Strong Primes are Easy to Find

John Gordon, Cybermation Ltd

Summary

A simple method is given for finding *strong*, random, large primes of a given number of bits, for use in conjunction with the RSA Public Key Cryptosystem. A *strong* prime p is a prime satisfying:

* $p = 1 \bmod r$

* $p = s-1 \bmod s$

* $r = 1 \bmod t$,

where r,s and t are all large, random primes of a given number of bits. It is shown that the problem of finding *strong*, random, large primes is only 19% harder than finding random, large primes.

Introduction

The most promising public key cryptosystem (PKC) since the idea was first published [1] is almost certainly the RSA scheme [2].

A brief description of the RSA scheme will now be given, but the interested reader should consult [2] for more details. In what follows, the terms *number*, and *integer* are both to be taken as indicating either a positive integer or zero.

In implementing this scheme a person (say Bob) makes for himself a set of three large numbers: m, E and D, (respectively the *modulus, Public key* and *Secret key*) with the properties:

$$\text{if } y = x^E \bmod m \qquad \text{then} \qquad x = y^D \bmod m$$

for all numbers x in the range (0, m-1).

The numbers E and m are published, and someone else (say Alice) wishing to send a secret message x (regarded for the purpose of encryption as a large integer) to Bob, calculates y from x and sends to Bob the *cryptogram* y. Since Bob knows D he can recover the message. Anyone else wishing to eavesdrop must find D, or else discover x some other way. Both of these recourses appear to be *computationally infeasible* for suitable choices of parameters.

Bob makes m, E and D as follows. He chooses two very large primes p and q at random with p and q of roughly equal size (of say 256 bits each). This choice of p and q is the subject of this paper. He chooses E at random, relatively prime to $(p-1)(q-1)$ and then finds D from the relationship:

$$ED = 1 \bmod (p-1)(q-1)$$

which he can do easily and quickly using *Euclid's algorithm* [3], [4]. Finally he forms m using:

$$m = pq.$$

A potential eavesdropper must it seems first find D, which appears to require the determination of p and q, which in turn seems to imply that he must be able to factorise m. To factorise a product $m=pq$ where p and q are very large primes of say 256 bits each is one of the hardest known common problems [2], [3].

The advanced techniques a cryptanalyst might use to factor m [3] break down when p (and similarly q) is not only prime but also has the properties that $p-1$ has a large prime factor say r, and $p+1$ has a large prime factor say s. To make the problem really hard $r-1$ should have a large prime factor as well. For logistical reasons it is also necessary to be able to choose p in some sense at random but with a given number of bits.

There is thus a considerable interest in the problem of finding primes with these desirable progerties. However there does not appear to be any published way of finding primes with all these properties, and in this paper we show how to satisfy all the requirements. In particular it is shown that the extra conditions imposed upon p add only 19% to the cost of the task of finding p.

The Technique

We seek therefore a computationally economical construction for a large, randomly seedable integer p of given number of bits and with the following properties:

* p is prime.

* $p-1$ has a large, prime factor, say r.

* $p+1$ has a large, prime factor, say s.

* $r-1$ has a large, prime factor, say t.

Numbers sastisfying these criteria will be known as *Strong Primes*. We now show, explicitly how to contruct strong primes. We begin with the following observations.

If $p-1$ has a large, prime factor r, then $p = Kr + 1$ for some K. If K is odd (and assuming that r is greater than 2 and hence an odd prime), then p will be even (since a product of odd numbers is odd), which is ridiculous since p is greater than r. Therefore K must be even. Since p, r s and t are all assumed to be large we are only interested in *odd* primes p whose properties are in effect:

(1) $p = 2jr + 1$ (or $p = 1 \bmod 2r$)

(2) $p = 2ks - 1$ (or $p = s-1 \bmod 2s$)

(3) $r = 2Lt + 1$ (or $r = 1 \bmod 2t$)

for some j, k, L where r,s and t are primes.

Our order of events will be:

* choose random seeds a and *b*

* from a and *b* generate random primes s and t

* from t construct r

* from r and s construct p.

It will be assumed that choosing random seeds with any required number of bits does not present any special problems and this aspect will not be addressed further.

Find s and r

Finding s (and t) which are just random primes greater than a given seed and of specified number of bits is relatively straightforward. Starting from random seed a, we will find the first prime s (or t) greater than a. We now estimate the computational effort and the time to complete tasks of this type.

We know from the *Prime Number Theorem* [4], that x/Ln(x) is a very good estimate of the number of primes less than x. Hence the density of primes in the neighbourhood of x is given by:

$$d/dx\ (x/Ln(x)) = (1/Ln(x)-1/Ln(x)^2)$$

which is close to 1/Ln(x) for large x. The mean separation between primes of magnitude x is therefore about Ln(x). If we search through only *odd* numbers for the next prime greater than s we will need to examine on average no more than Ln(x)/2 numbers. If $x=2^n$ this amounts to 0.35n integers. We are thus unlikely to need to examine say n integers before finding a prime.

Eliminating multiples of 3 reduces the search by a further factor of two-thirds and so on. Continuing in this way we find that eliminating multiples of the first 54 primes, (2,3,5,...,251), leaves only 10.035% of all integers for serious further examination. The reason for being interested in the first 54 primes resides in the fact that these are precisely those primes which can be represented in one 8-bit byte or less. They can be stored in single bytes and permit extremely rapid elimination using a division algorithm which efficiently exploits a 1-byte divisor. This algorithm need not form a quotient. The typical search will then pay serious attention to only 0.10035Ln(x) integers. If x is a n-bit number this amounts to less than 0.07n integers.

Our technique is to test these remaining, uneliminated integers using an efficient, statistical technique, for example *Algorithm-P* in [3]. Each such test consists of v passes through a procedure whose complexity is dominated by the need to perform a modular exponentiation u^q(mod.x) where v is about 5, and q and x are n-bit quantities.

In point of fact, non-primes are almost invariably eliminated on the *first* pass and so v=5 only for the number finally chosen. If most numbers are eliminated then v=1 is a more realistic estimate.

Modular exponentiation on a normal computer where multiprecision arithmetic must be used requires triple-nested loops at the bit or word level and the time to perform such an exponentiation is of order $O(n^3)$. Experiments on small computers using very efficient assembly language programming indicate that the time to exponentiate for large n is about

$$T_{exp}(n) = cTn^3/w$$

where c is a constant of size about 8, T is the time for one instruction and w is the word size. (On a special-purpose n-bit machine the time would be of the order $O(n^2)$). The time therefore needed to find s (or t) (ignoring the time for quick eliminations) is about 0.07n times $cvTn^3/w$ i.e. about:

$$T_{prime}(n) = 0.07cvTn^4/w.$$

When we have found s (or t), it is unlikely to have more bits than the seed a. We can virtually ensure this by picking our value of a in the range $(2^{n-1}, 2^{n-1}+2^{n-2}-1)$. This ensures that a starts with the two digits 10 which leaves a run of 2^{n-2} integers in which to find a prime before increasing the number of bits.

Find r

We now seek a prime r of the form 2Lt+1. Our technique is to search through (2Lt+1)-space for successive values of L. Since we will only be examining odd numbers, primes will appear to be twice as dense as among all numbers, but conversely twice as many will have non-trivial divisors. Thus the time to find r will again be be $T_{prime}(n)$ where n in the number of bits in r.

Controlling the size of r

We are likely to use about $n\mathrm{Ln}(2)/2 = 0.35n$ successive values of L before finding r. Every time L doubles another bit is added to $2Lt+1$. If this process is not to leave great uncertainties in the final number of bits in r we should start with a value L_o of of about n, so that L will increase on average by a factor of 1.35 which is less than 2. The final size of r will be very close to $\mathrm{Log}_2(2L_o) + n$ bits. We can increase the certainty of this by increasing L_o.

A more sophisticated approach is to arrange for 2t to be say $\mathrm{Log}_2(n)$ bits shorter than the desired length of r, then starting with unity, add in successive multiples or 2t until the desired length of r is reached, and then begin checking for primality at each subsequent addition of 2t.

Find p

We now come to the final part of the technique namely, given primes r and s, find a prime p, close in size to a given number of bits, and satisfying:

$$p = 2jr+1 = 2ks-1, \text{ for some } j \text{ and } k$$

or

$$p = 1 \bmod 2r = 2s-1 \bmod 2s. \qquad \{1\}$$

The key to solving the problem of finding primes with these properties is contained in the following theorem.

Theorem 1:

If r and s are odd primes, then p satisfies:

$$p = (1 \bmod 2r) = (s-1 \bmod 2s)$$

if and only if p is of the form:

$$p = p_o + 2krs \qquad \{2\}$$

where

$$p_o = u(r,s) \qquad :u(r,s) \text{ odd}$$

$$= u(r,s) + rs \qquad :u(r,s) \text{ even}$$

$$\text{and } u(r,s) = (s^{r-1} - r^{s-1}) \bmod rs. \qquad \{3\}$$

Proof:

Integers, prime or otherwise, satisfying {1} clearly also satisfy the weaker condition:

$$p = jr + 1 = ks - 1 \qquad \text{for some } j,k. \qquad \{4\}$$

Numbers satisfying {4} are alternately odd and even. Integers satisfying {1} are just the odd valued numbers satisfying {4}. The remainder of the proof consists in showing that numbers satisfying {4} are of the form u(r,s) + krs. Solving {4} is just a special case of an application of the *Chinese Remainder Theorem* [4].

Consider the number u(r,s) of the form {3} above.

Now by *Fermat's Theorem* [4], namely:

if q is prime and $0 <= x < q$, then

$$x^{q-1} = 1 \bmod q \qquad = kq + 1, \qquad \text{for some } k,$$

it is clear that $s^{r-1} = 1 \bmod r$, and similarly $r^{s-1} = 1 \bmod s$.

Also of course $s^{r-1} = 0 \bmod s$, and $r^{s-1} = 0 \bmod r$.

Finally $rs = (0 \bmod r) = (0 \bmod s)$.

Thus u(r,s) satisfies {4}.

We now show that numbers not of the form u(r,s) + ksr cannot satisfy {4}.

Let u and u' satisfy {4} and consider the difference:

$$u - u' \quad = \quad (1 \bmod r) = (1 \bmod r) = 0 \bmod r \qquad = kr$$

$$= \quad (s-1 \bmod s) - (s-1 \bmod s) = 0 \bmod s = k's$$

for some k and k'. Thus u-u' is a multiple of LCM(r,s) which is rs since r and s are prime. Since u(r,s) satisfies {4}, u and u' must be of the form u(r,s) + ksr. QED.

Finding u(r,s) and hence p_o requires two exponentiations at a cost of $2T_{exp}$. Finding p amounts to finding a prime in (p_o+2krs)-space and the same considerations apply here as did to the search for r in (2Lt+1)-space, namely that we should start with p_o and add in successive multiples of 2rs until the desired size is reached, then check for primality at each subsequent addition.

Size of p,r,s and t

The size of p is a few bits larger than the size of 2rs. The difference is entirely due to the need to excercise some control in the size of p. Thus we should start with 2rs of a suitable size, say $\text{Log}_2(n)$ bits less than n, the desired number of bits in p. This in turn tells us the size for r and s. Presumably it is desirable for r and s to be of about equal size which will be a few bits less than half the size of p. There is no problem with s, and the method of finding r of suitable size has already been dealt with earlier.

Time to Find p

The time spent searching for primes dominates. We need to mount searches for p, of n bits, and for t, r and s, each roughly of n/2 bits. Altogether, ignoring all times except those spent searching for primes, the time to find p should average

$$T_{prime}(n) + 3T_{prime}(n/2)$$

$$= 1^3/_{16}T_{prime}(n)$$

$$= 1.19 \times 0.07cvTn^4/w.$$

This represents an increase of only 3/16 (=19%) over the time to find a random prime of given size n bits.

Example

Using the technique described here, *strong* primes of about 256 bits such as:

p=7,918,324,333,004,779,287,780,879,909,121,159,911,537,
551,977,796,076,554,305,607,309,994,905,870,203

where t= 83,106,713,586,449,986,154,292,642,419,182,973
r= 7,645,817,649,953,398,726,194,923,102,564,833,517
and s=10,638,156,841,358,536,678,090,874,848,207,317,901

can be generated in about 20 minutes on a small microcomputer with 1MHz clock (Apple-II) using an extremely efficient modular arithmetic package (CyMAS).

Acknowledgements

I am indebted to Donald Davies of the National Physical Laboratories, England for his encouragement in this matter and for one or two observations incorporated above. I am also indebted to British Telecom for suggesting and sponsoring the problem.

References

[1] W Diffie and M.E. Hellman, "*New Directions in Cryptography*", IEEE Trans. Inform. Theory, vol. IT-22, No.6, 1976, 644-54.

[2] R.L. Rivest, A. Shamir and L. Adleman, "*A Method for obtaining Digital Signatures and Public Key Cryptosystems*", Comms ACM, 21,2, 121-126, 1978.

[3] D.E. Knuth, "*The Art of Computer Programming Volume 2, Seminumerical Algorithms*", 2nd Ed., Addison-Wesley, 1982.

[4] D.M. Burton, "*Elementary number theory*", Allyn and Bacon, 1980

J Gordon
Cybermation Ltd
High St
Wheathampstead, Herts
England