J. Cryptology (1995) 8: 101–114

Journal of
CRYPTOLOGY

Short RSA Keys and Their Generation

Scott A. Vanstone and Robert J. Zuccherato
Department of Combinatorics and Optimization, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1

Communicated by Johannes Buchmann

Received 22 July 1993 and revised 4 January 1994

Abstract. This paper deals with the problem of generating RSA moduli having a predetermined set of bits. It would appear to be of practical interest if one could construct their modulus so that, for example, some of the bits are the ASCII representation of their identification information (i.e., name, address, etc.). This could lead to a savings in both bandwidth for data transmission and storage. A theoretical question which arises in connection with this is to determine the maximum number of bits which can be specified so that the modulus can be determined in polynomial time and, of course, security is maintained.

Key words. RSA, Public-key cryptography, Prime number generation, Integer factorization, Security.

1. Introduction

The most well known and accepted public-key cryptosystems are those based on discrete logarithms in finite groups and integer factorization. In particular, the Diffie–Hellman key exchange [8], the El Gamal protocol [9] in $\mathbb{Z}_p^*$, p a prime, and the RSA system [27] for modulus $n = p \cdot q$, where p and q are primes, have been implemented worldwide. One disadvantage of these systems is that p and n must be relatively large (at least 512 bits) to attain an adequate level of security. For this reason researchers have looked for public-key schemes which reduce the size of the public key. An attractive and promising system is the Diffie–Hellman and El Gamal protocols defined in the group associated with the points on an elliptic curve over a finite field (see, for example, [1], [10], and [19]). It appears that a 155-bit elliptic curve scheme gives comparable security to a 1024-bit RSA modulus. Nevertheless, RSA remains a very viable and practical encryption and signing process. The purpose of this paper is to describe a

method for reducing the storage requirement of RSA public moduli without compromising security.

The problem of interest to us is to determine how many bits of a modulus of the form $n = p \cdot q$ can be specified. For n a 1024-bit number the following argument suggests that most of the bits could be prescribed. The number of 512-bit prime numbers is about $2^{512}/\log(2^{512}) - 2^{511}/\log(2^{511}) = (2^{502} \cdot 510)/(511 \cdot \log(2))$. Thus, the number of 1024-bit numbers of the form $p \cdot q$, where p and q are 512-bit primes, is about $(2^{502}/\log(2))^2/2$. Now the number of 1024-bit numbers is 2^{1023} and so the fraction of 1024-bit numbers of the form $p \cdot q$, where p and q are 512-bit primes, is roughly $2^{-19}/\log^2(2) \approx 4 \times 10^{-6}$.

These computations suggest that if we leave about 17 bits of freedom, then we should be able to find a 1024-bit integer n, the product of two primes, with over 1000 of the bits specified. The real question of practical importance then is how many bits can we specify in a modulus $n = p \cdot q$ so that p and q can be found in time proportional to a polynomial in $\log(n)$? To this end we propose several techniques and analyze their security. If the modulus n is an m-bit number we describe how to specify t bits of the number where t is suitably bounded. If the t bits can be random bits the situation is somewhat simplified. This scenario may arise where a group of users use the same t random bits and hence only $m - t$ bits need be stored for each user and one copy of the t random bits for the entire group. There may be situations where a user would like the t bits to be a binary representation of their user ID and other publicly available information. This situation can also be implemented by adjoining a small number of random bits after the information. We show that up to $m/2$ of the bits can always be specified, but specifying $m/2$ of the bits comes at the expense of longer key generation (i.e., prime generation). The methods discussed give various degrees of dependence between the primes p and q.

At present we see no way of specifying up to half of the bits for a public key used for the Diffie–Hellman and El Gamal protocols. One possibility is to specify t bits of the public key and then try about 2^t values of a until α^a has the specified bits. This is clearly impractical for t being very large.

The remainder of this paper is organized as follows. In Section 2 we describe a method for generating the primes after specifying t high-order bits in a 1024-bit RSA modulus. We have chosen 1024 as a reference bit length in order to make the description of the techniques as concrete as possible. We also give a generalization to a modulus n of arbitrary length. Section 3 describes a method which specifies half of the bits in the modulus at the expense of much longer key generation time. In Section 4 we give a compromise method using ideas from the previous two sections. Section 5 focuses on various scenarios and complications of specifying bits in the modulus. Maurer's method for generating primes is described in Section 6 and applied to our technique. In Section 7 we describe a method for specifying t low-order bits of the modulus while in Section 8 we explore the question of how many bits of the modulus can be specified. Section 9 discusses running times and Section 10 describes our implementations. Security issues are considered in Section 11 and Section 12 is our conclusion.

2. Specifying the First t Bits

2.1. *For $m = 1024$*

We first describe a method for specifying the first t bits of $n = p \cdot q$ with specific attention paid to the case where n is 1024 bits. In this scheme p and q are effectively chosen independently.

Let β be a fixed number of length $t = 2 \cdot k$ bits, and assume that β factors as $\beta = f_1 \cdot f_2$, where f_1 and f_2 are each of length k bits. We show in Section 5 how to choose an appropriate β. We can then let our primes p and q be of the form $p = 2^{512-k} \cdot f_1 + a_1$ and $q = 2^{512-k} \cdot f_2 + a_2$, where a_1 and a_2 are numbers of length l bits. When we multiply to get n we get

$$p \cdot q = 2^{1024-2 \cdot k} \cdot f_1 \cdot f_2 + 2^{512-k}(f_1 \cdot a_2 + f_2 \cdot a_1) + a_1 \cdot a_2.$$

It is easy to see that if the last two terms in the sum are less than $1024 - 2 \cdot k$ bits, then the first $2 \cdot k$ bits are exactly the number β. To guarantee this we need $512 - k + k + l + 2 < 1024 - 2 \cdot k$ or $l + 2 \cdot k < 510$.

To guarantee security we need to make sure that, given n, f_1, and f_2, it is difficult to determine p and q, or, equivalently, a_1 and a_2. It is thus necessary to require $a_1 \cdot a_2$ to be at least $512 - k$ bits. We wish to prevent a brute-force attack on a_1 and a_2. This would involve trying every possible combination of a_1 and a_2 until the corresponding values of p and q multiplied to give n. Thus in order to prevent this type of attack we require the overlap between $a_1 \cdot a_2$ and the $2^{512-k}(f_1 \cdot a_2 + f_2 \cdot a_1)$ term to be at least 60 bits. We therefore need $2 \cdot l > 512 - k + 60$ or $2 \cdot l + k > 572$.

The above two inequalities force a largest possible value for k of 148 and thus $t = 296$. This gives a corresponding value of 213 for l. We can therefore specify the first 296 bits of n while not compromising the security of our system.

2.2. *Producing the Primes*

We wish to produce primes of the form $p = 2^{512-k} \cdot f + a$ where f has a length of k bits and a is an l-bit number. To produce such a prime we choose random l-bit numbers a and test $p = 2^{512-k} \cdot f + a$ for primality using the Miller–Rabin primality test (see [20] and [25]). If p is not prime, we return to choosing suitable a's. To avoid a brute-force attack on n we choose l large enough to guarantee suitable interaction between the terms $a_1 \cdot a_2$ and $2^{512-k}(f_1 \cdot a_2 + f_2 \cdot a_1)$.

2.3. *Keys of Arbitrary Size*

If n is required to be a size other than 1024 bits, say m bits, then it is easy to verify (in the manner of above) that the following two inequalities must hold. If p and q are $m/2$ bits long, then $l + 2 \cdot k < m/2 - 2$ and $2 \cdot l + k > m/2 + \varepsilon(n)$, where $\varepsilon(n)$ is the number of bits of overlap required to make brute-force search less effective than other general-purpose factoring algorithms (i.e., we

should choose $\varepsilon(n)$ so that $2^{\varepsilon(n)} \approx O(e^{\sqrt{\ln(n)\cdot\ln(\ln(n))}})$. The largest possible value of t can then be taken as the maximum number of bits that can be specified.

3. Specifying Half the Bits

This method allows us to specify half of the bits of n but now q is completely dependent on the choice of p.

3.1. *For m = 1024*

Let α and β be given 256-bit numbers that are public knowledge. Let β be such that it factors as $\beta = f_1 \cdot f_2$ for 128-bit numbers f_1 and f_2. We then (privately) create primes p and q such that $p = 2^{384} \cdot f_1 + a_1$ and $q = 2^{384} \cdot f_2 + a_2$, where a_1 and a_2 are 248-bit numbers with $a_1 \cdot a_2 \equiv \alpha \pmod{2^{256}}$. Then this gives

$$n = 2^{768} \cdot f_1 \cdot f_2 + 2^{384}(f_1 \cdot a_2 + f_2 \cdot a_1) + a_1 \cdot a_2.$$

It is easily seen that $2^{384}(f_1 \cdot a_2 + f_2 \cdot a_1) + a_1 \cdot a_2$ is at most 763 bits so there is no carry into $2^{768} \cdot f_1 \cdot f_2$, and the high-order 256 bits of n are the number β. Also since $a_1 \cdot a_2 \equiv \alpha \pmod{2^{256}}$, the low-order 256 bits are the number α. The middle 512 bits of n can then be communicated, stored, and the full number can be retrieved.

To guarantee security we must be sure that given α, n, f_1, and f_2 finding a_1 and a_2 is a difficult problem. Since $a_1 \cdot a_2$ is a 496-bit number of which 384 bits are known from the form of n, this leaves 112 bits of flexibility, or 2^{112} possibilities for $a_1 \cdot a_2$. Exhaustive search techniques would therefore be infeasible by current technology.

3.2. *Producing the Primes*

To produce the primes p and q of the desired form, 248-bit odd numbers, a_1, should be randomly chosen until a p is found that is prime. The Extended Euclidean Algorithm is then used to produce an a_2 such that $a_1 \cdot a_2 \equiv \alpha \pmod{2^{256}}$. If a_2 is less than 2^{248} and q is prime, then the algorithm terminates, if not then return to choosing random values for a_1. There is a problem with this algorithm; obtaining a value for a_2 that is less than 2^{248}.

To deal with this problem we change the parameters in the following way. The numbers f_1 and f_2 are still 128-bit numbers, but now $p = 2^{376} \cdot f_1 + a_1$ and $q = 2^{392} \cdot f_2 + a_2$, where a_1 is a 240-bit number and a_2 is 256 bits. Then we get

$$n = 2^{768} \cdot f_1 \cdot f_2 + 2^{392} \cdot f_2 \cdot a_1 + 2^{376} \cdot f_1 \cdot a_2 + a_1 \cdot a_2.$$

Now $2^{392} \cdot f_2 \cdot a_1 + 2^{376} \cdot f_1 \cdot a_2 + a_1 \cdot a_2$ is at most 763 bits, so there is no carry over into $2^{768} \cdot f_1 \cdot f_2$ and $a_1 \cdot a_2$ is a 496-bit number with only 376 bits left unhidden, so there are 2^{120} possibilities for this value. This actually gives an improvement of the level of security obtained before. The algorithm can now be

changed to choose a_1 to be 240 bits and not to require a_2 to be less than 2^{248}. The primes p and q are now 504 and 520 bits, respectively.

3.3. *Keys of Arbitrary Size*

In many applications the size of n may be required to be larger or smaller than 1024 bits. To deal with this we present modified parameters that handle the more general situation. Assume we want our n to be $8 \cdot k$ bits long, where $k > 8$. Then p and q should each be approximately $4 \cdot k$ bits in length. Let α and β be specified numbers as before, of length $2 \cdot k$. Factor β as $f_1 \cdot f_2$, where f_1 and f_2 are k-bit numbers. We then require a_1 to be a $(2 \cdot k - 8)$-bit number and a_2 to be $2 \cdot k$ bits, so that $p = 2^{k_1} \cdot f_1 + a_1$ and $q = 2^{k_2} \cdot f_2 + a_2$, where $k_1 = 3 \cdot k - 4$ and $k_2 = 3 \cdot k + 4$. Choose a_1 and a_2 as before.

This gives

$$n = 2^{k_1+k_2} \cdot f_1 \cdot f_2 + 2^{k_2} \cdot f_2 \cdot a_1 + 2^{k_1} \cdot f_1 \cdot a_2 + a_1 \cdot a_2,$$

with β being the first $2 \cdot k$ bits and α being the last $2 \cdot k$ bits. There is an overlap of $k - 4$ bits between the $a_1 \cdot a_2$ term and the $2^{k_1} \cdot f_1 \cdot a_2$ term, so this product still remains hidden and hence exhaustive search techniques remain infeasible for large k.

4. A Compromise

There are certain advantages and disadvantages to both of the schemes described thus far. In Section 2 the primes p and q could be chosen independently. This resulted in a good running time but at the expense of only being able to specify about one-quarter of the bits. In Section 3 half the bits could be specified but the primes were completely dependent on each other resulting in a very poor running time. In this section we present a compromise scheme which has some of the advantages of both techniques.

4.1. *For m = 1024*

Let β be a 256-bit number such that it factors as $\beta = f_1 \cdot f_2$ for 128-bit numbers f_1 and f_2. Let α be a u-bit number, where $u < 248$. Our required primes will be of the form $p = 2^{384} \cdot f_1 + a_1$ and $q = 2^{384} \cdot f_2 + a_2$, where a_1 and a_2 are 248-bit numbers. We require the final u bits of n to be exactly the number α so we want $a_1 \cdot a_2 \equiv \alpha \pmod{2^u}$. Thus, as before, we get

$$n = 2^{768} \cdot f_1 \cdot f_2 + 2^{384}(f_1 \cdot a_2 + f_2 \cdot a_1) + a_1 \cdot a_2.$$

Again, it is easy to see that the top 256 bits of n are the number β and the bottom u bits are α.

As was shown in Section 3.1, there is an overlap of 112 bits between the last two terms of n. Therefore, obtaining the values a_1 and a_2, and thus obtaining the primes, is a difficult problem.

4.2. *Producing the Primes*

We produce the desired primes in the following way. As before, randomly choose 248-bit numbers a_1 and test $p = 2^{384} \cdot f_1 + a_1$ for primality. If p is prime, then by using the Extended Euclidean Algorithm we can produce a u-bit number c such that $a_1 \cdot c \equiv \alpha \pmod{2^u}$. Random $(248 - u)$-bit numbers b can then be chosen to produce $a_2 = b \cdot 2^u + c$ until the corresponding value of q is prime. Notice that now $a_1 \cdot a_2 \equiv \alpha \pmod{2^u}$.

When searching for q, we will be searching a particular residue class modulo 2^u. The distribution of primes in this class is about the same as for any interval of about the same size [14]. We must therefore choose the parameter u such that there are enough possibilities for b to guarantee, with high probability, the existence of a prime of this particular form. In this interval we would expect there to be approximately $2^{(248-u)}/\ln(2^{512})$ primes. It then follows that u should be chosen to be somewhere between 228 and 218 to give between about 20 and 30 bits of freedom to find a suitable b.

4.3. *Keys of Arbitrary Size*

Clearly, the analysis done in Section 3.3 also applies here. The size of α must now be u bits, and this size should be chosen so that a prime is likely to be found for any value of c.

5. How To Choose β

As we have seen before, β is a number of length $2 \cdot k$ bits which factors as $f_1 \cdot f_2$ for k-bit numbers f_1 and f_2. This is the case when the first $t = 2 \cdot k$ bits of n are being specified and when half the bits are being specified. Since there are no conditions on α, this value can be chosen to be any convenient number of the appropriate size.

On the other hand, β must be chosen so that it factors appropriately. If these t bits can be random bits there is no problem. Simply choose two numbers f_1 and f_2 of length k bits and multiply them together to get a $(2 \cdot k)$-bit number β. In this situation a number of people can all use the same β (and α if needed) thus reducing the amount of memory needed to store all of their public keys.

In some situations, however, the user may want these t bits to be a binary representation of their user ID and other publicly available information. This situation is slightly more complicated because this number may or may not factor as the product of two numbers of equal size. This problem can be overcome by leaving some random bits at the low end of β to allow more choice for the factors.

We need β to factor into two numbers about the same size as its square root. To accomplish this, we allow a difference of d bits in the size of the factors f_1 and f_2. Thus, the parameters k_1 and k_2 will have to be similarly changed to give two primes p and q of equal size. This can be done easily with no change in security or prime generation. We need an estimation of $r(u)$, the probability

that a random integer, N, has no prime factor greater than $N^{1/u}$. We use $r(u) \approx e^{-u \cdot \ln u}$ which is a crude approximation to the value given in [5] (see also [21]). If every prime factor of β has fewer than d bits, then it will factor into two integers differing in size by d bits. This is not the only way to guarantee that β will factor correctly but it is sufficient. We use it to get an upper bound on the number of random bits needed. If β is 256 bits, as is the case when n is 1024 bits, then this probability is approximately $e^{-(256/d)\cdot\ln(256/d)}$. It is then easy to see that if we allow a difference of 10 bits between the factors of β we will only require about 47 random bits to guarantee with high probability the existence of factors of the required type. Thus, the first 209 bits of β can be the publicly available information, while the final 47 bits can be varied until β factors as desired. Experimental results confirm that with 47 bits of randomness, β should factor as required with very little difficulty.

It may also be desirable to allow f_1 and f_2 to be the publicly available information. Only one multiplication would then be required to obtain β and thus the public modulus. This does not compromise security as f_1 and f_2 are already public.

6. Fast Generation of Primes

In this section we present Maurer's method [18] for fast generation of primes and describe how it is well suited to generating primes of the form required in the previous sections.

6.1. *Background*

In [18] Maurer describes how to generate provable primes in time only about one-third greater than the expected running time required for generating a pseudo-prime that passes the Miller–Rabin test for only one base. We briefly sketch his method here.

The following two lemmas form the basis for the algorithm. Lemma 1 is a special case of a theorem due to Pocklington [22] (see also [4] or [13]).

Lemma 1. *Let $n = 2RF + 1$, where F has a known prime factorization $F = q_1^{\beta_1} \cdot q_2^{\beta_2} \cdot \cdots \cdot q_r^{\beta_r}$. If there is an integer a satisfying*

$$a^{n-1} \equiv 1 \pmod{n}$$

and

$$\gcd(a^{(n-1)/q_j} - 1, n) = 1$$

for $j = 1, \ldots, r$, then each prime factor p of n is of the form $p = mF + 1$ for some integer $m \geq 1$. Moreover, if $F > \sqrt{n}$, or if $F > R$, then n is prime.

Lemma 2. *Let $p = 2RF + 1$ be a prime with $F = \prod_{j=1}^{r} q_j^{\beta_j}$, $F > R$ and $\gcd(2R, F) = 1$, where $q_1, \ldots, q_r$ are distinct primes. Then the probability that a randomly selected base $a \in \mathbb{Z}_p^*$ is successful in proving the primality of p by Lemma 1 is equal to $\varphi(F)/F$ which is at least $1 - \sum_{j=1}^{r} 1/q_j$.*

The primes are constructed recursively by first constructing small primes $q_1, \ldots, q_r$ and then picking random R until $p = 2RF + 1$ can be proved prime by an appropriate choice of the base a. Lemma 2 shows that if p is indeed prime, then finding a base a that is successful in proving this fact is easy. Also, if p is composite and does not contain a small prime factor that will reveal itself in trial division, then virtually every base a will satisfy $a^{p-1} \not\equiv 1 \pmod{p}$, and will be a witness for the compositeness of p, unless p is of a very special form (see [3] and [6]).

By choosing q_i of the appropriate size the prime p can be chosen to lie within a certain interval $[P_1, P_2]$. Again the reader is referred to [18] for a detailed description of the algorithm.

6.2. *Using This Method*

When we specify only the first $t = 2 \cdot k$ bits of the product n, the choice of p is independent of the choice of q (actually, the choice of a_1 is independent of the choice of a_2). To find a prime of the form $p = 2^{m/2-k} \cdot f + a$, where f is k bits long and a is l bits, simply produce a prime p by the above method in the interval $[2^{m/2-k} \cdot f, 2^{m/2-k} \cdot f + 2^l]$. This prime is of the desired form.

In the other schemes the primes p and q are at least partially dependent on each other and hence the generation of these primes is more complex and time consuming (see Section 9). The first prime p can be produced as described above, but then q is at least partially determined. As mentioned in Sections 3 and 4, the Extended Euclidean Algorithm must be used to obtain the appropriate value for a_2 and thus get q. The primality of q can then be checked using the Miller–Rabin test. If q is not prime, either a new p must be chosen or a new b must be chosen to give a new q.

7. Specifying the Last t Bits

A fourth way to specify some of the bits of an m-bit modulus n involves specifying the last t bits for some $t < m/2$. Let α be a known t-bit number. If we then let p and q be primes of size approximately $m/2$ bits with $p \cdot q \equiv \alpha \pmod{2^t}$, then the last t bits of n will be exactly α.

To generate these primes we first choose any random $(m/2)$-bit prime p. Using the Extended Euclidean Algorithm we can then obtain c such that $p \cdot c \equiv \alpha \pmod{2^t}$. If we then choose a random $(m/2 - t)$-bit number b such that $q = 2^t \cdot b + c$ is prime, it is easy to see that $p \cdot q \equiv \alpha \pmod{2^t}$ and n is the desired modulus.

As mentioned in Section 4.2 the distribution of primes in any congruence class is about the same as for any interval of about the same size. Thus we would expect there to be about $2^{512-t}/\ln(2^{512})$ primes in the congruence class searched. It seems reasonable then that if t is chosen to be somewhere between 492 and 482 we would expect to find a prime, q, of the desired form without much difficulty. If this fails we can then pick a new p and repeat the calculations.

8. How Many Bits Can Be Specified?

In this section we explore the question of how many bits of the modulus can be specified so that p and q can be found in time proportional to a polynomial in $\log(n)$. To do this we look at the problem in its most general form and introduce a new technique for factoring β where f_1 and f_2 are not specified ahead of time. This new technique does not add anything to any of the other schemes introduced in this paper, but is simply a generalization of the different methods presented and is included here as another way of looking at the problem.

Our modulus is, as usual, $n = p \cdot q$, where p and q are of the form $p = 2^c \cdot f_1 + a_1$ and $q = 2^c \cdot f_2 + a_2$. Here a_1 and a_2 are l-bit numbers while f_1 and f_2 are each k bits. Again we specify α and β, which are c- and t-bit numbers, respectively, ahead of time to be public. Now f_1 and f_2 are such that $f_1 \cdot f_2 = 2^u \cdot \beta + \delta$, where δ is a u-bit number. We determine f_1 and f_2 by varying δ and trying to factor $2^u \cdot \beta + \delta$ appropriately. This should be a relatively simple task as long as u is sufficiently large.

Now

$$n = p \cdot q = 2^{2 \cdot c} \cdot f_1 \cdot f_2 + 2^c \cdot (f_1 \cdot a_2 + f_2 \cdot a_1) + a_1 \cdot a_2$$
$$= 2^{2 \cdot c}(\beta \cdot 2^u + \delta) + 2^c(f_1 \cdot a_2 + f_2 \cdot a_1) + a_1 \cdot a_2.$$

In order that β appear as the first t bits we need the following constraints.

If m is the total number of bits in n then:

1. $2 \cdot c + 2 \cdot k = m$.
2. $c + k + l < m - t$ to avoid rippling into the high-order t bits.
3. $2 \cdot l > c + s$ to ensure at least s bits of rippling into the middle term.
4. $c + l + k > 2 \cdot c + s$ to ensure at least s bits of rippling into the first term.

Simplifying 1–4 we get:

2′. $l + t < m/2$.
3′. $2 \cdot l - c > s$.
4′. $l - 2 \cdot c > s - m/2$.

In order to determine the maximum number of bits that can be specified using this scheme we want to maximize the size of α and β, or, equivalently, $t + c$. We require the solution of the following integer linear program. (We assume that m and s are given and fixed.)

$$\textbf{Maximize} \quad B = t + c,$$
$$\textbf{subject to} \quad l + t < m/2,$$
$$c - 2 \cdot l < -s,$$
$$2 \cdot c - l < m/2 - s,$$

and all variables are nonnegative.

As simply a linear program the optimal value is given by $t = (m - s)/3$, $l = (m + 2 \cdot s)/6$, $c = (m - s)/3$, and $B = t + c = 2(m - s)/3$. Solving this

program for $m = 1024$ and $s = 64$, we get $t = 320$, $c = 320$, $l = 192$, and $k = 192$ allowing us to specify 640 bits of a 1024-bit modulus.

How many bits of p or q do we reveal by this method? Suppose, for any choice of δ, f_1 and f_2 both have a fixed number x of high-order bits. Consider that greater than one in two numbers with $2 \cdot k$ (the number of bits in $f_1 \cdot f_2$) bits factors appropriately. We have only $2^{2\cdot(k-x)}$ numbers.

The only problem with specifying this many bits is in generating the primes. Notice that α is $c = 320$ bits long and both a_1 and a_2 are $l = 192$ bits long. Thus for every a_1 that is chosen to make p prime, after applying the Extended Euclidean Algorithm to find a_2, we may get an a_2 of 320 bits. This choice for a_2 will obviously not work, as many different values of a_1 must then be tried until we get an a_2 that fits, and then, with high probability, it will not produce a q that is prime. To overcome this problem we add the constraint $c \leq l$ to our integer linear program. After solving the revised integer linear program we get $t = 446$, $c = 65$, $l = 65$, and $B = 511$. Hence this scheme offers no advantage over previously described methods that can specify 512 bits.

9. Running Time

In this section we give heuristic running times of the algorithms used to produce p and q for all three schemes described.

When only the first t bits are being specified, for $(m/2)$-bit primes p and q, the choice of p and q are independent and Maurer's fast prime-generation technique is used. As was shown in [18], the running time for this generation of primes is $O(m^3 \ln(\ln(m)))$ bit operations.

In the other schemes described for $(m/2)$-bit primes p and q the choice of p and q are not independent and hence the running time is slightly worse. We produce p using Maurer's fast prime-generation technique which has running time $O(m^3 \ln(\ln(m)))$ bit operations. We then produce a q and test it for primality, which must be done approximately $\ln(2^m)$ times before a prime q is found. We thus get the running time of producing appropriate primes p and q to be $O(m^3 \ln(\ln(m)) \ln(2^m))$ or $O(m^4 \ln(\ln(m)))$ bit operations.

10. Implementation

We present the results of our implementation of the ideas presented here. We implemented these schemes on a SUN-2 SPARC-station using MAPLE V. A naive prime-generating algorithm was used in which an odd random number of the appropriate size was chosen and then increased by two until it passed the Miller–Rabin primality test with five iterations. The times shown are average times over a number of different trials. All primes produced are 512-bit numbers. We include these results to show that the schemes presented in this paper are feasible.

The average time to produce one prime for a modulus in which the first 296 bits are specified is 205.9 s.

The average time to produce both primes when half the bits of the modulus are specified is 36,866.1 s.

The average time to produce both primes when the first 256 bits and the last 218 bits of the modulus are specified is 437.7 s.

The average time to produce both primes when the last 482 bits of the modulus are specified is 710.6 s.

Notice that for the three schemes in which the choice of q is not totally dependent on p, the time to generate both primes is about the same. When half the bits are specified, and the choice of q is totally dependent on p, the time increases dramatically.

11. Security Considerations

To compare the security of using our specially constructed primes with that of using general primes p and q we must consider the difficulty of factoring

$$n = 2^{k_1+k_2} \cdot f_1 \cdot f_2 + 2^{k_2} \cdot f_2 \cdot a_1 + 2^{k_1} \cdot f_1 \cdot a_2 + a_1 \cdot a_2,$$

where f_1 and f_2 are k-bit numbers, a_1 and a_2 are l_1 and l_2 bits, respectively, and everything is known except for a_1 and a_2. We also know that n factors as $p \cdot q = (2^{k_1} \cdot f_1 + a_1)(2^{k_2} \cdot f_2 + a_2)$.

To make n resistant to a Pollard-$(p-1)$ attack [23], $(p-1)$ and $(q-1)$ should each have at least one prime factor of about 15 digits [7]. The probability that a 512-bit number will have all of its prime factors less than about 15 digits is about 3×10^{-11} [12]. Thus, for all practical purposes, n is resistant to this type of attack. A practical question that arises is how can it be feasibly ensured that *both* $(p-1)$ and $(q-1)$ are divisible by a large prime, particularly in situations where the prime q is tightly constrained by the choice of p?

When specifying the last t bits of a 1024-bit modulus n, this problem can be satisfied as follows. We want $p \cdot q \equiv \alpha \pmod{2^t}$. Assume p has been constructed with a large prime factor. Then let Q be some other large prime (e.g., greater than 15 digits). We produce a q of the form $q = QR + 1$ for some positive integer R. The congruence $p(QR + 1) \equiv \alpha \pmod{2^t}$ can be solved for R. We can then search this equivalence class for R's that will give a prime q. For the other schemes mentioned, we cannot see how to do this feasibly, and leave it as an open question.

The number field sieve [15] factors numbers of the special form $n = r^e \pm s$ for small integers r and s. We cannot see how this applies to our method as the product produced here has no higher probability of being of this special form than a general product of primes.

The quadratic-sieve algorithm [24] is the most efficient general-purpose factoring algorithm known. It cannot, however, factor numbers of 1024 bits, so this attack also appears to be resisted. The elliptic curve factorization method [16] only works on integers with "small" prime factors and so does not apply here.

Can we exploit the special structure of the product to factor n? All that is required is to determine a_1 and a_2. We know $a_1 \cdot a_2 \pmod{\min(2^{k_1}, 2^{k_2})}$, but

little else as the other products are added together. Thus if $l_1 + l_2 > \min(k_1, k_2)$, in particular if $2^{l_1+l_2-\min(k_1,k_2)}$ is of the order of $e^{\sqrt{\ln(n)\cdot\ln(\ln(n))}}$ (the quadratic sieve running time) no advantage will be gained in trying brute force over the general factoring algorithms.

We thank one of the referees for pointing out the following two attacks. The first is as follows. We have $n = pq$, where n is $6k$ bits and p is $3k$ bits. We can now search a k bit range for p. Let p' be a guess at p fixed in the centre of the k-bit range. Now calculate n/p' including k fraction bits. Let $d = p - p'$, then the fraction $n/(p' + d)$ can be expanded in a power series as $n/p' - dn/p'^2 + d^2n/p'^3 - \ldots$. If this is an integer, then it is q. Since $|d| < 2^k$ the term d^2n/p'^3 is $O(2^{-k})$, so we can ignore it. If we calculate $2k$ fraction bits of n/p'^2 we can use an Extended Euclidean Algorithm to calculate possible d's by equating the fractional part of n/p' to that of $d(n/p'^2)$. The calculation is about the same work as expanding $2k$ bits of the fractional part of n/p^2 as a continued fraction.

In our schemes we have n is 1024 bits and p is 512 bits. We can assume the opponent knows all but the low-order 240 bits of p (all of the schemes can be modified so that this is the case). Assume that the opponent guesses the next 70 bits, leaving the low 170 bits to be determined. It would appear that the work to mount this type of attack is not significantly less than that which would be required to factor a 1024-bit number using the number field sieve.

For the second attack notice that our product is of the form $n = (2^{384}f_1 + a_1)(2^{384}f_2 + a_2)$, where f_1 and f_2 are 128 bits and a_1 and a_2 are at least 240 bits. We know that $f_1f_2 = \beta$ and is public knowledge. Now, $f_1f_2n = (2^{384}f_1f_2 + a_1f_2)(2^{384}f_1f_2 + a_2f_1)$ has two nearly equal factors. One can now attempt to factor this product as a difference of squares by writing $z^2 = y^2 - f_1f_2n$ and scanning through possible y's starting at $\sqrt{f_1f_2n}$ until a perfect square is found. This search will take about $z^2/2y$ trials to factor n. Since $z = |(f_1a_2 - f_2a_1)/2|$ is about 368 bits, and y is 640 bits, the scan will take about 2^{95} trials, which does not appear to be feasible.

Recently, Kaliski [11] broke Anderson's RSA trapdoor [2]. The proposed trapdoor is based on a secret value A which is 200 bits long and is used to produce 256-bit primes p of the form $p = r(q, A)A + q$. Here q is a prime less than $\sqrt{A}$, and r is 56 bits long and a function of A and q. Kaliski uses some "unusually good simultaneous Diophantine approximations" and lattice basis reduction to break the trapdoor. The approximations rely on the fact that $q < \sqrt{a}$, however. Our scheme can be compared to this with $r = f_1$, $A = 2^{384}$, and $q = a_1$. Since we do not have $a_1 < 2^{192}$, this technique cannot be used.

Both Maurer [17] and Rivest and Shamir [26] described schemes for factoring n with an oracle. Maurer's technique does not seem to apply here as his oracle answers questions regarding the order of elliptic curves modulo p. Rivest and Shamir, however, describe a method for factoring n if the high-order $m/3$ bits of p are known. In our schemes this number of bits of either factor is never revealed. We do not see how to generalize their attack and apply it to our moduli.

It would appear that this specially constructed n is no easier to factor than a general product of two primes and therefore will not compromise security.

12. Conclusion

In this paper we have given various methods for constructing moduli that are the product of two primes and which have a fixed number of bits predetermined. From this discussion there has emerged a very interesting question which requires further research. We conclude by summarizing it.

Open Question. For a fixed positive integer m, what is the largest value of x such that we can find, in time proportional to a polynomial in m, a modulus $n = p \cdot q$ of m-bits having x bits predetermined, where p and q are distinct primes of about the same size?

Acknowledgment

We would like to thank two anonymous referees for their careful reading of this paper, and for suggesting some useful improvements.

References

[1] G. B. Agnew, R. C. Mullin, and S. A. Vanstone, An implementation of an elliptic curve cryptosystem over $F_{2^{155}}$, *IEEE Journal on Selected Areas in Communications*, Vol. 6, 1993, pp. 3–13.

[2] R. Anderson, A practical RSA trapdoor, *Electronics Letters*, Vol. 29, No. 11, 1993, p. 995.

[3] P. Beauchemin, G. Brassard, C. Crépeau, C. Goutier, and C. Pomerance, The generation of random numbers that are probably prime, *Journal of Cryptology*, Vol. 1, No. 2, 1988, pp. 53–64.

[4] D. M. Bressoud, *Factorization and Primality Testing*, Berlin: Springer-Verlag, 1989.

[5] E. R. Canfield, P. Erdös, and C. Pomerance, On a problem of Oppenheim concerning "Factorisatio Numerorum," *Journal of Number Theory*, Vol. 17, No. 1, Aug. 1983, pp. 1–28.

[6] R. D. Carmichael, On composite numbers P which satisfy the Fermat congruence $a^{P-1} \equiv 1 \pmod{P}$, *American Mathematical Monthly*, Vol. 19, 1912, pp. 22–27.

[7] H. Cohen, *A Course in Computational Algebraic Number Theory*, Berlin: Springer-Verlag, 1993.

[8] W. Diffie and M. E. Hellman, New directions in cryptography, *IEEE Transactions on Information Theory*, Vol. 22, No. 6, 1976, pp. 644–654.

[9] T. El Gamal, A public key cryptosystem and a signature scheme based on discrete logarithms, *IEEE Transactions on Information Theory*, Vol. 31, No. 4, 1985, pp. 469–472.

[10] G. Harper, A. J. Menezes, and S. A. Vanstone, Public key cryptosystems with very small key size, *Advances in Cryptology—EUROCRYPT '92*, Lecture Notes in Computer Science, Vol. 658, Berlin: Springer-Verlag, 1993, pp. 163–173.

[11] B. S. Kaliski, Jr., Anderson's RSA trapdoor can be broken (preprint).

[12] D. E. Knuth and L. T. Pardo, Analysis of a simple factorization algorithm, *Theoretical Computer Science*, Vol. 3, 1976, pp. 321–348.

[13] E. Kranakis, *Primality and Cryptography*, Stuttgart: Teubner; New York: Wiley, 1986.

[14] Ch. J. de la Vallée Poussin, Démonstration simplifiée du théorème de Dirichlet sur la progression arithmétique, *Mémoires Couronnés et autres Mémoires* (8° Ed.), Vol. 53, 1895–96, No. 3, p. 59.

[15] A. K. Lenstra, H. W. Lenstra, Jr., M. S. Manasse, and J. M. Pollard, The number field sieve, *Proceedings of the 22nd ACM Symposium on Theory of Computing*, pp. 464–572, 1990.

[16] H. W. Lenstra, Jr., Factoring with elliptic curves, *Annals of Mathematics*, Vol. 126, 1987, pp. 649–673.

[17] U. M. Maurer, Factoring with an oracle, *Advances in Cryptology—EUROCRYPT '92*, Lecture Notes in Computer Science, Vol. 658, Berlin: Springer-Verlag, pp. 429–436.

[18] U. M. Maurer, Fast generation of prime numbers and secure public-key cryptographic parameters, *Journal of Cryptology* (to appear).

[19] A. Menezes and S. Vanstone, Elliptic curve cryptosystems and their implementation, *Journal of Cryptology*, Vol. 6, No. 4, 1994, pp. 209–224.

[20] G. L. Miller, Riemann's hypothesis and tests for primality, *Journal of Computer and System Sciences*, Vol. 13, No. 3, Dec. 1976, pp. 300–317.

[21] P. C. van Oorschot, A comparison of practical public key cryptosystems based on integer factorization and discrete logarithms, in *Contemporary Cryptology—The Science of Information Integrity*, G. J. Simmons, ed., New York: IEEE Press, 1991.

[22] H. C. Pocklington, The determination of the prime or composite nature of large numbers by Fermat's theorem, *Proceeding of the Cambridge Philosophical Society*, Vol. 18, 1914–1916, pp. 29–30.

[23] J. M. Pollard, Theorems on factorization and primality testing, *Proceedings of the Cambridge Philosophical Society*, Vol. 76, 1974, pp. 521–528.

[24] C. Pomerance, Analysis and comparison of some integer factoring algorithms, in *Computational Methods in Number Theory*, H. W. Lenstra, Jr., and R. Tijdeman, eds., Mathematical Centre Tracts, Vol. 154, Amsterdam: Mathematisch Centrum, 1982, pp. 89–139.

[25] M. O. Rabin, Probabilistic algorithms for testing primality, *Journal of Number Theory*, Vol. 12, 1980, pp. 128–138.

[26] R. L. Rivest and A. Shamir, Efficient factoring based on partial information, *Advances in Cryptology—EUROCRYPT '85*, Lecture Notes in Computer Science, Vol. 219, Berlin: Springer-Verlag, 1986, pp. 31–34.

[27] R. L. Rivest, A. Shamir, and L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, *Communications of the ACM*, Vol. 21, No. 2, 1978, pp. 120–126.