2-1}, \cdots, a_0$. Alice holds $a \in \mathcal{R}$ while Bob holds $b \in \mathcal{R}$. p is a prime and $q|p-1$, g is a generator group. The protocol to additively share ab over $\mathcal{R}$ is as follows:

1 - Offline initializations:

Bob performs the following offline computations:

- Picks a random $C \in Z_q$ and publishes it.
- Picks a random R, computes g^R and C^R.
- Picks ρ_2 random elements in $\mathcal{R}$, $s_0, \cdots, s_{\rho_2-1}$ and sets ρ_2 pairs of elements in $\mathcal{R}$, $(t_0^{(0)}, t_0^{(1)}), \cdots, (t_{\rho_2-1}^{(0)}, t_{\rho_2-1}^{(1)})$ such that, $t_i^{(j)} = j2^i b + s_i$ $\forall i = (0, \cdots, \rho_2 - 1), j = (0, 1)$.

Alice performs the following offline computations:

- Picks ρ_2 random values $K_0, \cdots, K_{\rho_2-1}$ and computes $pk_{a_i}^{(i)} = g^{K_i}$.
- Computes $pk_{1-a_i}^{(i)} = C/pk_{a_i}^{(i)}$ $\forall i = (0, \cdots, \rho_2 - 1)$.

2 - Online computations and transfers:

Bob sends g^R to Alice. Alice computes the decryption keys, $(g^R)^{K_i} = (pk_{a_i}^{(i)})^R$ $\forall i = (0, \cdots, \rho_2 - 1)$. Alice and Bob performs ρ_2 OT_2^1 oblivious transfer of strings. In the i-th invocation:

- Alice sends $pk_0^{(i)}$ to Bob.

Table 1: Computation complexity of the subroutine using $\mathbf{OT}_2^1$

Overheads	Alice	Bob
Offline Computations	ρ_2 exponentiations	2 exponentiations
Online Computations	ρ_2 exponentiations + ρ_2 decryptions	ρ_2 exponentiations + $2\rho_2$ encryptions.
Communications	ρ_2 group elements	One group element + $3\rho_2$ string elements

- Bob computes $(pk_0^{(i)})^R$ and $(pk_1^{(i)})^R = C^R/(pk_0^{(i)})^R$.
- Bob sends the two encryptions, $H[(pk_0^{(i)})^R, w_i, 0] \oplus t_i^{(0)}$ and $H[(pk_1^{(i)})^R, w_i, 1] \oplus t_i^{(1)}$ and the random w_i to Alice.
- Alice is able to decrypt her choice using the decryption key $pk_{a_i}^{(i)}$.

After the ρ_2 OT's are completed, Alice computes $x = \sum_{i=0}^{\rho_2-1} t_i^{(a_i)}$ while Bob computes $y = \sum_{i=0}^{\rho_2-1} s_i$. It follows that $x + y = ab$ over $\mathcal{R}$.

Complexity evaluation. By investigating the above subroutine, Alice performs ρ_2 offline modular exponentiations on the form g^{K_i} while Bob performs two offline modular exponentiations g^R and C^R. Considering the online computations, Alice performs ρ_2 modular exponentiations on the form $(g^R)^{K_i}$ and ρ_2 decryptions while Bob performs ρ_2 modular exponentiations on the form $(pk_0^{(i)})^R$ and $2\rho_2$ encryptions. Regarding communication overheads, Alice sends ρ_2 group elements $pk_0^{(i)}$ while Bob sends one group element g^R and $3\rho_2$ string elements. The results are summarized in Table 1.

9.2 The Subroutine Using a General Radix

The trick is to generalize the radix r, through which the secret parameter a is represented and to employ OT_r^1 instead of OT_2^1. Let $\rho_2 = \log_2 |\mathcal{R}|$, then a can be encoded into ρ_2 bits $a_{\rho_2-1}, \cdots, a_0$. Let $\rho_r = \log_r |\mathcal{R}|$ where r is a general radix. a can also be represented by ρ_r symbols (alphabets) $\alpha_{\rho_r-1}, \cdots, \alpha_0$, each alphabet $\alpha_i \in \{0, \cdots, r-1\}$ (e.g. $r = 16$ for the Hexadecimal representation). Now, one may write, $a = \sum_{i=0}^{\rho_r-1} r^i \alpha_i$. It is obvious that, $\rho_r = \rho_2/\log_2 r$. We show that such attempt improves the computation complexity of the protocol. Alice holds $a \in \mathcal{R}$ while Bob holds $b \in \mathcal{R}$, the protocol to additively share ab over $\mathcal{R}$ is as follows:

1 - Offline initializations:

Bob performs the following offline computations:

- Picks $r-1$ random values $C_1, \cdots, C_{r-1}$ and publishes them.
- Picks a random R and computes $g^R, C_0^R, \cdots, C_{r-1}^R$.
- Picks ρ_r random elements in $\mathcal{R}$, $s_0, \cdots, s_{\rho_r-1}$. He also defines ρ_r sets of elements in $\mathcal{R}$, $(t_0^{(0)}, \cdots, t_0^{(r-1)}), \cdots, (t_{\rho_r-1}^{(0)}, \cdots, (t_{\rho_r-1}^{(r-1)})$.
- Sets $t_i^{(j)} = jr^i b + s_i \; \forall i = (0, \cdots \rho_r - 1), j = (0, \cdots, r-1)$.

Alice performs the following offline computations:

- Picks ρ_r random values $K_0, \cdots, K_{\rho_r-1}$ and computes $pk_{\alpha_i}^{(i)} = g^{K_i} \; \forall i = (0, \cdots, \rho_r - 1)$ and computes $pk_0^{(i)} = C_{\alpha_i}/pk_{\alpha_i}^{(i)}$.
- Computes $pk_{j \neq a_i}^{(i)} = C_j/pk_0^{(i)} \; \forall i = (0, \cdots, \rho_2 - 1), j = (0, \cdots, r-1)$.

2 - Online computations and transfers:

Bob sends g^R to Alice. Alice computes the decryption keys, $(g^R)^{K_0}, \cdots, (g^R)^{K_{\rho_r-1}}$. Alice and Bob perform ρ_r OT_r^1's. In the i-th invocation:

- Alice sends $pk_0^{(i)}$ to Bob.
- Bob computes $(pk_0^{(i)})^R$ and without any further exponentiations, he computes $pk_j^{(i)} = C_j^R/(pk_0^{(i)})^R$ $\forall j = (1, \cdots, R-1)$.
- Bob sends the encryption of each $t_i^{(j)}$, $H[(pk_j^{(i)})^R, w_i, j] \oplus t_i^{(j)}$ and the random w_i to Alice.
- Alice decrypts her choice, α_i among the r choices using $pk_{\alpha_i}^{(i)}$.

After the ρ_r oblivious transfers are accomplished, Alice computes $x = \sum_{i=0}^{\rho_r-1} t_i^{(\alpha_i)}$ while Bob computes $y = \sum_{i=0}^{\rho_r-1} s_i$. It follows that $x + y = ab$ over $\mathcal{R}$.

Complexity evaluation. Considering the offline overheads, Alice performs ρ_r modular exponentiations on the form g^{K_i} while Bob performs r modular exponentiations on the form $g^R, C_1^R, \cdots, C_{r-1}^R$. When Alice and Bob come online, Alice performs ρ_r modular exponentiations on the form $(g^R)^{K_i}$ and ρ_r decryptions while Bob performs ρ_r modular exponentiations on the form $(pk_0^{(i)})^R$ and $r\rho_r$ encryptions. Alice sends ρ_r group elements while Bob sends $(r\rho_r + \rho_r)$ string elements. These complexities are summarized in Table 2.

Table 2: Complexity evaluation of the subroutine using a general radix

Overheads	**Alice**	**Bob**
Offline Computations	ρ_r exponentiations	r exponentiations
Online Computations	ρ_r exponentiations + ρ_r decryptions	ρ_r exponentiations + $r\rho_r$ encryptions
Communications	ρ_r group elements	1 group element + $\rho_r(1+r)$ string elements

Table 3: Complexity comparison and evaluation ($\rho_2 = 1024$ bits)

-	**Offline exp.**	**Online exp.**	**Enc. + Dec.**	**communications** / **group elements, string elements**
r	$\rho_r + r$	$2\rho_r$	$\rho_r(1+r)$	$\rho_r + 1$, $\rho_r(1+r)$
$r = 2$	1026	2048	3072	1025, 3072
$r = 4$	516	1024	2560	513, 2560
$r = 16$	272	512	4352	257, 4352
$r = 256$	384	256	32896	129, 32896
$r = 2^{16}$	65600	128	4194368	65, 4194368

9.3 Comparison and Evaluation

We prefer to compare our results through a numerical example. Typical numerical setting is $\rho_2 = 1024$ bits. In this case, $\rho_r = 1024/\log_2 r$. Table 3 describes the complexities of the protocol for different values of r.

Notice that when the radix $r = 4$, all overheads are reduced, this provides an absolute improvement over the conventional case, $r = 2$. when $r = 16$, the communication overheads start to grow but still comparable to the conventional case. When $r = 256$ the offline computation complexity starts to grow slightly whereas the communication overheads grow rapidly. In all cases, the required online exponentiations – which are computationally expensive – are significantly reduced as r increases.

10 Conclusions

In this paper, we introduced a simple yet an efficient idea to eliminate the quadratic slowdown in the joint generation of a two-prime RSA modulus. We allow the sharing of a two-prime RSA function in a running time of $\mathcal{O}(\ell)$. Although we restricted the discussion to the two-party case, the protocol can be extended to the multiparty case. We also introduced an idea by which we can speedup the underlying subroutine and further improve the computation complexity.

References

[1] J. Algesheimer, J. Camenisch and V. Shoup, "Efficient computation modulo a shared secret with application to the generation of shared safe-prime products," in *Proceedings of the Crypto'02*, pp. 417-432, 2002.

[2] M. Bellare and S. Micali, "Non-interactive oblivious transfer and applications," in *proceedings of the Crypto'89*, LCNS 435, pp. 547-557, Springer-Verlag, 1990.

[3] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation," in *Proceedings of the 20th ACM symposium on the theory of computing*, pp. 1-10, 1988.

[4] S. Blackburn, S. Blake-Wilson, M. Burmester, and S. Galbraith, *Shared Generation of Shared RSA Keys*, Technical Report CORR98-19, Department of Combinatorics and Optimization, University of Waterloo, 1998.

[5] D. Boneh and M. Franklin, "Efficient generation of shared RSA keys," in *Crypto'97*, pp. 425-439, 1997.

[6] D. Boneh and J. Horwitz, "Generating a Product of three Primes with an Unknown Factorization," in *Proceedings 3rd Algorithmic Number Theory Symposium (ANTS-III)*, Portland, USA, pp. 237-251, 1998.

[7] C. Cachin, S. Micali, and M. Stadler, "Computationally private information retrieval with polylogarithmic communication," in *Advances in Cryptography (Eurocrypt'99)*, pp. 402-414, 1999.

[8] D. Catalano, R. Gennaro, and S. Halevi, "Computing inverses over a shared secret modulus," in *Eurocrypt'00*, LNCS 1807, pp. 190-207, Springer-Verlag, 2000.

[9] B. Chor, O. Goldreich, E. Kushilevitz, and M. Susdan, "Private information retrieval," *Journal of the ACM*, vol. 45, no. 6, pp. 965-982, 1998.

[10] C. Cocks, "Split knowledge generation of RSA parameters," in *Cryptography and Coding 6th IMA Conference*, LNCS 1355, pp. 89-95, Springer-Verlag, 1997.

[11] C. Cocks, *Split Generation of RSA Parameters with Multiple Participants.* http://www.cesg.gov.uk/downlds/math/rsa2.pdf

[12] Y. Desmedt, "Threshold cryptography," *European Transactions on Telecommunications and Related technologies*, vol.5 no.4, pp. 35-43, July-Aug. 1994.

[13] Y. Desmedt, "Society and group oriented cryptography a new concept," in *Advances in Cryptology (Crypto'87)*, LNCS 293, pp. 120-127, Springer-Verlag, 1988.

[14] Y. Desmedt and Y. Frankel, “Threshold cryptosystem,” in *Crypto’89*, LNCS 435, pp. 307-315, Springer-Verlag, 1990.

[15] Y. Frankel and Y. Desmedt, *Parallel Reliable Threshold Multisignature*, Technical Report TR-92-04-02. Univ. of Wisconsin–Milwaukee, 1992.

[16] Y. Frankel, P. Mackenzie, and M. Yung, “Robust efficient distributed RSA-key generation,” in *Proceedings of 30th Stoc*, pp. 663-672, 1998.

[17] R. Gennaro, *Theory and Practice of Verifiable Secret Sharing*, PhD thesis, Massachusetts Institute of Technology (MIT), May 1996.

[18] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, “Robust threshold DSS signatures,” in *advances in Cryptology (Eurocrypt’96)*, LNCS 1070, pp. 354-371, Springer-Verlag, 1996.

[19] Y. Gertner, Y. Ishai, E. Kushilevitz, and T. Malkin, “Protecting data privacy in information retrieval schemes,” in *Proceedings of 30th STOC*, pp. 151-160, 1998.

[20] N. Gilboa, “Two party RSA key generation,” in *Proceedings of Crypto’99*, LNCS 1666, pp. 116-129, 1999.

[21] M. H. Ibrahim, “Verifiable threshold sharing of a large secret safe prime,” in *Proceedings of the International Conference on Information Technology Coding and Computing (ITCC’05)*, pp. 608-613, Las Vegas, Nevada, USA, 2005.

[22] M. H. Ibrahim, I. A. Ali, I. I. Ibrahim and A. H. El-Sawy, “Fast fully distributed and threshold RSA function sharing,” in *Proceedings of the Information Systems: New Generations (ISNG’04)*, pp. 13-18, LasVegas, USA, Nov. 2004.

[23] M. H. Ibrahim, I. I. Ibrahim and A. H. El-Sawy, “Fast three-party shared generation of RSA keys without distributed primality tests,” in *Proceedings of the Information Systems: New Generations (ISNG’04)*, pp. 7-12, LasVegas, USA, Nov. 2004.

[24] E. Kushilevitz and R. Ostrovsky, “Single-database computationally private information retrieval,” in *Proceedings of 38th FOCS*, pp. 364-373, 1997.

[25] S. Langford, “Threshold DSS signatures without a trusted party,” in *Crypto’95*, LNCS 963, Springer-Verlag, pp. 397-409, 1995.

[26] M. Noar and B. Pinkas, “Oblivious transfer and polynomial evaluation,” in *Proceedings of the Thirty-first Annual ACM Symposium on Theory of Computing*, pp. 245-254, 1999.

[27] M. Noar and B. Pinkas, “Efficient oblivious transfer protocols,” in *Proceedings of SIAM Symposium on Discrete Algorithms (SODA’01)*, pp. 7-9, Jan. 2001.

[28] G. Poupard and J. Stern, “Generation of shared RSA-keys by two parties,” in *Asiacrypt’98*, pp. 245-254, 1999.

[29] M. Rabin, *How to Exchange Secrets by Oblivious Transfer*, Technical Report TR-81, Harvard Aiken Computation Laboratory, 1981.

[30] J. Stern, “A new and efficient all-or-nothing disclosure of secrets protocol,” in *Asiacrypt’98*, pp. 357-371, Springer-Verlag, 1998.

[31] T. Straub, “Efficient two party multi-prime RSA key generation,” in *Proceedings of IASTED International Conference on Communication, Network, and Information Security*, pp. 100-105, New York, 2003.

Maged Hamada Ibrahim Received his BSc in communications and computers engineering from Helwan University, Cairo; Egypt. Received his MSc and PhD in Cryptography and Network security systems from Helwan University in 2001 and 2005 respectively. Currently, working as a lecturer and also joining several network security projects in Egypt. His main interest is Cryptography and network security. More specifically, working on the design of efficient and secure cryptographic algorithms. In particular, secure distributed multiparty computations. Other things that interest him are number theory and the investigation of mathematics for designing secure and efficient cryptographic schemes.