FAST GENERATION OF RANDOM, STRONG RSA PRIMES

Robert D. Silverman

RSA Laboratories

May 17, 1997

Abstract. A number of cryptographic standards currently under development call for the use of *strong* primes in the generation of an RSA key. This paper suggests a fast way of generating random strong primes that also satisfy a number of other cryptographic requirements. The method requires no more time to generate strong primes than it takes to generate random primes.

§1. Introduction.

In section 4.1.2 of the X9.31-1997 standard for public key cryptography there are a number of recommendations regarding the generation of primes that make up an RSA modulus. This paper shall examine these criteria, showing how random primes that satisfy most of the criteria may be quickly generated, and explaining why some criteria can be and should be ignored.

The position of RSA Laboratories is that virtually all of these requirements are unnecessary. The reasons for this are given in [10], which is included as an attachment to this document. We summarize the criteria here:

(1) If e, the public exponent is odd, then e shall be relatively prime to $p-1$ and $q-1$.

This is easily satisfied by choosing e.g. $e = 3$, or $e = 2^{16} + 1$. These are commonly used values. This criterion is necessary in order for encryption to work properly. When constructing the primes p and q, it is easy to ensure that e does not divide $LCM(p-1, q-1)$.

(2) If e is even, then it must be relatively prime to $(p-1)/2$ and $(q-1)/2$, and $p \not\equiv q \bmod 8$.

These criteria are easily satisfied by letting e be twice a prime, and then generating p

Typeset by $\mathcal{AMS}$-TEX

and q so that one of them is $\equiv 3 \bmod 8$ and the other is $\equiv 7 \bmod 8$ with e coprime to $LCM(p-1, q-1)$. These latter conditions are easily satisfied during prime generation and we show how to do so below.

Note: The public exponent e is selected prior to generation of the primes.

(3) The modulus shall have $1024 + 256x$ bits for $x = 0, 1, \ldots$.

As a result, the primes p and q shall then be $512 + 128x$ bits each. The choice of the value of x depends on the level of security required. Larger values of x give greater security. This requirement is a statement reflecting the current state of the art in factoring technology. The fastest method known is the Number Field Sieve. With it, 512-bit moduli are simply not secure today for new applications, 768 bits will not be secure within just a few years, and hence a minimum of 1024 is recommended for banking. Today, one thousand workstations working in parallel can factor a 512-bit modulus in 'about' 1-2 months. A 1024-bit modulus would be about 6 million times as difficult to factor.

(4) p and q shall each pass a probabalistic test where the probability of error is less than 2^{-100}.

One can also use a deterministic primality *proof* such as the Bosma-Cohen-Lenstra algorithm or the Atkins-Goldwasser-Killian algorithm. This simply states that we shall have chosen primes with a high degree of confidence; that we have either chosen a prime using a decision procedure and that the probability that the procedure is in error is less than $2^{-100} \sim 8 \times 10^{-31}$ or that we have a rigorous proof of primality.

(5) $p-1$, $q-1$, $p+1$, and $q+1$ shall each have large prime factors. Unfortunately, the standard does not define 'large'. From the results presented in [8, 10, 12], we suggest that 2^{100} is sufficiently large. This will put $p \pm 1$ factoring attacks well out of computer range. The size of the prime factors (101 bits) of $p \pm 1$ and $q \pm 1$ is much too large for these algorithms to succeed in the lifetime of the universe.

(6) $GCD(p-1, q-1)$ shall be small. The standard does not define 'small', but the method for generating primes satisfies this requirement sufficiently to guard against the relevant attacks (repeat encryption). The argument in section 9 of [10] shows that if r is a large prime factor dividing $LCM(p-1, q-1)$, then either the order of the public exponent

exceeds r [which renders repeat encryption attacks impossible because it requires too much computation] *or* the order of the encrypting exponent is small with probability less than $1/(4r)$. Since $p-1$ and $q-1$ both have prime factors at least 2^{100}, $GCD(p-1, q-1)$ can be no more than $s = \sqrt{N}/2^{100}$ and hence the probability that the public exponent has order less than $k = 2^{100}$ is ks^2/N which is less than 2^{-100}. In practice, the probability will be much lower. This is a worst case analysis.

(7) p/q shall not be near the ratio of two small integers and $|p - q| > 2^{412+128x}$.

Once again, 'near' is undefined, but an exact definition is not needed. The purpose of these requirements is to guard against Fermat and related (i.e. Lehman) factoring algorithms. This condition is easily checked once one has generated two primes simply by subtracting their 4 or 8 highest order bytes and checking that the difference is non-zero. However, the basis for the requirement is that if p and q are too close together, then Fermat's or Lehman's algorithm can factor the modulus with work load equal to $(p+q)/2 - \lfloor\sqrt{pq}\rfloor$. The algorithm works by finding x, y such that $x^2 - y^2 = N$. The work load cited assumes that the attacker will start with an initial guess for x at $\lfloor\sqrt{N}\rfloor$. However, there is no reason why the attacker can't choose some other starting guess for x, (say) z, and the work then becomes $(p+q)/2 - z$. Requiring $p - q$ to be large can not guard against choices for z other than $\lfloor\sqrt{N}\rfloor$, and hence adds no real security to the key. We view this requirement as irrelevant. Similarly, the requirement that p/q shall not be close to the ratio of small integers is also irrelevant. We do note however, that this last requirement derives from an attempt to guard against the Lehman algorithm. If r and s are small [which in this context means less than 100], then $ps - qr$ needs to be less than 2^{64} for the attack to be feasible and the chance of this happening is negligible.

(8) $p - q$ shall have a large prime factor. The reason for this condition is unclear to this author. It also seems to be impossible to satisfy, short of actually factoring $p - q$ or running sufficient trials of ECM to be satisfied that no small factor exists.

(9) There is also a suggestion that some forms of the modulus, such as $N = 2^{64x} \pm c$ will simplify the reduction and require less storage. This seems to have been put in place before

the discovery of the Number Field Sieve. Moduli of this form are readily susceptible now to the special version of NFS and are quite insecure. They should not be used.

§2. Randomly Generating Strong Primes

We shall randomly generate two strong primes, each of size $512+128x$ bits in such a way that their product is $1024+256x$ bits. The procedure for prime generation that is outlined below shall therefore be executed twice; once for p and once for q. The procedure shall refer to p only.

We start by randomly generating a number X of the correct size. Then, we randomly generate 101-bit factors p_1 and p_2 for $p \pm 1$. Using the Chinese Remainder Theorem with p_1 and p_2 we then construct a sequence of candidates, starting at our random point X, for p such that $p_1|p-1$ and $p_2|p+1$. We then remove all candidates divisible by small primes with a sieve. Finally, we test the remaining candidates following the sieve for primality.

2.1 Selection of starting point

Randomly select an integer X in the range

$$[\sqrt{2}\,2^{511+128x}, 2^{512+128x}-1]$$

Our prime p will be selected as the first integer greater than X which satisfies the strong prime requirements. The product, $N=pq$, of two of these randomly chosen primes will produce the public modulus which will have exactly $1024+256x$ bits.

2.2 Selection of large prime factors of $p \pm 1$

Start by randomly generating two 101-bit numbers, y_1 and y_2. Using a sieve procedure we shall generate a sequence of candidates for p_1 and p_2 by starting respectively at y_1 and y_2 and sieving out small primes. This will remove a substantial number of composite numbers that need not be tested for primality. We then test what survives the sieve for primality. These shall be p_1 and p_2.

Starting at each of y_1 and y_2 sieve out all small primes up to 10^5 over the range $[y_1, y_1+5\cdot 10^5]$, and $[y_2, y_2+5\cdot 10^5]$. The limit, 10^5, for the primes with which we sieve is somewhat arbitrary, and is chosen for reasons of performance, rather than security. Any number between 10^3 and 10^6

is acceptable. Similarly, the length of the sieve interval, $5 \cdot 10^5$, is somewhat arbitrary. One can choose any numbers which are convenient for the particular implementation of this procedure as dictated by resources such as the amount of computer memory available. The length of the sieve interval should be several times the largest prime that is sieved. The numbers selected are a good balance between the cost (in time) of the sieve procedure against the cost of testing candidates for primality. See section [1] of the appendix for a full description of a sieve procedure.

The sieve will 'strike out' many of the numbers in the sieve interval. The numbers that are removed are divisible by small primes and hence can not be candidates for primality testing. After sieving, test the remaining numbers in the sieve interval sequentially, starting at y_1, y_2 to see if they are prime. Apply 25 iterations of Miller-Rabin to each candidate. This will results in a chance of error of less than 2^{-100}. One can also rigorously prove they are prime (if desired) by applying the Selfridge improvements to the theorem of Proth, Pocklington, & Lehmer. [11]. This procedure will yield two primes p_1 and p_2 which will be used as the large prime factors of $p-1$ and $p+1$ respectively.

These primes correspond to the $B1$ limit discussed in [10]. It is well beyond computer range to apply the $p \pm 1$ algorithms up to $2^{100} \sim 10^{30}$. In fact, it can't be done within the lifetime of the universe with existing hardware.

2.3 **Searching for a strong prime**

At this point we use the Chinese Remainder Theorem to construct a sequence of integers Y, starting at X such that every integer in the sequence is congruent to 1 mod p_1 and -1 mod p_2. This means that every integer Y in the sequence will have $p_1|Y-1$ and $p_2|Y+1$. That is to say, p_1 and p_2 will be large prime factors of $Y-1$ and $Y+1$. In order to do this one computes

$$R = \left((p_2^{-1}) \bmod p_1\right) \cdot p_2 - \left((p_1^{-1}) \bmod p_2\right) \cdot p_1.$$

One then computes $Y_0 = X + (R - X \bmod p_1 p_2)$. This is the first integer greater than X which is 1 mod p_1 and -1 mod p_2. Starting at Y_0 one now sieves the integers $Y_0, Y_0 + p_1 p_2, Y_0 + 2p_1 p_2, \ldots$ by all small primes up to (say) 10^6. Once again, the value of 10^6 may be changed to anything that is convenient. The length of the sieve interval should be several times the largest prime in the

sieve factor base ($5 \cdot 10^6$ is a good choice). The integers untouched by the sieve will be candidates for primality testing and as a result of our use of the CRT, will automatically be 'strong' primes. One also sieves the public exponent e at this time, so that candidates p with $e|p-1$ are also removed.

2.4 **Even Exponents**

At the point where one constructs R via the CRT, one could also add in the condition that $R = 3 \bmod 8$ (for the first prime to be generated) or $R = 7 \bmod 8$ (for the second prime to be generated) in the case where one wanted to use an *even* integer as the public exponent. [The Rabin-Williams system]. One can therefore choose Rabin-Williams at essentially no computing cost if desired since the additional time computing the CRT is negligible. One also sieves the public exponent as in the odd case.

2.5 **Testing Candidates**

Once the set of candidates has been sieved by small primes, one can now test the numbers that have not been touched by the sieve for primality. There are several ways to do this. The set of possible methods has been greatly extended by new results which are discussed below. The basic criteria shall be that any method used must have an error rate no greater then $2^{-100} \sim 7.8 \times 10^{-31}$.

(1) A deterministic primality test

The two best current methods are the Cyclotomic ring test by Bosma-Cohen-Lenstra or the Elliptic Curve Primality Test by Atkin-Goldwasser-Killian. [2, 1] My personal recommendation is that while these are possibilities, they should not be used. The reason is that these algorithms are quite complicated to implement and that the likelihood of error in the implementation far exceeds the likelihood that a random method will return a composite.

(2) Use of the Miller-Rabin algorithm.

One applies sufficient tests so that the probability of a randomly generated candidate actually being composite, when multiple Miller-Rabin tests say 'prime' is less than 2^{-100}.

According to [3, 13] for 512-bit primes, 8 iterations suffice. For 640-bit primes, 6 iterations suffice. **This is the suggested method for this standard**. Other possibilities are given below.

(3) Use of the Miller-Rabin algorithm combined with a Lucas or Frobenius strong probable prime test [5]. According to a very recent paper of Grantham, if one uses a Frobenius probable prime test, the probability that a candidate is composite when the tests say 'prime' is less than 1/1770, as opposed to the 1/4 one gets with Miller-Rabin alone. If one performs a single Miller-Rabin test, followed by T Frobenius tests, the probability of error for 512-bit primes is then less than $1.5 \times 10^{-17}(1/1770)^T$ [6, equation 1.6]. To achieve a probability of 2^{-100}, $T = 4$ sufficies. One should be able to apply the analytical techniques of [3] to the Grantham algorithm to arrive at even stronger probability estimates. That is to say the number 1.5×10^{-17} can probably be reduced, but the method is as yet too new for anyone to have done this analysis. However, the correctness of the $1.5 \cdot 10^{-17}$ bound is not in question.

(4) It should be noted that there is no known composite integer which passes a single Miller-Rabin test, followed by a single Lucas strong probable prime test. Pomerance, Selfridge, and Wagstaff Jr. currently offer $640 for a counter-example. While a formal estimate of the probability of error for a combined Miller-Rabin/Lucas test is still lacking, heuristics suggest that counter-examples are *extremely* rare. This combination of tests was suggested in [9]. It is therefore **suggested** that following the Miller-Rabin tests a single Lucas test be performed.

(5) This subject area is changing rapidly. The purpose of the above discussions is simply to demonstrate that there are stronger alternatives to Miller-Rabin and they can be used if desired.

§3. GCD Attacks

A paper [4] submitted by Don Johnson at the December 1996 meeting of the X9.31 Standards Committee presented an attack on a collection of independently generated public moduli

by computing GCDs. Properly speaking this method attacks the underlying random number generator and/or a flawed prime generator implementation rather than the method or structure for generating the primes. This issue was addressed in a paper by Matyas [7]. RSA Laboratories supports the position taken by the Matyas paper.

We would also like to add that in our opinion unless the underlying random number generators are very badly flawed that the chance of this attack succeeding is negligible. Even assuming that an attack is *possible* because two keys exist with a common prime, it is unlikely to be *practical*. With N possible keys among which a collision exists, one would have to try, on average $\binom{N}{2}$ GCD computations. Collecting enough keys and computing the GCDs would require massive resources.

§4. **Choice of Random Number Generator**

Appendix B of X9.31 suggests several choices of random number generator. Any of these should be acceptable, but at least one should be *required*. However, the initial seed to any of these should be part of the audit information as well as the exact choice of algorithm. It should also be required that the initial seed have at least 256 bits of entropy. 80 bits is insuffient. Direct search of an 80-bit key space is within an order or magnitude or two of what can be achieved by factoring the modulus by the Number Field Sieve. 100 bits might be adequate.

References

1. Francois Morain, *Implementation of the Goldwasser-Killian-Atkin Primality Testing Algorithm*, Project ALGO, INRIA (1988).
2. Wieb Bosma, *Primality Proving with Cyclotomy*, Doctoral Dissertation Univ. of Amsterdam (1990).
3. I. Damgard, P. Landrock, and C. Pomerance, *Average case error estimates for the strong probable prime test*, Math. Comp. **61** (1993), 177-194.
4. Don Johnson, *A GCD attack on a weak RSA instantiation and some solutions or auditing RSA prime generation*, Certicom (Dec. 1996).
5. Jon Grantham, *A Frobenius probable prime test with high confidence*, http://www.math.uga.edu/ grantham/pseudo/ (to appear).

6. S.H. Kim and C. Pomerance, *The probability that a random probable prime is composite*, Math. Comp. **53** (1989), 721-742.

7. S.M. Matyas, *Comments on the GCD attack on RSA primes*, IBM (1997).

8. P.L. Montgomery & R.D. Silverman, *An FFT Extension to the $P-1$ Factoring Algorithm*, Math. Comp. **54** (1990), 839-854.

9. C. Pomerance, J.L. Selfridge, and S.S. Wagstaff Jr., *The pseudoprimes to $25 \cdot 10^9$*, Math. Comp. **35** (1980), 1003-1026.

10. R.L. Rivest and R.D. Silverman, *Are Strong Primes Needed for RSA?* (to appear).

11. J.Brillhart, D.H. Lehmer, and J.L. Selfridge, *New primality criteria and factorizations of $2^m \pm 1$*, Math. Comp. **29** (1975), 620-647.

12. R.D. Silverman & S.S. Wagstaff Jr., *A Practical Analysis of the Elliptic Curve Factoring Algorithm*, Math. Comp. **61** (1993), 445-462.

13. A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, Boca Raton, 1997.

Appendix

1. A *sieve procedure* is as follows. Start by selecting a *factor base* of all the primes p_i up to some selected limit L. Select a starting point for the sieve P, and a length for the sieve interval M. Compute $S_i = P \bmod p_i$ for all i. Initialize an array of length M to zero. Then starting at $P - S_i + p_i$ let every p_i^{th} element of the array be set to 1. Do this for the entire length of the array and for every i.

Now, every location in the array which has the value 1, is divisible some some small prime and is hence composite.

The array can be a bit array for compactness, when memory is small, or a byte array for speed, when memory is readily available. This is also no need to sieve the entire sieve interval at once before looking for candidations. One can partition the array into suitably small pieces, sieve each piece, look for candidates than go on to the next piece. Every location with the value 0 is a candidate for prime testing.

20 Crosby Drive

Bedford, MA 01730