Are ‘Strong’ Primes Needed for RSA?

Ronald L. Rivest*
Robert D. Silverman†

Dec 1, 1998

Abstract

We review the arguments in favor of using so-called “strong primes” in the RSA public-key cryptosystem. There are two types of such arguments: those that say that strong primes are needed to protect against factoring attacks, and those that say that strong primes are needed to protect against “cycling” attacks (based on repeated encryption).

We argue that, contrary to common belief, it is unnecessary to use strong primes in the RSA cryptosystem. That is, by using strong primes one gains a negligible increase in security over what is obtained merely by using “random” primes of the same size. There are two parts to this argument. First, the use of strong primes provides no additional protection against factoring attacks, because Lenstra’s method of factoring based on elliptic curves (ECM) circumvents any protection that might have been offered by using strong primes. The methods that ’strong’ primes are intended to guard against, as well as ECM, are probabalistic in nature, but ECM succeeds with higher probability. For RSA key sizes being proposed now, the probability of success of these methods is very low. Additionally, the newer Number Field Sieve algorithm can factor RSA keys with certainty in less time than these methods.

Second, a simple group-theoretic argument shows that cycling attacks are extremely unlikely to be effective, as long as the primes used are large. Indeed, even probabalistic factoring attacks will succeed much more quickly and with higher probability than cycling attacks.

1 Introduction

The RSA public-key cryptosystem [49] has received widespread acceptance in the commercial marketplace, in part because it has received intense scrutiny by academic cryptographers. The purpose of this paper is to extend this scrutiny by carefully examining the common recommendation that one should use “strong primes” when constructing keys for an RSA

*Supported by RSA Data Security, Inc. Address: Laboratory for Computer Science, Massachusetts Institute of Technology Cambridge, MA 02139; Email address: rivest@theory.lcs.mit.edu .
†Address: RSA Laboratories, 20 Crosby Dr., Bedford, MA. 01730, Email address: bobs@rsa.com

cryptosystem. We find that for practical purposes using large "random" primes offers security equivalent to that obtained by using "strong" primes. Current requirements for "strong" primes do not make them any more secure than randomly chosen primes of the same size. Indeed, these requirements can lead a general audience to believe that if a prime is "strong" that it is secure and that if it isn't "strong" then it must be "weak" and hence insecure. This simply is not true.

We begin in Section 2 with a general discussion of the notion of "weak keys".

Section 3 then provides a review of the essentials of the RSA public-key cryptosystem, mostly as a means of establishing notation.

In Section 4 we continue with a definition of a "strong prime." (Actually, we provide several definitions, corresponding to the various criteria that have been proposed in the literature.) In order to assure ourselves that strong primes are a practical possibility, we review two algorithms for finding strong primes in Section 5.

A quick survey of the history of the recommendation to use strong primes is given in Section 6. (Here the first author confesses to having played a part in this history of this recommendation.) We see that the recommendations are based on two basic concerns: a concern about the difficulty of factoring, and a concern about "cycling" attacks.

Section 7 discusses the requirement for "strong" keys, rather than just "strong" primes. Certain factoring attacks can exploit the relationship between the two primes that make up the key as opposed to exploiting the structure of the individual primes. The requirement for "strong" primes ignores these attacks completely. We shall show that suitably large random primes guard against these attacks.

In Section 8 we consider factoring attacks on the modulus. We see that before 1985 it appeared sensible to recommend using strong primes as a protection against certain factoring attacks, known as the "$p \pm 1$" attacks. Indeed, key sizes being proposed at that time *were* amenable to attack by these methods. The development of Lenstra's "elliptic curve method" [29] of factoring in 1985 is seen, however, to remove the motivation for using strong primes as a protection against the $p \pm 1$ factoring attacks. We shall present tables which give the probability of success for $p \pm 1$ and ECM attacks for a given level of effort.

In Section 9 we address the second concern: cycling attacks. Here we present a new theorem that shows that as long as the primes used are large, the cycling attacks are unlikely to succeed.

Finally, Section 10 closes with some recommendations.

We assume that the reader is familiar with elementary number theory, such as presented by Niven and Zuckerman [37], LeVeque [30], or Chapter 33 of Cormen et al. [8]. For surveys of factoring and number-theoretic algorithms, see Bach [2], Bressoud [6], Buell [7], Dixon [10], Guy [14], Knuth [22], Lenstra and Lenstra [26], Montgomery [36] Pomerance [40], and Riesel [44]. We assume that the reader is familiar with the elementary notions of cryptography (encryption, decryption, keys, etc.), as given, for example, in Davies and Price [9]. An understanding of cryptography is not essential, except as motivation for this paper. For discussions of the relationship between number theory and cryptography, see Kranakis [24], Pomerance [42], and Rivest [47].

2 Weak keys

In this section we review the notion of a "weak key," and attempt to characterize this notion more formally.

When a cryptographer selects a cryptographic key, he is creating an instance of a problem that his adversary would like to solve. Suppose the key selected is k, and that the set of all possible keys is denoted K. The adversary would like to determine k from some available data (messages encrypted with key k, etc.).

The adversary may employ either "general-purpose" or "special-purpose" methods in his attempt to determine k. Here we understand a general-purpose method as one that is always effective, no matter what k was chosen. An example of a general-purpose method is "brute-force search": trying each possible k in turn until the correct one is found. On the other hand, a special-purpose method is defined by the requirement that k be of a special form; if k is not of this form then the method will completely fail to find k. For this attack method, keys of the special form may be considered as "weak keys." For example, a particular attack method may only work against keys that terminate in 10 zeros; such keys would be "weak keys" for this attack method.

The notion of a "weak key" has appeared before in the literature. Specifically, the notion of a weak key has been defined for DES (the Data Encryption Standard). Davies and Price [9, Section 3.4] discuss the characteristics of weak (and semi-weak) DES keys and recommend that the DES user avoid using these keys in certain situations. We give this merely as an example, and do not pursue the issue of DES weak keys further in this paper.

When an adversary utilizes a special-purpose attack method, he is taking a gamble. He is betting that the cryptographer has been so careless (or unlucky) as to have chosen a weak key. If the cryptographer didn't choose a weak key, then the adversary gains nothing by his use of a special-purpose method—the effort will cost time and money and he would have been better off using a general-purpose method. On the other hand, the adversary may be lucky, the cryptographer may have chosen a weak key, and the adversary may be able to find k using his special-purpose method much more efficiently than he would using a general-purpose method. When is it worthwhile for the adversary to take such a gamble?

To answer this question, we propose the following "figure of merit" for an attack method: the expected amount of effort required for the adversary to find his first solution. We imagine that the adversary is given a sequence of problem instances to solve, each corresponding to a different cryptographic key. He applies his method to each instance in turn. He method may not be able to solve many of the instances; these may not be based on weak keys. He keeps trying until he has his first success. The total amount of effort he has expended up until his first success is the desired "figure of merit." We call this measure the *time-to-first-solution.*

We can make this notion more precise as follows. Let S denote the special-purpose method under consideration, and let K_S (a subset of K) denote the set of (weak) keys for which S is able to find a solution. The probability that the cryptographer has chosen a weak key is just $|K_S| \,/\, |K|$ (assuming that he chooses keys from K uniformly at random), and the expected number of instances the adversary has to consider before he solves one is $|K| \,/\, |K_S|$. Suppose that method S runs for time t_S on any problem instance, independent of whether or not the instance is based on a weak key. Then the expected value of S's time-to-first-solution is

$$T(S) = t_S \cdot \frac{|K|}{|K_S|} \ .$$

The general notion of "time-to-first-solution" is easily extended to handle variations on the above formalization (such as those cases where t_S depends on whether or not k is a weak key).

The notion of "time-to-first-solution" is a robust notion that can be applied to general-purpose methods as well as special-purpose methods. In this case, the notion becomes equivalent to that of the expected running time of the algorithm (since a general-purpose method always succeeds on the first instance).

We assert that a special-purpose method is of interest to an adversary only if its expected time-to-first-solution is less than that of other methods available, such as that of general-purpose methods. We call this assertion the "*Time-To-First-Solution Principle*." The Time-To-First-Solution Principle is not susceptible to proof, although various decision-analysis or utility arguments could be brought out in support of this principle. It is hard to imagine what possible reason an adversary could have for choosing a special-purpose method whose expected time-to-first-solution was greater than for other available methods. We assume the Time-To-First-Solution Principle in what follows.

We say a special-purpose method is *competitive* if its time-to-first-solution is smaller than that for the available general-purpose methods. Only if a special-purpose method S is competitive should its domain K_S be considered as being "weak" keys. In this case, the cryptographer should seriously consider modifying his key-selection procedure so as to avoid the weak keys in K_S.

As an example, consider a "partial brute-force search" method, which merely searches directly for k in a set K_S. Although the expected running time is only $t_S = |K_S|/2$ when $k \in K_S$, the time-to-first-solution of this method is just

$$(|K_S|/2)\frac{|K|}{|K_S|} = \frac{|K|}{2}$$

which is the expected running time of ordinary brute-force search. Therefore partial brute-force search is not competitive.

In the following sections we try to determine if there are any competitive special-purpose methods for attacking RSA, so that certain RSA keys should be deemed as "weak" and avoided.

We note here that truly weak keys can be constructed by anyone wanting to do so. One reason for doing so would be to allow repudiation of a message. If one deliberately generates a weak key, one could later claim that the key was weak, that someone else broke it and sent a bogus message. To guard against this, it should be a requirement that the random number seed, along with a specification of the exact procedure for generating primes be stored by the user. Now, an outsider could prove that the actual weak key was not generated according to the stated algorithm with the given seed.

Silverman and Wagstaff Jr. [51] analyze in detail the probability of success for ECM given the size of the prime and level of effort willing to be expended. Some of those results are summarized below in Section 7. In addition, they give estimates of the amount of time one should spend with ECM before changing to the Number Field Sieve in order to minimize the expected total time before one succeeds. The *time -to-first-solution* for special purpose methods is not competitive with what can be achieved with the Number Field Sieve.

3 The RSA public-key cryptosystem

We review the essentials of the RSA public-key cryptosystem, primarily as a means of introducing notation. See Rivest et al. [49] for a more complete description.

A user (say, Alice) of the RSA public-key cryptosystem secretly selects two large primes p and q, and computes their product n, known as her *public modulus*. She also selects an integer $e > 2$ such that

$$\gcd(e, (p-1)(q-1)) = 1 \ . \tag{1}$$

This e is known as her *encryption exponent*. Alice publishes the pair (e, n) as her *public key*. She then computes d such that

$$de \equiv 1 \pmod{\phi(n)} \ , \tag{2}$$

where $\phi(n)$ is Euler's totient function,

$$\phi(n) = |\{a : 0 < a \leq n \text{ and } \gcd(a, n) = 1\}| \tag{3}$$

which for $n = pq$ is $\phi(n) = (p-1)(q-1)$. The value d is her *decryption exponent*. Actually, equation (2) can be improved slightly to

$$de \equiv 1 \pmod{\lambda(n)} \ , \tag{4}$$

where

$$\lambda(n) = \mathrm{lcm}(p-1, q-1) \ . \tag{5}$$

Solving for d using equation (4) instead of equation (2) is slightly preferable because it can result in a smaller value for d. She retains the pair (d, n) as her *private key*.

Another user (say, Bob) can send Alice an encrypted message M, where $0 \leq M < n$, by sending her the ciphertext

$$C \equiv M^e \pmod{n}$$

computed using Alice's public key (e, n). When Alice receives C she can decrypt it using the equation

$$M \equiv C^d \pmod{n}$$

computed using her private key (d, n). Since no one but Alice possesses d, no one but Alice should be able to compute M from C.

The RSA system can also be used to sign messages. For reasons of efficiency the signing procedure typically uses a fixed public auxiliary *message digest* function H (also known as a *one-way* function or a *cryptographic hash* function). An example of such a function is given by Rivest [48]. The function H is fast to evaluate but computationally infeasible to invert. Alice computes her signature S for the message M using the equation

$$S = H(M)^d \pmod{n} \ .$$

The pair (M, S) form a $(message, signature)$ pair that can be verified by anyone as having been signed by Alice. To do so, one uses Alice's public key and checks that the equation

$$H(M) = S^e \pmod{n}$$

holds. If (and only if) the equation holds does one accept the signature as being a signature really produced by Alice. Since no one but Alice possesses d, no one but Alice can easily compute S from M, although anyone can easily verify the pair (M, S) as shown above.

In practice there arise many extensions and applications of the basic RSA scheme, such as the use of certificates and hybrid systems involving RSA and a conventional cryptosystem. Because our concern is with the security of the basic scheme, however, we do not discuss these issues here.

The security of RSA depends critically on the inability of an adversary to compute d from the public key (e, n). The problem of computing d from (e, n) is equivalent to the problem of factoring n into its prime factors p and q, as is proven by Bach et al. [3]. Therefore it is important for the RSA user to select primes p and q in such a way that the problem of factoring $n = pq$ is computationally infeasible for an adversary. Choosing p and q as "strong" primes has been recommended as a way of maximizing the difficulty of factoring n. In Section 8 we review the arguments in favor of using strong primes as a protection against factoring attacks.

The security of RSA also depends on the inability of a cryptanalyst to invert the basic RSA function even without knowing d. Some attacks along these lines, known as "cycling" attacks, have been proposed in the literature. In Section 9 we shall consider such attacks in detail.

4 What is a strong prime?

This section gives a definition of "strong prime" that includes all of the various conditions that have previously been proposed in the literature. We also provide terminology for some variant definitions.

We let $|p|$ denote the length of p in binary. The following definition uses English words such as "large prime" that are clarified with specific recommendations on sizes. (These are only nominal values; in practice one might wish to use even larger values.)

A prime p is considered to be a "strong prime" if it satisfies the following conditions.

- p is a large prime. (Say, $|p| \geq 256$.)
- The largest prime factor of $p - 1$, say p^-, is large. (Say, $|p^-| \geq 100$.) That is,
$$p = a^- p^- + 1 \tag{6}$$
for some integer a^- and large prime p^-.
- The largest prime factor of $p^- - 1$, say p^{--}, is large. (Say, $|p^{--}| \geq 100$.)
$$p^- = a^{--} p^{--} + 1 \tag{7}$$
for some integer a^{--} and large prime p^{--}.

- The largest prime factor of $p+1$, say p^+, is large. (Say, $|p^+| \geq 100$.) That is,

$$p = a^+ p^+ - 1 \tag{8}$$

for some integer a^+ and large prime p^+.

We denote the corresponding values for the prime q as q^-, q^{--}, q^+, b^-, b^{--}, and b^+.

Sometimes a prime is called strong if it satisfies only a subset of these conditions. To make our terminology precise, we say that a prime is

p^--*strong* if p^- is large,

p^{--}-*strong* if p^{--} is large,

p^+-*strong* if p^+ is large,

(p^-, p^+)-*strong* if both p^- and p^+ are large,

(p^-, p^{--}, p^+)-*strong,* or just *strong,* if all of p^-, p^{--}, and p^+ are large.

Sometimes a more stringent requirement is recommended (e.g., by Williams and Schmid [56]), specifying that we must have $a^- = 2$ (or $a^{--} = 2$, or $a^+ = 2$, etc.). We call such primes p^--*superstrong* (or p^{--}-*superstrong*, p^+-*superstrong*, etc.) Hellman and Bach [15] have also recommended that $p^+ - 1$ contain a large prime factor (which we would call p^{+-}).

5 Finding Strong Primes

Strong primes are somewhat rare. In practice, however, one can find strong primes without undue difficulty.

The original RSA paper [49] suggested the use of p^{--}-strong primes, which can be easily found as follows.

1. Find a large random prime p^{--} by testing large randomly chosen integers for primality until a prime is found.

2. Compute p^- as the least prime of the form

$$p^- = a^{--} p^{--} + 1 \tag{9}$$

   for some integer a^{--}. This can be accomplished by trying $a^{--} = 2, 4, 6, \ldots$ until a prime p^- is found. A probabilistic test for primality, such as the Miller-Rabin test [31, 43], can be used to test each candidate p^- for primality.

3. Compute p as the least prime of the form

$$p = a^- p^- + 1 \tag{10}$$

   for some integer a^-, in a similar manner.

The time required to find p in this manner is only about three times the time required to find a random prime of the same size.

Clearly, a prime p computed using the above procedure is p^{--}-strong. It may not be p^+-strong, however, and so Williams and Schmid [56] and Gordon [12] have suggested algorithms for finding primes that are p^+-strong as well (i.e., strong primes). We now give the algorithms for finding strong primes proposed by Williams and Schmid, and by Gordon.

5.1 The Williams/Schmid algorithm for finding strong primes

In 1979, Williams and Schmid proposed the following algorithm for finding strong primes.

1. Find p^{--} and p^+ as large random primes (say, 130 bits in length each).
2. Compute
$$r = -(p^{--})^{-1} \bmod p^+ \ . \tag{11}$$
Here $0 < r < p^+$. The inverse of p^{--} modulo p^+ can be computed using the extended form of Euclid's algorithm (see, for example, Cormen et al. [8, Section 33.4]).
3. Find the least a such that both
$$\begin{aligned} p^- &= 2ap^{--}p^+ + 2rp^{--} + 1 \ , \text{ and} && (12)\\ p &= 4ap^{--}p^+ + 4rp^{--} + 3 \\ &= 2p^- + 1 && (13) \end{aligned}$$
are prime.

It is easy to verify that p^+ is a factor of $p+1$. Note also that p is p^--superstrong, since $a^- = 2$. The lengths of p and p^- are about twice the lengths of p^{--} and p^+ with this procedure.

This procedure is noticeably *less* efficient than finding a random prime, since finding an a that simultaneously makes both p^- and p prime in equations (12)–(13) is much more difficult than, say, finding an a^- to make p prime in equation (10). (In general, one would expect that finding p^--superstrong primes would be much more difficult than merely finding large primes.) Nonetheless, this procedure is quite usable and effective for finding strong primes.

5.2 Gordon's algorithm for finding strong primes

In 1984 John Gordon [12, 13] proposed another procedure for finding strong primes. (It is not clear whether Gordon was aware of the previously published algorithm of Williams and Schmid.) Gordon argues that finding strong primes is only slightly harder than finding large random primes of the same size. His algorithm is more efficient than the Williams/Schmid algorithm because it does not create a p^--superstrong prime; the value of a^- will typically be considerably larger than 2 with Gordon's procedure. Gordon's algorithm is as follows.

1. Find p^{--} and p^+ as large random primes (say 130 bits in length each).

2. Compute p^- as the least prime of the form $p^- = a^{--}p^{--} + 1$, for some integer a^{--}.

3. Let
$$p_0 = \left((p^+)^{p^- - 1} - (p^-)^{p^+ - 1}\right) \bmod (p^-p^+) \ .$$
(For the correctness of Gordon's algorithm, note that Fermat's theorem implies that $p_0 \equiv 1 \pmod{p^-}$ and $p_0 \equiv -1 \pmod{p^+}$.)

4. Compute p as the least prime of the form
$$p = p_0 + ap^-p^+$$
for some integer a.

By controlling the sizes of the initial primes p^{--} and p^+, one can have reasonable control over the size of the prime p. Gordon explains how even greater control on the final size of p can be obtained by starting the searches based on a^{--} and a at larger starting values (e.g. starting a^{--} at 2^{12}). Note that the length of p^- and p^+ in bits will be approximately half the length of p, and that the length of p^{--} will be slightly less than that of p^-. In a typical application, one might have p^{--} of length 110 bits, p^- and p^+ of length 122 bits, and p of length 256 bits.

Gordon estimates the running time of his algorithm as follows. Let $T(k)$ denote the time to test a k-bit number for primality; we can assume that $T(k) = \Theta(k^3)$. Roughly k such k-bit numbers need to be tested for primality before a prime is found, by the prime number theorem. The naive algorithm for finding a k-bit prime by testing random k-bit numbers for primality thus requires time $\Theta(kT(k))$. Gordon's algorithm requires finding one k-bit prime after finding three $k/2$-bit primes, taking a total time of
$$kT(k) + 3(k/2)T(k/2) = 1.1875\,(kT(k)) \ ,$$
This justifies Gordon's claim that finding strong primes requires only 19% more work than the naive algorithm for finding random primes.

6 History of Strong Prime Recommendations

In this section we present, in chronological order, the various recommendations that have been made for the use of strong primes in the RSA cryptosystem.

In their original RSA paper, Rivest, Shamir, and Adleman [49] suggest finding primes p and q that are p^--strong "to gain protection against sophisticated factoring algorithms," and suggest the procedure given in the previous section for finding p^{--}-strong primes. At the time these suggestions were made, $p \pm 1$ was the state of the art in factoring algorithms. Furthermore, the size of keys being suggested at that time was such that the $p \pm 1$ methods actually did give a non-negligible chance of succeeding within a reasonable time. Improvements in algorithms and increases in key sizes have made this obsolete.

In 1977 Simmons and Norris [52] discussed the following "cycling" or "superencryption" attack on the RSA cryptosystem: given a ciphertext C, consider decrypting it by repeatedly encrypting it with the same public key used to produce it in the first place, until the message appears. Since the encryption operation effects a permutation of $\mathbf{Z}_n = \{0, 1, \ldots, n-1\}$, the message can eventually be obtained in this manner. Rivest [45] responds to their concern by (a) showing that the odds of success are minuscule if the n is the product of two p^{--}-strong primes, and (b) arguing that this attack is really a factoring algorithm in disguise, and should be compared with other factoring attacks.

In 1978 Herlestam [16] proposed a generalization of Simmons and Norris's attack, which can be viewed as a search for a multiple of the order of M, modulo n, using a trinomial equation to generate possible exponents. Rivest [46] argued that the generalization is no more effective than the original attack, when p and q are both p^{-}-strong.

In 1979, Blakley and Borosh [5] consider the number of messages M that are left unencrypted by RSA. That is, they consider messages that satisfy the equation

$$M^e \equiv M \pmod{n} .$$

For any modulus n there are at least 9 such messages (1, 0, and $n-1$ are examples). They prove that these 9 messages are the only such messages if the following condition holds in addition to equation (1):

$$\gcd(e-1, \mathrm{lcm}(p-1, q-1)) = 2 . \tag{14}$$

This condition is trivially satisfied if $e = 3$. If e is large and $e-1$ is divisible by some power 2^k of 2 for $k > 1$, then one might want to choose either p or q to be p^{-}-superstrong so that equation (14) holds. (This is possibly relevant for those who suggest standardizing the encryption exponent as $e = F_4 = 2^{16} + 1$.)

In 1979, Williams and Schmid [56] review the state of the art in factoring and cycling attacks, and generalize the Simmons/Norris attack to detect cycles modulo p, instead of cycles modulo n. They also propose a procedure (described above) for generating strong primes as a defense against such attacks.

In 1981, Knuth [22, page 387] argued that p and q should both be p^{-}-strong, to avoid Pollard's $p-1$ attack (described below in Section 8.1), and suggests checking to ensure that neither $p+1$ nor $q+1$ has only small prime factors. Knuth also describes a slight improvement to Pollard's $p-1$ method in exercise 4.5.4-19.

In 1982 Berkovits [4] apparently rediscovered the generalization of the Simmons/Norris attack given by Williams and Schmid. (He was apparently unaware of the Williams/Schmid paper, since he didn't reference it.) He concludes that if n is the product of two p^{--}-superstrong, p^{-}-superstong primes, then the attack is unlikely to succeed. He asserts that while the attack "is not much worse than guessing factors of n, it is not much better either."

In 1982 Hoogendoorn [17] recounted and summarized the above arguments and recommendations.

In 1983 Williams [55] suggested that n should be the product of two p^{-}-strong, p^{+}-strong primes p and q, to prevent n from being factored easily by a $p+1$ or a $p-1$ method. Williams notes, however, that "if p and q are large enough and selected at random, we should expect [this property] would automatically hold."

In 1984 Hellman and Bach [15] recommended that $p^+ - 1$ contain a large prime factor (which we would call p^{+-}). They do not, however, give a justification for this recommendation.

Jamnig [20] studies the Berkovits [4] attack, and recommends that a^- be small (on the order of $\log(p)$) and that b^- (the cofactor of q^- in $q-1$) be similarly small. On the basis of his analysis he concludes that Gordon's procedure is likely to yield keys that are vulnerable to a cycling attack since it produces a large value for a^-. (We disagree with his conclusion—the most he can say is that his proof technique is insufficient to prove Gordon's method secure, not that his proof technique demonstrates that Gordon's method is insecure.) Jamnig also disagrees with the referee's suggestion that Lenstra's elliptic curve method makes using strong primes irrelevant as a means of protecting against factoring attacks. (The first author supposes he should admit that he was the referee; this paper fleshes out his earlier suggestion to Jamnig.)

Most recently, the X.509 standards committee has published a draft standard [19] recommending the use of strong primes. Their motivation for this recommendation is Gordon's article [12], which references Knuth [22] and Rivest et al. [49] as justification. We remark that both of these references have been made obsolete by more recent factoring algorithms.

7 Strong Keys

In addition to requirements for "strong" primes in the literature, there have also appeared requirements for constructing "strong" keys. Two of the older factoring methods, one dating back to Fermat, and the other to 1974 by R. Sherman Lehman, attempt to factor a number by representing it as the [perhaps weighted] difference of two squares. These attacks succeed any time the *difference* between the two primes comprising the key is very small, or when their ratio is very close to the ratio of two small integers.

If the ratio of the two primes p/q is very near a/b where a and b are small integers, then the amount of work needed to factor pq is approximately $aq + bp - 2\sqrt{abpq}$. Even if p and q are very nearly identical (say their 100 most significant bits are identical), the amount of work needed is in excess of $2^{288} \sim 10^{86}$ computer operations. It would take 10^{66} 1000-MIPS computers about 100,000 years to do this computation. The probability that randomly chosen p and q match to the first k bits is 2^{1-k}. (since the first bit is always 1). Different values of a and b give similar results.

8 Strong primes as a protection against factoring

The first attack that a cryptanalyst might attempt to mount against an instance of the RSA cryptosystem is a factoring attack. To *factor* a natural number n is to produce a complete list of its prime factors. To *split* a natural number n is to produce two natural numbers, neither of which is 1 or n, whose product is n. When n is the product of exactly two primes, the notions of factoring n and splitting n are equivalent. Strictly speaking, most "factoring" algorithms are really "splitting" algorithms, but a splitting algorithm can be applied recursively to yield a factoring algorithm.

It useful to introduce the notion of "smoothness". We say that x is a *prime-power factor* of n if x divides n and $x = p^k$ for some prime p and integer k. We say that n is *B-smooth* if all of its prime-power factors are less than the *factor-bound* B.

There are many factoring algorithms available for an enemy cryptanalyst to use in an attempt to factor a public modulus n. The efficiency of these algorithms can depend on many things. Broadly speaking, we can classify algorithms into three categories according to what the running time of the algorithm depends upon. We assume that the number to be factored is $n = pq$, where p and q are prime.

1. *Algorithms whose running time depends mainly on the size of n.* The most notable algorithms in this class are the quadratic sieve algorithm [41] and the general number field sieve [27].

2. *Algorithms whose running time depends on the size of p and the size of q.* These algorithms are particularly good at factoring numbers when one of p or q is small. The naive heuristic of trial division and Pollard's "rho" heuristic [39] are members of this class. Lenstra's elliptic curve method [29] is the most notable algorithm in this class.

3. *Algorithms whose running time depends on the size of p^-, p^{--}, or p^+ (or of q^-, q^{--}, or q^+).* The $p-1$ and $p+1$ methods of factoring are the most notable members of this class.

4. *Algorithms whose running time depends on the 'closeness' of p and q* Fermat's method [22] and Lehman's method [25] fall into this category.

At the moment, for factoring large numbers the most effective algorithms seem to be Lenstra's elliptic curve method [29], [51], which is especially good if the number has small prime factors, Pomerance's quadratic sieve method, and the number field sieve. [41], [27]. While the number field sieve is asymptotically faster than the quadratic sieve, when it was first introduced it was thought that it did not become faster than QS until between 130 and 150 decimal digits. More recent work by Montgomery and Huizing has shown than the crossover point is somewhere between 100 and 110 decimal digits [18], although the exact crossover point will always be machine and implementation dependent.

We begin with a review of the factoring attacks that motivate the recommendation for strong primes, beginning with a review of the "$p-1$ method" of Pollard [38]. We then review the generalizations of the $p-1$ method: the $p+1$ method and Lenstra's elliptic curve method.

8.1 Pollard's $p-1$ Method for Factoring

Pollard's $p-1$ method [38] is the standard justification for recommending the use of p^--strong primes.

The heart of Pollard's $p-1$ method is a technique for splitting a given composite number n that is divisible by at least two distinct primes, using any given multiple m of $p-1$, for some prime factor p of n. The presentation here follows the more modern presentation of Bach et al. [3].

How can one obtain a multiple m of $p-1$? Suppose that p is "weak" in the sense that $p-1$ is B_1-smooth, for a suitably small factor-bound B_1. Although we don't know B_1, we can perhaps be lucky and guess a suitable value for B_1 (say, 10^6). Then if we let m be the product of all prime powers less than B_1, we have that m is a multiple of $p-1$.

Given a multiple m of $p-1$, the following method can now be used to split the odd number n.

1. Choose an element a at random from $\mathbf{Z}_n^* = \{1, 2, \ldots, n-1\}$.
2. Compute $d = \gcd(a, n)$. If $d > 1$, report that d is a factor of n and halt.
3. Compute $x = a^{mn} \bmod n$.
4. Compute $d = \gcd(x-1, n)$. If $d > 1$, report that d is a factor of n and halt.
5. If $x = 1$ and m is even, set $m \leftarrow m/2$ and return to step 3.
6. Report failure to find a factor. Halt.

Bach et al. prove that this algorithm has a 50% chance of picking an a that causes n to be split. By running this algorithm several times, then, we are virtually certain of splitting n.

The reason why $p-1$ is special is that $p-1$ is the size of the multiplicative group $\mathbf{Z}_p^*$ modulo p. Computing a power of a in step 3 is done modulo n, but the result is also correct modulo p by the Chinese remainder theorem. Since $a^{p-1} \equiv 1 \pmod{p}$ for all $a \in \mathbf{Z}_p^*$, we see that step 4 is likely to return p as a factor of n.

A second "FFT" stage of the algorithm, also due to Pollard [38], enables the method to factor n if $p-1$ contains a *single* prime factor p^- between B_1 and some second bound B_2, where $B_2 >> B_1$, and if all other prime factors of $p-1$ are less than B_1. For example, we might have $B_1 = 10^6$ and $B_2 = 10^{10}$. The approach, in its most effective form, makes use of the Fast Fourier Transform. Montgomery and Silverman [34] demonstrate the practicality of using a variant of the second stage that implements the FFT using residue arithmetic, but note that "on average the Elliptic Curve algorithm will still be more effective than the $p-1$ algorithm, even with our enhancements."

8.2 The $p+1$ Method for Factoring

A variation on the $p-1$ method, called the "$p+1$ method", has been described in 1975 by Guy [14], who attributes the idea to J. H. Conway, and in 1982 by Williams [54]. Using Lucas sequences instead of raising to large powers allows one to factor n if $p+1$ is smooth. Bach et al. [3] give an alternative interpretation of this method as the analogue of the $p-1$ method, working now over $GF^*(p^2)$, the multiplicative group in the extension field of degree two over $GF(p)$, rather than over $\mathbf{Z}_p^*$. The $p+1$ method motivates the use of p^+-strong primes, since if $p+1$ is smooth then n can be easily factored. The FFT second stage of the $p-1$ algorithm generalizes to the $p+1$ algorithm as well; see Montgomery and Silverman [34] for details.

8.3 The Elliptic Curve Method for Factoring

In 1985 Hendrik W. Lenstra, Jr., discovered a factoring algorithm [29] that upset all of the previous wisdom about "strong primes." Stephens [53] gives an alternative exposition of this method, and Montgomery [33], [35] describes some implementation details and improvements. For additional background, note that Koblitz [23], Kaliski [21] and Miller [32] propose some ways that elliptic curves can be useful in cryptography.

Based on "elliptic curves," this method generalizes the $p-1$ method even further, by replacing the multiplicative group modulo p with the group of points on a random elliptic curve E defined modulo p.

Specifically, an elliptic curve $\mathbf{E}_{a,b}$ is defined (modulo p) by two values a and b such that

$$4a^3 + 27b^2 \not\equiv 0 \pmod{p} . \tag{15}$$

To define the curve, we first define an equivalence relation on the set $\mathbf{Z}_p^3 - \{(0,0,0)\}$ of triples of values from $\mathbf{Z}_p$, not all zero, by saying that (x,y,z) is equivalent to (x',y',z') if there exists a nonzero c such that $x \equiv cx' \pmod{p}$, $y \equiv cy' \pmod{p}$, and $z \equiv cz' \pmod{p}$. The equivalence class of (x,y,z) is denoted $(x:y:z)$. The elliptic curve is then the set of points

$$\mathbf{E}_{a,b} = \{(x:y:z) : y^2 z \equiv x^3 + axz^2 + bz^3 \pmod{p}\} .$$

It is possible to define a binary operation $\oplus$ on $\mathbf{E}_{a,b}$ so that $(\mathbf{E}_{a,b}, \oplus)$ forms an abelian group, analogous to multiplication modulo p. (Details are omitted in this paper, see Lenstra [29].)

Lenstra's method can be viewed as a generalization of Pollard's $p-1$ method, where the operations are now taking place in the elliptic curve group $(\mathbf{E}_{a,b}, \oplus)$ rather than the multiplicative group $(\mathbf{Z}_p, \cdot)$ modulo p. As a result, the critical parameter now becomes $|\mathbf{E}_{a,b}|$ rather than $\left|\mathbf{Z}_p^*\right| = p-1$.

Interestingly, the size $|\mathbf{E}_{a,b}|$ depends not only on p, but also on a and b. If a and b are chosen from $\mathbf{Z}_p$, then $|\mathbf{E}_{a,b}|$ satisfies

$$p + 1 - 2\sqrt{p} \le |\mathbf{E}_{a,b}| \le p + 1 + 2\sqrt{p} .$$

If a and b are chosen at random from $\mathbf{Z}_p$ satisfying (15), then to a first approximation, $|\mathbf{E}_{a,b}|$ can take on more or less any value in this range (see Lenstra [29] for an accurate discussion).

Since $|\mathbf{E}_{a,b}|$ is now a random variable, however, one can repeatedly choose new elliptic curves $\mathbf{E}_{a,b}$ by choosing new values for a and b until a curve is found such that $|\mathbf{E}_{a,b}|$ is smooth. One can not do this with Pollard's method—if $p-1$ is not smooth you are stuck. Now, however, the factorer can generate curves until he achieves success. The fact that $p-1$ may have a large prime factor does not deter him. He will continue to generate elliptic curves until he finds one whose size $|\mathbf{E}_{a,b}|$ is smooth.

Let

$$L(n) = e^{\sqrt{\ln n \ln \ln n}} .$$

Then Lenstra estimates that the running time of the elliptic curve method on an input n, where p is the smallest prime factor of n, is

$$L(p)^{\sqrt{2}} .$$

Thus this method is particularly good at finding small prime factors; but is also asymptotically one of the fastest factoring algorithms known for arbitrary numbers.

Montgomery, in his doctoral thesis, presented a very effective form of FFT's that works with the second stage of ECM [35], which is much better than the inefficient approach suggested in [34].

Thus we see that it is useless to protect against factoring attacks by building in large prime factors into $p-1$ or $p+1$, since the enemy cryptanalyst can instead attempt to find an elliptic curve $\mathbf{E}_{a,b}$ such that $|\mathbf{E}_{a,b}|$ is smooth. The cryptographer has no control over whether $|\mathbf{E}_{a,b}|$ is smooth or not, since this is essentially a random number approximately equal to p. Only by making p sufficiently large can he adequately protect against such an attack.

The reader should not misinterpret this conclusion to mean that the elliptic curve factorization method is a useful attack against the RSA cryptosystem. In fact, the elliptic curve factorization method is not very efficient when used against numbers of the sort used in RSA cryptotsystems. But then, neither is the $p-1$ method very efficient, which has been proposed as a justification for using strong primes. The point is that if the cryptographer attempts to avoid a $p-1$ attack by using strong primes, then the adversary can merely switch to an elliptic curve factorization method, which has greater efficiency (because it can run in parallel) , independent of whether $p-1$ is a strong prime or not. Choosing a strong prime is like locking one door, but leaving the others unlocked. Choosing large primes is like locking all the doors.

8.4 Class Group factoring methods

During the 1980's there were a number of algorithms proposed for factoring which depended upon smoothness properties of yet another group, known as the *Class Group*. The technical details are beyond the scope of this paper, however we give a brief sketch of the ideas behind the methods because they give yet another reason why "strong primes" are obsolete.

If one considers the quadratic field $\mathbf{Q}(\sqrt{-kN})$ for some small integer k, there exists a group associated with this field known as the Class Group. An algorithm by Schnorr and Lenstra [50] and later improved and implemented by Atkin and Rickert will succeed in factoring N if the Class Group is smooth. It basically applies the $p-1$ method by replacing the multiplicative group mod p, with the Class Group. However, as one can vary k, one gets multiple choices of groups as with ECM, and in fact this method has the same asymptotic run time as ECM. While one can get lucky, as with $p \pm 1$, this algorithm has not proved effective for keys much beyond 60 decimal digits.

Nevertheless, there is no way to guard against the Class Group being smooth by special construction of the primes which form the modulus. One can only guard against this algorithm by choosing the primes to be sufficiently large.

8.5 Probability of success of special methods

References [34] and [51] discuss in detail the practical computing limits of ECM and $p \pm 1$. This section summarizes those results. We give the probability of success for these methods on 512-bit primes (1024 bit keys) for various levels of effort. Times are given in hours for a

MIPS-R10000 computer. As mentioned before, both $p \pm 1$ and ECM are 2-step algorithms. In the first step one hopes that the order of the relevant group is smooth up to a limit $B1$. In the second step one allows that the order of the group is divisible by a *single*, larger prime P, where $B1 < P << B2$. It is important to note that the first step of the $p \pm 1$ algorithms is inherently serial, while ECM may be run in parallel by giving separate curves to different machines. One can therefore increase the chance of success for $p \pm 1$ only by increasing elapsed time, while one can increase the chance of success for ECM by increasing the number of machines. The table below gives $B1$ the step-1 bound for both the $p \pm 1$ and ECM algorithms. It then gives the time required to execute the algorithm for $p \pm 1$ and for a single Elliptic Curve. The next column gives the probability of success for $p \pm 1$ or for ECM with a single curve (the probabilities are the same). Finally, the last colum gives the probability of success if one were to run 1000 Elliptic Curves in parallel.

$B1$	$P \pm 1$ Time	ECM Time	Prob(Success)	ECM w/ 1000 Curves
10^7	.6	4.6	3×10^{-30}	3×10^{-27}
10^8	6	46	2×10^{-25}	2×10^{-22}
10^9	60	460	8×10^{-22}	8×10^{-19}
10^{10}	600	4600	6×10^{-19}	6×10^{-16}
10^{11}	6000	46000	9×10^{-17}	9×10^{-14}
10^{12}	60000	460000	6×10^{-15}	6×10^{-12}

It should also be noted that for ECM, one can spend a fixed amount of time by lowering $B1$ and increasing the number of curves proportionally. For numbers that are within practical computing range of this method (say 40 decimal digits) it often makes sense to do this. However, for 512-bit primes, it does not make more sense to lower $B1$ and run more curves until $B1$ reaches about 10^{16} and this is way beyond computing range.

As indicated by the table, it takes about 4.6 hours to run step 1 to 10^7. ECM performs approximately like an $O(p^{1/5})$ algorithm for factors that are within reach of even extraordinary computer facilities [51]. The cost to find a 128-bit factor with probability $(1 - 1/e)$ is about 4900 curves with step 1 limit of 3.3×10^6. A single curve can be run in about 1/4 hour on a MIPS-R10000. One can therefore find a 128-bit factor of a 1024-bit modulus with probability about .63 with about 5000 such machines running in parallel. Doubling the number of machines will increase the chance of success to about .86. Finding a 256-bit factor with the same probabilities will take about $2^{128/5} \sim 5 \times 10^7$ much work. Finding a 512-bit factor will take about $2^{384/5} \sim 1.3 \times 10^{23}$ times as much work. Put another way, it will take 1 billion machines running for 3.75×10^{10} years to give a .63 chance of successfully factoring a 1024-bit modulus with ECM. This is roughly twice the age of the universe.

Since 1985, the entire worldwide group of people doing factoring has discovered exactly one 47 decimal digit prime factor with ECM. This work comprises dozens of researchers running many millions of curves. It will take approximately $2^{(512-156)/5} \sim 2.7 \times 10^{21}$ times this total effort to give a .63 chance of factoring a 1024-bit RSA key with ECM. The conclusion that is easily reached from this data is that 1024-bit RSA moduli are well beyond the computing range of the $p \pm 1$ and ECM methods within the lifetime of the universe.

9 Strong primes as protection against cycling attacks

The original "cycling" attack is due to Simmons and Norris [52], as noted above. Given a ciphertext $C = M^e \bmod n$, the cryptanalyst repeatedly re-encrypts the ciphertext until C is again obtained. If k re-encryptions are performed, then

$$C^{e^k} \equiv C \pmod{n} , \tag{16}$$

and so

$$C^{e^{k-1}} \equiv M \pmod{n} . \tag{17}$$

The generalization of this cycling attack proposed by Williams and Schmid [56] and Berkovits [4] is always at least as efficient as the Simmons/Norris attack. In the generalized cycling attack, rather than finding the least k such that $C^{e^k} \equiv C \pmod{n}$, we look for the least u such that

$$\gcd(n, C^{e^u} - C) > 1 .$$

Typically, the gcd computation will return one of the prime factors of n, and we have thereby factored n. In the rare case that the *gcd* computation returns n, we can still compute M, since in that case equation (17) holds. Therefore the generalized attack succeeds whenever the original attack would have succeeded and, in fact, it is likely to succeed much more quickly, since we are now detecting cycling modulo p (or modulo q) rather than cycling modulo n. Indeed, equation (16) implies both of the equations

$$C^{e^k} \equiv C \pmod{p} \tag{18}$$
$$C^{e^k} \equiv C \pmod{q} , \tag{19}$$

whereas the generalized attack succeeds (and factors n) if only one of these conditions holds. If these conditions are more-or-less independent (a reasonable assumption) and equally likely, then if the original cycling attack succeeds in time T, then the generalized cycling attack will succeed in time $\sqrt{T}$. And, moreover, it typically succeeds by factoring n, not just determining the message M, as in the original attack.

Since the generalized cycling attack is thus essentially a factoring attack, one can argue that it must not be very effective, on the grounds that if it were an efficient attack, then people would be using it as a general-purpose factoring method (which they are not). Indeed, this attack is equivalent to the P-1 factoring algorithm with the condition that all of the prime factors of $p-1$ are contained in e^k-1. The Table in section 8.5 discussed the likelihood of success of the $P-1$ method. While this is perhaps a sound argument against the efficacy of these cycling attacks, it begs the question of analyzing just how (in)efficient these attacks are. In the rest of this section we provide a group-theoretic analysis of the efficiency of the generalized cycling attack.

Even in the extremely unlikely event that one of $p-1$ or $q-1$ has only small primes factors, it is much more unlikely that *both* have only small factors. Suppose r is a moderately large prime exactly dividing $LCM(p-1, q-1)$. Without loss of generality, assume that r divides $p-1$ so $p-1 = Rr$. If r divides $ord(e) \bmod p$, then the order of e is automatically large and the amount of computation needed to effect the cycling attack is beyond reach.

Suppose r does *not* divide $ord(e)$ mod $\lambda(N)$. It follows immediately that e must be an r^{th} power mod p. This follows from Lagrange's Theorem: $ord(e)$ must divide $p-1$, and we have assumed that r divides $p-1$ but r does not divide $ord(e)$. Hence e must be an r^{th} power mod p. The number of solutions to $a^r = e \bmod p$ is at most r and the probability that e is an r^{th} power is at most r so the probability that e is an r^{th} power is at most r/p. If r is about 100 bits and p is 512 bits, this probability is 2^{-412}.

10 Conclusions

We see that the elliptic curve method of factoring removes any special status that $p-1$ or $p+1$ may have had because of the $p \pm 1$ methods of factoring. There is no advantage to the cryptographer in building "special structure" into $p-1$ and $p+1$ in terms of protection against factoring attacks.

If one expends the same level of effort with $p \pm 1$ as was spent factoring RSA-130 with the Number Field Sieve (roughly 50 years on a single Sparc-10), one can take step 1 of $p \pm 1$ to about 10^{12}. The probability of factoring a 1024-bit modulus with this level of effort is about 6×10^{-15}. Furthermore, this work is inherently serial; it can not be run in parallel, so while RSA-130 was finished in several months by using many machines, the time for $p \pm 1$ is elapsed time. While strong primes protect against the $p \pm 1$ methods, they do not protect against their generalization: the elliptic curve method. Using strong primes doesn't seem to be harmful, except for the extra cost of generating them, but using them doesn't seem to buy much protection either. The only real protection comes from choosing large enough primes p and q.

Actually, some harm does come from the use of "strong" primes, but it has nothing to do with the effective security of keys that use them. Instead, the harm comes from the fact that they lead an unsophisticated user into believing that because they are "strong" that they are more secure than keys that are not "strong", i.e. that any other key must be "weak".

Similarly, we see that strong primes buy little if anything in terms of protection against cycling attacks. The chance that a cycling attack will succeed is negligible, even if the primes are chosen randomly. Factoring attacks have a much higher probability of success.

The gist of this paper is thus that *strong primes offer little protection beyond that offered by random primes.* On the other hand, aside from the additional effort required to generate them, there seems to be no technical reason *not* to use strong primes if one so desires. But we see no rationale for *requiring* the use of strong primes in public-key RSA standards.

We note that the the conclusions in this paper reflect the current state of knowledge about factoring. It is entirely possible that future developments might change these conclusions—in either direction. A new factoring algorithm might again make strong primes seem desirable—or it might make strong primes seem particularly dangerous.

Acknowledgments

We would like to thank Burt Kaliski, Hendrik Lenstra, Jr., Dave McAllester, and Gary Miller for helpful discussions.

References

[1] L. M. Adleman. Factoring numbers using singular integers. Technical Report TR 90-20, U.S.C. Computer Science Department, September 1990.

[2] Eric Bach. Number-theoretic algorithms. In *Annual Review of Computer Science*, volume 4, pages 119–172. Annual Reviews, Inc., 1990.

[3] Eric Bach, Gary Miller, and Jeffrey Shallit. Sums of divisors, perfect numbers, and factoring. *SIAM Journal of Computing*, 15(4):1143–1154, November 1986.

[4] Shimshon Berkovits. Factoring via superencryption. *Cryptologia*, 6(3):229–237, July 1982.

[5] G. R. Blakley and I. Borosh. Rivest-Shamir-Adleman public key cryptosystems do not always conceal messages. *Computers & Mathematics with Applications*, 5(3):169–178, 1979.

[6] D. M. Bressoud Factorization and Primality Testing. *Springer-Verlag* Undergraduate Texts in Mathematics 1989

[7] Duncan A. Buell. Factoring: Algorithms, computations, and computers. *Journal of Supercomputing*, 1:191–216, 1987.

[8] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms.* MIT Press/McGraw-Hill, 1990.

[9] D. W. Davies and W. L. Price. *Security for Computer Networks: An Introduction to Data Security in Teleprocessing and Electronic Funds Transfer.* John Wiley and Sons, New York, 1984.

[10] John D. Dixon. Factorization and primality tests. *The American Mathematical Monthly*, 91(6):333–352, June-July 1984.

[11] Carl Friedrich Gauss. *Disquisitiones Arithmeticae.* Yale University Press, 1965. Translated from the 1801 Latin original by A. Clarke.

[12] J. Gordon. Strong RSA keys. *Electronics Letters*, 20(12):514–516, June 1984.

[13] John A. Gordon. Strong primes are easy to find. In T. Beth, N. Cot, and I. Ingemarsson, editors, *Proceedings of EUROCRYPT 84*, pages 216–223, Paris, 1985. Springer. Lecture Notes in Computer Science No. 209.

[14] Richard K. Guy. How to factor a number. In *Proceedings Fifth Manitoba Conference on Numerical Mathematics*, pages 49–89, 1975.

[15] Martin E. Hellman and Carl E[ric] Bach. Method and apparatus for use in public-key data encryption system. U.S. Patent 4,633,036, December 1986. (Filed May 31, 1984.).

[16] Tore Herlestam. Some critical remarks on public-key cryptosystems. *BIT*, 18:493–496, 1978.

[17] P. J. Hoogendoorn. On a secure public-key cryptosystem. In H. W. Lenstra, Jr. and R. Tijdeman, editors, *Computational Methods in Number Theory*, pages 159–168. Mathematical Centre Tracs 154, Amsterdam, 1982.

[18] M. Elkenbracht-Huizing An implementation of the Number Field Sieve Experimental Mathematics *to appear*

[19] ISO. Annex C – the RSA public key cryptosystem, in ISO/IEC 9594-8:1989(E). Part of ISO/IEC X.509 Draft Standard, 1989.

[20] Peter Jamnig. Securing the RSA-cryptosystem against cycling attacks. *Cryptologia*, XII(3):159–164, July 1988.

[21] Burton S. Kaliski, Jr. *Elliptic Curves and Cryptography: A Pseudorandom Bit Generator and Other Tools*. PhD thesis, MIT EECS Dept., January 1988. Published as MIT LCS Technical Report MIT/LCS/TR-411 (Jan. 1988).

[22] Donald E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, 1969. Second edition, 1981.

[23] N. Koblitz. Elliptic curve cryptosytems. *Mathematics of Computation*, 48(177):203–209, 1987.

[24] E. Kranakis. *Primality and Cryptography*. Wiley-Teubner Series in Computer Science, 1986.

[25] R. S. Lehman Factoring Large Integers *Mathematics of Computation*, 28(1974):637-646, 1974

[26] A. K. Lenstra and H. W. Lenstra, Jr. Algorithms in number theory. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science (Volume A: Algorithms and Complexity)*, chapter 12, pages 673–715. Elsevier and MIT Press, 1990.

[27] A.K. Lenstra & H.W. Lenstra (eds.)

The Development of The Number Field Sieve *Springer-Verlag* Lecture Notes in Mathematics #1554 1993

[28] A. K. Lenstra, H. W. Lenstra, Jr., M. S. Manasse, and J. M. Pollard. The number field sieve. In *Proc. 22nd ACM Symposium on Theory of Computing*, pages 564–572, Baltimore, Maryland, 1990. ACM.

[29] H. W. Lenstra, Jr. Factoring integers with elliptic curves. *Annals of Mathematics*, 126:649–673, 1987.

[30] W. J. LeVeque. *Fundamentals of Number Theory.* Addison-Wesley, 1977.

[31] Gary L. Miller. Riemann's hypothesis and tests for primality. *JCSS*, 13(3):300–317, 1976.

[32] Victor S. Miller. Use of elliptic curves in cryptography. In H. C. Williams, editor, *Proceedings CRYPTO 85*, pages 417–426. Springer, 1986. Lecture Notes in Computer Science No. 218.

[33] Peter L. Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of Computation*, 48(177):243–264, January 1987.

[34] Peter L. Montgomery and Robert D. Silverman. An FFT extension to the $p-1$ algorithm. *Mathematics of Computation*, 54(190):839–854, April 1990.

[35] P. L. Montgomery An FFT Extension of the Elliptic Curve Method of Factorization *UCLA dissertation*, 1992

[36] P.L. Montgomery A Survey of Modern Integer Factorization Algorithms *CWI Quarterly* 1995

[37] Ivan Niven and Herbert S. Zuckerman. *An Introduction to the Theory of Numbers.* John Wiley & Sons, fourth edition, 1980.

[38] J. M. Pollard. Theorems on factorization and primality testing. *Proceedings Cambridge Philosophical Society*, 76:521–528, 1974.

[39] J. M. Pollard. A Monte Carlo method for factorization. *BIT*, 15:331–334, 1975.

[40] C. Pomerance. Analysis and comparison of some integer factoring algorithms. In H. W. Lenstra, Jr. and R. Tijdeman, editors, *Computational Methods in Number Theory*, pages 89–139. Math. Centrum Tract 154, Amsterdam, 1982.

[41] Carl Pomerance. The quadratic sieve factoring algorithm. In T. Beth, N. Cot, and I. Ingemarrson, editors, *Advances in Cryptology*, volume 209 of *Lecture Notes in Computer Science*, pages 169–182. Springer-Verlag, 1984.

[42] Carl Pomerance, editor. *Proceedings of the AMS Symposia in Applied Mathematics: Computational Number Theory and Cryptography*. American Mathematical Society, to appear.

[43] M. Rabin. Probabilistic algorithms for testing primality. *J. Number Theory*, 12:128–138, 1980.

[44] Hans Riesel. *Prime Numbers and Computer Methods for Factorization.* Progress in Mathematics. Birkhäuser, 1985.

[45] Ronald L. Rivest. Remarks on a proposed cryptanalytic attack of the M.I.T. public-key cryptosystem. *Cryptologia*, 2(1):62–65, January 1978.

[46] Ronald L. Rivest. Critical remarks on 'Some critical remarks on public-key cryptosystems' by Herlestam. *BIT*, 19:274–275, 1979.

[47] Ronald L. Rivest. Cryptography. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science (Volume A: Algorithms and Complexity)*, chapter 13, pages 717–755. Elsevier and MIT Press, 1990.

[48] Ronald L. Rivest. The MD4 message digest algorithm. Technical Report MIT/LCS/TM-434, MIT Laboratory for Computer Science, October 1990. (To appear in Proceedings CRYPTO 1990.).

[49] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[50] C. P. Schnorr and H. W. Lenstra A Monte-Carlo factoring algorithm with linear storage *Math. Comp.*, 43:289-312, 1984

[51] R.D. Silverman & S.S. Wagstaff Jr. A Practical Analysis of the Elliptic Curve Factoring Algorithm *Math. Comp.*, 61(203):445–462, July 1993.

[52] Gustavus J. Simmons and Michael J. Norris. Preliminary comments on the MIT public-key cryptosystem. *Cryptologia*, 1(4):406–414, October 1977.

[53] N. M. Stephens. Lenstra's factorization method based on elliptic curves. In H. C. Williams, editor, *Proceedings CRYPTO 85*, pages 409–416. Springer, 1986. Lecture Notes in Computer Science No. 218.

[54] H. C. Williams. A $p{+}1$ method of factoring. *Mathematics of Computation*, 39(159):225–234, July 1982.

[55] H. C. Williams. An overview of factoring. In D. Chaum, editor, *Proceedings of CRYPTO 83*, pages 71–80, New York, 1984. Plenum Press.

[56] H. C. Williams and B. Schmid. Some remarks concerning the MIT public-key cryptosystem. *BIT*, 19:525–538, 1979.