Close to Uniform Prime Number Generation With Fewer Random Bits

Pierre-Alain Fouque and Mehdi Tibouchi

École normale supérieure
Département d'informatique, Équipe de cryptographie
45 rue d'Ulm, F-75230 Paris CEDEX 05, France
{pierre-alain.fouque,mehdi.tibouchi}@ens.fr

Abstract. In this paper we analyze a simple method for generating prime numbers with fewer random bits. Assuming the Extended Riemann Hypothesis, we can prove that our method generates primes according to a distribution that can be made arbitrarily close to uniform. This is unlike the PRIMEINC algorithm studied by Brandt and Damgård and its many variants implemented in numerous software packages, which reduce the number of random bits used at the price of a distribution easily distinguished from uniform.

Our new method is also no more computationally expensive than the ones in current use, and opens up interesting options for prime number generation in constrained environments.

Keywords: Public-key cryptography, prime number generation, RSA, efficient implementations, random bits.

1 Introduction

Prime number generation is a very sensitive issue for many public-key schemes and it may not have received an attention commensurate to its importance from a security standpoint.

There are several ways in which we could assess the quality of a random prime generation algorithm, such as its speed (time complexity), its accuracy (the probability that it outputs numbers that are in fact composite), its statistical properties (the regularity of the output distribution), and the number of bits of randomness it consumes to produce a prime number (as good randomness is crucial to key generation and not easy to come by [ESC05]).

In a number of works in the literature, cryptographers have proposed faster prime generation algorithms [BDL91,BD92,JPV00,JP06] or algorithms providing a proof that the generated numbers are indeed prime numbers [Mau89,Mau95,Mih94].

A number of these works also prove lower bounds on the entropy of the distribution of prime numbers they generate, usual based on very strong conjectures on the regularity of prime numbers, such as the prime r-tuple conjecture of Hardy-Littlewood [HL22]. However, such bounds on the entropy do not ensure that the resulting distribution is statistically close to the uniform distribution: for example, they do not preclude the existence of efficient

distinguishers from the uniform distribution, which can indeed be shown to exist in most cases.

But some cryptographic protocols (including most schemes based on the Strong RSA assumption, such as Cramer-Shoup signatures [CS00]) specifically require uniformly distributed prime numbers for the security proofs to go through. Moreover, even for more common uses of prime number generation, like RSA key generation, it seems preferable to generate primes that are almost uniform, so as to avoid biases in the RSA moduli themselves, even if it is not immediately clear how such biases can help an adversary trying to factor the moduli.

To the authors' knowledge, the only known prime generation algorithm for which the statistical distance to the uniform distribution can be bounded is the trivial one: pick a random (odd) integer in the desired interval, return it if it is prime, and try again otherwise. The output distribution of this algorithm is exactly uniform (or at least statistically close, once one accounts for the compositeness probability of the underlying randomized primality checking algorithm), but it has the drawback of consuming a very large amount of random bits, in addition to being rather slow.

By contrast, the PRIMEINC algorithm studied by Brandt and Damgård [BD92] (basically, pick a random number and increase it until a prime is found) only consumes roughly as many random bits as the size of the output primes, but we can show that its output distribution, even if it can be shown to have high entropy if the prime r-tuple conjecture holds, is also provably quite far from uniform, as we demonstrate in Section 3. It is likely that most algorithms that proceed deterministically beyond an initial random choice, including those of Joye, Paillier and Vaudenay [JPV00,JP06] or Maurer [Mau89,Mau95] exhibit similar distributional biases.

The goal of this paper is to achieve in some sense the best of both worlds: construct a prime generation algorithm that consumes much fewer random bits than the trivial algorithm while being efficient and having an output distribution that is provably close to the uniform one.

We present such an algorithm (with several variants) in Section 2. The main result presents a simple trade-off between the uniformity of the resulting distribution and the amount of randomness consumed, and shows that statistical indistinguishability can be achieved with much fewer random bits than required by the trivial algorithm. The proof is quite elementary (using only some character sum estimates) and only depends on the Extended Riemann Hypothesis—an assumption much less bold than the likes of the prime r-tuple conjecture.

We also verify in Section 4 that our prime generation algorithm is quite competitive performance-wise, both in terms of speed and of required random bits. It is even suitable for implementation in constrained environments like smart cards.

2 Our method

2.1 Description

Basic Method. A simple method to construct obviously uniformly distributed prime numbers up to 2^n is to pick random numbers in $\{0, \ldots, 2^n - 1\}$ and retry until a prime is found. However, this method consumes n bits of randomness per iteration (not counting the amount of randomness consumed by primality testing), and hence an expected amount of $n^2 \log 2$ bits of randomness to produce a prime, which is quite large.

A simple way to reduce the amount of randomness consumed is to pick a random odd number and to simply change the top ℓ bits until a prime is found. This method, described in Algorithm 1, consumes only ℓ bits of randomness per iteration, and an expected $(n - \ell) + (\ell n \log 2)/2$ bits overall. Despite the simplicity of this method, we prove in the next section that its resulting distribution achieves a statistical distance of less than about $2^{-\ell/2}$ to the uniform distribution.

Algorithm 1 Our basic method

1: $b \xleftarrow{\$} \{0, \ldots, 2^{n-\ell-1} - 1\}$
2: **repeat**
3: $\quad a \xleftarrow{\$} \{0, \ldots, 2^{\ell} - 1\}$
4: $\quad p \leftarrow 2^{n-\ell} \cdot a + 2b + 1$
5: **until** p is prime
6: **return** p

Note that this method can be combined with efficient trial division to speed up primality testing, since the reductions of the low-order bits $(2b + 1)$ modulo small primes can be precomputed. Thus, we only need to carry out trial division on a at each iteration of the **repeat** loop.

More efficient version. The previous method starts by fixing the reduction modulo $2^{n-\ell}$ of the prime it searches for. But we can improve the odds of finding a prime number by fixing the reduction b modulo a number m which is a product of many small primes instead. The number b should then be picked uniformly at random among integers less than m and coprime to m, which can be achieved using the unit generation algorithm of Joye and Paillier [JP06].

Including the Joye-Paillier algorithm, the whole prime generation process can be written as described in Algorithm 2. In this setting, m is usually picked as the product of all small primes up to a certain bound β chosen such that the bit size of m is $n - \ell$ for some appropriate ℓ.

In Algorithm 2, Steps 1–7 are the Joye-Paillier unit generation technique: at the end of this loop, b is distributed exactly uniformly at random in $(\mathbb{Z}/m\mathbb{Z})^*$. Then, Steps 8–11

Algorithm 2 More efficient method

1: $b \xleftarrow{\$} \{1, \ldots, m-1\}$
2: $u \leftarrow (1 - b^{\lambda(m)}) \bmod m$
3: **if** $u \neq 0$ **then**
4: $\quad r \xleftarrow{\$} \{1, \ldots, m-1\}$
5: $\quad b \leftarrow b + ru \bmod m$
6: $\quad$ **goto** step 2
7: **end if**
8: **repeat**
9: $\quad a \xleftarrow{\$} \{0, \ldots, \lfloor 2^n/m \rfloor - 1\}$
10: $\quad p \leftarrow am + b$
11: **until** p is prime
12: **return** p

are exactly analogous to Algorithm 1, but are executed a much smaller number of times on average.

More precisely, the probability that a number of the form $am+b < 2^n$ is prime is about $\frac{m}{\varphi(m)n\log 2}$ according to the prime number theorem for arithmetic progressions. Therefore, the expected number of primality testing loops in this case is only:

$$n \log 2 \cdot \frac{\varphi(m)}{m} = n \log 2 \prod_{p \leq \beta} \left(1 - \frac{1}{p}\right) \sim \frac{n \log 2 \cdot e^{-\gamma}}{\log \beta}$$

by Mertens' formula (where γ is the Euler-Mascheroni constant). Now, we have $\log m = \sum_{p \leq \beta} \log p \sim \beta$, so that $\beta \approx (n - \ell) \log 2$. Thus, the number of primality testing loops is down to about:

$$\frac{n \log 2 \cdot e^{-\gamma}}{\log(n - \ell) + \log \log 2}$$

which, for typical parameter sizes ($512 \leq n \leq 3072$ and $32 \leq \ell \leq 256$, say), is around 5 to 7 times fewer than with the previous approach. Accordingly, this part of the algorithm consumes 5 to 7 times less randomness, namely about:

$$\frac{\ell n \log 2 \cdot e^{-\gamma}}{\log(n - \ell) + \log \log 2} \text{ bits overall.}$$

This technique can also lead to significantly faster prime generation compared to an approach which uses trial division to reduce calls to the more costly primality testing algorithm, depending on how efficient modular reductions are in the environment of interest—for example, Joye and Paillier report speed-ups of one order of magnitude on smart cards using their algorithm [JP06, Fig. 2], which is slightly more computationally demanding than ours since it requires a large modular multiplication at each iteration, instead of the small integer multiplication from Step 10.

On the other hand, the unit generation part of the algorithm does consume more randomness than the single-step generation of b in Algorithm 1. This has limited influence, however, since the number of iterations in this part is quite small. Indeed, it is easy to see that b is relatively prime to some $p|m$ as soon as one of the random elements of $\mathbb{Z}/m\mathbb{Z}$ returned by the random number generator is itself coprime to p (more precisely: the assignment $b \leftarrow b+ur$ preserves coprimality with p, since $(b,p)=1$ implies $u \equiv 0 \pmod{p}$; but when $b \equiv 0 \pmod{p}$, then u is invertible mod p, and thus b becomes non zero mod p if r is). As a result, the probability that this part of the algorithm takes $\leq k$ iterations (including the initial random assignment of b) is exactly $z_k = \prod_{p \leq \beta}(1-p^{-k})$, which converges rapidly to 1. The expected number of iterations is thus $E_\beta = \sum_{k \geq 1} k(z_k - z_{k-1})$ which we can evaluate numerically for any given β. An upper bound is given by the limit as $\beta \to +\infty$, namely:

$$E_\infty = \sum_{k \geq 2} k \left(\frac{1}{\zeta(k)} - \frac{1}{\zeta(k-1)} \right) \leq 2.71 \tag{1}$$

This means in particular that the amount of randomness required by unit generation, $(n-\ell) \cdot E_\beta$, is reasonably small (and typically small enough that this method is better than the previous one in terms of random bits).

An added benefit of this approach is that the bound on the statistical distance to the uniform distribution is expressed in terms of $\varphi(m)$ rather than m itself: this is smaller when m is a product of many small primes than when m is a power of two as before.

2.2 Main result

Both of the previous methods generate primes of the form $am+b$ where b is first picked uniformly at random among integers less than m and coprime to m: in the first method, m is a power of two, whereas in the second method, it is a product of small primes, but the principle remains the same.

We now prove that the distribution of primes generated in this fashion is close to the uniform distribution among all primes in the allowable interval $\{1, \ldots, x-1\}$ with $x = m \cdot \lfloor 2^n/m \rfloor \approx 2^n$.

Theorem 1. *Assume the Extended Riemann Hypothesis. Suppose further that $x \geq 17$ and $x/m \geq e$ (the base of natural logarithms). Then the statistical distance Δ between the uniform distribution over primes less than x and the distribution resulting from the previous method is bounded as:*

$$\Delta < 3 \log^2 x \cdot \sqrt{\frac{\varphi(m)}{x}}$$

The rest of this section is devoted to the proof of this theorem. As usual, note $\pi(x)$ the number of primes up to x, and $\pi(x, m, b)$ the number of primes p up to x such that $p \equiv b \pmod{m}$. Furthermore, for any Dirichlet character χ modulo m, let:

$$\pi(x, \chi) = \sum_{p \leq x} \chi(x)$$

(where the sum is over prime numbers p, as will be the case for all sums over p henceforth). Then clearly, for any b coprime to m, we have:

$$\pi(x, m, b) = \frac{1}{\varphi(m)} \sum_{\chi \bmod m} \overline{\chi(b)} \cdot \pi(x, \chi)$$

and in this sum, the contribution of the trivial character χ_0 is:

$$\frac{1}{\varphi(m)} \pi(x, \chi_0) = \frac{\pi(x) - s(m)}{\varphi(m)}$$

where $s(m)$ is the sum of all distinct prime divisors of m. Hence:

$$\frac{1}{\varphi(m)} \sum_{\chi \neq \chi_0} \overline{\chi(b)} \cdot \pi(x, \chi) = \pi(x, m, b) - \frac{\pi(x) - s(m)}{\varphi(m)} \tag{2}$$

Therefore, we obtain the following result.

Lemma 1. *The statistical distance Δ satisfies:*

$$\Delta \leq \frac{1}{\pi(x)} \left(2s(m) + \frac{1}{\varphi(m)} \sum_{b \in (\mathbb{Z}/m\mathbb{Z})^*} \left| \sum_{\chi \neq \chi_0} \overline{\chi(b)} \pi(x, \chi) \right| \right)$$

Proof. Indeed, the probability that a given prime $p \leq x$ is chosen by our algorithm is 0 if p divides m, and $1/(\varphi(m)\pi(x, m, b))$ if $p \equiv b \pmod{m}$ for b coprime to m (b is first chosen uniformly among the $\varphi(m)$ classes in $(\mathbb{Z}/m\mathbb{Z})^*$, and p is then chosen uniformly among the $\pi(x, m, b)$ possible primes of the form $am + b$). Thus:

$$\begin{aligned}
\Delta &= \frac{s(m)}{\pi(x)} + \sum_{b \in (\mathbb{Z}/m\mathbb{Z})^*} \pi(x, m, b) \cdot \left| \frac{1}{\varphi(m)\pi(x, m, b)} - \frac{1}{\pi(x)} \right| \\
&= \frac{1}{\pi(x)} \left(s(m) + \sum_{b \in (\mathbb{Z}/m\mathbb{Z})^*} \left| \frac{\pi(x)}{\varphi(m)} - \pi(x, m, b) \right| \right) \\
&= \frac{1}{\pi(x)} \left(s(m) + \sum_{b \in (\mathbb{Z}/m\mathbb{Z})^*} \left| \frac{s(m)}{\varphi(m)} - \frac{1}{\varphi(m)} \sum_{\chi \neq \chi_0} \overline{\chi(b)} \pi(x, \chi) \right| \right) \qquad \text{by (2)} \\
&\leq \frac{1}{\pi(x)} \left(2s(m) + \frac{1}{\varphi(m)} \sum_{b \in (\mathbb{Z}/m\mathbb{Z})^*} \left| \sum_{\chi \neq \chi_0} \overline{\chi(b)} \pi(x, \chi) \right| \right)
\end{aligned}$$

as required. □

We can now estimate the sum over $b \in (\mathbb{Z}/m\mathbb{Z})^*$ using the Cauchy-Schwarz inequality. Indeed, let:

$$S(x) = \frac{1}{\varphi(m)} \sum_{b \in (\mathbb{Z}/m\mathbb{Z})^*} \left| \sum_{\chi \neq \chi_0} \overline{\chi(b)} \pi(x, \chi) \right|$$

Applying Cauchy-Schwarz, we get:

$$\begin{aligned} S(x) &\leq \frac{1}{\varphi(m)} \cdot \sqrt{\varphi(m)} \sqrt{\sum_{b \in (\mathbb{Z}/m\mathbb{Z})^*} \left| \sum_{\chi \neq \chi_0} \overline{\chi(b)} \pi(x, \chi) \right|^2} \\ &\leq \frac{1}{\sqrt{\varphi(m)}} \sqrt{\sum_{b \in (\mathbb{Z}/m\mathbb{Z})^*} \sum_{\chi, \chi' \neq \chi_0} \overline{\chi(b)} \chi'(b) \cdot \pi(x, \chi) \overline{\pi(x, \chi')}} \end{aligned}$$

But the sum over $b \in (\mathbb{Z}/m\mathbb{Z})^*$ of $\overline{\chi(b)}\chi'(b)$ is $\varphi(m)$ if $\chi = \chi'$ and vanishes otherwise. Thus:

$$S(x) \leq \sqrt{\sum_{\chi \neq \chi_0} \left| \pi(x, \chi) \right|^2} \tag{3}$$

We can thus derive a bound on $S(x)$, and hence on Δ, from an estimate of $\pi(x, \chi)$ for non trivial Dirichlet characters χ. Such an estimate can be obtained by locating the zeros of the L-function $L(s, \chi)$ in the critical strip (see e.g. [Dav80, Chapter 20]). In particular, a well-known consequence of the Extended Riemann Hypothesis, due to Titchmarsh, is that $\pi(x, \chi) = O\big(\sqrt{x} \log(mx)\big)$, where the implied constant is absolute. One can actually give an effective version of this estimate:

Lemma 2 (Oesterlé, [Oes79]). *Assuming the Extended Riemann Hypothesis, we have, for all $x \geq 2$ and all non trivial Dirichlet characters χ modulo m:*

$$\left| \pi(x, \chi) \right| \leq \sqrt{x} (\log x + 2 \log m)$$

Therefore, under ERH, estimate (3) becomes:

$$S(x) \leq \sqrt{\varphi(m) x} (\log x + 2 \log m)$$

since $m \leq x$. Putting this together with the result of Lemma 1, we get:

$$\begin{aligned} \Delta &\leq \frac{2s(m) + \big(3 \log x - 2 \log(x/m)\big) \sqrt{\varphi(m) x}}{\pi(x)} \\ &\leq \frac{3 \log x \sqrt{\varphi(m) x}}{\pi(x)} + \frac{2}{\pi(x)} \big(s(m) - \sqrt{e \cdot \varphi(m) m}\big) \\ &\leq \frac{3 \log x \sqrt{\varphi(m) x}}{\pi(x)} \end{aligned}$$

since it is easy to check that $s(m) \leq \sqrt{e \cdot \varphi(m) m}$. Finally, a classical estimate [BS96, Th. 8.8.1] states that $\pi(x) > x/\log x$ for $x \geq 17$. Therefore, we obtain the stated bound on Δ:

$$\Delta < 3 \log^2 x \cdot \sqrt{\frac{\varphi(m)}{x}}$$

□

Remark 1. 1. Perhaps surprisingly, while the Extended Riemann Hypothesis provides strong enough bounds to obtain this non trivial estimate for the statistical distance, it is not actually sufficient to prove that the algorithm as presented above *terminates* in all cases. Indeed, the best bound that can be established with ERH (e.g. using Lemma 2) for the smallest prime p in the arithmetic progression $am + b$ is of the form $p < C \cdot (m \log m)^2$ for some constant C ($C = 2$ is enough). In particular, ERH isn't sufficient to prove that $\pi(x, m, b) \neq 0$ unless $x \gg m^2$, which is not usually the case in our setting, where we would like to choose m only a few bits shorter than x.
However, a conjecture by Wagstaff [Wag79], made more precise by McCurley [McC86], ensures that the algorithm terminates on any input provided that m is chosen such that $x > (2+\varepsilon) \cdot \varphi(m) \log^2 m$. This is satisfied, in particular, whenever the bound given in Theorem 1 is non trivial! An alternate way to avoid this problem without relying on assumptions stronger than ERH is to modify the algorithm slightly by starting over if finding a prime appears to take significantly longer than the expected number of iterations.

2. The analysis above assumes that the primality testing algorithm used in the algorithm checks for exact primality. In practice, however, efficiency reasons make probabilistic algorithms like Miller-Rabin, which have a small probability of returning `true` on input of a composite number, much more suitable for actual implementations. However, the number of Miller-Rabin rounds carried out ensures that this probability of erroneously recognizing a composite as prime is negligible. This means that the actual output distribution is in fact statistically close to the ideal distribution for which the primality testing algorithm is perfect. The "real" Δ is thus at most negligibly larger than the ideal one considered in this section.
3. For simplicity, we have only considered the generation of primes smaller than a given bound, but it is not difficult to extend our algorithms to the generation of primes in a fixed interval, using the same approach as [JP06].

3 Why entropy bounds may not be good enough

Previous works on the generation of prime numbers, such as [BD92,Mau95,JP06], provide a proof (based on rather strong assumptions) that the output distribution of their algorithm has an entropy not much smaller than the entropy of the uniform distribution. This is a reasonable measure of the inability of an adversary to guess which particular prime was

output by the algorithm, but it doesn't rule out the possibility of gaining some information about the generated primes. In particular, it doesn't rule out the existence of an efficient distinguisher between the output distribution and the uniform one.

Consider for example the PRIMEINC algorithm studied by Brandt and Damgåard in [BD92]. In essence, it consists in picking a random integer $y < x$ and returning the smallest prime greater or equal to y. There are some slight technical differences between this description and the actual PRIMEINC[1], but they have essentially no bearing on the following discussion, so we can safely ignore them.

It is natural to suspect that the distribution of the output of this algorithm is quite different from the uniform distribution: for example, generating the second prime of a twin prime pair, i.e. a prime p such that $p-2$ is also prime, is abnormally unlikely. More precisely, if one believes the twin prime conjecture, the proportion of primes of that form among all primes up to x should be $\sim 2c_2/\log x$, where $c_2 \approx 0.66$ is the twin prime constant. On the other hand, PRIMEINC outputs such a p if and only if it initially picks y as p or $p-1$. Therefore, we expect the frequency of such primes p in the output of PRIMEINC to be much smaller, about $4c_2/(\log x)^2$. This is indeed well verified in practice, as seen in Table 1, where we also check that our proposed algorithm doesn't exhibit such a bias.

n	64	128	256	512
Expected number	298	149	74	37
PRIMEINC	13	2	1	1
Trivial algorithm	275	129	76	34
Algorithm 1 with $\ell = n/8$	299	158	86	35

Table 1. Number of n-bit primes p such that $p-2$ is also prime output among 10000 primes generated by PRIMEINC, compared to the expected amount and the amount returned by more "uniform" algorithms. PRIMEINC is clearly an outlier.

More generally, it is easy to see that the method used in [BD92] to obtain the lower bound on the entropy of the output distribution of PRIMEINC can also provide a relatively large constant lower bound on the statistical distance to the uniform distribution! Indeed, the method relies on the following result, obtained by a technique first proposed by Gallagher [Gal76].

Lemma 3 ([BD92, Lemma 5]). *Assume the prime r-tuple conjecture, and let $F_h(x)$ denote the number of primes $p \leq x$ such that the largest prime less than p satisfies $p-q \leq h$.*

[1] To wit, Brandt and Damgåard restrict their attention to odd numbers $x/2 < y < x$, and set an upper bound to the number of iterations in the algorithm

Then for any constant λ,

$$F_{\lambda \log x}(x) = \frac{x}{\log x}\left(1 - e^{-\lambda}\right)(1 + o(1))$$

as $x \to +\infty$.

Now, the probability that the PRIMEINC algorithm outputs a fixed p can clearly be written as $d(p)/x$, where $d(p)$ is the distance between p and the prime that immediately precedes it. Let Δ' be the statistical distance between the output distribution of PRIMEINC and the uniform distribution. We have:

$$\Delta' = \sum_{p \le x} \left| \frac{d(p)}{x} - \frac{1}{\pi(x)} \right| > \sum_{\substack{p \le x \\ d(p) > 2 \log x}} \left| \frac{2 \log x}{x} - \frac{\log x}{x} \right|$$

for $x \ge 17$ in view of the already mentioned classical bound $\pi(x) > x/\log x$ for $x \ge 17$. By Lemma 3, this gives:

$$\Delta' > \frac{\log x}{x} F_{2 \log x}(x) = \left(1 - e^{-2}\right)(1 + o(1)) > 0.86 + o(1)$$

as $x \to +\infty$, and in particular, Δ' admits a relatively large constant lower bound (at least if one believes the prime r-tuple conjecture on which the entropy bound is based).

Since the entropy lower bounds for other prime generation algorithms like [JP06] are based on the same techniques, it is very likely that their output distributions can similarly be shown to be quite far from uniform. However, we have not been able to obtain the proof of the lower bounds stated in that paper to check whether the approach above generalizes to that setting.

4 Performance comparison

Let us simplify and summarize the performance estimates of Section 2.1 by giving asymptotic formulas for the efficiency of the various algorithms mentioned above, in terms of number of primality tests, random bits used, and statistical distance to the uniform distribution. These formulas are collected in Table 2.

Concretely speaking, we can precisely compute those efficiency parameters for the generation of, say, 512 or 1024-bit primes. We report those numbers in Table 3 and 4, together with timings for straightforward implementations of these algorithms using PARI/GP [PAR06].

Note that there is, of course, a trade-off between the number of random bits used and the uniformity of the distribution: both increase with ℓ (resp. m). A reasonable compromise is probably to choose ℓ just under the number of bits produced by each call to the random number generator of the system, so that a single word of randomness is produced for each

	Trivial	PRIMEINC	Joye-Paillier	Algorithm 1	Algorithm 2
Expected primality tests	$\frac{\log 2}{2} \cdot n$	$\frac{\log 2}{2} \cdot n$	$e^{-\gamma} \log 2 \cdot \frac{n}{\log n}$	$\frac{\log 2}{2} \cdot n$	$e^{-\gamma} \log 2 \cdot \frac{n}{\log n}$
Expected random bits	$\frac{\log 2}{2} \cdot n^2$	n	$E_\infty \cdot n$	$\frac{\log 2}{2} \cdot n\ell$	$e^{-\gamma} \log 2 \cdot \frac{n\ell}{\log n}$
Statistical distance	$= 0$	$\gtrsim 0.86$	$>$ constant?	$< 1.02n^2 \cdot 2^{-\ell/2}$	$< \frac{1.09n^2}{\log n} \cdot 2^{-\ell/2}$

Table 2. Comparison of the asymptotic efficiencies of the previous algorithms (assuming $1 \ll \ell \ll n$). The constant $E_\infty \approx 2.7$ is defined in Equation (1).

	Trivial	PRIMEINC	Joye-Paillier	Algorithm 1	Algorithm 2
Expected primality tests	177	177	35	177	35
Expected random bits	90,852	512	1098	11,805	3320
Statistical distance	$= 0$	$\gtrsim 0.86$	$>$ constant?	$< 2^{-12.9}$	$< 2^{-13.3}$
Average CPU time	0.19 s	0.20 s	0.04 s	0.19 s	0.04 s

Table 3. Practical efficiency of the various algorithms for the generation of 512-bit primes.

	Trivial	PRIMEINC	Joye-Paillier	Algorithm 1	Algorithm 2
Expected primality tests	355	355	61	355	61
Expected random bits	363,409	1024	2405	22,965	6205
Statistical distance	$= 0$	$\gtrsim 0.86$	$>$ constant?	$< 2^{-10.9}$	$< 2^{-11.4}$
Average CPU time	2.03 s	2.02 s	0.44 s	1.92 s	0.41 s

Table 4. Practical efficiency of the various algorithms for the generation of 1024-bit primes.

iteration of the algorithm. In our case, we picked $\ell = 64$ in Algorithm 1, and chose m as the product of all primes up to 337 or 691 (for 512 and 1024 bits) in Algorithm 2 ($\lfloor 2^n/m \rfloor$ is then 62 bit long in both cases).

As we can see, our algorithms are as efficient as comparable ones in terms of speed, and achieve provable bounds on the statistical distance at a much smaller cost than the only other such example, the trivial uniform generator. In fact, Algorithm 2 is as fast as the Joye-Paillier algorithm and has provably good statistical properties at a mere threefold increase in random bits requirement (as opposed to thousandfold for the trivial uniform generator).

References

BD92. Jørgen Brandt and Ivan Damgård. On generation of probable primes by incremental search. In Ernest F. Brickell, editor, *CRYPTO*, volume 740 of *Lecture Notes in Computer Science*, pages 358–370. Springer, 1992.

BDL91. Jørgen Brandt, Ivan Damgård, and Peter Landrock. Speeding up prime number generation. In Hideki Imai, Ronald L. Rivest, and Tsutomu Matsumoto, editors, *ASIACRYPT*, volume 739 of *Lecture Notes in Computer Science*, pages 440–449. Springer, 1991.

BS96. Eric Bach and Jeffrey Shallit. *Algorithmic Number Theory*, volume 1. MIT Press, 1996.

CS00. Ronald Cramer and Victor Shoup. Signature schemes based on the strong rsa assumption. *ACM Trans. Inf. Syst. Secur.*, 3(3):161–185, 2000.

Dav80. Harold Davenport. *Multiplicative Number Theory*, volume 74 of *Graduate Texts in Mathematics*. Springer, 2nd edition, 1980.

ESC05. D. Eastlake 3rd, J. Schiller, and S. Crocker. Randomness Requirements for Security. RFC 4086 (Best Current Practice), June 2005.

Gal76. Patrick X. Gallagher. On the distribution of primes in short intervals. 23:4–9, 1976.

HL22. Godfrey H. Hardy and John E. Littlewood. Some problems of 'partitio numerorum': III. on the expression of a number as a sum of primes. 44:1–70, 1922.

JP06. Marc Joye and Pascal Paillier. Fast generation of prime numbers on portable devices: An update. In Louis Goubin and Mitsuru Matsui, editors, *CHES*, volume 4249 of *Lecture Notes in Computer Science*, pages 160–173. Springer, 2006.

JPV00. Marc Joye, Pascal Paillier, and Serge Vaudenay. Efficient generation of prime numbers. In Çetin Kaya Koç and Christof Paar, editors, *CHES*, volume 1965 of *Lecture Notes in Computer Science*, pages 340–354. Springer, 2000.

Mau89. Ueli M. Maurer. Fast generation of secure rsa-moduli with almost maximal diversity. In *EUROCRYPT*, pages 636–647, 1989.

Mau95. Ueli M. Maurer. Fast generation of prime numbers and secure public-key cryptographic parameters. *J. Cryptology*, 8(3):123–155, 1995.

McC86. Kevin S. McCurley. The least r-free number in an arithmetic progression. *Trans. Amer. Math. Soc.*, 293:467–475, 1986.

Mih94. Preda Mihailescu. Fast generation of provable primes using search in arithmetic progressions. In Yvo Desmedt, editor, *CRYPTO*, volume 839 of *Lecture Notes in Computer Science*, pages 282–293. Springer, 1994.

Oes79. Joseph Oesterlé. Versions effectives du théorème de Chebotarev sous l'hypothèse de Riemann généralisée. *Astérisque*, 61:165–167, 1979.

PAR06. The PARI Group, Bordeaux. *PARI/GP, version 2.3.5*, 2006. Available from http://pari.math.u-bordeaux.fr/.

Wag79. Samuel S. Wagstaff, Jr. Greatest of the least primes in arithmetic progressions having a given modulus. *Math. Comp.*, 33:1073–1080, 1979.