Fast generation of prime numbers and secure public-key cryptographic parameters

Report**Author(s):**

Maurer, Ueli

Publication date:

1993

Permanent link:

<https://doi.org/10.3929/ethz-a-000915151>

Rights / license:

In Copyright - Non-Commercial Use Permitted

Originally published in:

ETH, Eidgenössische Technische Hochschule Zürich, Departement Informatik, Institut für Theoretische Informatik 201

Eidgenössische
Technische Hochschule
Zürich

Departement Informatik
Institut für
Theoretische Informatik

Ueli Maurer

**Fast Generation of
Prime Numbers and
Secure Public-Key
Cryptographic Parameters**

November 1993

Eidg. Techn. Hochschule Zürich
Informatikbibliothek
ETH-Zentrum
CH-8092 Zürich

ETH Zürich
Departement Informatik
Institut für Theoretische Informatik
Prof. Dr. J. Nievergelt

Author's address:

Prof. U.M. Maurer, Institut für Theoretische Informatik, ETH Zentrum,
CH-8092 Zurich, Switzerland. e-mail: maurer@inf.ethz.ch

© 1993 Departement Informatik, ETH Zürich

Abstract

A recursive algorithm for generating nearly random provable primes is presented. Heuristic arguments suggest that the generated primes are close to uniformly distributed over the set of primes in the specified interval, with only a small fraction of the primes being excluded. The expected time for generating a prime is less than 60% greater than the expected time required for generating a pseudo-prime that passes the Miller-Rabin test for only one base, and this number can even be reduced to 5% when certain deviations from the uniform distribution can be tolerated. For generating pseudo-primes, several Miller-Rabin tests with independent bases must be applied for achieving a sufficient confidence level; hence the presented algorithm is even faster than presently-used algorithms for generating only pseudo-primes.

Security constraints on the prime parameters of certain cryptographic systems are discussed, and in particular a detailed analysis of the iterated encryption attack on the RSA public-key cryptosystem is presented. The prime generation algorithm can easily be modified to generate nearly random primes or RSA-moduli that satisfy these security constraints. Further results described in this paper include an analysis of the optimal upper bound for trial division in the Miller-Rabin test as well as an analysis of the distribution of the number of bits of the smaller prime factor of a random k -bit RSA-modulus, given a security bound on the size of the two primes.

1. Introduction

A variety of cryptographic systems, including public-key distribution systems [25], [40], [52], public-key cryptosystems [26], [32], [42], [72], digital signature schemes [26], [72], [73], [75], [83] and identification protocols [28], [35], and a large number of variations of some of these systems have recently been proposed. The security of most public-key schemes is based on the (conjectured) difficulty of certain number-theoretic problems such as the factorization of large integers or the discrete logarithm problem in some finite group, and their public and/or private parameters include one or several large prime numbers. The purpose of this paper is to briefly review previous approaches to the generation of cryptographic primes and to present a new algorithm for generating prime numbers and secure public-key parameters. Our algorithm has the properties that it yields provable primes (as opposed to only probable primes or pseudo-primes), that the primes can be expected to be chosen randomly with sufficiently uniform distribution from the set of primes in a specified interval and, more importantly, that it is faster than all previous methods for generating even only pseudo-primes for cryptographic applications. Moreover, the algorithm is easily modified to generate primes that satisfy certain important cryptographic security constraints, without increasing the expected running time and without causing significant further deviation of the distribution from the uniform distribution over all primes in a given interval satisfying these conditions.

A large prime number can in principle be generated by repeatedly choosing an integer n at random from the specified interval and testing n for primality. The simplest primality test is to divide the given number n by all primes less than or equal to $\sqrt{n}$, but this approach is completely infeasible when the length of n exceeds 15-20 decimal digits. There exist several sophisticated general-purpose algorithms for testing primality [17], [57] (see also [44]). According to [59], the current record in primality testing is held by F. Morain [58] who proved the primality of a 1505-digit number of a general form using massive parallel computational resources.

The history of theoretical results on primality testing is long. Pratt [68] showed that the primes were recognizable in non-deterministic polynomial time, Miller [55] proved that the Riemann hypothesis for Dirichlet L-functions implied that the primes were recognizable in deterministic polynomial time, Adleman, Pomerance and Rumely [2] showed that the primes were recognizable in deterministic time $O((\log n)^{c \log \log \log n})$ for some constant c , and finally Adleman and Huang proved in a seminal report [1] that the primes were recognizable in random polynomial time. A significant step towards this result was achieved by Goldwasser and Kilian [31]. It is interesting to note that the primality tests used in practice [17], [57] have super-

polynomial running time and that the algorithm of [1] is superior only for very large numbers that are by far out of reach for presently-available computational resources.

There exist special purpose primality tests for numbers of certain special forms (for instance for Mersenne numbers which are of the form $2^q - 1$ where q is a prime [43]), but these primes can generally not be used in cryptography for security reasons. For instance, Mersenne prime factors of a given integer can be found easily.

In most present implementations of public-key cryptographic systems the primes are generated by application of a probabilistic compositeness test [55],[69],[78]. The most popular such test is the so-called Miller-Rabin test [69]. Let n be an integer to be tested and let $n - 1 = 2^u v$ with v odd. n passes the test for the base b if and only if either

$$b^v \equiv 1 \pmod{n}$$

or

$$b^{2^i v} \equiv -1 \pmod{n}$$

for some i satisfying $0 \leq i < u$. One can show that every composite number n can pass this test for at most $1/4$ of the bases b in the interval $[1, \dots, n-1]$. Hence the probability that a composite integer is not detected by t applications of the Miller-Rabin test with independent randomly chosen bases is at most $(1/4)^t$ [56]. In fact, much stronger results can be proved because for most composite integers the fraction of bases satisfying the above conditions is much smaller than $1/4$. Let $p_{k,t}$ denote the probability that, when odd k -bit integers are selected at random until one passes t consecutive independent Miller-Rabin tests, this integer is prime. Note that it does not follow from the described bound that $p_{k,t} \leq (1/4)^t$ [9]. However, Kim and Pomerance [38] and Damgård and Landrock [20] proved much stronger bounds on $p_{k,t}$. For instance, $p_{256,6} \leq 2^{-52}$ [20]. While for large enough t the error probability $p_{k,t}$ can be made sufficiently small for all practical purposes, the primality of an integer cannot be proved by a feasible number of such compositeness tests. However, such a proof would follow from the unproven extended Riemann hypothesis (see [55]). Alfons, Granville and Pomerance (cf. [34]) proved that for every given set of bases there exist composite numbers that pass the Miller-Rabin test for these bases. Bleichenbacher [11] exhibits a 55-digit composite number that passes the test for all bases ≤ 100 . One of the results proved by Bleichenbacher and Maurer [12] is that the Miller-Rabin test for the bases 2, 3, 5, 7, 11, 13 and 23 is a correct primality test for numbers $\leq 10^{16}$.

In this paper we consider the problem of generating random primes together with a *certificate of primality*. The generation of provable primes has previously been considered [19],[61],[76] but the major advantages of our algorithm are that it is

faster and that the diversity of primes that can be generated is much larger. Heuristic arguments suggest that a generated prime is close to uniformly distributed over a specified interval where only a small fraction of the primes is excluded for efficiency reasons. Moreover, our algorithm can, at no extra computational cost, be modified to generate a prime p that satisfies certain cryptographic security constraints.

The paper is organized as follows. Section 2 summarizes some number-theoretic results. The proposed algorithm for generating primes is described in Section 3. A running time analysis for this algorithm and for generating pseudo-primes is presented in Section 4, where the optimal limit for trial division in these algorithms is derived. Cryptographic security constraints on primes and RSA-moduli are discussed in Section 5. In particular, a detailed analysis of the iterated encryption attack against the RSA cryptosystem [72] is presented in Section 5.3, and the distribution of the size of the smaller prime factor of an RSA-modulus of a given size is analyzed in Section 5.4. Section 6 suggests an efficient method for generating nearly random secure RSA-moduli.

2. Number-theoretic Preliminaries

Throughout this paper, Z_n^* denotes the multiplicative group modulo n , $\text{ord}_n(x)$ denotes the order of x in Z_n^* , i.e., the smallest positive integer t satisfying $x^t \equiv 1 \pmod{n}$, and $\varphi(n)$ denotes Euler's totient function, i.e., the number of positive integers smaller than and relatively prime to n , with the exception $\varphi(1) = 1$. (a, b) denotes the greatest common divisor of a and b and the cardinality of a finite set S is denoted by $\#S$. All logarithms are to the natural base e . A basic fact about multiplicative orders is that for every $x \in Z_m^*$,

$$n|m \implies \text{ord}_n(x) | \text{ord}_m(x). \quad (1)$$

The following lemma, which is a key fact used in our algorithm, is a special case of a theorem due to Pocklington [62]. (see also [14] or [43]).

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

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

and

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

for $j = 1, \dots, r$, 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 is odd and $F > R$, then n is prime.

Proof. Let p be any prime dividing n . From the first and second condition on a it follows that $\text{ord}_p(a)$ divides $n - 1$ and that $\text{ord}_p(a)$ does not divide $(n - 1)/q_j$ for $j = 1, \dots, r$, respectively. Therefore $\text{ord}_p(a)$ is a multiple of $q_j^{\beta_j}$ for $j = 1, \dots, r$, hence also of F , and thus so is $p - 1$. The last claim of the lemma follows from the facts that at most one prime factor of a number can be greater than its square root and that when F is odd, the smallest possible prime factor of n is $2F + 1$; hence n can be composite only if $n \geq (2F + 1)^2$, which contradicts $F > R$. $\square$

The following lemma allows to prove the primality of an integer n even when the factored part is only greater than $\sqrt[3]{n}$ rather than $\sqrt{n}$. It will be pointed out that this lemma can be used to speed up the prime generation algorithm described in the following section at the expense of somewhat distorting the uniformity of the distribution of the generated primes.

Lemma 2. *Let n, R, F and a be as in Lemma 1 and let $x \geq 0$ and y be defined by $2R = xF + y$ and $0 \leq y < F$. If $F \geq \sqrt[3]{n}$ and if $y^2 - 4x$ is neither 0 nor a perfect square, then n is prime.*

Proof. Because $n \leq F^3$ and since according to Lemma 1 every prime factor of n is at least $F + 1$, n can have at most two prime factors. Assume n is composite, i.e., $n = (m_1F + 1)(m_2F + 1) = m_1m_2F^2 + (m_1 + m_2)F + 1$ for some $m_1 \geq m_2$. Hence $2R = m_1m_2F + m_1 + m_2$. Since $m_1m_2 < F$ and the choice $\langle m_1, m_2 \rangle = \langle F - 1, 1 \rangle$ violates $n \leq F^3$, it follows that $m_1 + m_2 < F$ and hence that $x = m_1m_2$ and $y = m_1 + m_2$. Substituting m_2 by $y - m_1$ in $x = m_1m_2$ gives $m_1^2 - ym_1 + x = 0$, which has a solution for m_1 in integers if and only if $y^2 - 4x$ is a perfect square or 0. $\square$

To show that a large integer is not a perfect square it suffices to find a small prime modulo which the number is a quadratic non-residue. Results that are similar to this lemma are described in [15] and [19], but the proofs appear to be more complicated. Note that Lemmas 1 and 2 can easily be generalized to allow a to be different for each q_j [14], but we will only make use of the special case because the verification of the simpler condition is computationally more efficient. Moreover, it will be demonstrated by Lemma 4 that in our application virtually every base a is successful in proving a number prime so that the generalization is unnecessary.

A few basic facts about Euler's φ -function are

$$\frac{\varphi(n)}{n} = \prod_{p|n} (1 - 1/p) \geq 1 - \sum_{p|n} 1/p, \quad (2)$$

where the product and summation are over all (distinct) prime divisors of n ,

$$\varphi(ab) \geq \varphi(a)\varphi(b) \quad (3)$$

with equality if and only if $(a, b) = 1$, and

$$\sum_{d|n} \varphi(d) = n. \quad (4)$$

Z_p^* is a cyclic group for every prime p and hence, for any divisor d of $p-1$,

$$\#\{x \in Z_p^* : \text{ord}_p(x) = d\} = \varphi(d). \quad (5)$$

Lemma 3. *Let p be a prime and d a divisor of $p-1$. Then*

$$\#\{x \in Z_p^* : d | \text{ord}_p(x)\} \geq \frac{\varphi(d)}{d}(p-1)$$

with equality if and only if $(d, (p-1)/d) = 1$.

Proof. Using (5), (3) and (4) one obtains

$$\begin{aligned} \#\{x \in Z_p^* : d | \text{ord}_p(x)\} &= \sum_{d' : d|d'|(p-1)} \varphi(d') = \sum_{k : k | \frac{p-1}{d}} \varphi(kd) \\ &\geq \sum_{k : k | \frac{p-1}{d}} \varphi(k)\varphi(d) = \varphi(d) \sum_{k : k | \frac{p-1}{d}} \varphi(k) = \varphi(d) \frac{p-1}{d}. \end{aligned}$$

The inequality holds with equality if and only if $(d, (p-1)/d) = 1$. $\square$

Remark: The proof demonstrates that Lemma 3 holds for any cyclic group when $\text{ord}_p(x)$ and $p-1$ are replaced by the order of x in the group and the order of the group, respectively, but this more general result will not be used in the paper.

The following lemma demonstrates that if n is prime then virtually every base a can successfully be used in Lemma 1 to prove this fact, provided that all q_j 's are sufficiently large (which will always be the case in our application).

Lemma 4. *Let $p = 2RF + 1$ be a prime with $F = \prod_{j=1}^r q_j^{\beta_j}$, $F > R$ and $(2R, F) = 1$, where $q_1, \dots, q_r$, are distinct primes. Then the probability that a randomly selected base $a \in Z_p^*$ is successful in proving the primality of p by Lemma 1 is equal to $\varphi(F)/F$ which is at least $1 - \sum_{j=1}^r 1/q_j$.*

Proof. $a^{p-1} \equiv 1 \pmod{p}$ is satisfied for every $a \in Z_p^*$. If $(2R, F) = 1$ then the two statements $F | \text{ord}_p(a)$ and

$$a^{(p-1)/q_j} \not\equiv 1 \pmod{p}$$

for $1 \leq j \leq r$ are equivalent. Application of Lemma 3 with $d = F$ and of (2) completes the proof. $\square$

3. A Recursive Algorithm for Generating Nearly Random Primes

An algorithm for generating provable primes with approximately uniform distribution over the set of primes in a given interval is described in this section. For efficiency reasons, a small fraction of the primes p in the interval will be excluded, namely those of the form $p = 2ap' + 1$ for a small a and with p' prime. In Section 5 we will discuss how the algorithm can be modified to generate nearly random primes satisfying certain security constraints

3.1. On the relative size of the prime factors of large integers

In the following, the ratio of the lengths of two integers will be an important quantity. Let the *relative size* of an integer a with respect to an integer b be defined as $\log a / \log b$. (Note that this definition is independent of the base to which the logarithms are computed.)

Let $p_i(n)$ denote the i th largest prime factor of the integer n and let $\omega_i(N, x)$ be the number of positive integers less than or equal to N for which the i th largest prime factor is at most N^x , i.e., let

$$\omega_i(N, x) = \#\{n : 1 \leq n \leq N, p_i(n) \leq N^x\}.$$

Knuth and Trabb Pardo [39] showed that

$$\lim_{N \rightarrow \infty} \frac{\omega_i(N, x)}{N} = F_i(x),$$

where the functions $F_i(x)$ are for $i \geq 1$ defined by the integral equations

$$F_i(x) = 1 - \int_x^1 \frac{dt}{t} \left(F_i \left(\frac{1}{1-t} \right) - F_{i-1} \left(\frac{1}{1-t} \right) \right)$$

with the convention that $F_0(x) = 0$ for all x and $F_i(x) = 1$ for $x \geq 1$, for $i \geq 1$. For example, if $1/2 \leq x \leq 1$ we have $t/(1-t) \geq 1$ for all $t \geq x$ and hence

$$F_1(x) = 1 - \int_x^1 \frac{dt}{t} = 1 + \log x$$

for $1/2 \leq x \leq 1$. It follows for instance that the probability that a randomly selected large integer¹ has a prime factor greater than its square root is $1 - F_1(1/2) = \log 2 = 0.693$. The functions $F_1(x)$, $F_2(x)$ and $F_3(x)$ are tabulated in [39]. A few more values of F_1 are $F_1(0.25) = 0.00491$, $F_1(1/3) = 0.0486$ and $F_1(0.4) = 0.130$. The function $x \mapsto F_1(1/x)$ is also known as the rho-function studied by Dickman [24]. A good algorithm for computing the Dickman rho-function is described in [21].

Consider the following process, suggested to the author by Eric Bach [5], for generating real-valued random variables $s_1, s_2, \dots$. We make use of auxiliary random variables $u_1, u_2, \dots$. First, u_1 is chosen uniformly from the interval $[0, 1]$, then u_2 is chosen uniformly from $[0, 1 - u_1]$, then u_3 is chosen uniformly from $[0, 1 - u_1 - u_2]$, and so on. The numbers $u_1, u_2, \dots$ are maintained in a list ordered in decreasing order. The elements of the ordered list are the numbers $s_1, s_2, \dots$. Although this is a conceptually infinite process, it can be stopped after the first r elements in the ordered list, $s_1, \dots, s_r$, are fixed and will not be modified anymore. After the generation of $u_1, \dots, u_d$, the values $s_1, \dots, s_r$ are fixed as soon as

$$s_r > 1 - \sum_{i=1}^d u_i$$

because in this case $u_{d+1}, u_{d+2}, \dots$ will be inserted into the list after s_r . The described procedure for generating $s_1, \dots, s_r$ satisfying (6) will be used in the algorithm presented in the remainder of Section 3, where it will be called `GenerateSizeList`. However, for reasons discussed below we will use the stronger condition $s_r > 1 - \sum_{i=1}^r u_i$, which is equivalent to

$$s_r > 1 - \sum_{i=1}^r s_i. \quad (6)$$

It is not difficult to see that the cumulative distributions $G_1, G_2, \dots$ of $s_1, s_2, \dots$, where $G_i(x) = \text{Prob}[s_i \leq x]$ for $i \geq 1$, satisfy the following integral equations:

$$G_i(x) = \int_0^x \left(G_i \left(\frac{x}{1-t} \right) + G_{i-1} \left(\frac{x}{1-t} \right) \right) dt$$

with the convention that $G_0(x) = 0$ for all x and $G_i(x) = 1$ for $x \geq 1$, for $i \geq 1$. Using the variable substitution $y := x/(1-t)$ one can show that $G_i(x) = F_i(x)$ for $i \geq 1$. Thus the random variables $s_1, s_2, \dots$ generated according to the described process are distributed according to $F_1, F_2, \dots$ subject to the conditions $s_{i+1} \leq s_i$

¹Of course, it is impossible to choose integers with uniform distribution. This imprecise wording should here and below be understood as meaning to choose an integer at random from the interval $[1, N]$ or $[cN, N]$ for some $c < 1$, and consider the limit as N goes to infinity.

for $i \geq 1$ and $\sum_{i=1}^{\infty} s_i = 1$. If this process is stopped after $s_1, \dots, s_r$ have been generated, then $s_1, \dots, s_r$ are distributed according to the joint distribution of the relative sizes of the r largest prime factors of an integer chosen uniformly from $[1, N]$ (or, equivalently, from $[cN, N]$ for a fixed interval spread $c < 1$), for very large N .

3.2. Outline of the algorithm

Lemma 1 suggests to construct a large prime by choosing some primes $q_1, \dots, q_r$, computing $F = \prod_{i=1}^r q_i^{\beta_i}$ for some exponents $\beta_1, \dots, \beta_r$, and repeatedly choosing integers $R < F$ at random until $n = 2RF + 1$ can be proved to be prime by an appropriate choice of the base a in Lemma 1. Lemma 4 shows that if n is prime, then finding such a base is easy. On the other hand, if n is composite, then virtually every base a will satisfy $a^{n-1} \not\equiv 1 \pmod{n}$ and hence be a witness for the compositeness of n , unless n is of a very special form (see [9],[16]). (Of course, in a reasonable implementation, n is first tested for small prime divisors before applying a modular exponentiation.)

Instead of choosing F and R sufficiently large at the beginning, the described construction approach can also be applied repeatedly by using some generated primes as factors of a larger F , thereby constructing larger and larger primes [76]. However, one major problem with this approach is that it is difficult to control the diversity of reachable primes and that this might endanger the security of a cryptosystem. For instance, factoring the product of two such primes could be significantly easier than solving a general instance of the factoring problem. Moreover, the algorithm of [76] is less efficient than our algorithm.

The goal of our algorithm, although it is based on the above construction, is that the primes be selected randomly with reasonably uniform distribution from the set of primes in a given interval I . This goal is achieved by generating sufficiently many of the largest prime factors $q_1, q_2, \dots$ of $(n-1)/2$ (only one or two are needed in most cases), each of appropriate size as described below. These prime factors are generated by recursive application of the algorithm. Depending on whether Lemma 1 or Lemma 2 is used for the primality proof of n , the factored part of $n-1$ must exceed $\sqrt{n}$ or $\sqrt[3]{n}$, respectively.

The sizes of the q_i , relative to $(n-1)/2$, are chosen according to the procedure described in Section 3.1 and are hence distributed according to the (asymptotic) distributions of the relative sizes of the largest, second largest, etc. prime factors of a very large integer. Note that one would actually have to use the conditional distribution of the relative sizes of the prime factors of an integer x , given that $2x+1$ is prime. Heuristic arguments for showing that this condition does not change the

asymptotic distribution are given in [51]. Further results supporting the idea that when p is prime, the factorization pattern of $p - 1$ does not differ greatly from that of a “random” number can be found in [8], [27], [29], [30], [36], [37], [41], [65], [67] and [82].

It may appear to the reader that Bach’s algorithm [4] for generating random factored integers is strongly related to this paper. However, in Bach’s approach the top goal is to achieve a uniform distribution whereas in our approach the top goal is efficiency. In particular, Bach’s algorithm makes use of a general primality test, and avoiding the use of such a test is exactly the point of this paper.

3.3. Description of the procedure RandomPrime

A listing of the PROCEDURE RandomPrime is shown in Figure 1. The listing is intended to serve as a guideline rather than a blueprint for an implementation, and some additional hints for an actual implementation are given below and in Section 3.4. We use a Pascal-like notation, where keywords are in capital letters. The same variable and constant names are used somewhat differently in the text and in the listing, but we believe that this should cause no confusion. Names with subscripts in the text are used in the listing by incorporating the subscript into the name (e.g., P_1 in the text corresponds to P1 in Figure 1). The functions Sqrt, Exponentiate, Random, TrialDivision, g_opt and CheckLemma1 as well as the procedures PrimeTest and GenerateSizeList are described in the text and the listing of Figure 1 shows only the function and procedure declarations, without implementations.

A variable of the type LongInt can represent integers of the size needed in a cryptographic context (e.g., up to 1024 bits). Such a type is often implemented (if no special-purpose hardware is available) by an array of integers whose first component contains the number d of active (non-zero) array components and whose first d components represent the integer in some fixed base (e.g., base 2^{16}). A set of procedures implementing the basic arithmetic operations for integers of the type LongInt are assumed to be available, but the calls to these procedures are not shown explicitly in Figure 1. Instead, the usual notation for integer operations (+, -, *, DIV) is used. Moreover, for the sake of simplicity, we allow numbers of type LongInt to be multiplied and divided by REAL numbers.

The FUNCTION Sqrt(a: LongInt): LongInt returns the square root of a (equivalent to a in the function declaration), rounded to the nearest integer or, in a more efficient implementation, some integer approximation of the square root of a depending only on the most significant bits and the length of a . The FUNCTION

Exponentiate(a: LongInt; e: REAL): LongInt returns the largest integer not greater than a^e , or a good approximation of this number. The **FUNCTION Random(a, b: LongInt): LongInt** selects an integer at random from the interval $[a, b]$ with uniform distribution.

The **PROCEDURE PrimeTest(a: LongInt): BOOLEAN** returns the value TRUE if and only if a is a prime, and needs to be efficient only for relatively small integers. For instance, PrimeTest could be implemented as trial division up to the square root of the tested number. This procedure is needed to end the recursion in the procedure RandomPrime when the primes to be generated are sufficiently small.

The **FUNCTION TrialDivision(a, b: LongInt): BOOLEAN** returns the value TRUE if and only if a is not divisible by a prime smaller or equal to b . This procedure requires a list of small primes, e.g., the primes less than or equal to $2^{16} = 65536$. The **FUNCTION g_opt(a: LongInt): LongInt** returns the optimal trial division bound (cf. Section 4) which minimizes the total time for detecting the compositeness of an integer.

The **FUNCTION GenerateSizeList(VAR rsl: RelSizeList; VAR r: INTEGER)** generates an ordered list of relative sizes of prime factors of an integer according to the procedure described in Section 3.1, where the number of prime factors is returned in the variable r .

The **FUNCTION CheckLemma1(n, a: LongInt, L: PFactorList; r: INTEGER): BOOLEAN** takes as input two integers n and a and a list $L = [q_1, \dots, q_r]$ of prime factors of $n - 1$, where the length of the list is given by the parameter r . It returns the value TRUE if and only if the two conditions of Lemma 1 are satisfied, which proves that n is prime. Note that in our algorithm, with probability about $\log 2 = 0.693$ only one prime factor q_1 is in the list L (i.e., $r = 1$), in which case the consecutive computation of $a^{(n-1)/q_1}$ and a^{n-1} corresponds to only one full modular exponentiation. If $r > 1$, the procedure CheckLemma1 requires slightly more than a full modular exponentiation. We refer to Section 4 for a running time analysis of the procedure CheckLemma1.

For given P_1 and P_2 , the **PROCEDURE RandomPrime(P1, P2: LongInt; VAR p: LongInt)** generates and returns a prime number p in the interval $[P_1, P_2]$ (for example the interval $[2^{255}, 2^{256} - 1]$ of 256-bit integers). When P_2 is smaller than a given constant P_0 (e.g., $P_0 = 10^7$), then the prime p can be generated by selecting integers at random from $[P_1, P_2]$ until a prime is found, which is checked by using the procedure PrimeTest, i.e., for instance by trial division up to its square root. This part of the procedure RandomPrime is needed to end the recursion described below.

When $P_2 > P_0$, the construction approach described in Section 3.2 is used. Let $P = \sqrt{(P_1 - 1)(P_2 - 1)}/2$ be the (approximate) geometric midpoint of the interval $[(P_1 - 1)/2, (P_2 - 1)/2]$. The number of primes in F is equal to the parameter r returned by the procedure `GenerateSizeList`, and the relative sizes $s_1, \dots, s_r$ of $q_1, \dots, q_r$ are chosen according to the list of relative sizes returned by the procedure `GenerateSizeList` in the variable `sl`. For each of these primes q_i with relative size $s_i = \text{sl}[i]$, the actual approximate size $Q \approx P^{s_i}$ is computed by the statement $Q := \text{Exponentiate}(P, \text{sl}[i])$. Q is taken as the geometric midpoint of an interval $[Q/c_{\text{int}}, Q \cdot c_{\text{int}}]$ where $c_{\text{int}} > 1$ is a small constant (e.g., $c_{\text{int}} = 1.2$). Then a prime is selected (approximately) at random from this interval by recursive application of the procedure `RandomPrime`.

After $F = \prod_{i=1}^r q_i$ is generated, integers R are chosen uniformly from the interval $[I_1, I_2]$, with $I_1 = (P_1 - 1)/(2F)$ and $I_2 = (P_2 - 1)/(2F)$, until $n = 2RF + 1$ is prime. Candidates n are tested first by trial division up to a bound determined by the procedure `g_opt` and then by the procedure `CheckLemma1` which checks whether the conditions of Lemma 1 for primality of n are satisfied.

3.4. Implementation issues and further comments

The described implementation of `RandomPrime` assumes that the spread P_2/P_1 of the interval $[P_1, P_2]$ is reasonably small (e.g., less than 4, but typically less than 2). If a prime should be selected uniformly from a larger interval, it is advisable to cut the interval into subintervals of reasonable spread, to select one of the intervals at random where the corresponding probability is assigned to each interval, and to use `RandomPrime` to generate a prime in the selected interval.

The reason why condition (6) is needed is to ensure that R , which is of relative size $1 - \sum_{i=1}^r s_i$, does not contain a prime factor greater than q_r . If it would, then the distribution of the relative sizes of the prime factors of $(n - 1)/2$ would differ from that of a random integer of the same size, which may be undesirable.

A problem with the procedure `RandomPrime` as described above is that when the relative size $1 - \sum_{i=1}^r s_i$ of R is too small, then the interval $[I_1, I_2]$ may be too small to contain an R for which $2RF + 1$ is prime. Of course, an endless execution of the `WHILE` loop must be prevented, for example by restricting the number of iterations. Furthermore, it must be avoided with high probability that the interval $[I_1, I_2]$ contains no prime factor because in this case F (or at least the smallest prime factor of F) would have to be regenerated. Allowing F to be rejected with non-negligible probability would strongly increase the running time of the algorithm, in particular because the rejection could happen at every level of the recursion. We

therefore recommend two modification to the procedure `RandomPrime` which are not described in Figure 1.

1. The output of the procedure `GenerateSizeList` should only be accepted if the sum of the relative sizes, $1 - \sum_{i=1}^r s_i$, is less than a given bound. We suggest to use the bound $1 - C_1/(\log P + C_2)$ for $C \approx 10$ and $C_2 \approx 50$. This modification reduces the diversity of reachable primes slightly, but this can be tolerated in most applications. In particular, primes p for which $(p - 1)/2$ is the product of a small number and a prime or the product of a small number and two primes of similar size cannot be reached. These numbers include the primes often referred to as *safe* primes, an attribute not justified sufficiently in the author's opinion. Discarding these special primes somewhat distorts the uniformity of the distribution, but has essentially no influence on the security of a cryptographic scheme. It is even conceivable that the so-called safe primes form a small class of primes that are actually insecure.
2. It is further advisable to let the interval constant c_{int} depend on P_2 , increasing when P_2 decreases, to ensure that the spread of the intervals passed to the procedure `RandomPrime` are always sufficiently large to guarantee that for the largest possible sum of relative sizes (as described above), the expected number of R 's resulting in a prime $2RF + 1$ is at least 5 for example. We suggest the choice $c_{int} = 1 + C_3/(\log Q + C_4)$ for $C_3 \approx 20$ and $C_4 \approx 10$.

Several ways to speed up the procedure `RandomPrime` by allowing slight further deviations from the uniform distribution are described in Section 4.4. Finally, it should be pointed out that the code of Figure 1 can be slightly improved in several ways (known to a good programmer) depending on the available processor and memory size.

4. Running Time Analysis for Generating Probable and Provable Primes

4.1. Efficient generation of pseudo-primes

Consider the problem of randomly selecting a k -bit strong pseudo-prime, i.e., a k -bit integer that passes the Miller-Rabin test for some base b . In a reasonable implementation, a selected odd candidate is tested for small prime divisors below a certain bound g before the Miller-Rabin test involving a computationally expensive full modular exponentiation is invoked. This can be done either by sequentially

dividing by 3, 5, 7, 11, 13, ... up to the greatest prime $\leq g$ or by computing greatest common divisors with products of several of the small primes. If too few small primes are tested, then an exponentiation is required in too many cases, but if too many small primes are tested, then the trial division step starts dominating. Let g_{opt} be the trial division bound that minimizes the expected running time for generating a pseudo-prime. Of course, g_{opt} depends on the size of the integers and on the particular implementation of long-integer arithmetic. However, one can show that, almost independently of the implementation, for all sufficiently large k (including the cases of interest in cryptography), $g_{opt}(k) \approx t_{exp}(k)/t_{div}(k)$ where $t_{exp}(k)$ and $t_{div}(k)$ are the times required for a full k -bit modular exponentiation and for ruling out one small prime as divisor of a k -bit integer, respectively. All the running times analyzed in this section are functions of the number k of bits of the integers, but for ease of notation the variable k will often be omitted.

Let $\pi(x)$ denote the number of primes less than or equal to x . It is well-known that the density of primes among the integers on the order of x is approximately $1/\log x$ and that $\pi(x) \sim x/\log x$. The probability that a randomly selected odd k -bit integer is prime is approximately $2/(k \log 2) = 2.89/k$.

Let R be a positive integer-valued random variable. The expected value of R is defined by $E[R] = \sum_{i=1}^{\infty} i \cdot P[R = i]$ and it is easy to verify that

$$E[R] = \sum_{i=1}^{\infty} P[R \geq i]. \quad (7)$$

A random odd integer has no prime factor smaller or equal to g with probability

$$d(g) \triangleq \prod_{3 \leq p \leq g} (1 - 1/p),$$

where here and in the following the variable p indicates that the product (or summation) is only over primes in the specified range. $d(g)$ is very well approximated by

$$d(g) \approx \frac{2e^{-\gamma}}{\log g},$$

where $\gamma \approx 0.5772$ is Euler's constant [70]. Hence $2e^{-\gamma} = 1.123$.

When a random odd integer is tested for compositeness by dividing by all primes less than or equal to g , starting with 3, the expected number of divisions that must be performed is, according to (7), given by

$$e(g) \triangleq 1 + \sum_{3 \leq p \leq g} d(p) = \alpha(g) \frac{g}{\log^2 g},$$

where $\alpha(g)$ depends only slightly on g and ranges from 1.87 to 1.40 when g ranges from 100 to 10^6 .

A random composite number n that fails to be detected by trial division by primes $\leq g$ is detected with overwhelming probability by the first Miller-Rabin test which requires one full exponentiation. The expected time required for detecting the compositeness of an odd integer is hence

$$E[t_c] = e(g)t_{div} + d(g)t_{exp}. \quad (8)$$

On the other hand, when the selected integer is prime, the time required to establish it as a pseudo-prime (for one base) is

$$t_p = (\pi(g) - 1)t_{div} + t_{exp}. \quad (9)$$

When a sequence of independent random experiments is performed where each experiment has a success probability ρ , then the expected number of trials required for one success is $1/\rho$. Hence the expected number of composite integers that need to be tested and discarded before a prime is found is well approximated by $k \log 2/2 = 0.347 k$, so that the total expected time for finding a pseudo-prime is

$$E[t_{pp}] \approx (\log 2/2) k E[t_c] + t_p. \quad (10)$$

In the following we determine which choice for the parameter g minimizes $E[t_{pp}]$. t_p depends only slightly on g and furthermore, t_p is negligible in (10) for large k . Hence we can almost equivalently determine g_{opt} minimizing $E[t_c]$ given in (8). Assume that the parameter g is increased to include the next larger prime $q > g$. If a number contains a prime $\leq g$, this modification has no effect on the running time. We therefore only consider the case where a number contains no prime $\leq g$, in which one has to perform one extra division by the additional prime. Given this case, the conditional probability that one can save an exponentiation is simply the probability that a number is divisible by the extra prime, which is $1/q \approx 1/g$. The expected running time is minimized when the increase and decrease in expected running time balance out, i.e., when $t_{div} = (1/g)t_{exp}$. Hence we have

$$g_{opt} = \frac{t_{exp}}{t_{div}}.$$

For this choice we can now express the expected running time $E[t_c]$ as a function of only t_{exp} :

$$E[t_c] = (e(g)/g + d(g)) t_{exp} \approx \left(\frac{2e^{-\gamma}}{\log g} + \frac{\alpha(g)}{\log^2 g} \right) t_{exp}$$

where $g = g_{\text{opt}} = t_{\text{exp}}/t_{\text{div}}$. For reasonably large k we thus have

$$\begin{aligned} E[t_{\text{pp}}(k)] &\approx \left(2e^{-\gamma} + \frac{\alpha(g(k))}{\log^2 g(k)} \right) \frac{\log 2}{2} k t_{\text{exp}}(k) \\ &\approx (0.39 + 0.55/\log g(k)) \frac{k}{\log g(k)} t_{\text{exp}}(k) \end{aligned} \quad (11)$$

where $g(k) = g_{\text{opt}}(k) = t_{\text{exp}}(k)/t_{\text{div}}(k)$. We have neglected $t_p(k)$ and assumed $\alpha(g(k)) = 1.6$

Let us find realistic figures for $E[t_{\text{pp}}]$ for integers of 100 and 200 decimal digits, i.e., $k = 332$ and $k = 664$, respectively. For the somewhat arbitrary but realistic values $g_{\text{opt}}(332) = 1000$ and $g_{\text{opt}}(664) = 4000$ we obtain $E[t_{\text{pp}}(332)] \approx 22.5 t_{\text{exp}}(332)$ and $E[t_{\text{pp}}(664)] \approx 36.5 t_{\text{exp}}(664)$. Note that the ratio $E[t_{\text{pp}}]/t_{\text{exp}}$ increases when a better implementation of exponentiation is used (for instance when exponentiation is performed on special-purpose hardware).

4.2. Analysis of the procedure CheckLemma1

The procedure CheckLemma1 receives the list $L = [q_1, \dots, q_r]$ of prime factors of $n - 1$ as a parameter, where the sizes $s_1, \dots, s_r$ of $q_1, \dots, q_r$ relative to $n - 1$ are generated by the procedure GenerateSizeList as described in Section 3.1. The verification of the conditions of Lemma 1 requires the computation of $a^{(n-1)/q_i}$ for $i = 1, \dots, r$ as well as $a^{(n-1)}$. We have $s_1 > 0.5$ with probability $\log 2 \approx 70\%$, in which case the list contains only one prime factor q_1 and the computation of $a^{(n-1)/q_1}$ and $a^{(n-1)}$ corresponds to one full modular exponentiation. When $r = 2$, i.e., $s_2 < s_1 < 0.5$ but $s_2 > 1 - s_1 - s_2$ (which happens with probability $\approx 25.8\%$), then the conditions of Lemma 1 can be checked by computing consecutively $A = a^{(n-1)/q_1 q_2}$, $B = A^{q_1} = a^{(n-1)/q_2}$, $C = B^{q_2} = a^{(n-1)}$ and $D = A^{q_2} = a^{(n-1)/q_1}$, where computing the first three terms is equivalent to one full exponentiation and where computing the last term corresponds to s_2 times a full exponentiation. For the general case $r > 2$ it is straight-forward to arrange the computation of $a^{(n-1)/q_i}$ for $i = 1, \dots, r$ and $a^{(n-1)}$ as a sequence of steps corresponding to $1 + \sum_{i=2}^r (i-1)s_i$ times a full modular exponentiation. A careful analysis shows that $E[1 + \sum_{i=2}^r (i-1)s_i] < 1.16$ for the algorithm described in Section 3.1. The probabilities that $r = 3$, $r = 4$ and $r = 5$ are approximately 4.4%, 0.45% and 0.035%, respectively.

4.3. Running time analysis for the procedure RandomPrime

We now consider the expected running time $E[t_{\text{RP}}(k)]$ of the procedure RandomPrime

described in the previous section. $E[t_{RP}(k)]$ is the sum of the expected time for generating the integer $F = \prod_{i=1}^r q_i$ and the expected time for generating a prime $p = 2RF + 1$ by random choices of R . It was demonstrated in the previous section that the detection of composite candidates by using the procedure `CheckLemma1` requires about 16% more time than by a simple exponentiation as used in the Miller-Rabin test; this is the price one has to pay for obtaining a proof of primality. Therefore, the expected time required for finding a suitable R is approximately $1.16 E[t_{pp}(k)]$. Under the simplifying but for this analysis admissible assumption that at each level of the recursion of `RandomPrime`, F consists of a single prime factor of relative size δ , $E[t_{RP}(k)]$ can be approximated by

$$\begin{aligned} E[t_{RP}(k)] &\approx E[t_{RP}(\delta k)] + 1.16 E[t_{pp}(k)] \\ &\approx 1.16 \sum_{i=0}^{\infty} E[t_{pp}(\delta^i k)] \\ &\approx \frac{1.16}{1 - \delta^{3.585}} E[t_{pp}(k)], \end{aligned}$$

where the term $1/(1 - \delta^{3.585})$ is obtained for the Karatsuba-Ofman implementation of long-integer arithmetic (see Section 4.5) with $E[t_{pp}(k)] = O(k^{3.585}/\log k)$, neglecting the $1/\log k$ factor. The above simplifying assumption implies $r = 1$ and hence that the procedure `CheckLemma1` would require exactly one modular exponentiation, but the factor 1.16 is included to take into account the possibility for multiple factors of F (i.e., $r > 1$).

The average of $\alpha^{3.585}$, where α is distributed according to $F_1(x)$ (cf. Section 3.1.), is approximately 0.26. When $\delta^{3.585}$ in the above expression is replaced by the average of $\alpha^{3.585}$, i.e., by 0.26, we obtain

$$E[t_{RP}(k)] \approx 1.57 \cdot E[t_{pp}(k)].$$

Simulations have suggested that this approximation is quite accurate [22], [79] i.e., that the expected running time of `RandomPrime` is less than 60% greater than the time required for generating a pseudo-prime. For a straight-forward (as opposed to Karatsuba-Ofman) implementation of long-integer arithmetic, the factor is smaller than 1.57.

4.4. Speeding up the procedure `RandomPrime`

The procedure `RandomPrime` can be sped up in different ways. In order to speed up the trial divisions when F is generated and candidates $2RF + 1$ are tested for several R , the remainders of F modulo the small primes can be stored such that for

every choice of R , only the remainders of R (rather than of n) modulo the small primes need to be computed. However, because most of the time is consumed by the exponentiations and not by the trial divisions, the achievable improvement is limited.

When the uniform distribution is not of central concern, the following modifications are possible. These modifications do not seem to endanger the security of a system significantly, but it should be pointed out that because the primes are generated recursively, deviations from the uniform distribution are amplified at each level of the recursion.

1. The factor 1.16 derived in Section 4.2 and appearing in the analysis of Section 4.3 can be avoided by restricting the relative size s_1 of q_1 to be larger than $1/2$ (i.e., $q_1 > \sqrt{(n-1)/2}$). As pointed out in Section 3.1., this condition is satisfied in about 70% of the cases and hence does not entail a dramatic distortion of the distribution.
2. A significant speedup can be achieved by using Lemma 2 for the primality proof (instead of Lemma 1), which requires only that the factored part F of $p-1$ be greater than $\sqrt[3]{p}$. In particular, when the relative size s_1 of q_1 is for instance restricted to be in the range $[1/3, 1/2]$, then $E[t_{RP}(k)]$ is only about 5% greater than $E[t_{pp}(k)]$.
3. It was pointed out by Mihailescu [54] that instead of generating R 's at random until $2RF+1$ is prime, it is somewhat more efficient to search for the prime in an appropriate interval of the arithmetic progression $1, 2F+1, 4F+1, \dots$. Note, however, that searching primes in an arithmetic progression has the effect that the probability that a certain prime is selected is proportional to the length of the interval of composite numbers preceding it in the progression, and that these intervals can vary significantly in length. It appears reasonable in most applications to tolerate the resulting distortion of the uniform distribution.

4.5. Asymptotic running time analysis

We now investigate the asymptotic running time of our algorithm. Let $M(k, l)$ denote the time required for multiplying a k -bit integer with an l -bit integer. A straight-forward implementation of integer multiplication has running time $M(k, l) = O(kl)$. In contrast, a sophisticated but not practical algorithm due to Schönhage and Strassen [74] (see also [3], pp. 270-274) has an asymptotic running time $M(k, k) = O(k \cdot \log k \cdot \log \log k)$ for multiplying two k -bit integers, which is only

slightly better than for FFT-based methods which, in contrast to the Schönhage-Strassen, are practical. However, in practical implementations for cryptographic purposes where the numbers have at most a few hundred decimal digits, it is preferable to use the asymptotically slower recursive algorithm of Karatsuba and Ofman (cf. [3], pp. 62-64) which multiplies two k -bit integers in time $O(k^{1.585})$.

Modular reduction can be implemented by a multiplication with the inverse of the modulus rounded to sufficient precision. Hence, based on the asymptotically fastest algorithm, we have

$$t_{\text{exp}}(k) = O(k \cdot M(k, k)) = O(k^2 \cdot \log k \cdot \log \log k).$$

We further have $t_{\text{div}}(k) = O(k \cdot \log(g(k)))$ which for the choice $g(k) = O(k \log \log k)$ is $t_{\text{div}}(k) = O(k \log k)$. Using (8) and (10) we thus obtain

$$E[t_{pp}(k)] = O(k^3 \log \log k)$$

whereas for an implementation based on the Karatsuba-Ofman algorithm we obtain $E[t_{pp}(k)] = O(k^{3.585} / \log k)$. A straight-forward implementation of integer arithmetic would result in $E[t_{pp}(k)] = O(k^4 / \log k)$. We refer to [7] for further analyses of prime generation algorithms.

5. Security Constraints for Public-key Cryptographic Parameters

The security of many cryptographic systems is based on the conjectured difficulty of solving a certain number-theoretic problem. For each of these problems there exist some special-purpose algorithms that can feasibly solve certain sets of special instances. It depends on the density of such special instances and on the security policy whether it is necessary to *guarantee*, by an appropriate design of the system parameters that a certain special-purpose algorithm is infeasible, or whether it is sufficiently secure to choose the parameters at random. It is often suggested to choose the system parameters such as to create the most difficult instance for some special-purpose algorithm. For example, it is suggested in [71] to choose primes p for the RSA-system of the form $2ap' + 1$ with $p' = 2bp'' + 1$ where p' and p'' are also primes and a and b are very small integers (e.g., $a = b = 1$), or it is suggested to choose primes p such that $p + 1$ contains a very large prime factor [33], [60]. However, it is conceivable that there exist special-purpose algorithms for solving instances in such a severely restricted parameter space, while the general problem may still be computationally intractable. Therefore, it is important to

balance reasonably between the diversity of the parameters and the feasibility of all the known special-purpose algorithms for solving the problem on which a system's security is resting.

Systems based on discrete logarithms and on factoring are discussed in Sections 5.1 and 5.2, respectively. In Section 5.3, the iterated encryption attack on the RSA system is analyzed and it is shown that such an attack can easily be thwarted by a simple modification in the procedure RandomPrime. It is often suggested to choose both primes in the RSA-system of the same length in bits. In Section 5.4 we investigate the implications of such a restriction by analyzing the distribution of the relative size of the smaller prime in an RSA-modulus, given a security bound on the size of the primes.

5.1. Systems based on discrete logarithms modulo p

Many systems are based on the discrete logarithm problem modulo a large prime p (e.g., [13], [25], [26], [35], [73], [83]). The fastest known general algorithm for computing discrete logarithms modulo p [18] seems at present to be infeasible for primes of more than 100 digits. We refer to [53] and [47] for a discussion of discrete logarithm algorithms.

The running time of the algorithm of Pohlig and Hellman [63] is on the order of the square root of the largest prime factor of $p - 1$ and hence it is a necessary condition for security that $p - 1$ contains at least one sufficiently large prime factor q_1 . The probability that a randomly selected integer has no prime factor greater than its 6-th, 8-th or 10-th root is $F_1(1/6) = 1.96 \cdot 10^{-5}$, $F_1(1/8) = 3.23 \cdot 10^{-8}$ or $F_1(1/10) = 2.8 \cdot 10^{-11}$, respectively (cf. [39]). A heuristic justification is given in [51] for the conjecture that integers of the form $(p - 1)/2$, where p is a prime, have the same distribution of the sizes of prime factors as random integers. Note that for instance with probability $3 \cdot 10^{-8}$, the largest prime factor of a 512-bit prime, reduced by 1, has at most 64 bits and that the Pohlig-Hellman algorithm appears to be feasible in this case.

While it is not necessary that $p - 1$ contains an extremely large prime factor, it appears nevertheless advisable for systems based on the discrete logarithm in Z_p^* to choose p of the form $p = 2Rq + 1$ where the relative size of q is at least $1/2$, or even higher (e.g. 0.9). It is often suggested to choose $R = 1$. This choice creates the most difficult instances for the Pohlig-Hellman algorithm, but may on the other hand be vulnerable against another (yet undiscovered) special-purpose discrete-logarithm algorithm. As described before, it may therefore be safer to require only that R be relatively small (e.g. having not more than 10-20 decimal digits), which also

offers the advantage that generating a prime p is faster. For a fixed choice of R , an expected number $(\log p)/2$ of primes q must be generated until $2Rq + 1$ is prime. In contrast, when R is allowed to lie in a certain range, only one prime q needs to be generated and R can be varied until $2Rq + 1$ is a prime.

In a discrete-logarithm based system the base b is usually suggested to be a generator of the multiplicative group of $GF(p)$. However, because of the algorithm of [63], it is almost equivalent from a security point of view to require only that q divides $\text{ord}_p(b)$, if the relative size of q is large. The algorithm of Section 3 for generating primes can easily be adapted to generate a prime with nearly uniform distribution over the set of primes p in a given interval for which $p - 1$ has a prime factor q of at least a certain specified size. The base a used in Lemma 1 for proving the primality of p satisfies $q \mid \text{ord}_p(a)$ and can thus be used as the base in discrete-logarithm based systems. Furthermore, when R is small and hence its factorization is easily obtained, the base a can be proved to be primitive (if it is) by checking that in addition to the conditions of Lemma 1, for every prime factor r of R , $a^{(n-1)/r} \not\equiv 0 \pmod{p}$. These additional checks can be performed very efficiently. According to Lemma 3, a random a is primitive with probability $\varphi(p-1)/(p-1)$ which is close to $1/2$ when R contains no very small prime factors, and slightly smaller if it does.

It is straight-forward to modify the procedure `RandomPrime` to generate a prime and a generator for the group satisfying the constraints for the Schnorr scheme [73] or the NIST proposal for a digital signature standard (DSS) [83].

5.2. Systems based on factoring

Another collection of systems is based on the difficulty of factoring a composite modulus [28], [42], [42], [72], [75]. The largest size of integers of general form that can presently be factored using massively parallel computation have on the order of 120 decimal digits [45]. These factoring records are achieved using variations of the quadratic sieve algorithm (e.g., see [46]), but the asymptotically fastest factoring algorithm is the number field sieve described in [48]. We refer to [47] and [66] for a discussion of factoring algorithms.

There exist many special-purpose factoring algorithms. Lenstra's elliptic curve algorithm [49] is successful in finding "small" factors having up to 40 decimal digits. Pollard's algorithm [64] finds factors p for which $p - 1$ has only relatively small prime factors. This algorithm was generalized by Williams [80] to primes for which $p + 1$ has no large prime factor and by Bach and Shallit [6] to primes for which any cyclotomic polynomial evaluated at p has no large prime factor, i.e., for which either $p - 1$, $p + 1$, $p^2 \pm p + 1$, $p^4 + p^3 + p^2 + p + 1$, etc., has no large prime factor.

It is therefore often recommended (e.g., [33]) to generate primes for which it is guaranteed that some of these expressions, in particular $p-1$ and $p+1$, each contains at least one large prime factor. However, it should be pointed out that, in view of the elliptic curve factoring algorithm [49], these conditions make little sense. For every fixed choice of elliptic curve parameters a and b , it is about equally likely that for instance $p+1$ is smooth with respect to a certain bound and that the order of the corresponding elliptic curve, $E_p(a, b)$, is smooth with respect to the same bound (see also [54]). The fact that the order of the elliptic curve cannot be given explicitly as an algebraic expression in p has no impact on the validity of this observation. However, it will be shown in Section 5.3 that a non-smoothness condition on $p-1$ can be justified for a different reason.

One way of deciphering ciphertexts in the RSA public-key cryptosystem [72] without factoring the modulus is by iterated encryption [77]. Section 5.3 gives a detailed analysis of this attack, and Theorem 6 states sufficient non-restrictive conditions on p and q that allow to foil this attack for any given public exponent e . These conditions can be satisfied at no extra computational cost by a simple modification in the procedure `RandomPrime`.

5.3. The iterated-encryption attack against the RSA system

The encryption transformation in the RSA-system is defined by

$$y \equiv x^e \pmod{m},$$

where x, y, e and m are the plaintext, ciphertext, public encryption exponent and public modulus, respectively. Because this transformation is known publicly, it can be iterated without knowledge of the secret key. Iterated t -fold encryption in an RSA cryptosystem reveals the plaintext x if and only if

$$x^{(e^u)} \equiv x \pmod{m}$$

for some $u \leq t$, i.e., if and only if

$$e^u \equiv 1 \pmod{\text{ord}_m(x)}$$

for some $u \leq t$. Hence the minimal number of encryptions needed to recover the plaintext is $\text{ord}_{\text{ord}_m(x)}(e)$; it is required for security reasons that this number be large for virtually all x . The following theorem is needed to prove Theorem 6 which states sufficient conditions for the iterated encryption attack to be infeasible.

Theorem 5. Let $m = pq$ be an RSA-modulus where $p-1 = 2R_p F_p$ and $q-1 = 2R_q F_q$ and where the prime factorizations of F_p and F_q are $F_p = \prod_{i=1}^r p_i^{\alpha_i}$ and $F_q = \prod_{i=1}^s q_i^{\beta_i}$, respectively. Then the fraction f of plaintexts $x \in Z_m^*$ for which $\text{ord}_m(x)$ is at least $\text{lcm}(F_p, F_q)$ satisfies

$$f \geq \frac{\varphi(F_p)}{F_p} \cdot \frac{\varphi(F_q)}{F_q} \geq 1 - \sum_{i=1}^r 1/p_i' + \sum_{i=1}^s 1/q_i'.$$

Proof. Lemma 3 states that

$$\#\{x \in Z_p^* : F_p | \text{ord}_p(x)\} \geq (p-1) \frac{\varphi(F_p)}{F_p}$$

and

$$\#\{x \in Z_q^* : F_q | \text{ord}_q(x)\} \geq (q-1) \frac{\varphi(F_q)}{F_q}.$$

For $x \in Z_m^*$ the conditions $F_p | \text{ord}_p(x)$ and $F_q | \text{ord}_q(x)$ together with (1) imply that $\text{lcm}(F_p, F_q)$ divides $\text{ord}_m(x)$. Because $Z_m^* = Z_p^* \times Z_q^*$ we have

$$\#\{x \in Z_m^* : \text{lcm}(F_p, F_q) | \text{ord}_m(x)\} \geq (p-1)(q-1) \frac{\varphi(F_p)}{F_p} \cdot \frac{\varphi(F_q)}{F_q}.$$

The last inequality of the theorem follows from (2). $\square$

Theorem 6. Let $m = pq$ be an RSA-modulus as in Theorem 5 where $p_i' - 1 = 2R_{p_i'} F_{p_i'}$ for $1 \leq i \leq r$ and $q_i' - 1 = 2R_{q_i'} F_{q_i'}$ for $1 \leq i \leq s$ and where the prime factorizations of $F_{p_i'}$ and $F_{q_i'}$ are $F_{p_i'} = \prod_{j=1}^{r_i} p_{ij}''^{\alpha_{ij}}$ for $1 \leq i \leq r$ and $F_{q_i'} = \prod_{j=1}^{s_i} q_{ij}''^{\beta_{ij}}$ for $1 \leq i \leq s$. For every integer a relatively prime to $(p-1)(q-1)$ and satisfying

$$a^{(p_i'-1)/p_{ij}''} \not\equiv 1 \pmod{p_i'}$$

for $1 \leq i \leq r$ and $1 \leq j \leq r_i$, as well as

$$a^{(q_i'-1)/q_{ij}''} \not\equiv 1 \pmod{q_i'}$$

for $1 \leq i \leq s$ and $1 \leq j \leq s_i$, the fraction of plaintexts $x \in Z_m^*$ for which $\text{ord}_{\text{ord}_m(x)}(a)$ is not a multiple of $\text{lcm}(F_{p_1'}, \dots, F_{p_r'}, F_{q_1'}, \dots, F_{q_s'})$ is at most $\sum_{i=1}^r 1/p_i' + \sum_{i=1}^s 1/q_i'$.

Proof. Similar arguments as used in the proof of Lemma 1 allow one to show that the first condition on a implies $F_{p_i'} | \text{ord}_{p_i'}(a)$ for $1 \leq i \leq r$. Hence $F_{p_i'} | \text{ord}_{p_i^{\alpha_i}}(a)$ and also $F_{p_i'} | \text{ord}_{F_p}(a)$ for $1 \leq i \leq r$. Thus $\text{lcm}(F_{p_1'}, \dots, F_{p_r'})$ divides $\text{ord}_{F_p}(a)$. Similarly

one obtains $\text{lcm}(F_{q'_1}, \dots, F_{q'_t}) \mid \text{ord}_{F_q}(a)$. It follows from (1) that $\text{lcm}(F_{p'_1}, \dots, F_{p'_t}, F_{q'_1}, \dots, F_{q'_t})$ divides $\text{ord}_{\text{lcm}(F_p, F_q)}(a)$. According to Theorem 5 we have for a fraction at least $1 - \sum_{i=1}^r 1/p'_i + \sum_{i=1}^s 1/q'_i$ of the $x \in Z_m^*$ that $\text{lcm}(F_p, F_q) \mid \text{ord}_m(x)$, which together with (1) implies that $\text{ord}_{\text{lcm}(F_p, F_q)}(a) \mid \text{ord}_{\text{ord}_m(x)}(a)$ and hence that $\text{lcm}(F_{p'_1}, \dots, F_{p'_t}, F_{q'_1}, \dots, F_{q'_t})$ divides $\text{ord}_{\text{lcm}(F_p, F_q)}(a)$, as was to be shown. $\square$

Theorem 6 illustrates that, in order to prevent decipherability by iterated encryption, the condition, suggested by Rivest [71] and others, that $p' - 1$ (where p' is the largest prime factor of $p - 1$) must also have a very large prime factor p'' , is unnecessary.

When the procedure `RandomPrime` is used to generate p and q , the primes p'_1 and q'_1 are generated on the first, and the primes p''_{ij} and q''_{ij} are generated on the second level of the recursion. Note that the conditions on a required by Theorem 6 to ensure that decryption by iterated encryption is infeasible, are satisfied automatically when $F_{p'_1}, \dots, F_{p'_t}, F_{q'_1}, \dots, F_{q'_t}$ are pairwise relatively prime and when the public encryption exponent e is used as the parameter a in the procedure `CheckLemma1` at the second level of the recursion, i.e., for proving the primality of the p''_{ij} and q''_{ij} . Hence the conditions of Theorem 6 can be satisfied at no extra computational cost, simply by controlling the choice of the parameter a and by avoiding the repeated use of primes.

Similar conditions (based on the factorization of $p + 1$ and $q + 1$ rather than $p - 1$ and $q - 1$) for preventing feasible decryption by iterated encryption can be derived for the elliptic-curve public-key cryptosystem of [42].

5.4. Random RSA moduli

It is usually recommended to implement the RSA system with a modulus $m = pq$ with two primes of equal length in bits. For a given size of the modulus this choice results in the most difficult instances for the elliptic curve factoring algorithm and also makes an implementation more symmetric when Chinese remaindering is used for decryption. This does not appear to be a severe restriction from a security point of view, but in the following we nevertheless investigate the problem of choosing an RSA-modulus $m = pq$ (with $p < q$) at random from the set of integers in a given interval $[cN, N]$ (with $0 < c < 1$) that are the product of two distinct primes and satisfy the constraint that both primes are sufficiently large, i.e., $p \geq N^\gamma$ for some γ satisfying $0 < \gamma < 1/2$ (e.g., $\gamma = 0.45$).

Let $\tau_2(N, \gamma)$ denote the number of integers $\leq N$ that are the product of exactly

two distinct primes, both greater or equal to N^γ , where γ is fixed with $0 < \gamma < 1/2$:

$$\tau_2(N, \gamma) = \#\{m : 1 \leq m \leq N, m = pq, N^\gamma \leq p < q\}.$$

Here and below p and q refer to primes. The following theorem was stated in [51] as a conjecture. It was pointed out to the author by Eric Bach [5] that it can be proved along the lines of the heuristic arguments given in [51] for its justification, by carefully estimating the error terms.

Theorem 7. For $0 < \gamma < 1/2$,

$$\lim_{N \rightarrow \infty} \tau_2(N, \gamma) \frac{\log N}{N} = \log(1 - \gamma) - \log \gamma.$$

Remark. The theorem also holds if $\tau_2(N, \gamma)$ is defined similarly, but for intervals $[cN, N]$ with arbitrary fixed positive spread $c < 1$, when the denominator N is replaced by $(c - 1)N$.

Proof. We have

$$\begin{aligned} \tau_2(N, \gamma) &= \sum_{N^\gamma \leq p < \sqrt{N}} \#\{q : p < q \leq N/p\} \\ &= \sum_{N^\gamma \leq p < \sqrt{N}} \pi(N/p) - \sum_{N^\gamma \leq p < \sqrt{N}} \pi(p). \end{aligned} \quad (12)$$

The second sum is at most equal to

$$\sum_{p < \sqrt{N}} \pi(p) < (\pi(\sqrt{N}))^2 = O(N/\log^2 N). \quad (13)$$

For p in the range of interest, N/p increases without bound. Therefore

$$\sum_{N^\gamma \leq p < \sqrt{N}} \pi(N/p) \sim \sum_{N^\gamma \leq p < \sqrt{N}} \frac{N/p}{\log(N/p)}.$$

Replacing this sum by a Stieltjes integral and using (12) and (13) we obtain

$$\frac{\tau_2(N, \gamma)}{N} \sim \int_{N^\gamma}^{\sqrt{N}} \frac{d\pi(x)}{x \log(N/x)} + O\left(\frac{1}{\log^2 N}\right). \quad (14)$$

Using $\pi(x) = \int_2^x dt/(\log t) + \epsilon(x)$ where $\epsilon(x) = O(x/\log^2 x)$ (see [23]), we obtain

$$\int_{N^\gamma}^{\sqrt{N}} \frac{d\pi(x)}{x \log(N/x)} = \int_{N^\gamma}^{\sqrt{N}} \frac{dx}{x \log x \log(N/x)} + \int_{N^\gamma}^{\sqrt{N}} \frac{d\epsilon(x)}{x \log(N/x)}. \quad (15)$$

The first integral can be computed by using the variable substitution $y = (\log x)/(\log N)$, with $dy = dx/(x \log N)$, which gives

$$\int_{N^\gamma}^{\sqrt{N}} \frac{dx}{x \log x \log(N/x)} = \frac{1}{\log N} \int_\gamma^{1/2} \frac{dy}{y(1-y)} = \frac{\log(1-\gamma) - \log \gamma}{\log N}. \quad (16)$$

Using the rule for integration in parts, $\int g(x)df(x) = f(x)g(x) - \int f(x)dg(x)$, for $f(x) = \epsilon(x)$ and $g(x) = 1/(x \log(N/x))$, allows us to transform the second integral in (15):

$$\int_{N^\gamma}^{\sqrt{N}} \frac{d\epsilon(x)}{x \log(N/x)} = \left[\frac{\epsilon(x)}{x \log(N/x)} \right]_{N^\gamma}^{\sqrt{N}} + \int_{N^\gamma}^{\sqrt{N}} \epsilon(x) \frac{1 + \log(N/x)}{x^2 \log^2(N/x)} dx. \quad (17)$$

We now make use of the bound $\epsilon(x) = O(x/\log^2 x)$ which shows that the first term on the right side of (17) is $O(1/\log^3 N)$. In order to show that the second term is $O(1/\log^2 N)$, note that the function to be integrated is $O(1/(x \log^2 x \log N))$ which for $N^\gamma \leq x \leq \sqrt{N}$ is also $O(1/(x \log^3 N))$, and that $\int_{N^\gamma}^{\sqrt{N}} dx/x = O(\log N)$. The proof of the theorem is completed by combining (14) - (17) $\square$

Consider now the distribution of the relative size of the smaller prime in a random RSA-modulus m of a certain size, given a security bound on the size of the primes. Theorem 7 states that when m is chosen at random from the set of integers $\in [cN, N]$ (for some fixed c) that are the product of exactly two primes, both $\geq N^\gamma$, then the probability that the smaller prime factor is at most N^α is, asymptotically, given by

$$1 - \frac{\log(1-\alpha) - \log \alpha}{\log(1-\gamma) - \log \gamma}. \quad (18)$$

It can easily be seen that for $\gamma > 0.4$, the relative size of the smaller prime factor of an RSA-modulus is close to uniformly distributed over the interval $[\gamma, 1/2]$, with sizes in the range of γ being slightly more likely than larger sizes in the range of $1/2$.

6. Generating random secure RSA moduli

Consider the problem of generating an RSA-modulus $m = pq$ on the order of N (e.g., $N = 2^{512}$ or $N = 2^{1024}$) for a given public encryption exponent e (e.g. $e = 3$). Let $p < q$. The primes p and q can be generated by the procedure RandomPrime by specifying suitable intervals and using e as the base in the primality proof on the second level of recursion.

A reasonable set of security constraints seems to be that

- (1) p as well as q must be greater than a given bound $L = N^\gamma$ for some $\gamma > 0.4$ and
- (2) $p-1$ and $q-1$ must contain distinct large prime factors p'_1 and q'_1 , respectively, with $p'_1 \geq L'$ and $q'_1 \geq L'$ for a given bound $L' = N^{\gamma'}$ for some γ' with $0.3 < \gamma' < \gamma$.

These two conditions, which are only mildly restrictive, imply that $p'_1 > \sqrt{p-1}$ and $q'_1 > \sqrt{q-1}$ and thus that only one prime factor p'_1 of $p-1$ and one prime factor q'_1 of $q-1$ must be generated. Moreover, since the factored parts $F_{p'_1}$ of p'_1 and $F_{q'_1}$ of q'_1 are larger than $\sqrt{p'_1-1}$ and $\sqrt{q'_1-1}$, respectively, the lower bound $\text{lcm}(F_{p'_1}, \dots, F_{p'_t}, F_{q'_1}, \dots, F_{q'_t}) = \text{lcm}(F_{p'_1}, F_{q'_1})$ of Theorem 6 is greater than $N^{1/4}$ when $(F_{p'_1}, F_{q'_1}) = 1$.

The above conditions can be satisfied by making appropriate use of (18) for selecting an interval for the smaller prime p , and by restricting the size of the largest prime factors of $p-1$ and $q-1$. When p is generated, the interval for the prime q is $[N_1/p, N_2/p]$, where $[N_1, N_2]$ is the specified interval for m .

7. Concluding Remarks

An important issue in the generation of cryptographic parameters is the trade-off between security constraints that must be placed on the parameters and the diversity of the parameters, i.e., the probability distribution according to which they are selected. We have provided a detailed analysis of this trade-off for the major cryptographic systems based on large prime numbers. A somewhat surprising additional benefit of our algorithm for generating nearly random provable primes satisfying given security constraints is that it is also more efficient than the widely-used algorithms for generating only pseudo-primes.

Acknowledgements

It is a great pleasure to thank Eric Bach for two significant contributions to the paper mentioned in the text as well as for suggesting several improvements for the presentation of the paper. I would also like to thank Daniel Bleichenbacher for many helpful discussions.

References

- [1] L.M. Adleman and M.A. Huang, Primality testing and abelian varieties over finite fields, *Lecture Notes in Mathematics*, Vol. 1512, Springer Verlag, 1992.
- [2] L.M. Adleman, C. Pomerance and R.S. Rumely, On distinguishing prime numbers from composite numbers, *Annals of Mathematics*, Vol. 117, pp. 173-206, 1983.
- [3] A.V. Aho, J.E. Hopcroft and J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Reading, MA: Addison-Wesley, 1974.
- [4] E. Bach, How to generate factored random numbers, *SIAM Journal on Computing*, Vol. 17, No. 4, pp. 173-193, 1988.
- [5] E. Bach, personal communication, April 1992.
- [6] E. Bach and J. Shallit, Factoring with cyclotomic polynomials, *Mathematics of Computation*, Vol. 52, pp. 201-219, 1989. *Proc. 26th IEEE Symp. on Foundations of Computer Science*, pp. 443-450, 1985.
- [7] E. Bach and J. Shallit, *Algorithmic number theory, Volume I: Efficient Algorithms*, MIT Press, to appear.
- [8] A. Balog, $p + a$ without large prime factors, *Seminaire de theorie des nombres de Bordeaux*, No. 31, 1983.
- [9] 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, pp. 53-64, 1988.
- [10] B. Blakely and G.B. Blakely, Security of number theoretic cryptosystems against random attacks, I, *Cryptologia*, Vol. 2, No. 4, pp. 305-320, Oct. 1978.
- [11] D. Bleichenbacher, On the power of pseudo-primality tests, Tech. Rep., Dept. of Computer Science, ETH Zurich, Sept. 1993.
- [12] D. Bleichenbacher and U.M. Maurer, Finding All Strong Pseudoprimes $\leq x$, preprint, 1993.
- [13] M. Blum and S. Micali, How to generate cryptographically strong sequences of pseudo-random bits, *SIAM Journal on Computing*, Vol. 13, No. 4, pp. 850-864, 1984.
- [14] D.M. Bressoud, *Factorization and Primality Testing*, Berlin: Springer-Verlag, 1989.
- [15] J. Brillhart, D.H. Lehmer and J.L. Selfridge, New primality criteria and factorizations of $2^m \pm 1$, *Mathematics of Computation*, Vol. 29, pp. 620-647, 1975.

- [16] R.D. Carmichael, On composite numbers P which satisfy the Fermat congruence $a^{P-1} \equiv 1 \pmod{P}$, *American Math. Monthly*, Vol. 19, pp. 22-27, 1912.
- [17] H. Cohen and A.K. Lenstra, Implementation of a new primality test, *Mathematics of Computation*, Vol. 48, No. 177, pp. 103-121, 1987.
- [18] D. Coppersmith, A.M. Odlyzko and R. Schroepel, Discrete Logarithms in $GF(p)$, *Algorithmica*, vol. 1, pp. 1-15, 1986.
- [19] C. Couvreur and J.J. Quisquater, An introduction to fast generation of large prime numbers, *Philips Journal of Research*, Vol. 37, pp. 231-264, 1982, (errata: id, Vol. 38., p. 77, 1983).
- [20] I. Damgård, P. Landrock and C. Pomerance, Average Case Error Estimates for the Strong Probable Prime Test, *Mathematics of Computation*, Vol. 61, pp. 177-194, 1993.
- [21] J. van de Lune and E. Wattel, On the numerical solution of a differential-difference equation arising in analytic number theory, *Mathematics of Computation*, Vol. 23, pp. 417-421, 1969.
- [22] R. De Moliner, Effiziente Konstruktion zufälliger grosser Primzahlen, Diploma Thesis, Inst. for Signal and Information Processing, Swiss Federal Institute of Technology, Zurich, 1989.
- [23] H.G. Diamond, Elementary methods in the study of the distribution of prime numbers, *Bulletin Am. Math. Soc. (New Series)*, Vol. 7, No. 3, 1982.
- [24] K. Dickman, On the frequency of numbers containing prime factors of a certain relative magnitude, *Arkiv for Matematik, Astronomi och Fysik*, Vol. 22A, No. 10, pp. 1-14, 1930.
- [25] W. Diffie and M.E. Hellman, New directions in cryptography, *IEEE Transactions on Information Theory*, Vol. 22, No. 6, pp. 644-654, 1976.
- [26] T. El-Gamal, A public key cryptosystem and a signature scheme based on the discrete logarithm, *IEEE Transactions on Information Theory*, Vol. 31, No. 4, pp. 469-472, 1985.
- [27] P. Erdős, On the normal number of prime factors of $p-1$ and some related problems concerning Euler's φ -function, *Quarterly Journal of Mathematics*, Oxford, Vol. 6, pp. 205-213, 1935.
- [28] A. Fiat and A. Shamir, How to prove yourself: practical solution to identification and signature problems, *Advances in Cryptology - CRYPTO '86*, Lecture Notes in Computer Science, Vol. 263, pp. 186-194, Berlin: Springer-Verlag, 1987.

- [29] J.B. Friedlander, Shifted primes without large prime factors, in *Number theory and applications*, R.A. Mollin (ed.), Kluwer Academic Publishers, pp. 393-401, 1989.
- [30] M. Goldfeld, On the number of primes p for which $p + a$ has a large prime factor, *Mathematika*, Vol. 16, pp. 23-27, 1969.
- [31] S. Goldwasser and J. Kilian, Almost all primes can be quickly certified, *Proc. of the 18th Annual ACM Symposium on the Theory of Computing*, pp. 316-329, 1986.
- [32] S. Goldwasser and S. Micali, Probabilistic encryption, *J. of Computer and System Sciences*, Vol. 28, pp. 270-299, 1984.
- [33] J. Gordon, Strong RSA Keys, *Electronics Letters*, Vol. 20, No. 12, 1984.
- [34] A. Granville, Primality Testing and Carmichael Numbers, *Notices of the American Math. Society*, pp. 696-700, 1992.
- [35] L.C. Guillou and J.-J. Quisquater, A practical zero-knowledge protocol fitted to security microprocessor minimizing both transmission and memory, *Advances in Cryptology - EUROCRYPT '88*, Lecture Notes in Computer Science, Vol. 330, pp. 123-128, Berlin: Springer-Verlag, 1988.
- [36] G.H. Hardy and J.E. Littlewood, Some problems of 'partitio numerorum'; III: on the expression of a number as a sum of primes, *Acta Mathematica*, Vol. 44, pp. 1-70, 1922.
- [37] C. Hooley, On the largest prime factor of $p + a$, *Mathematika*, Vol. 20, pp. 135-143, 1973.
- [38] S.H. Kim and C. Pomerance, The probability that a random probable prime is composite, *Mathematics of Computation*, Vol. 53, pp. 721-741, 1989.
- [39] D.E. Knuth and L. Trabb Pardo, Analysis of a simple factorization algorithm, *Theoretical Computer Science*, Vol. 3, pp. 321-348, 1976.
- [40] N. Koblitz, *A Course in Number Theory and Cryptography*, Berlin: Springer-Verlag, 1987.
- [41] N. Koblitz, Primality of the number of points on an elliptic curve over a finite field, *Pacific Journal of Mathematics*, Vol. 131, No. 1, pp. 157-165, 1988.
- [42] K. Koyama, U.M. Maurer, T. Okamoto and S.A. Vanstone, New public-key cryptosystem based on elliptic curves over the ring Z_n , *Advances in Cryptology - CRYPTO '91*, Lecture Notes in Computer Science, Vol. 576, pp. 252-266, Berlin: Springer-Verlag, 1992.

- [43] E. Kranakis, *Primality and Cryptography*, Stuttgart: Teubner, and New York: John Wiley & Sons, 1986.
- [44] A.K. Lenstra, Primality testing, in *Cryptology and computational number theory*, C. Pomerance (ed.), Proc. of Symp. in Applied Math., Vol. 42, pp. 13-25, American Mathematical Society, 1990.
- [45] A.K. Lenstra, Announcement by electronic mail on June 11, 1993.
- [46] A.K. Lenstra and M.S. Manasse, Factoring with two large primes, *Advances in Cryptology - EUROCRYPT '90*, Lecture Notes in Computer Science, Vol. 473, pp. 69-80, Berlin: Springer-Verlag, 1991.
- [47] A.K. Lenstra and H.W. Lenstra, Algorithms in number theory, Chapter 12 in *Handbook of Theoretical Computer Science*, J. van Leeuwen (ed.), MIT Press and Elsevier Science Publishers, 1990.
- [48] A.K. Lenstra, H.W. Lenstra, M.S. Manasse and J.M. Pollard, The number field sieve, *Proc. 22nd ACM Symposium on Theory of Computing*, pp. 564-572, 1990.
- [49] H.W. Lenstra, Jr., Factoring integers with elliptic curves, *Annals of Mathematics*, Vol. 126, pp. 649-673, 1987.
- [50] U.M. Maurer, Fast generation of secure RSA-moduli with almost maximal diversity, *Advances in Cryptology - EUROCRYPT '89*, Lecture Notes in Computer Science, Vol. 434, pp. 636-647, Berlin: Springer-Verlag, 1990.
- [51] U.M. Maurer, Some number-theoretic conjectures and their relation to the generation of cryptographic primes, in *Cryptography and Coding II*, C. Mitchell (ed.), pp. 173-191, Oxford University Press, 1992.
- [52] U.M. Maurer and Y. Yacobi, Non-interactive public-key cryptography, *Advances in Cryptology - EUROCRYPT '91*, Lecture Notes in Computer Science, Vol. 547, pp. 498-507, Berlin: Springer-Verlag, 1991.
- [53] K. McCurley, The discrete logarithm problem, in *Cryptology and computational number theory*, C. Pomerance (ed.), Proc. of Symp. in Applied Math., Vol. 42, pp. 49-74, American Mathematical Society, 1990.
- [54] P. Mihăilescu, Provable primes are fastest when seeked in arithmetic progressions, preprint, June 1993.
- [55] G.L. Miller, Riemann's hypothesis and tests for primality, *Journal of Computer and System Sciences*, Vol. 13, pp. 300-317, 1976.

- [56] L. Monier, Evaluation and comparison of two efficient probabilistic primality testing algorithms, *Theoretical Computer Science*, Vol. 12, pp. 97-108, 1980.
- [57] F. Morain, Distributed primality proving and the primality of $(2^{3^{539}} + 1)/3$, *Advances in Cryptology - EUROCRYPT '90*, Lecture Notes in Computer Science, Vol. 473, pp. 110-123, Berlin: Springer-Verlag, 1991.
- [58] F. Morain, Prime values of partition numbers and the primality of $p(1840926)$, Tech. Report LIX/92/RR/11, Laboratoire d'Informatique de l'Ecole Polytechnique (LIX), F-91128 Palaiseau Cedex, FRANCE, 1992.
- [59] F. Morain, personal communication, September 1993.
- [60] M. Ogiwara, A method for generating cryptographically strong primes, Research Reports on Information Sciences, No. C-93, Dept. of Information Sciences, Tokyo Institute of Technology, April 1989.
- [61] D.A. Plaisted, Fast verification, testing, and generation of large primes, *Theoretical Computer Science*, Vol. 9, pp. 1-16, 1979, (errata: id., Vol 14., p. 345, 1981).
- [62] H.C. Pocklington, The determination of the prime or composite nature of large numbers by Fermat's theorem, *Proceedings of the Cambridge Philosophical Society*, Vol. 18, pp. 29-30, 1914-1916.
- [63] S.C. Pohlig and M.E. Hellman, An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance, *IEEE Transactions on Information Theory*, Vol. 24, No. 1, pp. 106-110, 1978.
- [64] J.M. Pollard, Theorems on factorization and primality testing, *Proceedings of the Cambridge Philosophical Society*, Vol. 76, pp. 521-528, 1974.
- [65] C. Pomerance, Popular values of Euler's function, *Mathematika*, Vol. 27, pp. 84-89, 1980.
- [66] C. Pomerance, Factoring, in *Cryptology and computational number theory*, C. Pomerance (ed.), Proc. of Symp. in Applied Math., Vol. 42, pp. 27-47, American Mathematical Society, 1990.
- [67] K. Prachar, Über die Anzahl der Teiler einer natürlichen Zahl, welche die Form $p - 1$ haben, *Monatshefte für Mathematik*, Vol. 59, pp. 91-97, 1955.
- [68] V.R. Pratt, Every prime has a succinct certificate, *SIAM Journal on Computing*, Vol. 4, No. 3, pp. 214-220, 1975.
- [69] M.O. Rabin, Probabilistic algorithm for testing primality, *Journal of Number Theory*, Vol. 12, pp. 128-138, 1980.

- [70] H. Riesel, *Prime numbers and computer methods for factorization*, Boston, Basel, Stuttgart: Birkhäuser, 1985.
- [71] R.L. Rivest, Remarks on a proposed cryptanalytic attack on the M.I.T. public key cryptosystem, *Cryptologia*, Vol. 2, No. 1, pp. 62-65, Jan. 1978.
- [72] 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, pp. 120-126, 1978.
- [73] C.P. Schnorr, Efficient identification and signatures for smart cards, *Advances in Cryptology – CRYPTO '89*, Lecture Notes in Computer Science, Vol. 435, pp. 239-252, Berlin: Springer-Verlag, 1990.
- [74] A. Schönhage and V. Strassen, Schnelle Multiplikation grosser Zahlen, *Computing*, Vol. 7, pp. 281-292, 1971.
- [75] A. Shamir, Efficient signature schemes based on birational permutations, to appear in *Proc. of CRYPTO '93*.
- [76] J. Shawe-Taylor, Generating strong primes, *Electronics Letters*, Vol. 22, No. 16, pp. 875-877, 1986.
- [77] G. Simmons and M. Norris, Preliminary comments on the M.I.T public key cryptosystem, *Cryptologia*, Vol. 1, No. 4, pp. 406-414, Oct. 1977.
- [78] R. Solovay and V. Strassen, A fast Monte-Carlo test for primality, *SIAM Journal on Computing*, Vol. 6, No. 1, pp. 84-85, 1977 (errata: id., Vol. 7, p. 118, 1978).
- [79] G. Trenta, Werkzeuge zur Realisierung eines RSA-Kryptosystems, Diploma Thesis, Dept. of Computer Science, Swiss Federal Institute of Technology, March 1990.
- [80] H.C. Williams, A $p + 1$ method of factoring, *Mathematics of Computation*, Vol. 39, No. 159, pp. 225-234, 1982.
- [81] H.C. Williams and B. Schmid, Some remarks concerning the M.I.T. public-key cryptosystem, *BIT*, Vol. 19, pp. 525-538, 1979.
- [82] K. Wooldridge, Values taken many times by Euler's phi-function, *Proceedings of the AMS*, Vol. 76, pp. 229-234, 1979.
- [83] Specifications for a digital signature standard, US Federal Register, Vol. 56, No. 169, August 30, 1991.

```

PROCEDURE RandomPrime(P1,P2: LongInt; VAR p: LongInt);
CONST c_int = 1.2; rmax = 10; PO = 10000000;
TYPE PFactorList = ARRAY [1..rmax] OF LongInt;
    RelSizeList = ARRAY [1..rmax] OF REAL;
VAR a,n,I1,I2,P,Q,F,R: LongInt;
    i,g,r: INTEGER;
    b: BOOLEAN;
    sl: RelSizeList;
    pfl: PFactorList;
FUNCTION Sqrt(a: LongInt): LongInt;
FUNCTION Exponentiate(a: LongInt; e: REAL): LongInt;
FUNCTION Random(a,b: LongInt): LongInt;
PROCEDURE PrimeTest(a: LongInt): BOOLEAN;
FUNCTION TrialDivision(a,b: LongInt): BOOLEAN;
FUNCTION g_opt(a: LongInt): LongInt;
PROCEDURE GenerateSizeList(VAR rsl: RelSizeList; VAR r: INTEGER);
FUNCTION CheckLemma1(u,v: LongInt; L: PFactorList; r: INTEGER): BOOLEAN;
BEGIN
    IF P2 <= PO THEN BEGIN
        REPEAT
            n := Random(P1,P2);
        UNTIL PrimeTest(n);
        p := n; END;
    ELSE BEGIN
        GenerateSizeList(sl,r);
        P := Sqrt((P1-1)*(P2-1)) DIV 2;
        F := 1; g := g_opt(P);
        FOR i := 1 TO r DO BEGIN
            Q := Exponentiate(P,sl[i]);
            RandomPrime(Q/c_int,Q*c_int,pfl[i]);
            F := F*pfl[i];
        END;
        b := FALSE; I1 := (P1-1) DIV (2*F); I2 := (P2-1) DIV (2*F);
        WHILE NOT(b) DO BEGIN
            R := Random(I1,I2);
            n := 2*R*F+1;
            a := Random(2,P);
            IF TrialDivision(n,g) THEN b := CheckLemma1(n,a,pfl,r);
        END;
        p := n;
    END;
END.

```

Figure 1: The procedure RandomPrime in a Pascal-like notation. Only the declaration of functions and procedures are given; their implementations are described in Section 3.3.