Constructive Methods for the Generation of Prime Numbers

(— Submission to NESSIE —)

[Published in S. Murphy, Ed., *Proc. of 2nd Open NESSIE Workshop*, Egham, UK, September 12–13, 2001.]

Marc Joye[1] and Pascal Paillier[2]

[1] Gemplus Card International
Parc d'Activités de Gémenos, 13881 Gémenos Cedex, France
marc.joye@gemplus.com – http://www.geocities.com/MarcJoye/
[2] Gemplus Card International
34 rue Guynemer, 92447 Issy-les-Moulineaux, France
pascal.paillier@gemplus.com

http://www.gemplus.com/smart/

Abstract. Public-key cryptography is rapidly becoming ubiquitous in many aspects of "electronic" life. It is used to provide various security services such as data integrity, confidentiality, authentication and non-repudiation. Most public-key cryptographic methods require the generation of random prime numbers. A naive solution to obtain a prime number consists in randomly choosing a number and testing it for primality. Such a solution is however not satisfactory in all respects as it is not efficient. This note is aimed at making strides towards the provision of practical (i.e., efficient) solutions for generating prime numbers.

Keywords. Prime numbers, prime generation, RSA, DSA, public-key cryptography.

1 Introduction

When efficiency is not a concern, the best way to generate a random prime number is to select a random number q and test it for primality; if the test is unsuccessful then it is reiterated with $q \leftarrow q + 1$. Noting that all prime numbers (except 2) are odd, a straightforward improvement is to choose q odd and to update candidate q as $q \leftarrow q + 2$. This note specifies a technique which chooses a candidate *constructively* coprime to lots of small primes and updates it in a way that it still remains coprime with those factors. A method for producing coprime candidates is also specified.

Current standards do not really consider how to generate random primes but rather propose methods to test the primality of a random number. For

example, the IEEE P1363 standard for public-key cryptography simply suggests to choose a random odd number and to check its primality [1, § A.15.5]. More efficient constructive techniques for prime generation are only addressed in a recent ISO/IEC working draft [2] where the aforementioned sieving method is described (cf. [2, Section 6]). However, no practical implementation is given. The purpose of this note is to fill the gap by providing a very efficient sieving method.

2 Two Useful Propositions

Let $\mathbb{Z}_m$ be the ring of integers modulo m and let ${\mathbb{Z}_m}^*$ be its group of units (i.e., the set of invertible elements modulo m). We further denote λ the Carmichael function defined, for $m = \prod_i {p_i}^{\delta_i}$, as $\lambda(m) = \mathrm{lcm}[\lambda({p_i}^{\delta_i})]_i$, and $\lambda({p_i}^{\delta_i}) = \phi({p_i}^{\delta_i}) = {p_i}^{\delta_i - 1}(p_i - 1)$ for an odd prime p_i, $\lambda(2) = 1$, $\lambda(4) = 2$ and $\lambda(2^{\delta_i}) = \frac{1}{2}\phi(2^{\delta_i}) = 2^{\delta_i - 2}$ for $\delta_i \geq 3$.

Proposition 1. *For all $k \in \mathbb{Z}_m$, $k \in {\mathbb{Z}_m}^*$ if and only if $k^{\lambda(m)} \equiv 1 \pmod{m}$.*

Proof. We have $k \in {\mathbb{Z}_m}^*$ if and only if, for all primes $p_i \mid m$, $\gcd(k, p_i) = 1 \iff k^{p_i - 1} \equiv 1 \pmod{p_i}$ so that $k^{\lambda(m)} \equiv 1 \pmod{m}$ by Chinese remaindering. □

Proposition 2. *For all k and $r \in \mathbb{Z}_m$ s.t. $\gcd(r, k, m) = 1$, we have*

$$[k + r(1 - k^{\lambda(m)})] \in {\mathbb{Z}_m}^* \ .$$

Proof. Let $\prod_i {p_i}^{\delta_i}$ denote the prime factorization of m. Define $K_r := [k + r(1 - k^{\lambda(m)})] \in \mathbb{Z}_m$. Let p_i be a prime factor of m. Suppose that $p_i \mid k$ then $K_r \equiv r \not\equiv 0 \pmod{p_i}$ since $\gcd(r, p_i)$ divides $\gcd(r, \gcd(k, m)) = \gcd(r, k, m) = 1$. Suppose now that $p_i \nmid k$ then $k^{\lambda(m)} \equiv 1 \pmod{p_i}$ and so $K_r \equiv k \not\equiv 0 \pmod{p_i}$. Therefore for all primes $p_i \mid m$, we have $K_r \not\equiv 0 \pmod{p_i}$ and thus $K_r \not\equiv 0 \pmod{{p_i}^{\delta_i}}$, which, by Chinese remaindering, concludes the proof. □

Remark 1. Note that $\mathbb{B}_m(k) := \{r \in \mathbb{Z}_m : \gcd(r, k, m) = 1\} \supseteq {\mathbb{Z}_m}^*$.

3 Generating Random Primes

3.1 Scope

The goal is to generate a prime number q within the interval $[q_{\min}, q_{\max}]$ in compliance with the ISO/IEC standard ([2]). In most cases, one has $q_{\max} = 2^n - 1$, and $q_{\min} = 2^{n-1} + 1$ when generating n-bit primes or $q_{\min} = \lceil \sqrt{2^{2n-1} + 1} \rceil$ when generating $2n$-bit RSA moduli of the form $N = pq$ with p, q prime.

Our proposal consists in a pair of algorithms: the prime generation algorithm itself and an algorithm for generating invertible elements. We assume that a random number generator is at disposal along with a primality testing oracle.

3.2 Prime generation

Let $0 < \epsilon \leq 1$ denote a quality parameter (a typical value for ϵ is 10^{-3}). The setup phase requires a product of primes $\pi = \prod_i p_i$ such that there exist integers t, v, w satisfying

(P1) $1 - \epsilon < \dfrac{w\pi - 1}{q_{\max} - q_{\min}} \leq 1$;
(P2) $v\pi + t \geq q_{\min}$;
(P3) $(v + w)\pi + t - 1 \leq q_{\max}$; and
(P4) the ratio $\phi(\pi)/\pi$ is as small as possible.

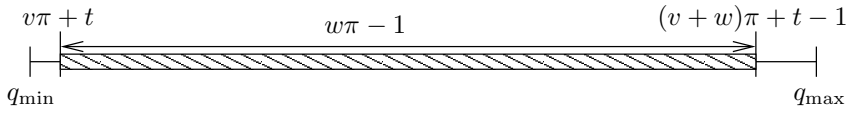

Fig. 1. Targeted Interval.

The primes output by our algorithm lie, in fact, in the sub-interval $[v\pi + t, (v + w)\pi + t - 1] \subseteq [q_{\min}, q_{\max}]$. The error in the approximation is captured by the value of ϵ: a smaller ϵ gives better results (cf. Property (P1)). The minimality of the ratio $\phi(\pi)/\pi$ in Property (P4) ensures that π contains a maximum number of primes.

Input: parameters $l = v\pi$, $m = w\pi$, t, and $a \in \mathbb{Z}_m{}^*$
Output: a prime $q \in [q_{\min}, q_{\max}]$

1. Randomly choose $k \in \mathbb{Z}_m{}^*$
2. Set $q \leftarrow [(k - t) \bmod m] + t + l$
3. If (q not prime) then
   (a) Set $k \leftarrow ak \pmod m$
   (b) Go to Step 2
4. Output q

Fig. 2. Prime Generation Algorithm.

It is worthwhile noticing that if $a, k \in \mathbb{Z}_m{}^*$ so does their product, ak, since $\mathbb{Z}_m{}^*$ is a (multiplicative) group. Therefore, throughout the algorithm, k remains coprime to m and thus to π (remember that π contains a large number of prime factors by (P4)). This, in turn, implies that q is coprime to π as $q \equiv [(k -$

$t) \bmod m] + t + l \equiv k \pmod{\pi}$ and $k \in {\mathbb{Z}_\pi}^*$. Consequently, the probability that candidate q is prime at Step 3 is high.

We now specialize the previous algorithm to make it as fast as possible. The optimal value for t is $t = 0$. Moreover, it is advantageous to choose a so that multiplication by a modulo m is not very costly. The best possible value is $a = 2$. Unfortunately, 2 must belong to ${\mathbb{Z}_m}^*$ and owing to Condition (P4), 2 is a factor of π and so of m, a contradiction. Our idea is to choose m odd (so that $2 \in {\mathbb{Z}_m}^*$) and to slightly modify the previous algorithm in order to ensure that prime candidate q is always odd. We require $\pi = \prod_i p_i$ (with $p_i \neq 2$) and integers v and w (odd) satisfying:

(P2') $v\pi + 1 \geq q_{\min}$;
(P3') $(v + w)\pi - 1 \leq q_{\max}$.

Input: parameters $l = v\pi$ and $m = w\pi$ (m odd)
Output: a prime $q \in [q_{\min}, q_{\max}]$

1. Randomly choose $k \in {\mathbb{Z}_m}^*$
2. Set $q \leftarrow k + l$
3. If (q even) then $q \leftarrow m - k + l$
4. If (q not prime) then
   (a) Set $k \leftarrow 2k \pmod{m}$
   (b) Go to Step 2
5. Output q

Fig. 3. Faster Prime Generation Algorithm.

Note that if $k+l$ is even then $m-k+l$ is odd since $m-k+l \equiv m+(k+l) \equiv m \equiv 1 \pmod 2$. Hence, as before, a so-generated q is prime to 2π: $\gcd(q, 2) = 1$ as q is odd; and $\gcd(q, \pi) = 1$ as $q \equiv \pm k \pmod{\pi}$ and $\pm k \in {\mathbb{Z}_\pi}^*$.

3.3 Unit generation

The prime generation algorithms given in Figs. 2 and 3 require a random element of ${\mathbb{Z}_m}^*$. This section provides an algorithm to efficiently produce such an element. The correctness of the algorithm follows from Proposition 1.

Input: parameter m
Output: a unit $k \in \mathbb{Z}_m{}^*$

1. Randomly choose $k \in [1, m)$
2. Set $U \leftarrow (1 - k^{\lambda(m)}) \bmod m$
3. If $(U \neq 0)$ then
   (a) Choose a random $r \in [1, m)$
   (b) Set $k \leftarrow k + rU \pmod m$
   (c) Go to Step 2
4. Output k

Fig. 4. Unit Generation Algorithm.

This algorithm presents the advantage of being self-correcting. As soon as k is relatively prime to some factor of m, it remains prime to this factor after the updating step $k \leftarrow k + rU$ (see Proposition 2).

4 Generating Constrained Random Primes

Some applications require that the primes found by our algorithm satisfy additional properties such as being strong or compliant with the ANSI X9.31 standard. We refer the reader to [3] for a collection of techniques allowing to produce such primes.

Acknowledgements

The authors are grateful to Walter Fumy for sending a copy of [2].

References

1. IEEE Std 1363-2000. IEEE Standard Specifications for Public-Key Cryptography. IEEE Computer Society, August 29, 2000.
2. ISO/IEC WD 18032. Prime number generation. Working draft, April 18, 2001.
3. Marc Joye, Pascal Paillier, and Serge Vaudenay. Efficient generation of prime numbers. In Ç. K. Koç and C. Paar, Eds., *Cryptographic Hardware and Embedded Systems – CHES 2000*, vol. 1965 of *Lecture Notes in Computer Science*, pp. 340–354, Springer-Verlag, 2000.