Two Party RSA Key Generation

(extended abstract)

Niv Gilboa

Computer Science Department
Technion, Haifa 32000, Israel
e-mail:gilboa@cs.technion.ac.il

Abstract. We present a protocol for two parties to generate an RSA key in a distributed manner. At the end of the protocol the public key: a modulus $N = PQ$, and an encryption exponent e are known to both parties. Individually, neither party obtains information about the decryption key d and the prime factors of N: P and Q. However, d is shared among the parties so that threshold decryption is possible.

1 Introduction

We show how two parties can jointly generate RSA public and private keys. Following the execution of our protocol each party learns the public key: $N = PQ$ and e, but does not know the factorization of N or the decryption exponent d. The exponent d is shared among the two players in such a way that joint decryption of cipher-texts is possible.

Generation of RSA keys in a private, distributed manner figures prominently in several cryptographic protocols. An example is threshold cryptography, see [12] for a survey. In a threshold RSA signature scheme there are k parties who share the RSA keys in such a way that any t of them can sign a message, but no subset of at most $t-1$ players can generate a signature. A solution to this problem is presented in [11]. An important requirement in that work is that both the public modulus N and the private key are generated by a dealer and subsequently distributed to the parties. The weakness of this model is that there is a single point of failure– the dealer himself. Any adversary who compromises the dealer can learn all the necessary information and in particular forge signatures.

Boneh and Franklin show in [4] how to generate the keys without a dealer's help. Therefore, an adversary has to subvert a large enough coalition of the participants in order to forge signatures. Several specific phases of the Boneh-Franklin protocol utilize reduced and optimized versions of information theoretically private multi-party computations [1, 6]. Those phases require at least three participants: Alice and Bob who share the secret key and Henry, a helper party, who knows at the end of the protocol only the public RSA modulus N.

Subsequent works [8, 9, 20] and [13] consider other variants of the problem of jointly generating RSA keys. In [8] Cocks proposes a method for two parties

to jointly generate a key. He extends his technique to an arbitrary number of parties k [9]. The proposed protocol suffers from several drawbacks. The first is that the security is unproven and as Coppersmith pointed out (see [9]) the privacy of the players may be compromised in certain situations. The second is that the protocol is far less efficient than the Boneh-Franklin protocol. In [20] Poupard and Stern show a different technique for two parties to jointly generate a key. Their method has proven security given standard cryptographic assumptions. Some of the techniques employed in the current work are similar to the ideas of [20] but the emphasis is different. Poupard and Stern focus on maintaining robustness of the protocol, while we emphasize efficiency. In [13] Frankel, Mackenzie and Yung investigate a model of malicious adversaries as opposed to the passive adversaries considered in [4, 8, 9] and in our work. They show how to jointly generate the keys in the presence of any minority of misbehaving parties.

The current work focuses on joint generation of RSA keys by two parties. We use the Boneh-Franklin protocol and replace each three party sub-protocol with a two party sub-protocol. We construct three protocols. The first is based on $\binom{2}{1}$ oblivious transfer of strings. Thus, its security guarantee is similar to that of general circuit evaluation techniques [22, 16]. The protocol is more efficient than the general techniques and is approximately on par with Cocks' method and slightly faster than the Poupard-Stern method. The second utilizes a new intractability assumption akin to noisy polynomial reconstruction that was proposed in [19]. The third protocol is based on a certain type of homomorphic encryption function (a concrete example is given By Benaloh in [2, 3]). This protocol is significantly more efficient than the others both in computation and communication. It's running time is (by a rough estimate) about 10 times the running time the Boneh-Franklin protocol.

There are several reasons for using 3 different protocols and assumptions. The first assumption is the mildest one may hope to use. The second protocol has the appealing property that unlike the other two protocols it is not affected by the size of the modulii. In other words the larger the RSA modulus being used, the more efficient this protocol becomes in comparison with the others. Another interesting property of the first two protocols is that a good solution to an open problem we state at the end of the paper may make them nore efficient in terms of computation than the homomorphic encryption protocol.

We assume that an adversary is passive and static. In other words, the two parties follow the protocol to the letter. An adversary who compromises a party may only try to learn extra information about the other party through its view of the communication. Furthermore, an adversary who takes over Alice at some point in the execution of the protocol cannot switch over to Bob later on, and vice versa.

The remainder of the work is organized as follows. In section 2 we describe some of the tools and techniques developed previously and used in this paper. In section 3 we give an overview of the Boneh-Franklin protocol and of where we diverge from it. In section 4 we describe how to compute a modulus N in a distributed fashion using each of the three protocols. In section 5 we show

how to amortize the cost of computing several candidate modulii N and how to perform trial divisions of small primes. In section 6 we show how to compute the decryption exponent d. In section 7 we describe how to improve the efficiency of the homomorphic encryption protocol. In section 8 we discuss some performance issues.

2 preliminaries

Notation 1: The size of the RSA modulus N is σ bits (e.g $\sigma = 1024$).

2.1 Smallest primes

At several points in our work we are interested in the j smallest distinct primes $p_1, \ldots, p_j$ such that $\pi_{i=1}^{j} p_i > 2^\sigma$. The following table provides several useful parameters for a few typical values of σ.

σ	j	p_j	$\sum_{i=1}^{j} \lceil \log p_i \rceil$
512	76	383	557
1024	133	751	1108
1536	185	1103	1634
2048	235	1483	2189

2.2 Useful techniques

In this subsection we review several problems and techniques that were researched extensively in previous work, and which we use here.

Symmetrically private information retrieval: In the problem of *private information retrieval* presented by Chor et al. [7] k databases ($k \geq 1$) hold copies of the same n bit binary string x and a user wishes to retrieve the i-th bit x_i. A PIR scheme is a protocol which allows the user to learn x_i without revealing any information about i to any individual database. *Symmetrically private information retrieval*, introduced in [15], is identical to PIR except for the additional requirement that the user learn no information about x apart from x_i. This problem is also called 1 out of m oblivious transfer and all or nothing disclosure of secrets (ANDOS). The techniques presented in [15] are especially suited for the multi-database ($k \geq 2$) setting. In a recent work Naor and Pinkas [19] solve this problem by constructing a SPIR scheme out of any PIR scheme, in particular single database PIR schemes. The trivial single database PIR scheme is to simply have the database send the whole data string to the user. Clever PIR schemes involving a single database have been proposed in several works: [18, 5, 21]. They rely on a variety of cryptographic assumptions and share the property that for "small" values of n their communication complexity is worse than that of the trivial PIR scheme.

We now give a brief description of the SPIR scheme we use, which is the Naor-Pinkas method in conjunction with the trivial PIR scheme. In our scenario the data string x is made up of n substrings of length ℓ. The user retrieves

the i-th substring without learning any other information and without leaking information about i. The database randomly chooses $\log n$ pairs of seeds for a pseudo-random generator G: $(s_1^0, s_1^1), \ldots, (s_{\log n}^0, s_{\log n}^1)$. Every seed s_j^b $(1 \leq j \leq \log n,\ b \in \{0,1\})$ is expanded into $n\ell$ bits $G(s_j^b)$, which can be viewed as n substrings of length ℓ. It then prepares a new data string y of n substrings. Suppose the binary representation of i is $i_{\log n} \ldots i_1$. The i-th substring of y is the exclusive-or of the i-th substring of x and the i-th substring of each of $G(s_1^{i_1}), G(s_2^{i_2}), \ldots, G(s_{\log n}^{i_{\log n}})$. The user and database combine in $\log n$ $\binom{2}{1}$-OT of strings to provide the user with a single seed from every pair. Finally, the database sends y to the user, who is now able to learn a single substring of x. The parameters of the data strings we use are such that the running time is dominated by the $\log n$ $\binom{2}{1}$-OTs and the communication complexity is dominated by the $n\ell$ bits of the data string, which are sent to the user.

Dense probabilstic and homomorphic encryption: we are interested in an encryption method that provides two basic properties: (1) Semantic security: as defined in [17]. (2) Additive homomorphism: we can efficiently compute a function f such that $f(\mathrm{ENC}(a), \mathrm{ENC}(b)) = \mathrm{ENC}(a+b)$. Furthermore, the sum is modulo some number t, where t can be defined flexibly as part of the system.

As a concrete example we use Benaloh's encryption [2, 3]. The system works as follows. Select two primes p, q such that: $m \stackrel{\triangle}{=} pq \approx 2^\sigma$, $t|p-1$, $\gcd(t, (p-1)/t) = 1$ and $\gcd(t, q-1) = 1$ [1]. The density of such primes along appropriate arithmetic sequences is large enough to ensure efficient generation of p, q (see [2] for details). Select $y \in \mathcal{Z}_m^*$ such that $y^{\phi(m)/t} \not\equiv 1 \bmod m$. The public key is m, y, and encryption of $M \in \mathcal{Z}_t$ is performed by choosing a random $u \in \mathcal{Z}_m^*$ and sending $y^M u^t \bmod m$.

In order to decrypt, the holder of the secret key computes at a preprocessing stage $T_M \stackrel{\triangle}{=} y^{M\phi(m)/t} \bmod m$ for every $M \in \mathcal{Z}_t$. Hence, t is small enough that t exponentiations can be performed. Decryption of z is by computing $z^{\phi(m)/t} \bmod m$ and finding the unique T_M to which it is equal. The scheme is semantically secure based on the assumption that *deciding higher residuosity* is intractable [3]. Most of our requirements are met by the weaker assumption that *deciding prime residuosity* is intractable [2].

Oblivious polynomial evaluation: In this problem, presented by Naor and Pinkas in [19] Alice holds a field element $\alpha \in \mathcal{F}$ and Bob holds a polynomial $B(x)$ over $\mathcal{F}$. At the end of the protocol Alice learns only $B(\alpha)$ and Bob learns nothing at all. The intractability assumption used in [19] is new and states the following. Let $S(\cdot)$ is a degree k polynomial over $\mathcal{F}$ and let $m, d_{Q,x}$ be two security parameters $(d_{Q,x} > k)$. Given $2d_{Q,x}+1$ sets of m field elements such that in each set there is one value of S at a unique point different than 0 and $m-1$ random field elements, the value $S(0)$ is pseudo-random.

We now give a brief description of the protocol presented in [19] as used in our application where the polynomial B is of degree 1. Bob chooses a random bivariate polynomial $Q(x, y)$ such that the degree of y is 1, the degree of x is $d_{Q,x}$

[1] Therefore t is odd.

and $Q(0,\cdot) = B(\cdot)$. Alice chooses a random polynomial S of degree $d_{Q,x}$ such that $S(0) = \alpha$. Define $R(x)$ as the degree $2d_{Q,x}$ polynomial $R(x) = Q(x, S(x))$. Alice chooses $2d_{Q,x}+1$ different non-zero points x_j for $j = 1, \ldots, 2d_{Q,x}+1$. For each such j Alice randomly selects $m-1$ field elements $y_{j,1}, \ldots, y_{j,m-1}$ and sends to Bob x_j and a random permutation of the m elements $S(x_j), y_{j,1}, \ldots, y_{j,m-1}$ (denoted by $z_{j,1}, \ldots, z_{j,m}$). Bob computes $Q(x_j, z_{j,i})$ for $i = 1, \ldots, m$. Alice and Bob execute a SPIR scheme in which Alice retrieves $Q(x_j, S(x_j))$. Given $2d_{Q,x}+1$ such pairs of $x_j, R(x_j)$ Alice can interpolate and compute $R(0) = B(\alpha)$.

The complexity of the protocol is $2d_{Q,x} + 1$ executions of the SPIR scheme for data strings of m elements.

3 Overview

In this section we give an overview of our protocol. The stages in which we use the Boneh-Franklin protocol exactly are the selection of candidates and the full primality test (the other stages require a third party in [4]). The protocol is executed in the following steps.

1. **Choosing candidates** Alice chooses independently at random two $\sigma/2-1$ bit integers $P_a, Q_a \equiv 3 \bmod 4$, and Bob chooses similarly $P_b, Q_b \equiv 0 \bmod 4$. The two parties keep their choices secret and set as candidates $P = P_a + P_b$ and $Q = Q_a + Q_b$.
2. **Computing** N Alice and Bob compute $N = (P_a + P_b)(Q_a + Q_b)$. We show how to perform the computation using three different protocols and three different intractability assumptions.
3. **Initial primality test** For each of the smallest k primes $p_1, \ldots, p_k$ the participants check if $p_i \mid N$ $(i = 1, \ldots, k)$. This stage is executed in conjunction with the computation of N. If N fails the initial primality test, computing a new candidate N is easier than computing it from scratch (as is the case following a failure of the full primality test)
4. **Full primality test** The test of [4] is essentially as follows: Alice and Bob agree on $g \in \mathcal{Z}_N^*$. If the Jacobi symbol $\left(\frac{g}{N}\right)$ is not equal to 1 choose a new g. Otherwise Alice computes $v_a = g^{(N-P_a-Q_a+1)/4} \bmod N$, and Bob computes $v_b = g^{(P_b+Q_b)/4} \bmod N$. If $v_a = v_b$ or $v_a = -v_b \bmod N$ the test passes.
5. **Computing and sharing d** In this step we compute the decryption exponent d assuming that e is known to both parties and that $\gcd(e, \phi(N)) = 1$. Alice receives d_a and Bob receives d_b so that $d = d_a + d_b \bmod \phi(N)$ and $de \equiv 1 \bmod m$. Boneh and Franklin describe two protocols for the computation of d. The first is very efficient and can be performed by two parties, but leaks $\phi(n) \bmod e$. Therefore, this method is suitable for small public exponents and not for the general case. The second protocol computes d for any e but requires the help of a third party.

4 Computing N

Alice holds P_a, Q_a and Bob holds P_b, Q_b. They wish to compute

$$N = (P_a + P_b)(Q_a + Q_b) = P_aQ_a + P_aQ_b + P_bQ_a + P_bQ_b.$$

We show how to carry out the computation privately using three different protocols.

4.1 Oblivious transfers

Let $\mathcal{R}$ be a publicly known ring and let $a, b \in \mathcal{R}$. Denote $\rho = \log|\mathcal{R}|$ (each element in $\mathcal{R}$ can be encoded using ρ bits). Assume Alice holds a and Bob holds b. They wish to perform a computation by which Alice obtains x and Bob obtains y such that $x + y = ab$ where all operations are in $\mathcal{R}$. Furthermore, the protocol ensures the privacy of each player given the existence of oblivious transfers. In other words the protocol does not help Alice and Bob to obtain information about b and a respectively. The protocol:

1. Bob selects uniformly at random and independently ρ ring elements denoted by $s_0, \ldots, s_{\rho-1} \in \mathcal{R}$. Bob proceeds by preparing ρ pairs of elements in $\mathcal{R}$: $(t_0^0, t_0^1), \ldots, (t_{\rho-1}^0, t_{\rho-1}^1)$. For every i $(0 \leq i \leq \rho - 1)$ Bob defines $t_i^0 \stackrel{\triangle}{=} s_i$ and $t_i^1 = 2^i b + s_i$.
2. Let the binary representation of a be $a_{\rho-1} \ldots a_0$. Alice and Bob execute ρ $\binom{2}{1}$-OTs. In the i-th invocation Alice chooses $t_i^{a_i}$ from the pair (t_i^0, t_i^1).
3. Alice sets $x \stackrel{\triangle}{=} \sum_{i=0}^{\rho-1} t_i^{a_i}$ and Bob sets $y \stackrel{\triangle}{=} -\sum_{i=0}^{\rho-1} s_i$.

Lemma 1. *$x + y = ab$ over the ring $\mathcal{R}$.*

Proof. Since $a_{\rho-1}, \ldots, a_0$ is the binary representation of a we can write $a = \sum_{i=0}^{\rho-1} a_i \cdot 2^i$.

$$\begin{aligned} x + y &= \sum_{i=0}^{\rho-1} t_i^{a_i} - \sum_{i=0}^{\rho-1} s_i \\ &\equiv \sum_{i=0}^{\rho-1} (a_i \cdot 2^i b + s_i) - \sum_{i=0}^{\rho-1} s_i \\ &\equiv b \sum_{i=0}^{\rho-1} a_i \cdot 2^i \\ &\equiv ab \end{aligned}$$

□

In the following protocol for computing N the ring $\mathcal{R}$ is $\mathcal{Z}_{2^\sigma}$, the integers modulo 2^σ.

1. Alice and Bob use the previous protocol twice to additively share $P_aQ_b = x_1 + y_1 \bmod 2^\sigma$ and $P_bQ_a = x_2 + y_2 \bmod 2^\sigma$. Alice holds x_1, x_2 and Bob holds y_1, y_2.
2. Bob sends $y \stackrel{\triangle}{=} y_1 + y_2 + P_bQ_b \bmod 2^\sigma$ to Alice.
3. Alice computes $P_aQ_a + y \bmod 2^\sigma$. Alice now holds $N \bmod 2^\sigma$, which is simply N due to the choice of σ.

Lemma 2. *The transcript of the view of the execution of the protocol can be simulated for both Alice and Bob and therefore the protocol is secure.*

Proof. We denote the messages that Alice receives during the sharing of P_aQ_b by $t_0^{a_0}, \ldots, t_{\sigma-1}^{a_{\sigma-1}}$ and the messages received while sharing P_bQ_a by $t_\sigma^{a_\sigma}, \ldots, t_{2\sigma-1}^{a_{2\sigma-1}}$. In the same manner we denote Bob's random choices for the sharing of P_aQ_b and P_bQ_a by $s_0, \ldots, s_{\sigma-1}$ and $s_\sigma, \ldots, s_{2\sigma-1}$ respectively.

Bob's view can be simulated because the only messages Alice sent him were her part of 2σ independent oblivious transfers.

Alice receives $2\sigma + 1$ elements in $\mathcal{Z}_{2^\sigma}$:

$$t_0^{a_0}, \ldots, t_{2\sigma-1}^{a_{2\sigma-1}}, y.$$

The uniformly random and independent choices by which $s_0, \ldots, s_{2\sigma-1}$ are selected ensure that the messages Alice receives are distributed uniformly subject to the condition that

$$\sum_{i=0}^{2\sigma-1} t_i^{a_i} + y \equiv N - P_aQ_a \bmod 2^\sigma.$$

Since Alice can compute $N - P_aQ_a$ a simulator S_a can produce the same distribution as that of the messages Alice receives, given N, P_a, Q_a. □

Lemma 3. *The computation time and the communication complexity of the protocol are dominated by 2σ oblivious transfers. The transfered strings are of length σ.*

4.2 Oblivious polynomial evaluation

Alice and Bob agree on a prime $p > 2^\sigma$ and set $\mathcal{F}$ to be $GF(p)$. They employ the following protocol to compute N:

1. Bob chooses a random element $r \in \mathcal{F}$. He prepares two polynomials over $\mathcal{F}$: $B_1(x) = P_bx + r$ and $B_2 = Q_bx - r + P_bQ_b$.
2. Alice uses the oblivious polynomial evaluation protocol of [19] to attain $B_1(Q_a)$ and $B_2(P_a)$. Alice computes $N = P_aQ_a + B_1(Q_a) + B_2(P_a)$.

The security of the protocol depends on the security of the cryptographic assumption outlined in subsection 2.2 and of a similar argument to the proof of lemma 2.

Lemma 4. *The computational complexity of the protocol is dominated by the execution of $2 \log m(2d_{Q,x} + 1)$ oblivious transfers, where m and $d_{Q,x}$ are the security parameters. The communication complexity is less than $3m(2d_{Q,x}+1)\sigma$.*

4.3 Benaloh's encryption

We now compute N by using the homomorphic encryption described in subsection 2.2. Let $p_1, \ldots, p_j$ be the smallest primes such that $\pi_{i=1}^{j} p_i > 2^\sigma$. The following protocol is used to compute $N \bmod p_i$:

1. Let $t \stackrel{\triangle}{=} p_i$. Alice constructs the encryption system: an appropriate p, q, y, and sends the public key $y, m = pq$ to Bob. Alice also sends the encryption of her shares, i.e $z_1 \stackrel{\triangle}{=} y^{P_a} u_1^t \bmod m$ and $z_2 \stackrel{\triangle}{=} y^{Q_a} u_2^t \bmod m$, where $u_1, u_2 \in \mathcal{Z}_m^*$ are selected uniformly at random and independently .
2. Bob computes the encryption of $P_b Q_b \bmod t$, which is denoted by z_3, calculates
$$z \stackrel{\triangle}{=} z_1^{Q_b} \cdot z_2^{P_b} \cdot z_3 \bmod m$$
and sends z to Alice.
3. Alice decrypts z, adds to the result $P_a Q_a$ modulo t and obtains $N \bmod t$.

The two players repeat this protocol for each p_i, $i = 1, \ldots, j$. Alice is able to reconstruct N from $N \bmod p_i$, $i = 1, \ldots, j$ by using the Chinese remainder theorem.

Lemma 5. *Assuming the intractability of the prime residuosity problem, the transcript of the views of both parties in the protocol can be simulated.*

Proof. The distribution of Bob's view can be simulated by encrypting two arbitrary messages assuming the intractability of prime residuosity. Therefore, Alice's privacy is assured.

The distribution of Alice's view can be simulated as follows. Given N, $N \bmod p_i$ can be computed for every i. The only message that Alice receives is $z \stackrel{\triangle}{=} z_1^{Q_b} \cdot z_2^{P_b} \cdot z_3 \bmod m$. By the definition of z_3 and the encryption system $z_3 = y^{P_b Q_b} u^t \bmod m$ where u is a random in Z_m^*. Thus z is a random element in the appropriate coset (all the elements whose decryption is $N - P_a Q_a \bmod t$). □

Lemma 6. *The running time of the protocol is dominated by the single decryption Alice executes, the communication complexity is 3σ and the protocol requires one round of communication.*

5 Amortization and initial primality test

The initial primality test consists of checking whether a candidate N is divisible by one of the first k primes $p_1, \ldots, p_k$. If it is then either $P = P_a + P_b$ or $Q = Q_a + Q_b$ is not a prime. This test can be carried out by Alice following the computation of N.

If a candidate N passes the initial primality test, Alice publishes its value and it becomes a candidate for the full primality test of [4]. However, if it fails the test a new N has to be computed. In this section we show how to efficiently find a new candidate following a failure of the initial test by the previous candidate. The

total cost of computing a series of candidates is lower than using the protocols of section 4 each time anew. We show two different approaches. One for the oblivious transfer and oblivious polynomial evaluation protocols, and the other for the homomorphic encryption protocol.

5.1 OT and oblivious polynomial evaluation

Suppose that after Alice and Bob discard a certain candidate N they compute a new one by having Alice retain the previous P_a, Q_a and having Bob choose new P_b, Q_b. In that case, as we show below, computing the new N can be much more efficient than if both parties choose new shares. The drawback is that given both values of N Bob can gain information about P_a, Q_a. Therefore, in this stage (unlike the full primality test) Alice does not send the value of N to Bob.

Assume Bob holds two sequences of strings: $(a_1^0, \ldots, a_n^0)$, $(a_1^1, \ldots, a_n^1)$ and Alice wishes to retrieve one sequence without revealing which one to Bob and without gaining information about the second sequence. Instead of invoking a $\binom{2}{1}$-OT protocol n times the players agree on a pseudo-random generator G and do the following:

1. Bob chooses two random seeds s_1, s_2.
2. Alice uses a single invocation of $\binom{2}{1}$-OT to gain s_b, where $b \in \{0, 1\}$ denotes the desired string sequence.
3. Bob sends to Alice the first sequence masked (i.e bit by bit exclusive-or) by $G(s_1)$ and the second sequence masked by $G(s_2)$.

Alice can unmask the required sequence while the second sequence remains pseudo-random. In the protocol of subsection 4.1 N is computed using only oblivious transfers in which Alice retrieves a set of 2σ strings from Bob. Alice's choices of which strings to retrieve depend only on her input P_a, Q_a. Therefore if Alice retains P_a and Q_a while Bob selects a sequence of inputs $(P_b^1, Q_b^1), \ldots, (P_b^n, Q_b^n)$, the two players can compute a sequence of candidates $N^1, \ldots, N^n$ with as many oblivious transfers as are needed to compute a single N.

The same idea can be used in the oblivious polynomial evaluation protocol, as noted in [19]. The evaluation of many polynomials at the same point requires as many oblivious transfers as the evaluation of a single polynomial at that point. Thus, computing a sequence of candidates N requires only $2 \log m(2d_{Q,x} + 1)$ computations of $\binom{2}{1}$-OT.

5.2 Homomorphic encryption

Alice and Bob combine the two stages of computing N and trial divisions by using the protocol of subsection 4.3 flexibly. Let $p_1, \ldots, p_{j'}$ be the j' smallest distinct primes such that $\pi_{i=1}^{j'} p_i > 2^{(\sigma-1)/2}$. Alice and Bob pick their elements at random in the range $0, \ldots, \pi_{i=1}^{j'} p_i - 1$ by choosing random elements in each $\mathcal{Z}_{p_i}$ for $i = 1, \ldots, j'$.

Alice and Bob compute $N \bmod p_i$ as described in subsection 4.3. If $N \equiv 0 \bmod p_i$ then at least one of the elements $P = P_a + P_b$ or $Q = Q_a + Q_b$ is divided by p_i. In that case, Alice and Bob choose new, random elements: $P_a, P_b, Q_a, Q_b \bmod p_i$, and recompute $N \bmod p_i$. The probability of this happening is less than $2/p_i$. Thus the expected number of re-computations is less than $\sum_{i=1}^{j'} 2/p_i$. This quantity is about 3.1 for $\sigma = 1024$ (2 does not cause a problem because $P_a \equiv Q_a \equiv 3 \bmod 4$ and $P_b \equiv Q_b \equiv 0 \bmod 4$).

Setting $P_a \bmod p_i$ for $i = 1, \ldots, j'$ determines P_a, and by the same reasoning the other 3 shares that Alice and Bob hold are also set. The two players complete the computation of N by determining the value of $N \bmod p_i$ (using the protocol of subsection 4.3) for $i = j' + 1, \ldots, j$, where $\pi_{i=1}^{j} p_i > 2^{\sigma}$. If for one of these primes $N \equiv 0 \bmod p_i$ Alice and Bob discard their shares and pick new candidates [2].

6 Computing d

Alice and Bob share $\phi(N)$ in an additive manner. Alice holds $\phi_a \stackrel{\triangle}{=} N - P_a - Q_a + 1$, Bob holds $\phi_b = -Q_b - P_b$ and $\phi_a + \phi_b = \phi(N)$. The two parties agree on a public exponent e. Denote $\eta \stackrel{\triangle}{=} \lceil \log e \rceil$. We follow in the footsteps of the Boneh-Franklin protocol and employ their algorithm to invert e modulo $\phi(N)$ without making reductions modulo $\phi(N)$:

1. Compute $\zeta = -\phi(N)^{-1} \bmod e$.
2. Compute $d = (\zeta\phi(N) + 1)/e$.

Now $de \equiv 1 \bmod \phi(N)$ and therefore d is the inverse of e modulo $\phi(N)$.

As a first step Alice and Bob change the additive sharing of $\phi(N)$ into a multiplicative sharing modulo e, without leaking information about $\phi(N)$ to either party. At the end of the sub-protocol Alice holds $r\phi(N) \bmod e$ and Bob holds $r^{-1} \bmod e$, where r is a random element in $\mathcal{Z}_e^*$.

1. Bob chooses uniformly at random $r \in \mathcal{Z}_e^*$. Alice and Bob invoke the protocol of subsection 4.1, setting $\mathcal{R} \stackrel{\triangle}{=} \mathcal{Z}_e$, $a \stackrel{\triangle}{=} \phi_a$ and $b \stackrel{\triangle}{=} r$. At the end of the protocol Alice holds x and Bob holds y such that $x + y \equiv \phi_a r \bmod e$.
2. Bob sends $y + \phi_b r \bmod e$ to Alice.
3. Alice computes $x + y + \phi_b r \equiv r\phi(N) \bmod e$, and Bob computes $r^{-1} \bmod e$.

Lemma 7. *The computation time and the communication complexity of the protocol are dominated by η oblivious transfers.*

After completing the sub-protocol we described above, Alice and Bob finish the inversion algorithm by performing the following steps:

[2] An interesting optimization is not to discard the whole share (P_a, P_b, Q_a, Q_b), but for each specific share, say P_a, only to select a new $P_a \bmod p_i$ for $i = j' - c, \ldots, j'$, where c is a small constant. The probability is very high that the new N thus defined is not a multiple of p_i.

1. The two parties hold multiplicative shares of $\phi(N) \bmod e$. They compute the inverse of their shares modulo e and thus have ζ_a, ζ_b respectively such that $\zeta_a \cdot \zeta_b \equiv -\phi(N)^{-1} \equiv \zeta \bmod e$.
2. Alice and Bob re-convert their current shares into additive shares modulo e, i.e ψ_a, ψ_b such that $\psi_a + \psi_b \equiv \zeta \bmod e$. Bob chooses randomly $\psi_b \in \mathcal{Z}_e$ and the two parties combine to enable Alice to gain $\psi_a \stackrel{\triangle}{=} -\psi_b + \zeta_a\zeta_b \bmod e$. This is done by employing essentially the same protocol we used for transforming ϕ_a, ϕ_b into a multiplicative sharing. If we replace ζ_a by ϕ_a, ζ_b by r and $-\psi_b$ by $r\phi_b$ we get the same protocol.
3. The two parties would like to compute $\zeta\phi(N)$. What they actually compute is $\alpha \stackrel{\triangle}{=} (\psi_a+\psi_b)(\phi_a+\phi_b)$. The result is either exactly $\zeta\phi(N)$ or $(\zeta+e)\phi(N)$. The computation is carried out similarly to the computation of N in subsection 4.1. The ring used is $\mathcal{Z}_k$ where $k > 4e \cdot 2^\sigma$. We modify the protocol in two ways. The first modification is that α is not revealed to Alice but remains split additively over $\mathcal{Z}_k$ among the two players. In other words they perform step 1 of the protocol in subsection 4.1 and additively share $\psi_a\phi_b + \psi_b\phi_a$. Alice adds $\psi_a\phi_a$ to her share and Bob adds $\psi_b\phi_b$ to his.
   The sum of the two shares *over the integers* is either α or $\alpha + k$. The second option is unacceptable (unlike the two possibilities for the value of α). In order to forestall this problem we introduce a second modification. The sharing of α results in Alice holding $\alpha + y' \bmod k$ and Bob holding $-y' \bmod k$. Furthermore Bob selects y' by his random choices. We require Bob to make those choices subject to the condition that $y' < k/2$. Since y' is not completely random, Alice might gain a slight advantage. However, that advantage is at most a single bit of knowledge about $\phi(N)$ (which can be guessed in any attempt to discover $\phi(N)$).
4. Alice sets $d_a \stackrel{\triangle}{=} \lceil(\alpha + y')/e\rceil$ and Bob sets $d_b \stackrel{\triangle}{=} \lfloor(-y' + 1)/e\rfloor$ (these calculations are over the integers). Hence, either $d_a + d_b = (\zeta\phi(N) + 1)/e$ or $d_a + d_b = (\zeta\phi(N) + 1)/e + \phi(N)$.

7 Improvements

In this section we suggest some efficiency improvements.

Off-line preprocessing: The performance of the protocol based on Benaloh's encryption can be significantly improved by some off-line preparation. Obviously, for any t used as a modulus in the protocol a suitable encryption system has to be constructed (i.e a suitable $m = pq$ has to be found, a table of all values $T_M = y^{M\phi(m)/t} \bmod m$ has to be computed etc.).

Further improvement of the online communication and computational complexity can be attained at the cost of some space and off-line computation. Instead of constructing a separate encryption system for t_1 and t_2, Alice constructs a single system for $t = t_1t_2$. The lookup table needed for decryption is formed as follows. Alice computes $T_M = y^{M\phi(m)/t} \bmod m$ for $M = 0, \ldots, t_1 - 1$ and $T_{\bar{M}} = y^{\bar{M}t_1\phi(m)/t} \bmod m$ for $\bar{M} = 0, \ldots, t_2 - 1$. The entries of the table are

obtained by calculating $T_M T_{\bar{M}} \bmod m$ for every pair $M, \bar{M}$. Constructing this table takes more time than constructing the two separate tables for t_1, t_2. The additional time is bounded by the time required to compute $t_2(t_1 + \log t_1 \log t_2)$ modular multiplications over $\mathcal{Z}_m$ (computing $T_{\bar{M}}$ involves $\log t_1 \log t_2$ multiplications in comparison with $\log t_2$ multiplications in the original table).

The size of the table is $t \log m$ (slightly more than $t\sigma$). This figure which might be prohibitive for large t can be significantly reduced. After computing every entry in the table it is possible by using perfect hashing [14] to efficiently generate a 1-to-1 function h from the entries of the table to $0, \ldots, 3t-1$. A new table is now constructed in which instead of original entry T_M an entry $(h(T_M), M)$ is stored. Decryption of z is performed by finding the entry holding $h(z^{\phi(m)/t})$ mod m and reading the corresponding M. The size of the stored table is $2t \log t$. As an example of the reduction in space complexity consider the case $t = 3 \cdot 751 = 2253$. The original table requires more than 2^{21} bits while the hashing table requires less than 2^{16} bits.

It is straightforward to use $t = t_1 t_2$ instead of t_1 and t_2 separately in subsections 4.3 and 5.2. The protocols in both subsections remain almost without change apart from omitting the sub-protocols for t_1 and t_2 and adding a sub-protocol for t. In subsection 5.2 it is not enough to check whether $N \equiv 0 \bmod t$. It is necessary to retain the two tests of $N \bmod t_1$ and $N \bmod t_2$.

Note that here we need the stronger *higher residuosity* intractability assumption replaces the *prime residuosity* assumption.

Alternative computation of d: The last part of generating RSA keys is constructing the private key. Using Benaloh's encryption we can sometimes improve the computation and communication complexity of the construction in comparison with the results of section 6. The improvement is possible if the parameter t of a Benaloh encryption can be set to e (that is, the homomorphism is modulo e) so that efficient decryption is possible. Therefore, e has to be a product of "small" primes, see [3]. The protocol for generating and sharing d is a combination of the protocols of subsection 4.3 and section 6. We leave the details to the full version of the paper.

8 Performance

The most resource consuming part of our protocol, in terms of computation and communication, is the computation of N together with trial divisions. We use trial divisions because of the following result by DeBruijn [10]. If a random $\sigma/2$ bit integer passes the trial divisions for all primes less than B then asymptotically:

$$\Pr[p \text{ prime} \mid p \not\equiv 0 \bmod p_i, \forall p_i \leq B] = 5.14 \frac{\ln B}{\sigma}(1 + o(\frac{2}{\sigma})).$$

We focus on the performance of the more efficient version of our protocol, using homomorphic encryption. We also assume that the off-line preprocessing suggested in section 7 is used. Let j', j be defined as in section 4.3. We pair off the first j' primes and prepare encryption systems for products of such pairs (as

in section 7.) The number of exponentiations (decryptions) needed to obtain one N is on average about $j + 3 - j'/2$. The probability that this N is a product of two primes is approximately $(5.14 \ln p_{j'}/\sigma)^2$.

Another obvious optimization is to divide the decryptions between the two parties evenly. In other words for half the primes the first party plays Alice and for the other half they switch.

If $\sigma = 1024$ and we pair off the first $j' = 76$ primes the running time of our protocol is (by a rough estimate) less than 10 times the running time of the Boneh-Franklin protocol. The communication complexity is (again very roughly) 42MB. If the paricipants are willing to improve the online communication complexity in return for space and pair off the other $j - j'$ needed to compute N the communication complexity is reduced to about 29MB.

Open problem: Boneh and Franklin show in [4] how to test whether N is a product of two primes, where both parties hold N. It would be interesting to devise a distributed test to check whether N is a product of two primes if Alice holds N, P_a, Q_a and Bob only has his private shares P_b, Q_b. The motivation is that in the oblivious transfer and oblivious polynomial evaluation protocols we presented P_a, Q_a will have to be selected only once. Thus the number of oblivious transfers in the whole protocol is reduced to the number required for computing a single candidate N.

Acknowledgements: I'd like to thank the anonymous referees for several enlightening comments, Benny Chor for suggesting this area of research, Josh Benaloh for his help and Yuval Ishai for his usual insights.

References

1. M. Ben-OR, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proc. of the 20th Annu. ACM Symp. on the Theory of Computing*, pages 1–10, 1988.
2. J. Benaloh. *Verifiable Secret-Ballot Elections.* PhD thesis, Yale University, 1987.
3. J. Benaloh. Dense probabilstic encryption. In *Proc. of the Workshop on Selected Areas of Cryptography*, pages 120–128, May 1994.
4. D. Boneh and M. Franklin. Efficient generation of shared rsa keys. In *Advances in Cryptology - CRYPTO '97*, pages 425–439. Springer-Verlag, 1997. The full version appears on the web at theory.stanford.edu/ dabo/pubs.html.
5. C. Cachin, S. Micali, and M. Stadler. Computationally private information retrieval with polylogarithmic communication. In *Advances in Cryptology - EUROCRYPT '99*, pages 402–414, 1999.
6. D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols(extended abstract). In *Proc. of the 20th Annu. ACM Symp. on the Theory of Computing*, pages 11–19, 1988.
7. B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. In *Proc. of the 36th Annu. IEEE Symp. on Foundations of Computer Science*, pages 41–50, 1995.
8. C. Cocks. Split knowledge generation of rsa parameters. In M. Darnell, editor, *Cryptography and Coding, 6th IMA international conference*, pages 89–95. Springer-verlag, December 1997.

9. C. Cocks. Split generation of rsa parameters with multiple participants, 1998. On-line version at www.cesg.gov.uk/downlds/math/rsa2.pdf.
10. N. DeBruijn. On the number of uncancelled elements in the sieve of eratosthenes. *Proc. Neder. Akad.*, 53:803–812, 1950. Reviewed in Leveque Reviews in Number Theory, Vol. 4, Section N-28, p. 221.
11. A. DeSantis, Y. Desmedt, Y. Frankel, and M. Yung. How to share a function securly. In *Proc. of the 26th Annu. ACM Symp. on the Theory of Computing*, pages 522–533, 1994.
12. Y. Desmedt. Threshold cryptography. *European Transactions on Telecommunications and Related Technologies*, 5(4):35–43, July-August 1994.
13. Y. Frankel, P. D. MacKenzie, and M. Yung. Robust efficient distributed rsa-key generation. In *Proc. of the 30th Annu. ACM Symp. on the Theory of Computing*, pages 663–672, 1998.
14. M. Fredman, J. Komlos, and E. Szemeredi. Storing a sparse table in o(1) worst case access time. *Journal of the ACM*, 31:538–544, 1984.
15. Y. Gertner, Y. Ishai, E. Kushilevitz, and T. Malkin. Protecting data privacy in private information retrieval schemes. In *Proc. of the 30th Annu. ACM Symp. on the Theory of Computing*, pages 151–160, 1998.
16. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game (extended abstract). In *Proc. of the 19th Annu. ACM Symp. on the Theory of Computing*, pages 218–229, 1987.
17. S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and systems science*, 28:270–299, 1984.
18. E. Kushilevitz and R. Ostrovsky. Single-database computationally private information retrieval. In *Proc. of the 38th Annu. IEEE Symp. on Foundations of Computer Science*, pages 364–373, 1997.
19. M. Naor and B. Pinkas. Oblivious transfer and polynomial evaluation. In *Proc. of the 31th Annu. ACM Symp. on the Theory of Computing*, pages 245–254, 1999.
20. Guillaume Poupard and Jacques Stern. Generation of shared rsa keys by two parties. In *ASIACRYPT'98*, pages 11–24. Springer-Verlag LNCS 1514, 1998.
21. J. P. Stern. A new and efficient all–or-nothing disclosure of secrets protocol. In *ASIACRYPT'98*, pages 357–371. Springer-Verlag, 1998.
22. A.C. Yao. How to generate and exchange secrets. In *Proc. of the 27th Annu. IEEE Symp. on Foundations of Computer Science*, pages 162–167. IEEE Press, 1986.