CHAPTER 5

A Comparison of Practical Public Key Cryptosystems Based on Integer Factorization and Discrete Logarithms*

PAUL C. VAN OORSCHOT
Bell-Northern Research

*Partial support for this work was provided by the University of Waterloo, Waterloo, Ontario, and by Newbridge Microsystems (a division of Newbridge Networks Corporation), Kanata, Ontario.

***Abstract*—based on the current literature, this survey carries out a detailed analysis of a version of the multiple polynomial quadratic sieve integer factorization algorithm, and of the Coppersmith algorithm for computing discrete logarithms in GF(2^n). This is used for a practical security comparison between the Rivest–Shamir–Adleman (RSA) cryptosystem and the El Gamal cryptosystem in fields of characteristic 2. Other aspects of the cryptosystems are also compared. In addition, the security of elliptic curve cryptosystems over GF(2^n) is discussed, and related to that of the previously mentioned cryptosystems.**

1 INTRODUCTION

Since its inception in the mid 1970s, public key cryptography has flourished as a research activity, and significant theoretical advances have been made. In more recent years, many public key concepts have gained acceptance in the commercial world. Without question, the best-known public key cryptosystem is the RSA cryptosystem of Rivest, Shamir, and Adleman [46]. Although not as well known, another public key cryptosystem of practical interest is that of El Gamal [17]. The latter system and its variations use a basic extension of Diffie-Hellman key exchange [15] for encryption, together with an accompanying signature scheme.

The security of the RSA and El Gamal cryptosystems is generally equated to the difficulty of integer factorization and that of the computation of discrete logarithms in finite fields, respectively. Each of these problems has been the subject of extensive research in recent years, and significant progress has been made. An early survey of progress in integer factorization is given by Davis, Holdridge, and Simmons [14]. Since then, the quadratic sieve algorithm has been carefully examined with an eye to specialized hardware [45] and networks of distributed processors [11,28]. More recently, dramatic theoretical progress has been made using elliptic curves [30] and algebraic number fields [27]. Each of the factoring algorithms resulting from these latter two advances is most efficient when applied to specific classes of composite integers as will be discussed below—although the number field sieve may eventually prove to be a competitive factoring technique for general integers of a size to be of cryptographic interest. Regarding discrete logarithms, for reasons discussed below we restrict our attention primarily to fields of characteristic 2. Early work on the problem of computing logarithms in GF(2^n) resulted in the acknowledgment that the field GF(2^{127}) is totally inadequate for cryptographic security [5]; 127 bits, which corresponds to 38 digits, is simply insufficient. Subsequent work [12,41] has led to further improvements in the subexponential-time index-calculus techniques for computing discrete logarithms in GF(2^n). Progress in solving large sparse linear systems over finite fields applies to both problems [24,52].

Significant advances have also been made, in theory and in practice, on techniques for efficient implementation of these cryptosystems, including custom very large-scale integration (VLSI) chips and very efficient digital signal processor software implementations for modular exponentiation for RSA [7], and custom VLSI chips for arithmetic operations in the El Gamal cryptosystem in GF(2^n) [47].

Asymptotic running times for many integer factorization algorithms are often given in the form $L(N)^c$, where $L(N)$, $= \exp(\sqrt{\log N \log\log N})$, with analysis carried out with sufficient accuracy to derive the correct value of c (see [43]). Such analysis is very important, and allows one to gauge the relative difficulty of problems of different sizes using a fixed algorithm; furthermore, given an actual running time obtained by applying an implementation to one problem instance, this permits reasonable extrapolations yielding running time estimates for similar implementations on larger problem instances. However, while asymptotic formulas such as $L(N)^c$ by themselves suffice for theoreticians, they leave much to be desired by those interested in estimating the best achievable running time of an algorithm in practice; and without further investigation, they are of limited use in comparing the difficulty of a particular instance of one problem (e.g., integer factorization) with a particular instance of another problem (e.g., the discrete logarithm problem).

Earlier surveys discussing both integer factorization and the discrete logarithm problem include work by Lenstra and Lenstra [26], Blake, Van Oorschot, and Vanstone [6], and Bach [1]. As noted by Bach, little work has been done on attaining more precise running time formulas for algorithms for these problems. Two notable exceptions are the discrete analysis by Odlyzko [41] of the Coppersmith algorithm for computing discrete logarithms in GF(2^n), and a detailed analysis of a version of the quadratic sieve integer factorization algorithm by Pomerance, Smith, and Tuler [45] in a paper that proposes a hardware design for an integer factorization machine. Regarding the former, fields of characteristic 2 have traditionally been of much interest in practice, as arithmetic in such fields is particularly amenable to efficient hardware implementation.

The computation of discrete logarithms in odd prime fields GF(p) is discussed in [13] and [25]. The latter paper concludes, based on empirical results—the computation of discrete logarithms in a prime field GF(p) of size 192 bits—that the computation of discrete logarithms in GF(p), using the best currently known techniques, is slightly harder than factorization of integers N (where $N \approx p$) via the multiple polynomial quadratic sieve. This conclusion is in line with earlier asymptotic analysis of these algorithms, and gives an indication of the relative security of RSA versus that of cryptosystems whose security relies on the difficulty of computing discrete logarithms in prime fields GF(p).

This raises the question of the relative difficulty of integer factorization and the computation of discrete logarithms in fields of characteristic 2. This issue has apparently not been addressed in the literature. Asymptotic running time formulas for the best currently known algorithms for each of these problems are well known, but as noted above are of limited use in "head-to-head" comparisons of particular instances of different problems. For example, a cryptographer might ask, "How large a field GF(2^n) must be chosen to make the computation of discrete logarithms in that field as difficult as the factorization of an m-bit integer?" While large-scale implementations provide valuable information from which to base extrapolations, answers to such questions cannot in general be obtained empirically, since problem instances of cryptographic interest should by definition be well beyond computational capabilities; furthermore, distinct implementations may (due to quality, software, and architecture details) vary dramatically with respect to running times, and thus provide only upper bounds on the difficulty of a problem instance even for a given algorithm. A worthwhile endeavor would be the refinement of existing asymptotic formulas, incorporating running time constants and currently missing factors to bring these formulas into agreement with actual (absolute) running times observed for the largest problem instances solved to date; however, this has not been done to date (nor is it done herein).

In this chapter, based on the current literature we carry out a practical comparison of the relative difficulty of integer factorization and the computation of discrete logarithms in GF(2^n). By "practical" here we mean a comparison suitable for dealing with particular problem instances of practical interest, rather than dwelling exclusively on asymptotic complexities. This facilitates a security comparison between the El Gamal cryptosystem in GF(2^n) and RSA. In addition to the relative security, we consider the practical efficiency of the systems, discuss further aspects, and report on recent advances. We also briefly consider the related elliptic curve cryptosystems, including the cryptographic significance of recent progress on the problem upon which their security rests—the computation of elliptic curve logarithms. Elliptic curve cryptosystems over GF(2^n) have recently received considerable attention as cryptographic alternatives to other candidate cryptosystems, offering greater security at shorter key lengths [22,34]. The recent reduction [35] of the elliptic curve logarithm problem, for certain elliptic curves, to the discrete logarithm problem in extension fields of the underlying field, has made an understanding of the relative difficulty of the discrete logarithm problem in GF(2^n) and that of integer factorization of even greater importance.

The analysis of the Coppersmith algorithm for computing discrete logarithms here is largely based on that of Odlyzko [41], with suitable modifications made to facilitate comparison. Similarly, our practical analysis of the quadratic sieve factorization algorithm is based on the analysis in [45], again with suitable modifications and subject to several assumptions. We emphasize that for our purposes, we are interested only in the relative difficulty of these different problems, and make no estimates as to absolute running times (e.g., number of days or weeks on a particular machine). As our model, we count the number of primitive machine instructions (such as addition and exclusive-or) on a single-processor machine such as a modern workstation; operation counts for instances of the different problems can then be compared to determine relative running times within this model. We base the security comparison on the best currently known algorithms for factorization and logarithms; future algorithmic advances cannot be predicted. Advances in technology (e.g., processor speeds) are not taken into account; these should affect the running times of the underlying problems similarly. To simplify the task, advanced architectures are not taken into account either, although these may or may not affect both algorithms similarly—for example, both can take advantage of large networks of loosely coupled workstations during the relation collection phase, but the quadratic sieve algorithm appears better suited to take full advantage of vectorized supercomputers [14]. The analysis we carry out involves several assumptions and estimates (see Sections 2 and 3). Regarding absolute running times, which are often strongly influenced by implementation, processor and architecture details, we refer the reader to [11], [27–29], and [51] for factorization, and to [12] for logarithms in GF(2^n); however, meaningful extrapolations of absolute running times from the latter may be difficult, since the largest implementation carried out to date is $n = 127$.

The remaining sections of this chapter are organized as follows. In Section 2 we review the index-calculus techniques currently available for computing discrete logarithms in GF(2^n). As is well known, these techniques consist of two stages, the first of which dominates the running time and memory requirements, involving construction of a large database through the generation and solution of a large sparse linear system. The costly operations in this first stage are the testing of a large number of pairs of polynomials for smoothness with respect to a particular degree bound, and the solution of the linear system over $\mathbb{Z}_M$, $M = 2^n - 1$. We review a discrete estimate of the running time (in number of shifts and adds) of the algorithm, and also consider the size of the linear system and the resulting database. Estimates of algorithmic parameters, the size of the resulting system, and running times are tabulated for a small number of fields of sizes representative of systems of practical interest.

In Section 3 we consider the fastest practical general purpose method for factoring integers N currently available—the multiple polynomial quadratic sieve algorithm. Analogous to the smoothness testing required in computing discrete logarithms, the quadratic sieve algorithm requires that a large number of residues (mod N) be generated and then tested for smoothness with respect to an appropriate integer bound. In this case, the smoothness testing is done by "sieving." A sufficient number of smooth residues provide a sufficient number of relations, from which a linear dependency in a large binary sparse linear system is sought. We estimate the running time (in number of single precision adds) of the algorithm, and also consider the size of the linear system. Tabulations are given estimating algorithmic parameters, system size, and running time for integers N of 100 and 155 digits. (The latter corresponds to 512 bits, a number arising as often as any other in discussions of RSA modulus size.) The currently best implementations and systems of quadratic sieve factoring can handle numbers at the low end of this range, and we restrict our examination to the specified upper limit as we feel that extrapolations beyond this point are not possible (for us) to make with any confidence, and would have little practical meaning. Furthermore, it is not clear what the range of practical utility of the quadratic sieve algorithm is.

In Section 4, we first briefly review the RSA and El Gamal cryptosystems. The estimates of Sections 2 and 3 are used to compare the running times for integer factorization and the computation of discrete logarithms in GF(2^n), for a limited set of bitlengths. This facilitates a first attempt at estimation of the relative security of the El

Gamal cryptosystem in GF(2^n), and RSA, at particular bitlengths of practical interest. We examine many further aspects of these respective cryptosystems and the underlying problems, including currently available throughput for implementations of exponentiation in $\mathbb{Z}_N$ and exponentiation in GF(2^n).

In Section 5 we survey recent work in the area of elliptic curve cryptosystems as mentioned above, and consider the cryptographic impact of recent progress in computing elliptic curve logarithms.

2 DISCRETE LOGARITHMS IN FIELDS OF CHARACTERISTIC 2

As a measure of the security of the El Gamal cryptosystem in GF(2^n), we consider the difficulty of computing discrete logarithms in fields of characteristic 2. In this section, we review the current techniques available for this problem [12,24,41,52]. We rely heavily on the authoritative analysis of Odlyzko [41]. Other surveys discussing the computation of discrete logarithms include [1], [26], and [32].

Elements of GF(2^n) are represented as polynomials over GF(2) of degree at most $n - 1$, with polynomial arithmetic done modulo a fixed binary irreducible polynomial $f(x) = x^n + f_1(x)$, deg $f_1(x) \leq n - 1$. A simple heuristic argument indicates that we may expect to find such $f(x)$ with deg $f_1(x) \approx \log(n)$, and henceforth we assume the degree of $f_1(x)$ to be about $\log(n)$.* Let $g(x)$ be a primitive element of GF(2^n). Then given any polynomial $b(x)$ in GF(2^n), the *discrete logarithm problem* in this setting is to find the unique integer y, $0 \leq y \leq 2^n - 2$, such that $b(x) \equiv g(x)^y \pmod{f(x)}$. We begin by reviewing the basic algorithm, outline the improvements that have been made, and review the running time and size of the resulting linear system.

2.1 Basic Algorithm

The standard index-calculus approach consists of two stages. The first is the precomputation (carried out only once for a given field) of a large database containing the logarithms of all irreducible polynomials of degree at most m, for appropriate m. The second stage then reuses this database each time the logarithm of a particular field element $b(x)$ is required.

Stage 1. In the first stage of the basic algorithm, an appropriate bound m ($m \approx n^{1/2} (\ln n)^{1/2}$) is selected, and then S is defined to be the set of all irreducible polynomials over GF(2) of degree at most m. A polynomial all of whose irreducible factors have degree at most m is said to be *smooth with respect to m* (or simply *smooth*, when m is understood). The objective of stage 1 is to compute the logarithms (with respect to $g(x)$) of the elements in S. This is done by obtaining approximately $|S|$ linear equations relating $|S|$ unknowns, the unknowns being the required logarithms, and then solving this system. To obtain the equations, select a random integer c, $1 \leq c \leq 2^n - 1$, and

*We use $\ln x$ to denote the natural logarithm of the number x, and log (x) to denote the logarithm of x to the base 2. For discrete logarithms of finite field elements, we also use log to denote the logarithm with respect to a primitive element $g(x)$, where $g(x)$ is implicitly assumed to be known and fixed.

compute $h(x)$, where deg $h(x) < n$ and

$$h(x) \equiv g(x)^c \pmod{f(x)} \tag{2.1}$$

Check if $h(x)$ is smooth. If so, factor $h(x)$ to obtain the irreducible polynomial decomposition

$$h(x) = \prod_{s \in S} s^{i_s}, \qquad i_s \geq 0$$

(s is a binary polynomial here.) Taking logs with respect to $g(x)$ then yields

$$c \equiv \sum_{s \in S} i_s \cdot \log(s) \qquad (\text{mod } 2^n - 1) \tag{2.2}$$

This equation relates the (unknown) logarithms of elements in S. $|S|$ such equations are obtained (or slightly more, in the case there are linear dependencies—but the resulting equations are largely expected to be independent here, and this expectation is confirmed in practice), and the linear system over the integers modulo $2^n - 1$ is solved to obtain the unknown logs.

Stage 2. In the second stage of the basic algorithm, we start with the knowledge of the logarithms of (almost) all elements in S. Given $b(x)$, we seek y, $0 \leq y \leq 2^n - 2$, such that $b(x) \equiv g(x)^y \pmod{f(x)}$. To find y, select random c, $1 \leq c \leq 2^n - 1$, and compute

$$h(x) \equiv b(x)g(x)^c \pmod{f(x)} \tag{2.3}$$

Check if the reduced polynomial $h(x)$ factors over S, and if so, factor $h(x)$ to obtain

$$h(x) = \prod_{s \in S} s^{i_s}, \qquad i_s \geq 0$$

Then

$$\log b(x) + c \equiv \sum_{s \in S} i_s \cdot \log(s) \qquad (\text{mod } 2^n - 1)$$

and one can determine $y = \log b(x)$ from the known quantities $\log(s)$.

2.2 Coppersmith Algorithm

The probability that a polynomial $h(x)$ is smooth (e.g., in Eqs. (2.1) and (2.3)) increases as the degree of $h(x)$ decreases. Blake, Fuji-Hara, Mullin, and Vanstone [5] noted that one way to take advantage of this is to make use of the extended Euclidean algorithm. This, together with another observation of Blake et al., concerning "systematic equations," motivated Coppersmith to conceive a far more powerful method, resulting in a dramatic decrease in the expected running time [12]. This latter variation is now described.

Stage 1. Coppersmith's algorithm obtains stage 1 equations more rapidly via the following technique. Positive integers k and B are carefully selected, and k is used to define h, such that

$$2^k \approx n^{1/3}(\ln n)^{-1/3}, \qquad B \approx n^{1/3}(\ln n)^{2/3}, \qquad h = \lceil n/2^k \rceil$$

Here h is the least integer greater than or equal to $n/2^k$ ($h \approx n^{2/3}(\ln n)^{1/3}$). Next select a pair of coprime polynomials $u_1(x)$, $u_2(x)$ of degrees at most B. Define

$$\begin{aligned} w_1(x) &= x^h \cdot u_1(x) + u_2(x) \\ w_2(x) &\equiv w_1(x)^{2^k} \pmod{f(x)} \end{aligned} \tag{2.4}$$

Note deg $w_1(x) \approx n^{2/3}$, and

$$\begin{aligned} w_2(x) &\equiv x^{h2^k} \cdot u_1(x)^{2^k} + u_2(x)^{2^k} \pmod{f(x)} \\ &= x^{h2^k-n} \cdot f_1(x)\, u_1(x^{2^k}) + u_2(x^{2^k}) \end{aligned} \tag{2.5}$$

and hence deg $w_2(x) \approx n^{2/3}$ also (since $h2^k - n = O(2^k) \approx O(n^{1/3})$, and $B2^k \approx n^{2/3}$). Now check if both $w_1(x)$ and $w_2(x)$ are smooth. (This is far more likely than $h(x)$ being smooth in Eq. (2.1), since $h(x)$ has degree $\approx n$ there.) If so, factor $w_1(x)$ and $w_2(x)$, and note

$$\log(w_2(x)) \equiv 2^k \cdot \log(w_1(x)) \pmod{2^n - 1} \tag{2.6}$$

is an equation that can be used in place of Eq. (2.2).

Whereas $m \approx n^{1/2}(\ln n)^{1/2}$ is optimal for the basic algorithm, $m \approx n^{1/3}(\ln n)^{2/3}$ is optimal for the Coppersmith algorithm, and the database is significantly smaller.

Stage 2. Stage 2 of the Coppersmith algorithm falls into two steps. Given a polynomial $b(x)$ whose logarithm is to be obtained, the first step reduces the problem to that of finding several logarithms of smaller-degree polynomials. The second step recursively decomposes each of these subproblems into that of finding the logarithms of a number of still smaller degree polynomials, until eventually the logs of these can be looked up from the stage 1 database. The time required to determine the log of a given polynomial $b(x)$ via this second stage is far less than that required for the first stage; we do not consider the details further here.

2.3 Faster Generation of Coppersmith Equations (Due to Odlyzko)

Odlyzko noted that the equation generation in the first stage of Coppersmith's variation can be further expedited (significantly in practice, but not asymptotically), as follows. Instead of selecting $u_1(x)$, $u_2(x)$, for use in Eq. (2.4), to be any coprime pair of polynomials as above, construct candidate pairs $u_1(x)$, $u_2(x)$ more carefully so as to increase the probability that $w_1(x)$ and $w_2(x)$ are smooth. This is done as follows. Select (possibly from a previously prepared table) a pair of polynomials $v_1(x)$, $v_2(x)$ each of degree at most (but near) $B - 1$, each known to be smooth with respect to m. The pair $v_1(x)$, $v_2(x)$ is used to determine (again for use in Eq. (2.4)) a pair of polynomials $u_1(x)$, $u_2(x)$, each of degree at most B, such that

$$v_1(x) \mid w_1(x) \quad \text{and} \quad v_2(x) \mid w_2(x) \tag{2.7}$$

These two conditions define at most $2B - 2$ (=deg $v_1(x)$ + deg $v_2(x)$) linear equations in $2B$ unknowns (the latter being the defining coefficients of $u_1(x)$ and $u_2(x)$). Solving this homogeneous binary system of order $2B - 2 \times 2B$ (e.g., by standard Gaussian elimination) yields three or more nontrivial solution pairs $u_1(x)$, $u_2(x)$. (Note it is essential here that the solution of this system does not take more time than that necessary to test the resulting $w_1(x)$, $w_2(x)$ pairs for smoothness, otherwise the purpose of the exercise will have been defeated.)

Using this technique, however, several pairs $v_1(x)$, $v_2(x)$ will result in the same pair of smooth polynomials $w_1(x)$, $w_2(x)$, and hence yield the same equation. To compensate for this, we increase the expected requirement from the previous value of $|S|$ equations to $\alpha|S|$ equations (with $\alpha > 1$), that $|S|$ of these be linearly independent. It can be shown that $|S|$ independent equations are expected if the number of smooth $w_1(x)$, $w_2(x)$ pairs obtained by this method is $\alpha|S|$, where $\alpha \approx 1.6$. To accommodate this, it may be necessary to increase the degree bound B by 1.

2.4 Smoothness Testing of Polynomials

As discussed above, it is necessary to determine whether certain polynomials (e.g., $w_i(x)$) are smooth with respect to some degree bound m. Once such smoothness is determined, the factorization of a polynomial can be achieved by standard techniques (e.g., via a square-free decomposition followed by a distinct-degree factorization [21] and then Berlekamp's classic Q-matrix method, or via the probabilistic method of Cantor and Zassenhaus [10]). Since far more polynomials fail to be smooth than require factorization, the cost of smoothness testing dominates that of factorization.

Regarding testing the smoothness of a polynomial $w(x) = \Pi\, y_i(x)^{e_i}$ with distinct irreducible factors $y_i(x)$, Odlyzko notes the following technique. (For an alternative method of smoothness testing, see [12].) Use standard techniques (e.g., see [31]) to compute the square-free component

$$w^{(0)}(x) = \prod_i y_i(x)$$

of $w(x)$. (This operation itself reveals partial information about the factorization of $w(x)$, including possibly individual factors $y_i(x)$ of degree m or less, which may be excluded from $w^{(0)}(x)$ right away.) Given $w^{(0)}(x)$, consider calculating

$$w^{(i)}(x) = \frac{w^{(i-1)}(x)}{(w^{(i-1)}(x),\, x^{2^i} - x)}, \quad 1 \le i \le m \tag{2.8}$$

It is easily seen that $w^m(x) = 1$ if and only if $w^{(0)}(x)$ is smooth with respect to m, since the ith iteration removes from consideration all irreducible factors whose degrees divide i. Furthermore, the effect is more efficiently achieved by replacing Eq. (2.8) by the sequence

$$w^{(M)}(x) = \frac{w^{(0)}(x)}{(w^{(0)}(x),\, x^{2^M} - x)}$$

and for $M + 1 \le M + i \le m$, where $M = \lfloor (m-1)/2 \rfloor + 1$

$$w^{(M+i)}(x) = \frac{w^{(M+i-1)}(x)}{(w^{(M+i-1)}(x),\, x^{2^{M+i}} - x)} \tag{2.9}$$

Within Eq. (2.9), the polynomials $x^{2^{M+i}} - x$ can be computed efficiently noting that if

$$r_i(x) \equiv x^{2^i} - x \pmod{w^{(i-1)}(x)},$$

then

$$x^{2^{i+1}} \equiv (r_i(x) + x)^2, \text{ so } x^{2^{i+1}} - x \equiv r_i(x^2) + x^2 - x \pmod{w^{(i-1)}(x)},$$

and since $w^{(i)}(x) \mid w^{(i-1)}(x)$, this last congruence is true modulo $w^{(i)}(x)$ also.

2.5 Solution of Linear System

In stage 1, once a sufficient number of linear equations have been determined via Eq. (2.6), the linear system over $\mathbb{Z}_{2^n-1}$ must be solved. (We overlook issues that arise from the fact that $2^n - 1$ may be composite.) The stage 1 database, containing the logarithms of irreducible polynomials of degree at most m, has $|S| \approx 2^{m+1}/m$ elements. To solve for these logarithms, $|S|$ independent equations are required. Using Odlyzko's method for generating stage 1 equations as noted above, about $N = 1.6|S|$ equations (in $|S|$ unknowns) must be generated in order to obtain a system of full rank.

Standard Gaussian elimination, at $O(N^3)$ operations, would dominate all other costs in the algorithm; for this reason, it is bypassed in favor of faster solution techniques which take advantage of the fact that the resulting linear system is sparse. These techniques include the conjugate gradient and Lanczos algorithms [13,24,41], and Wiedemann's technique [52], which have expected running times not much greater than $O(N^2)$. Structured or so-called *intelligent* Gaussian elimination can be used to reduce an original sparse system to a smaller system, before applying one of these techniques. Unfortunately, it is not known how to efficiently distribute the linear algebra among a large number of processors, as can be done with the equation collection phase; a single machine appears necessary. Due to memory constraints, access to a supercomputer appears desirable for carrying out this linear algebra for large problem instances. The recent work of LaMacchia and Odlyzko [24] shows that the linear algebra stages arising in both integer factorization and the discrete logarithm problem no longer appear to be a running-time bottleneck in practice.

2.6 Practical Analysis of Coppersmith Algorithm

In our analysis, we estimate the number of operations in terms of the number of shifts and adds required on a 32-bit single-processor machine.

Stage 1 of the Coppersmith variation, with generation of equations using Odlyzko's technique, is summarized below.

1. Carefully choose $m \approx n^{1/3}(\ln n)^{2/3}$, implying $|S| \approx 2^{m+1}/m$.
2. While less than $\approx 1.6|S|$ equations have been generated,
   a. Generate pairs $w_1(x)$, $w_2(x)$ of the form Eq. (2.4), via Eq. (2.7);
   b. Test each pair for smoothness; if smooth, obtain an equation of form Eq. (2.6).
3. Solve the large sparse linear system to obtain the logarithms of the elements in S.

Running time. Since stage 1 takes far more time than stage 2, the running time for computing discrete logs in GF(2^n) is that of stage 1. It is assumed that the time required for the solution of the linear system in stage 1 is less than that required for the equation generation phase of stage 1. This assumption seems reasonable, in light of progress made in solving similar large sparse systems [24]. However, there is limited practical experience with large systems of cryptographic interest ($n \approx 600$ and greater). The largest sparse system solved to date is that by LaMacchia and Odlyzko [25] as previously mentioned, resulting from a 192-bit prime field.

Although intelligent Gaussian elimination works better when there are many more equations than unknowns [24], we consider only the time required to generate enough equations to allow for solution of the system. Generating additional equations would apparently make the linear algebra stage run faster, at the expense of the equation generation phase.

Regarding the time required for the generation of equations, following the analysis in [41] we neglect the time required to generate the $w_1(x)$, and equate the running time of the equation generation part of stage 1 to that of testing smoothness for the resulting pairs $w_1(x)$, $w_2(x)$. To determine this cost, let deg $u_i(x) \approx d_i$ and deg $v_i(x) \approx d_i - 1$ (where $d_i \approx B$; minor variations between d_1 and d_2 can result in a small savings). From Eqs. (2.4) and (2.5) note that deg $w_1(x)/v_1(x) \approx h + 1$, and deg $w_2(x)/v_2(x) \approx M$ where

$$M = \max(h2^k - n + \deg f_1(x) + d_1 2^k - d_2 + 1,\ d_2(2^k - 1) + 1)$$

To satisfy the requirement of a sufficient number of independent equations, d_1 and d_2 must be sufficiently large such that

$$3 \cdot 2^{d_1+d_2}(1 - \varepsilon)^2 \cdot p(h + 1, m) \cdot p(M, m) \geq (1.6) \cdot 2^{m+1}/m$$

where ε depends on how far we allow the degrees of $v_i(x)$ to fall below $B - 1$; we simplify this to

$$p(h + 1, m) \cdot p(M, m) \cdot 2^{d_1+d_2+1} \geq 2^{m+2}/m \tag{2.10}$$

Here $p(r, m)$ is the probability that all irreducible factors of a binary polynomial of degree exactly r have degree at most m, that is, the probability of smoothness with respect to m. Formulas for $p(r, m)$ are readily available (e.g., see [5], [41, Appendix A]), and these exact probabilities are easily computed.

Now testing $w_i(x)$ for smoothness requires about $m/2$ operations of the form Eq. (2.9), involving the squaring of one polynomial modulo another of about the same degree (say t), a greatest common divisor (GCD) of two polynomials of this degree, and a division (the cost of the division being negligible since the denominator is typically unity). The squaring and GCD can be performed in about $6t$ t-bit vector shifts and adds (exclusive-ors), or $6t \cdot (t/32)$ simple machine operations assuming the work is done on a machine with 32-bit shifts and adds. The use of 32-bit machines, as opposed to larger-wordsize supercomputers, is a reasonable assumption for our purposes, in light of recent projects involving the networking of hundreds of workstations to carry out various pieces of the computations in quadratic sieve, elliptic curve, and number field sieve factorizations [27,28]. Since in the vast majority of cases $w_1(x)$ will fail to be smooth and $w_2(x)$ then need not even be tested, only the cost of testing one polynomial from each pair is charged. This leads to an operation estimate for smoothness testing of

$$m/2 \cdot 6t(t/32) \cdot (1.6) \cdot 2^{m+1}/m \cdot p(h + 1, m)^{-1} \cdot p(M, m)^{-1}$$

We have deg $w_i(x) \approx h$, and use $t = h$ here, although the degrees of the polynomials decrease during a smoothness test, and on average are thus somewhat less than h. Our estimate becomes about

$$h^2 \cdot 2^{m-2} \cdot p(h + 1, m)^{-1} \cdot p(M, m)^{-1} \qquad (2.11)$$

32-bit shifts/adds. Tables 1, 2, and 3 illustrate sample parameters and estimated operation counts (Eq. (2.11)) for the smoothness testing in stage 1 of the algorithm, for $n = 400$, 700, and 850, respectively. (The reason for choosing to focus on these particular bitlengths will become clear in Section 4. If one is willing to work with a million or more equations, then $2^k = 8$ is worth considering.)

Memory requirements. The tables below include figures indicating the size of the linear systems arising from various parameter choices. The resulting stage 1 database itself contains about $b = 2^{m+1}/m$ entries, each being the n-bit logarithm of an element in GF(2^n) (e.g., for $n = 1000$, each entry is 125 bytes). Hence roughly bn bits are required to store the completed stage 1 database. During the stage 1 computation the linear system itself requires storage on the order of the number of nonzero entries in the system (assuming a sparse representation is used), each nonzero entry being a coefficient in a linear relation (Eq. (2.6)); storage is required for the coefficient itself plus overhead associated with the sparse representation. Working memory required by the sparse solution techniques is of about the same order as that required for the representation of the system.

The sparseness of the relations depends on the number of distinct irreducible factors of $w_1(x)$ and $w_2(x)$ in Eq. (2.6). These are polynomials of degree about $n^{2/3}$ which are smooth with respect to m. The number of irreducible factors in each polynomial is thus $O(n^{2/3})$, and in some cases as small as $O(n^{2/3}/m)$.

Further speedups in the computation of discrete logarithms. We note below two further methods by which the Coppersmith variation, as modified by Odlyzko, can be improved. These improvements have no asymptotic impact, but can speed things up in practice. Details are given in [41].

TABLE 1. ESTIMATES FOR SMOOTHNESS TESTING ($n = 400$)

deg $f_1(x) = 10$, $2^k = 4$, $h = 100$

m	d_1	d_2	$b = 2^{(m+1)}/m$	bn	Operations
14	20	23	2.3406e + 03	9.3623e + 05	8.8509e + 16
15	18	21	4.3691e + 03	1.7476e + 06	6.7538e + 15
16	17	19	8.1920e + 03	3.2768e + 06	1.1959e + 15
17	16	18	1.5420e + 04	6.1681e + 06	3.4882e + 14
18	17	17	2.9127e + 04	1.1651e + 07	3.4324e + 14
19	15	17	5.5188e + 04	2.2075e + 07	8.8542e + 13
20	15	17	1.0486e + 05	4.1943e + 07	6.6796e + 13
21	16	16	1.9973e + 05	7.9892e + 07	8.8178e + 13
22	15	16	3.8130e + 05	1.5252e + 08	5.7013e + 13
23	15	16	7.2944e + 05	2.9178e + 08	5.7346e + 13
24	15	16	1.3981e + 06	5.5924e + 08	6.1942e + 13
25	16	16	2.6844e + 06	1.0737e + 09	9.2897e + 13
26	16	16	5.1622e + 06	2.0649e + 09	1.1056e + 14

Previous Chapter Previous Page Home Next page Next Chapter

TABLE 2. ESTIMATES FOR SMOOTHNESS TESTING ($n = 700$)

$\deg f_1(x) = 10, \quad 2^k = 4, \quad h = 175$

m	d_1	d_2	$b = 2^{(m+1)}/m$	bn	Operations
20	25	28	$1.0486e + 05$	$7.3400e + 07$	$6.3195e + 20$
21	26	26	$1.9973e + 05$	$1.3981e + 08$	$3.1137e + 20$
22	25	25	$3.8130e + 05$	$2.6691e + 08$	$8.2279e + 19$
23	23	25	$7.2944e + 05$	$5.1061e + 08$	$1.6711e + 19$
24	23	24	$1.3981e + 06$	$9.7867e + 08$	$9.9167e + 18$
25	23	23	$2.6844e + 06$	$1.8790e + 09$	$6.5894e + 18$
26	22	23	$5.1622e + 06$	$3.6136e + 09$	$3.2556e + 18$
27	22	23	$9.9421e + 06$	$6.9594e + 09$	$2.4706e + 18$
28	21	23	$1.9174e + 07$	$1.3422e + 10$	$1.5647e + 18$
29	22	22	$3.7026e + 07$	$2.5918e + 10$	$1.9366e + 18$
30	22	22	$7.1583e + 07$	$5.0108e + 10$	$1.8354e + 18$
31	22	22	$1.3855e + 08$	$9.6983e + 10$	$1.8454e + 18$
32	22	22	$2.6844e + 08$	$1.8790e + 11$	$1.9555e + 18$
33	22	22	$5.2060e + 08$	$3.6442e + 11$	$2.1720e + 18$
34	21	23	$1.0106e + 09$	$7.0741e + 11$	$1.9733e + 18$
35	22	23	$1.9634e + 09$	$1.3744e + 12$	$2.8893e + 18$
36	22	23	$3.8177e + 09$	$2.6724e + 12$	$3.6044e + 18$

1. *Large irreducible factor method.* Motivated by the "large prime" variations of the quadratic sieve and earlier integer factorization algorithms, a "large irreducible factor" variation of the index-calculus algorithms for computing discrete logarithms follows. The basic idea is that when testing a polynomial $w_i(x)$ for smoothness with respect to a degree-bound m, if it fails smoothness due to a single irreducible factor

TABLE 3. ESTIMATES FOR SMOOTHNESS TESTING ($n = 850$)

$\deg f_1(x) = 10, \quad 2^k = 4, \quad h = 213$

m	d_1	d_2	$b = 2^{(m+1)}/m$	bn	Operations
21	30	33	$1.9973e + 05$	$1.6977e + 08$	$9.2274e + 23$
22	30	31	$3.8130e + 05$	$3.2411e + 08$	$2.6795e + 23$
23	28	30	$7.2944e + 05$	$6.2003e + 08$	$3.5360e + 22$
24	27	29	$1.3981e + 06$	$1.1884e + 09$	$9.3776e + 21$
25	27	28	$2.6844e + 06$	$2.2817e + 09$	$4.3878e + 21$
26	27	27	$5.1622e + 06$	$4.3879e + 09$	$2.3381e + 21$
27	25	27	$9.9421e + 06$	$8.4507e + 09$	$6.7622e + 20$
28	26	26	$1.9174e + 07$	$1.6298e + 10$	$6.2617e + 20$
29	25	26	$3.7026e + 07$	$3.1472e + 10$	$3.2172e + 20$
30	24	26	$7.1583e + 07$	$6.0845e + 10$	$1.8677e + 20$
31	24	26	$1.3855e + 08$	$1.1777e + 11$	$1.5459e + 20$
32	25	25	$2.6844e + 08$	$2.2817e + 11$	$1.8362e + 20$
33	25	25	$5.2060e + 08$	$4.4251e + 11$	$1.6989e + 20$
34	25	25	$1.0106e + 09$	$8.5899e + 11$	$1.6618e + 20$
35	25	25	$1.9634e + 09$	$1.6689e + 12$	$1.7090e + 20$
36	25	25	$3.8177e + 09$	$3.2451e + 12$	$1.8390e + 20$
37	25	25	$7.4291e + 09$	$6.3148e + 12$	$2.0613e + 20$
38	25	25	$1.4467e + 10$	$1.2297e + 13$	$2.3977e + 20$

whose degree exceeds m but is at most $m + d$ (for a suitably small integer d), then retain the equation resulting from this polynomial where it otherwise would have been discarded. If, over the entire equation collection phase, $t \geq 2$ such equations result from the same large irreducible factor, then these can be combined to yield $t - 1$ "ordinary" equations (involving only factors whose degrees are less than the original bound m). This technique may lead to improvements in running time for equation collection by a factor of at most 3, for fields of cryptographic interest.

2. *Early abort strategy.* This technique is similarly motivated by an analogous technique used in integer factorization—the continued fractions algorithm. The reasoning here is that most of the polynomial pairs tested for smoothness will fail; if failure can be detected at an earlier stage, then running time will be saved. The method relies on the fact that a polynomial is unlikely to be smooth unless it has some smaller-degree factors. The idea is to abort the smoothness test (guessing it is destined to fail) once it is found that the sum of the degrees of irreducible factors of degree at most k, in the polynomial $w_i(x)$ being tested, is less than R, for appropriately chosen bounds k and R. The abort decision process can be efficiently combined with the smoothness testing technique of Section 2.4. This technique may lead to improvements in running time of the equation collection by a factor of about 2, for fields of cryptographic interest.

3 INTEGER FACTORIZATION

For the purpose of comparative analysis, the method we focus on for integer factorization is the quadratic sieve [43] and its variations. This is a logical choice since the quadratic sieve is the most efficient general purpose (i.e., applicable to composite integers of no special form) factoring algorithm in 1990. Other algorithms, such as variations of Lenstra's elliptic curve technique [30] and the number field sieve [27], have factored numbers of special form much larger than can be factored at present using the quadratic sieve, but such numbers can be easily avoided in cryptographic applications. Hence for our purpose in comparing the relative cryptographic security of schemes based on the difficulty of factoring to schemes based on the difficulty of extracting discrete logarithms, the comparison should be based on the quadratic sieve. General descriptions of the algorithm are readily available in the literature, and much practical experience has been documented (e.g., [11,14,28,29,44,51]). We include here an outline of the basic ideas for convenience, followed by a practical analysis of running time.

3.1 Outline of the Quadratic Sieve Algorithm

Let N be the composite integer to be factored. Define $FB = \{p_1, p_2, \ldots, p_b\}$, the *factor base,* to consist of $p_1 = -1$, $p_2 = 2$, and all odd primes $p_i < B$ (where B is a suitably chosen bound) for which N is a quadratic residue modulo p_i. Consider pairs of integers a_i, r_i such that $a_i^2 \equiv r_i \pmod{N}$, with $-N/2 \leq r_i \leq N/2$, where r_i factors over the factor base, that is,

$$r_i = \prod_{j=1}^{b} p_j^{\alpha_{ij}}, \qquad \alpha_{ij} \geq 0$$

We say that such integers are *smooth* (with respect to p_b). A sufficient number of such pairs (a_i, r_i) are found, such that there exists a product of a subset of the r_i's that is a perfect square. To find such a product, since only the parity of the exponents α_{ij} is of import, associate with each r_i a binary vector

$$v_i = (v_{i1}, v_{i2}, \ldots, v_{ib}) \quad \text{where} \quad v_{ij} \equiv \alpha_{ij} \pmod 2$$

After at most $b + 1$ (a_i, r_i) pairs are found, a linear dependency will exist among the corresponding b-tuples v_i, in which case there exists a subset S of indexes such that $\Sigma_{s\in S}\, v_i \equiv (0, 0, \ldots, 0)$, implying $\Pi_{i\in S}\, r_i$ is a perfect square (call it x^2). Then letting $y^2 = \Pi_{i\in S} a_i{}^2$ yields the relation $x^2 \equiv y^2 \pmod N$. Now if $x \not\equiv \pm y \pmod N$, which would be expected at least half the time for composite N and randomly chosen such x, y, then GCD$(x + y, N)$ is a proper divisor of N.

To obtain such (a_i, r_i) pairs with r_i smooth, the quadratic sieve algorithm generates r_i candidates as follows. Carefully choose a quadratic polynomial $r(x)$, such that for $x = i$ in a suitable range, say $-M \le i \le M$, $r(i)$ is small in absolute value. For example, for the (ordinary) quadratic sieve as described in [43], Pomerance originally suggested $r(x) = ([\sqrt{N}] + x)^2 - N$. (Note that the only odd primes p that the factor base need contain are those for which N is a quadratic residue mod p, for if p divides $r(i)$, then $([\sqrt{N}] + x)^2 \equiv N \pmod p$.) Consider an array $R[\,]$ of $2M + 1$ cells, with the cell indexed by i corresponding to the value $r_i = r(i)$. To check whether r_i is smooth, proceed as follows. Solve the quadratic congruence

$$r(x) \equiv 0 \pmod q, \qquad \text{for each prime power} \qquad q = p^\alpha < N,\ p \in FB \qquad (3.1)$$

There are two solutions x_j for each odd p in the factor base; the case $p = 2$ requires special consideration (e.g., see [14]). Since $r(x + kq) \equiv r(x) \pmod q$, given one solution x_j to Eq. (3.1), $(x_j + kq)$ is also a solution for all k, implying that every qth cell is divisible by q. To efficiently record this, initialize all cells in $R[\,]$ to 0, and add $\log(p)$ to cell $R[i]$ whenever it is found that $q = p^\alpha$ divides $r(i)$. For each solution x_j to Eq. (3.1), this is done by "sieving" the array $R[\,]$—modifying every qth cell. Once this process (via Eq. (3.1)) has been followed for all primes and prime powers, those cells $R[i]$ with value near $\log|r_i|$ are likely smooth. This is verified by factoring the corresponding r_i explicitly.

3.2 Practical Analysis of the Quadratic Sieve

In our analysis, we count what we refer to as *simple operations*—a wordsize (16- or 32-bit) arithmetic operation, equivalent to an add, shift, or move, on a single-processor machine. (In actual implementations, sieving operations may in fact be byte operations; this saves memory, but not time.) We emphasize that the analysis below does not attempt to establish the actual real time required to carry out the algorithm, but rather serves to derive an estimated operation count to which a similar analysis of discrete logarithm algorithms can be compared.

In practice, the condition $q = p^\alpha < N$ in Eq. (3.1) is relaxed, for example, to $q = p^\alpha < B$. It is not expected that this results in the loss of many smooth residues [45, Section 3]. Since for each solution x_j, $j = 1, 2$ to Eq. (3.1) the sieve processes the interval using stride q, we are interested in the sum

$$S = \sum_{q=p^a<B;\ p\in FB} \frac{1}{q} \tag{3.2}$$

(We shall use S in the analysis which follows.) S can be approximated for our purposes by

$$S_1 = \sum_{p<B;\ p\in FB} \frac{1}{p} \tag{3.3}$$

It is well known [19, Section 22.7] that $\Sigma_{p\leq B}p^{-1} = \ln\ln(B) + S_2 + o(1)$, where $S_2 = \gamma + \Sigma_p\{\ln(1 - p^{-1}) + p^{-1}\}$, $\gamma = 0.57721566\ldots$; for our purposes $S_2 \approx 0.26$ more than suffices for this convergent sum. Since about half the primes less than B are in the factor base, we use $S_1 \approx (1/2)\ln\ln(B)$.

A disadvantage of using a single polynomial $r(x)$ is that the magnitude of $r_i = r(i)$ increases linearly with i, and this is undesirable since the probability that a number is smooth decreases as its size increases. For the *multiple polynomial* version of the quadratic sieve, many distinct polynomials $r(x)$ are constructed and employed, allowing a smaller interval size M to be used on each of these. The penalty paid is the cost of determining the new polynomial, the solution of the congruences (Eq. (3.1)) for each new polynomial, and the reinitialization of the sieve interval. In practice the pay-off more than compensates for these costs; furthermore, multiple polynomials facilitate parallelization.

While other factors bear on the asymptotic running time of the algorithm (notably finding the linear dependency), it appears in practice that the sieving step dominates. Caron and Silverman [11, Section 3] report that sieving takes 80–85% of the running time on a machine with 32 × 32-bit arithmetic, but the time required for changing polynomials dominates on machines with only 16 × 16-bit arithmetic. Pomerance, Smith, and Tuler [45, Section 5] suggest the algorithm parameters be adjusted so that the time for sieving matches the time for preparing a polynomial and sieve initialization data. Once in possession of a sufficient number of smooth residues, finding a linear dependency/postprocessing does not appear to be a time bottleneck in practice ([11, Section 1], [45, Section 5], and most recently [24]—see Section 2.5). Again, while equation collection can be done using a loosely coupled network of workstations, use of an appropriately large machine is necessary to effectively handle the resulting large sparse linear system. In light of these observations, we shall take as a measure of the running time of the quadratic sieve algorithm the time required for the sieving process. This appears reasonable provided an appropriately large interval size M is selected (since the cost of changing polynomials is reduced by staying with each polynomial for a longer period of time), however, this assumption requires further examination, which we will not pursue here. If the cost of changing polynomials is as great as sieving itself, the operation count for the algorithm will be two or three times that of the sieving stage, depending on the effort required to deal with the linear system. Regarding algorithmic parameters, asymptotic estimates for the optimal size of the factor base, as a function of N, can be established; in practice, B and M are best determined and refined by experimentation, and are strongly influenced by available memory resources (e.g., see [28,51]).

In our analysis here, we consider the version of the quadratic sieve described in [45, Section 8], using multiple polynomials as suggested by Peter Montgomery [51], and multiplier 1. (The use of an appropriate "multiplier" (e.g., see [45, Section 4], [38, Sections 4.5, and 5.3]) can be used to skew the factor base in favor of more small primes, at the cost of residues of somewhat larger magnitude.) The residues then have maximum size $M\sqrt{N/2}$, and for the purpose of estimating the probability of smoothness, it appears reasonable to assume "typical" residues obtained have magnitude $Z = (1/3)M\sqrt{N/2}$. Let $r(u)$ be the probability that a residue is smooth with respect to B (notation to be justified shortly). We require an estimate for $r(u)$. Let $\Psi(x, y)$ denote the number of positive integers $\leq x$ that are free of prime divisors $> y$. $\Psi(Z, Z^{1/u})/Z$ is essentially the probability we require, for $B = Z^{1/u}$. This function has received much attention in the literature. It is well known that for fixed $u \geq 1$, $\lim_{x\to\infty}\Psi(x, x^{1/u})/x = \rho(u)$, where $\rho(u)$, is *Dickman's function* (see [40, Section 3] for a comprehensive survey on $\Psi(x, y)$). Knuth and Trabb Pardo [20] provide an error bound on the estimation of $\Psi(x, x^{1/u})/x$ by $\rho(u)$, and tabulate Dickman's function for discrete values in the range $1 \leq u \leq 10$. The bound on this error term is somewhat larger than desirable for larger values of u, but is apparently difficult to tighten. Canfield, Erdös, and Pomerance [9, Corollary to Theorem 3.1] have established that if $\varepsilon > 0$ is arbitrary and $3 \leq u \leq (1 - \varepsilon)\ln x/\ln\ln(x)$, then

$$\Psi(x, x^{1/u})/x = e^{-u(\ln u+(\ln\ln u-1)(1+1/\ln u)+\mathrm{E}(x,\,u))}, \text{ where } |\mathrm{E}(x, u)| \leq c_\varepsilon \left(\frac{\ln\ln u}{\ln u}\right)^2$$

and the constant c_ε depends only on ε. The tabulations of Dickman's function have been used by others to estimate smoothness probability (e.g., [45, 54]). We use as our approximation simply $r(u) \approx e^{-u\ln u}$, which appears to be as good an estimate as any currently available, agrees reasonably well with Dickman's function in the range $3 \leq u \leq 10$, and is a crude approximation to the result of Canfield et al. for $u > 10$. To increase the accuracy of our estimates, we would like to obtain better estimates to $r(u)$ for discrete values of x and u, but these are unavailable at present. Bach [1, Section 4] also comments on this situation, and offers suggestions.

For fixed N and a chosen value M, selecting $u > 1$ determines $B = Z^{1/u}$, and then we have $b \approx B/2\ln B$ from the prime number theorem and the fact that about half the primes less than B are in the factor base. (Better approximations for $\pi(x)$ are available, for example the Chebychev formula $\pi(x) \approx \int_2^x dx/\ln x$, but are unnecessary for our purposes.) Note that Z, and thus B and b, depend on M here.

We now estimate the number of simple operations required for the sieving phase. We do not factor in the time required to access data structures, etc. For each solution x_j, $j = 1, 2$ to Eq. (3.1) the sieve processes the interval using stride q. We charge three operations for the process of updating a cell each time it is "hit" during the sieving (as this typically will involve a read-add-write). Fix M and let K be the number of intervals of size $2M$ (= the number of polynomials) expected to be required to acquire b smooth residues. The number of operations is then roughly $12KMS$, where S is given by Eq. (3.2). Now to obtain b smooth residues, approximately $b/r(u)$ cells need be processed (meaning $K \approx r(u)^{-1}\, b/2M$). From this and Eqs. (3.2)–(3.3), the number of simple machine operations for sieving is estimated to be

$$3\ \ln\ln B \cdot b \cdot r(u)^{-1} \tag{3.4}$$

Recall that b here is the number of primes in the factor base, and hence the approximate number of relations required in the linear system; from this, an estimate of the memory requirements can be derived. Lenstra and Manasse [29] have used a factor base of 50,000 primes for typical 100-digit numbers; they have also employed a factor base of 65,500 elements to factor a 107-digit integer, using the "two-large-primes" variation.

Numeric determinations of optimal parameters for a fixed-size problem have been published by Wunderlich [54] for the continued fractions algorithm, but as noted by Morrison and Brillhart already in [38], optimizing parameters seems to be mainly a matter of experience. The situation has not changed (Silverman makes a similar comment [51]) and in fact appears more difficult for the multiple polynomial quadratic sieve, as the latter involves a greater number of parameters. Tables 4 and 5 illustrate sample parameters and estimated sieving operation counts (Eq. (3.4)) for the quadratic sieve algorithm as outlined above, for implementations for 100- and 155-digit composite integers N. Although crude, and based on a hypothetical model, they give a rough indication as to what parameter choices are possible, and their effects. From the tables, the sizes of factor bases that minimize the running time of the sieving stage, and/or are practical, can be estimated.

We note that the figures under the heading "Operations" in Tables 4 and 5 are estimates for the sieving phase only. For the larger values of b in these tables, the running time of the sieving phase will be dominated by the time taken (which is $O(b^2)$) to deal with the larger sparse linear systems; in these cases the figures under the "operations" heading are not attainable for the algorithm as a whole, and hence would not be used. Practical considerations (e.g., memory constraints, machine addressing limitations, etc.) also rule out the use of unreasonably large factor bases.

Speedups to quadratic sieve factorization. Other techniques are known for speeding up the basic version of the multiple polynomial quadratic sieve. As in the case for other similar integer factorization algorithms, and in the computation of discrete

TABLE 4. ESTIMATES FOR SIEVING PHASE $N \approx 10^{100}$ ($M = 10^7$)

u	b	B	Estimated $r(u)$	K	Operations
10.00	1.6708e + 04	4.3375e + 05	1.00000e − 10	8.3540e + 06	1.2849e + 15
9.75	2.2723e + 04	6.0504e + 05	2.27617e − 10	4.9916e + 06	7.7532e + 14
9.50	3.1430e + 04	8.5888e + 05	5.14785e − 10	3.0527e + 06	4.7892e + 14
9.25	4.4273e + 04	1.2425e + 06	1.15662e − 09	1.9139e + 06	3.0332e + 14
9.00	6.3610e + 04	1.8348e + 06	2.58117e − 09	1.2322e + 06	1.9731e + 14
8.75	9.3380e + 04	2.7705e + 06	5.72044e − 09	8.1619e + 05	1.3207e + 14
8.50	1.4033e + 05	4.2859e + 06	1.25875e − 08	5.5742e + 05	9.1169e + 13
8.25	2.1635e + 05	6.8079e + 06	2.74951e − 08	3.9343e + 05	6.5053e + 13
8.00	3.4302e + 05	1.1131e + 07	5.96046e − 08	2.8775e + 05	4.8110e + 13
7.75	5.6085e + 05	1.8787e + 07	1.28207e − 07	2.1873e + 05	3.6986e + 13
7.50	9.4856e + 05	3.2833e + 07	2.73552e − 07	1.7338e + 05	2.9659e + 13
7.25	1.6654e + 06	5.9635e + 07	5.78828e − 07	1.4386e + 05	2.4903e + 13
7.00	3.0478e + 06	1.1303e + 08	1.21427e − 06	1.2550e + 05	2.1988e + 13
6.75	5.8405e + 06	2.2463e + 08	2.52464e − 06	1.1567e + 05	2.0519e + 13
6.50	1.1783e + 07	4.7062e + 08	5.20072e − 06	1.1329e + 05	2.0352e + 13
6.25	2.5185e + 07	1.0461e + 09	1.06108e − 05	1.1868e + 05	2.1600e + 13
6.00	5.7442e + 07	2.4854e + 09	2.14335e − 05	1.3400e + 05	2.4717e + 13
5.75	1.4101e + 08	6.3663e + 09	4.28460e − 05	1.6455e + 05	3.0773e + 13
5.50	3.7633e + 08	1.7763e + 10	8.47238e − 05	2.2209e + 05	4.2126e + 13

Previous Chapter Previous Page Home Next page Next Chapter

TABLE 5. ESTIMATES FOR SIEVING PHASE $N \approx 10^{155}$ ($M = 10^9$)

u	b	B	Estimated $r(u)$	K	Operations
12.25	3.1698*e* + 05	1.0233*e* + 07	4.68094*e* − 14	3.3858*e* + 09	5.6503*e* + 19
12.00	4.3462*e* + 05	1.4323*e* + 07	1.12157*e* − 13	1.9376*e* + 09	3.2574*e* + 19
11.75	6.0426*e* + 05	2.0337*e* + 07	2.67335*e* − 13	1.1302*e* + 09	1.9143*e* + 19
11.50	8.5263*e* + 05	2.9320*e* + 07	6.33833*e* − 13	6.7260*e* + 08	1.1479*e* + 19
11.25	1.2222*e* + 06	4.2963*e* + 07	1.49463*e* − 12	4.0887*e* + 08	7.0323*e* + 18
11.00	1.7819*e* + 06	6.4059*e* + 07	3.50494*e* − 12	2.5419*e* + 08	4.4062*e* + 18
10.75	2.6451*e* + 06	9.7303*e* + 07	8.17258*e* − 12	1.6183*e* + 08	2.8274*e* + 18
10.50	4.0032*e* + 06	1.5077*e* + 08	1.89458*e* − 11	1.0565*e* + 08	1.8608*e* + 18
10.25	6.1861*e* + 06	2.3867*e* + 08	4.36597*e* − 11	7.0844*e* + 07	1.2580*e* + 18
10.00	9.7755*e* + 06	3.8658*e* + 08	1.00000*e* − 10	4.8877*e* + 07	8.7519*e* + 17
9.50	2.6292*e* + 07	1.0945*e* + 09	5.14785*e* − 10	2.5537*e* + 07	4.6512*e* + 17
9.00	7.9162*e* + 07	3.4784*e* + 09	2.58117*e* − 09	1.5335*e* + 07	2.8427*e* + 17
8.50	2.7224*e* + 08	1.2666*e* + 10	1.25875*e* − 08	1.0814*e* + 07	2.0418*e* + 17
8.00	1.0966*e* + 09	5.4206*e* + 10	5.96046*e* − 08	9.1987*e* + 06	1.7703*e* + 17
7.50	5.3409*e* + 09	2.8161*e* + 11	2.73552*e* − 07	9.7620*e* + 06	1.9165*e* + 17
7.00	3.2770*e* + 10	1.8513*e* + 12	1.21427*e* − 06	1.3494*e* + 07	2.7050*e* + 17

logarithms, savings in running time (at the expense of memory) are possible due to a "large prime variation." A speedup by a factor of about 2.5 appears possible (see [28], Fig. 1). A "two-large-prime" variation appears to give a further speedup by about the same factor, at the expense of a denser linear system—making its solution more time-consuming, and significantly more memory for the storage of relations [29]. A "small prime variation" may slightly reduce the time spent sieving, perhaps by 20% [45]. Other variations, including use of a small multiplier (as briefly mentioned earlier), and use of different classes of polynomials, may offer further advantages.

3.3 Recent Advances

Very recently, dramatic theoretical progress has been made on the problem of integer factorization; for integers of special form, this has resulted in a very efficient factorization algorithm. The algorithm, originated by Pollard and refined by H. W. Lenstra, has been implemented by A. K. Lenstra and Manasse [27]. It makes use of algebraic number fields, and is referred to as the (special) *number field sieve*. It applies to numbers of the form $N = r^e \pm s$ for small integers r and s (and in fact, to a somewhat larger class of "special" numbers), and runs in heuristic expected time $\exp((c + o(1))(\ln N)^{1/3}$ $(\ln\ln N)^{2/3})$, where $c \approx 1.526$. In mid-June 1990, the factorization of a special-form 155-digit number (the ninth Fermat number, $2^{512} + 1$) was completed using this algorithm. As in previous factorizations (see [28]), this factorization employed hundreds of workstations communicating by electronic mail, and involved a sparse linear system of over 200,000 equations. The theory has been extended to the factorization of general integers, with a larger running time constant of $c = 3^{2/3} \approx 2.08$ (see further comments below). While asymptotically significantly faster than all previous general factoring algorithms, it does not currently appear that this (general) number field sieve is practical for integers of cryptographic interest; for numbers within the reach of present computational power, the multiple polynomial quadratic sieve reigns. In summary, this work in its present state does not impact the cryptographic security of RSA, although it now appears that integer factorization is asymptotically an easier problem than previously believed.

Of related theoretical interest is the work of Gordon [18], which argues that the ideas of number field sieve factorization can be applied to the computation of discrete logarithms in odd prime fields GF(p), resulting in a heuristic expected asymptotic running time for the latter problem which is of the same form as that of the former. To record asymptotic running times succinctly, we define

$$L_x[v, c] = e^{(c+o(1))x^v(\ln x)^{1-v}}$$

The heuristic expected running times for the best algorithms for the problems of interest can then be summarized as follows:

Factoring an integer N—(general) number field sieve:	$L_{\ln N}[1/3, 2.08]$
Discrete logarithms in GF(p)—via number field sieve:	$L_{\ln p}[1/3, 2.08]$
Discrete logarithms in GF(2^n)—Coppersmith algorithm:	$L_n[1/3, c]$, $1.3507 \le c \le 1.4047$

Note that the critical parameter v is 1/3 in all cases, although only the latter algorithm—for extracting discrete logarithms in GF(2^n)—is of practical use at the current time.

The constant $c = 2.08$ for the (general) number field sieve noted above has recently been improved to $c = 1.923$, by work due to J. Buhler, H. W. Lenstra, and C. Pomerance, with contributions from Len Adleman, and further reduced to $c = 1.902$ by Coppersmith [12a]. The theoretical crossover for the quadratic sieve and the (general) number field sieve is then in the general area of the often-suggested 512-bit moduli for RSA implementations. This is, however, presently only of theoretical interest since neither algorithm is currently capable of factoring integers this large. Also of theoretical interest, following a massive precomputation on the order of $L_{\ln N}[1/3, 2.01]$, Coppersmith's work allows number field sieve factorization with a per-factorization running time of $L_{\ln N}[1/3, 1.639]$.

Of great interest is the degree to which theoretical factorization advances have actually been of use in practice. M. Manasse and A. Lenstra have carried out extensive implementations over the past few years. As of mid-summer 1990, the state of the art in practice is as follows [31a]. Using only the idle time on a network of 200 loosely coupled engineering workstations over one month, any of the following tasks is currently feasible:

1. (Special) number field sieve factorization of 155-digit Cunningham numbers (of form $r^e \pm s$; r, s small)
2. (Two-large-prime) multiple polynomial quadratic sieve factorization of 116-digit general integers
3. Elliptic curve factorization to extract up to 40-digit factors from up to 200-digit general integers

We emphasize that, similar to the (special) number field sieve discussed above, the remarkable power of elliptic curve factorization is limited to a special class of integers—in this case, integers containing moderately large prime factors. Neither of these algorithms apply directly to RSA moduli, which should be carefully chosen in light of such known techniques.

4 COMPARING EL GAMAL IN GF(2^n) VERSUS RSA

In this section, we first briefly review the RSA and El Gamal cryptosystems, and then consider their relative security based on the discussions in Sections 2 and 3, above. We then compare other aspects of the cryptosystems.

4.1 Review of RSA and El Gamal Cryptosystems

In the RSA cryptosystem [46], each user has three integer parameters e, d, and n, where $n = pq$ with p and q large, suitably chosen primes, and the pair e, d satisfying the relation $ed \equiv 1 \pmod{\phi(n)}$, ϕ being the Euler totient function. Suppose we have two users, Alice and Bob. Alice has a public key consisting of the pair (e_A, n_A), and private key d_A; likewise, Bob has (e_B, n_B) and d_B. If Alice wishes to send the encrypted message M to Bob, she encrypts it using Bob's public key, computing $C = M^{e_B} \pmod{n_B}$. If Alice wishes to send the signed message M to Bob, she signs it using her own private key, computing $C = M^{d_A} \pmod{n_A}$. Encryption and signing of messages can be composed. By considering M to be a key K, it is clear that the system can be used for key exchange.

To describe the El Gamal system [17] in the field GF(q), where q is a prime or prime power (with $q = 2^n$ of interest here), let α be a primitive element in the field. Alice now has a private key a and a public key α^{-a}; likewise Bob has b and α^{-b}. To send the message M to Bob encrypted, Alice selects a random integer r, and sends the pair $(\alpha^r, \alpha^{-br}m)$. Here m is the field element with integer representation M. Bob computes $(\alpha^r)^b \cdot \alpha^{-br}m$ to recover m. If M is a key K chosen by Alice, then this serves as a key exchange mechanism. (This is not an *authenticated* key exchange, since an imposter Charlie could masquerade as Alice to Bob, but the protocol can be modified to prevent this.) To send a signed message M to Bob, Alice again selects a random integer r and computes α^r. Let R be an integer representation of α^r, with $0 \leq R \leq q - 1$. Alice also computes $s = (M + aR)r^{-1} \pmod{q - 1}$, and sends the message M along with the signature (α^r, s). Bob verifies the signature by computing $(\alpha^{-a})^R(\alpha^r)^s$, and checking this is equal to α^M. For security reasons (see [17]), a new value r must be chosen for each signature. To avoid the cost of computing r^{-1} each time, one might alter the protocol slightly as follows: Rather than as above, have Alice compute $s = -(M - rR)a^{-1} \pmod{q - 1}$, and as verification have Bob check that $(\alpha^r)^R(\alpha^{-a})^s$ is equal to α^M. Recently, a new signature protocol has been proposed by Schnorr [48], resulting in a shorter signature which is more efficient to construct and verify.

Since their inception, the difficulty with using most public key systems for encryption has been their throughput—until now, available implementations have operated at bit rates too low to accommodate even standard 64K bits/sec speech, let alone megabit and higher data rates. While throughputs are increasing (see Section 4.3), and elliptic curve systems offer hope (see Section 5), symmetric systems still apparently dominate for the encryption function, with public key systems of primary use for key exchange and signatures and authentication.

4.2 Security Comparison: Discrete Logs in GF(2^n) Versus Integer Factorization

In this section, we compare the running times for fixed sizes of integer factorization and the computation of discrete logarithms in GF(2^n). For our model of comparison, we refer back to Section 1. We focus on a relative comparison based on the present level of security offered by 512-bit integer factorization.

Asymptotic analysis for the quadratic sieve factorization of an integer N [43] and the Coppersmith discrete logarithm algorithm in GF(2^n) [41] gives respective (heuristic expected) asymptotic running times proportional to

$$e^{\sqrt{\ln N \ln\ln N}} \quad \text{and} \quad e^{c \cdot n^{1/3}(\ln n)^{2/3}}, \quad c \approx 1.35 \tag{4.1}$$

(For the Coppersmith algorithm, the asymptotic running time actually exhibits periodic oscillations with $1.3507 \leq c \leq 1.4047$ [41]; being cryptographically conservative we use 1.35.) Without additional knowledge, simply plugging numbers into guideline functions such as these and expecting to obtain accurate absolute running times is unreasonable; trying to compare algorithms for different problems based on the use of two such unrelated expressions is most unreasonable. Such formulas do not take into account running time constants, which might vary wildly for the different algorithms, nor do they reflect the difficulty of one "operation." However, before consulting the more careful analysis of previous sections, out of interest we pause to check what these guideline functions would suggest if interpreted literally. For 512-bit numbers N (Eq. (4.1)) gives a guideline for the difficulty of factoring as 6.69×10^{19} "operations" (where a factoring operation here is not clearly defined). To match this using the expression in Eq. (4.1) for discrete logarithms literally, requires $n \approx 850$. Similarly, 332-bit integer factorization in Eq. (4.1) yields 2.34×10^{15} "operations," which is matched by $n \approx 475$ for discrete logarithms in Eq. (4.1).

We now resort to the discrete analysis carried out in Sections 2 and 3, from which a more reliable comparison is expected since there the algorithms are analyzed on a more equal footing. Recall that the analysis in those sections estimated operation counts for the smoothness testing and sieving stages only, respectively; to facilitate comparison, we presume that these are the overall running times of the algorithms. Table 5 is not far off the value of Eq. (4.1) for the difficulty of factoring, with the analysis indicating that the sieving process as analyzed in our model could be carried out perhaps an order of magnitude faster than by Eq. (4.1) taken literally. However, this depends on the choice of database size used; note that dealing with a linear system involving over a million unknowns would appear to be formidable. On the other hand, Tables 2 and 3 suggest that Eq. (4.1) taken literally underestimates the difficulty of taking logarithms within the 700- to 850-bit range, as compared to our model. Tables 3 and 5 together suggest that n in the range of 700+ bits would make the smoothness testing in the computation of discrete logs about as difficult as the sieving phase in quadratic sieve factoring of 512 bits. This analysis suggests that it is reasonable to estimate that the GF(2^n) discrete logarithm problem of equivalent difficulty to 512-bit integer factorization (with respect to running time) is somewhere in the range of 700–800 bits. Similarly, Tables 1 and 4 indicate that 400-bit discrete logarithms appear to be of approximately the same difficulty as 100-digit factorization. Thus the analysis in Sections 2 and 3 suggests that for equivalent levels of security to integer factorization at

100 and 155 digits, the discrete logarithm problem in GF(2^n) requires somewhat smaller bitlengths n than those that result from aligning the expressions of Eq. (4.1) literally. Table 6 summarizes this discussion.

TABLE 6. ESTIMATED BITLENGTHS FOR EQUIVALENT SECURITY

Factoring	Discrete Logs (Characteristic 2)
332	400 +
512	700 +

The similarities between integer factorization and discrete logarithm algorithms have been discussed by many. For example, the sieving of residues is analogous to the smoothness testing in Coppersmith's algorithm; in both cases, the objective is to find a sufficient number of linear relations, and in both cases, large sparse linear systems must be dealt with. We now consider and contrast a number of items regarding these algorithms.

Largest problem instances to date. The current champion among general integers factored by the quadratic sieve algorithm is 116 digits ($\approx$ 386 bits), using the two-large-prime method [8a, 29]. The largest field for which the Coppersmith algorithm has been applied for extracting discrete logarithms is GF(2^{127})—that is, 127 bits.* While discrete logarithm problems in far larger fields GF(2^n) are tractable with this algorithm and current technology, computational number theorists have apparently not found it as fashionable a problem to pursue, one possible reason being the absence of a motivational analogue of the Cunningham tables [8], which have stimulated much of the work in integer factorization. However, with renewed interest in fields of characteristic 2, it is anticipated that researchers will begin to explore larger-scale implementations of the Coppersmith algorithm. For the harder problem of discrete logarithms in prime fields, a larger problem instance has been solved—a 192-bit field by LaMacchia and Odlyzko, as noted earlier [25]. This problem, which was motivated by a cryptographic scheme in actual use by a major computer corporation, provided among other things, experience solving a system of 300,000 equations in 100,000 unknowns.

Modifications to "basic" algorithms. The running time of the quadratic sieve algorithm can be improved by a factor of about 2.5 by the large prime variation, and another factor of about 2.5 by the two-large-primes variation, as noted in Section 3.2. The running time of the Coppersmith algorithm (modified as described in Section 2.3) for computing logs in GF(2^n) can be improved by a factor of up to 3 by the large irreducible factor method, and further by a factor of up to 2 by early abort strategies, as noted in Section 2.6. As these factors would appear to balance out with respect to running time, they are not factored into our analysis, as our objective is a relative comparison. In addition, for very large problem instances, it is unclear which, if any, of the large prime variations would be used, as they result in somewhat denser linear systems, and substantially increased memory requirements.

Attention given to problems. Both the problem of integer factorization and that of extracting discrete logarithms have been well-studied, the former having received more attention in both theory and in practice. In light of this, it might be suggested that current algorithms for computing discrete logarithms—particularly in GF(2^n)—may afford more room for improvement than those for integer factorization. While this might

*Footnote added in proof: Dan Gordon and Kevin McCurley (Sandia National Laboratories) have (1992) completed the Coppersmith precomputation for extracting discrete logarithms in GF(2^{401}) and have calculated all of the relations needed to do the precomputation for GF (2^{503}).

be true with respect to implementation (as opposed to asymptotic) speedups, it is interesting to note that the number field sieve for factoring [27] evolved from the Gaussian integer method proposed for computing discrete logarithms in GF(p) [13], and in turn led to an algorithm for computing discrete logarithms in GF(p) by Gordon [18]. Unfortunately, for neither the factorization nor the logarithm problem is there definitive evidence at hand that any of the currently known algorithms are even close to optimal. The inherent complexity of these problems appears most difficult to establish, and accordingly the optimality of the best current algorithms is most difficult to judge.

Difficulty of solving additional problem instances. In integer factorization and RSA, once the factors of the modulus N are determined, the secret RSA key corresponding to the public key can be found directly, and messages signed with this RSA key pair are then vulnerable with essentially no additional work. For discrete logarithm-based cryptosystems, once the stage 1 database is built, individual logarithms can be computed in a relatively short time (compared to the time required to compute the database), but the database itself, which is nontrivial in size for larger problem instances, must be retained and accessed during the computation of each individual logarithm.

Solution of the linear system. In the quadratic sieve algorithm, a linear dependency must be found in a binary linear system. This can be efficiently implemented by packing, for example, 32 coefficients in one word on a 32-bit machine, and making use of exclusive-or machine instructions. For the Coppersmith algorithm in GF(2^n), the linear system must actually be solved, modulo a prime (or several primes) on the order of n bits. The latter requires more time and space for systems with the same number of unknowns.

Large memory requirements arise in both algorithms in relation to the large sparse linear systems. Caron and Silverman [11] point out that in practice, most of the memory requirement arises from storing the large-prime factorizations in the large-prime variation of the quadratic sieve. The situation is worse in the two-large-prime variation [29]. Similar results would be expected for similar variations of the Coppersmith algorithm. In [41], Odlyzko notes that in the Coppersmith algorithm, for large problems the equation solution phase is limited more by memory than by running time. Thus for very large problems, memory becomes a real barrier affecting both the variation of an algorithm that is used, and the parameters chosen.

For the algorithms in question, note that running time and memory differ in one important respect. The running time is dominated by the sieving and smoothness testing stages in these algorithms, which can be performed using very large numbers of loosely coupled processors, as noted earlier; using more machines of the same size (perhaps through the help of additional anonymous friends with idle workstations) can compensate for increased running time demands of larger problems. However, for the linear algebra stage, one cannot similarly pool memory resources, since a loosely coupled system does not work for that stage—larger problems apparently require machines with larger memory capabilities.

4.3 El Gamal in GF(2^n) Versus RSA

Having examined the difficulty of the underlying number-theoretic problems on which the security of the cryptosystems in question is based, we now compare other aspects of the cryptosystems.

Exponentiation throughput. As modular exponentiation and discrete exponentiation in the field are required to implement the RSA and El Gamal cryptosystems, the throughput that is presently attainable for these operations is of interest. We consider this below.

Regarding modular exponentiation as used in RSA, a 1989 survey of hardware implementations is given by Brickell [7]. The RSA chips cited range up to 17K bits/sec for 512-bit exponentiation at 14 MHz. Recently, Shand et al., [50] have achieved a 226K bits/sec implementation of 508-bit RSA modular exponentiation. This timing is for RSA exponentiation using the Chinese remainder theorem (CRT). The technology used is *programmable active memory* (PAM) [4], and their system consists of a host processor driving three distinctly configured PAM boards, each PAM being on a 25×25 cm^2 printed circuit board; two of these are noted as 38-ns units. To achieve the cited throughput, two of the PAM boards are "programmed"—via an automated process taking about 30 min—with data customized for the particular RSA modulus being employed. This implementation gives a dramatic order of magnitude speedup over previous RSA implementations, although its nature makes comparison somewhat difficult, particularly in real-life cryptosystems where RSA moduli may change periodically. Other recent work includes [37].

Software implementations of modular exponentiation on digital signal processors rival custom-hardware speeds. An implementation on the Motorola DSP56000 by Michael Wiener of BNR achieves 13.4K bits/sec for 512-bit exponentiation using CRT, at a clock rate of 20.48 MHz; without the CRT, estimated throughput is 5.4K bits/sec. On the same processor, Dussé and Kaliski report software achieving 11.6K bits/sec for 512-bit exponentiation using CRT, and 4.6K bits/sec without CRT [16]. Shand, Bertin, and Vuillemin [50] report a 10.3K bits/sec software implementation of 512-bit RSA on a 25 MHz RISC-based workstation.

For exponentiation in GF(2^n) with $n = 593$, a single-chip 2-micron complementary metal-oxide-semiconductor (CMOS) implementation is available with average throughput of 150K bits/sec at 10 MHz; the chip has also been run at 20 MHz yielding 300K bits/sec [47]. These timings are for limited Hamming weight exponentiation, using exponents of maximum weight 30 to decrease the number of multiplications required in an exponentiation. The hardware implementation makes use of a so-called *optimal normal basis* [39] to reduce the complexity of multiplication in the field GF(2^{593}). It is projected that using 1-micron technology, a clock rate of 33 MHz, and a one-pass multiply (instead of a two-pass multiply), an average throughput of over 1M bits/sec, and perhaps 1.5M bits/sec, is achievable for such limited Hamming weight exponentiation.

Both modular exponentiation and discrete exponentiation in GF(2^n) can be implemented using hardware which performs n-bit multiplication (respectively, modular and in the field) in $O(n)$ clock cycles—in fact, in $n + c$ clocks for small constants c. Exponentiation in GF(2^n) can take advantage of the fact that, using a normal basis representation of fields elements, squaring an element is simply a circular shift of that element's coefficient vector, which cuts down the average number of multiplications in the standard "square-and-multiply" exponentiation technique from $3n/2$ to $n/2$; there is no analogy to this in modular exponentiation. To be fair, it should be noted that to facilitate the hardware implementation of normal basis multiplication, fields GF(2^n) should be sought which exhibit optimal normal bases (see [39] for further details). The more dramatic reason that throughputs mentioned for exponentiation in GF(2^n) are high is due to the use of limited Hamming weight exponents. The use of this technique for encryption in El Gamal–like cryptosystems appears safe—provided the Hamming weight is sufficiently large to rule out exhaustive-search attacks—and significantly cuts down on the number of multiplications required in an exponentiation.

Of course, in evaluating the speed of cryptosystems, one must consider the "types" of exponentiation that are required for both encryption and the application of signatures, and correspondingly, for decryption and the checking of signatures. For example in RSA, encryption is typically carried out with a small public exponent—$2^{16} + 1$ has been suggested by many—but this weight-2 exponent is not used to determine the throughput of RSA, since decryption is more costly. The corresponding secret exponent is in general full-length, and of expected weight $n/2$ for an n-bit modulus. Fortunately, decryption time can be reduced here through use of the Chinese remainder theorem, as the holder of the secret key can exploit knowledge of the factors p and q of his modulus N. (We note the work of Wiener [53] warning against the use of short decrypting exponents.) Thus in evaluating throughput, one should take into account the types and number of exponentiations required, and the weights of the exponents involved, as dictated by each stage of the particular encryption and signature protocols used.

Message expansion. For RSA encryption, there is no message expansion; the encrypted message is the same size as the original. Similarly, signed RSA messages have no message expansion. For the El Gamal system as described in [17] and above, encryption results in message expansion by a factor of 2, and signing a message results in message expansion by a factor of 3. As noted earlier, more recently proposed signature schemes for El Gamal–like systems have smaller message expansion [48].

Choice of arithmetic systems. It has been suggested as an option by some that the duration of use of each RSA key pair be restricted by a fixed "cryptoperiod," after which time the key pair be changed. This results in a new RSA modulus N, moving future computations into a new arithmetic system, giving an opponent less incentive to devote considerable resources to attacking a particular modulus N and key pair. Of course, the system is either impervious to a factoring attack or it is not; if a particular modulus size is insecure, moving to a new modulus does not prevent an attacker from continuing to factor the previous modulus, but does limit the damage to the lifetime of that modulus. Altering the length of the cryptoperiod may also provide system administrators with some degree of flexibility for adjusting the security of the system. A change of RSA key pair and modulus is easily accommodated by most modular exponentiation implementations (i.e., by those that handle general n-bit moduli), and there are many n-bit RSA moduli N offering the same degree of security.

The situation differs for the El Gamal cryptosystem in GF(2^n). Custom hardware is typically built to handle a particular representation of a field of a fixed dimension n. But more importantly, from the attacker's point of view, there is effectively only a single n-bit arithmetic system to attack for each value n, since all fields with 2^n elements are isomorphic and mappings between these fields are easily established. The lack of choice of arithmetic systems here might be viewed as a weakness relative to RSA, or El Gamal in GF(q)—for a fixed value n, a single database suffices for computing logarithms in GF(2^n). On the other hand, if n is chosen sufficiently large to resist the most concerted discrete logarithm attack, then there is no need to move to a new system. If, for reasons of implementation efficiency, it is desired to use fields $GF(2^n)$ that admit optimal normal basis representations, then the choice of values n is also restricted (see [39]).

Use of extension fields of $GF(2^n)$. To increase the security of the El Gamal cryptosystem in $GF(2^n)$, larger fields are used. Given hardware for multiplication (exponentiation) in $GF(2^n)$, the quadratic extension $GF(2^{2n})$ is appealing because the hardware for the smaller field can be reused. The multiplicative group here has $2^{2n} - 1 = (2^n - 1)(2^n + 1)$ elements. To guard against a combined discrete logarithm attack using a generalized Pohlig–Hellman algorithm [42] on the cyclic group of $2^n + 1$ elements, and index-calculus on the subgroup of $2^n - 1$ elements, it should be ensured that $2^n + 1$ has a large prime factor. Although it has been suggested that the presence of the subfield $GF(2^n)$ in $GF(2^{2n})$ may make computation of discrete logarithms in $GF(2^{2n})$ easier than in a similar-sized field that does not contain a subfield, aside from the above-mentioned idea no algorithm has been reported that gives weight to this suggestion. Presuming there is no weakness introduced, other extension fields of $GF(2^n)$ are similarly available. Analogously, the security of RSA can be increased by resorting to moduli N of greater bitlength, through use of appropriate modular multiplication (exponentiation) hardware or software for the larger bitlength.

5 RECENT WORK REGARDING ELLIPTIC CURVE CRYPTOSYSTEMS

In this section we survey recent work related to elliptic curve cryptosystems, and consider the cryptographic impact of recent progress in the computation of discrete logarithms over elliptic curves.

We first briefly review some background. Given a finite field F, the points of an elliptic curve E over F form an abelian group. A well-known theorem of Hasse states that the number of points k on an elliptic curve E over the field $F = GF(q)$ is of the form $k = q + 1 - t$, where $|t| \leq 2\sqrt{q}$, that is, the order of the group is $k \approx q$. The Diffie–Hellman key exchange technique, and the related El Gamal cryptosystem, are based on exponentiation, which is simply the repeated application of a group operation (multiplication in the original case). Therefore the elliptic curve group may be used to construct cryptosystems, as outlined by Miller [36] and Koblitz [22]. In the case of the elliptic curve group, the group is additive, with the addition operation requiring a small number of operations in underlying field F; the additive analogue of exponentiation is multiplication, that is, repeated addition in the group.

In an arbitrary group G, the *general discrete logarithm problem* is as follows: Given a specified element $g \in G$ of (typically maximum) order r, and any other group element $b \in G$, determine the unique y, $0 \leq y \leq r - 1$, such that $b = g^y$, if such an integer exists. The *elliptic curve logarithm problem* then follows: Given a point P of (typically maximum) order r in the elliptic curve group, and another point R in the group, determine unique y, $0 \leq y \leq r - 1$, such that $R = yP$, if such an integer exists. The elliptic curve analogues of the Diffie–Hellman key exchange and the El Gamal cryptosystem are insecure if the elliptic curve logarithm problem is easy.

Until recently, the best discrete logarithm attacks known which applied to elliptic curves were general methods applicable in any group, having running time $O(\sqrt{k})$ in a group of order k (e.g., see [1,26,32]). To avoid vulnerability to a Pohlig–Hellman attack [42], elliptic curves should be chosen such that the order of the relevant group has a large prime divisor. The order of the group in question can be determined via a polynomial time, but not very practical, algorithm due to Schoof [49] (see [23] for a further discussion); alternatively, subclasses of elliptic curves are known whose group orders can be easily determined, thus avoiding the need to use Schoof's algorithm ([3, 34]). The above-mentioned "square root" attacks are very weak, and the apparent absence of stronger general attacks has resulted in the belief that elliptic curve cryptosystems with relatively short keylengths may afford greater security than alternative cryptosystems with larger keylengths. To reiterate the point with an example, suppose it *were* true that the best elliptic curve logarithm attack on a particular curve having an elliptic curve group of order about 2^{132} (i.e., the curve is over a field GF(2^n) with $n \approx 132$) *was* one of running time precisely $\sqrt{k}$ "operations"; then extracting elliptic logarithms would take about 7.4×10^{19} such operations. Shorter keylengths translate into simpler implementations of arithmetic, and smaller bandwidth and memory requirements. Among other applications, this significantly impacts the design and feasibility of smart card systems.

Practical implementation of elliptic curve cryptosystems, in particular the elliptic curve analogue of the El Gamal cryptosystem, has recently been studied by Menezes and Vanstone [34]. They have explored the feasibility of hardware implementation of an arithmetic processor for carrying out elliptic curve computations over fields of characteristic 2. The class of curves of so-called *zero* j*-invariant* have received special attention as they offer significant additional computational advantages in implementing the group operation. Furthermore, these are convenient in that the order of the elliptic curve groups of curves over GF(2^n) of zero j-invariant are known [33]: for n odd, the order is one of $q + 1$, $q + 1 \pm \sqrt{2q}$; for n even, the order is one of $q + 1$, $q + 1 \pm \sqrt{q}$, $q + 1 \pm 2\sqrt{q}$. Of interest in what follows, it is also known that there are precisely three isomorphism classes of elliptic curves with j-invariant 0 when n is odd, and precisely seven such classes for n even (see [33] for a representative from each class). Finally, the case when n is odd is preferred because a point on the curve, which is in general an ordered pair (x, y) of elements of GF(2^n), can in that case be represented by the x-component and a single bit of the y-component. This can be used to reduce message expansion for encryption in the elliptic El Gamal system to a factor of $3/2$ [34, Section 5].

More recently, Menezes, Okamoto, and Vanstone [35] have made the first significant progress on the elliptic curve logarithm problem. For certain classes of curves over fields GF(q), they show how to reduce the elliptic curve logarithm problem in a curve E over GF(q) to the discrete logarithm problem in an extension field GF(q^k) of GF(q). In general, k is exponentially large and the reduction takes exponential time. However, for *supersingular* elliptic curves, k is small (k = 1, 2, 3, 4, or 6) and the reduction is probabilistic polynomial time, yielding a (probabilistic) subexponential-time elliptic curve logarithm algorithm. (If E is an elliptic curve over a field GF(q) where $q = p^m$, and the elliptic curve group has order $q + 1 - t$, then E is supersingular if p divides t. Hence for $p = 2$, the supersingular curves are precisely those curves with j-invariant equal to 0.) For curves over GF(2^n) of zero j-invariant, when n is odd the technique applies with $k = 2$ for one of the isomorphism classes of curves, and for $k = 4$ for the other two classes; when n is even the technique also applies to each of the seven classes of curves with k = 3, 2, or 1. Hence among curves over GF(2^n) of zero j-invariant, the classes of curves recommended for cryptographic use are those for which k is exactly 4, and in this case computing elliptic curve logarithms is essentially no more difficult than computing discrete logarithms in the quartic extension GF(2^{4n})—which would be done via the Coppersmith algorithm. As a very rough guideline, to match the level of security offered by 512-bit RSA, if computing discrete logarithms in GF(2^n) for $n = 700$ or 800 is as difficult as factoring 512-bit numbers, then for an elliptic curve over GF(2^m) which has $k = 4$ as discussed above, one would require $m \approx 175$ or 200.

To summarize, in light of recent results, in using elliptic curve cryptosystems over GF(2^n), one must now (i) take special care in the particular choice of elliptic curve, and (ii) compensate for the attack by using appropriately larger fields to preserve the security. These larger fields may still be smaller than those required for equivalent security in other types of cryptosystems, so that elliptic curve cryptosystems remain attractive in practice, albeit to a somewhat lesser degree than prior to discovery of this new technique. Arithmetic operations in elliptic curve systems are easier due to the fact that smaller fields can be used, but this is partially offset by the fact that several arithmetic operations in the underlying field are required for one ''elliptic curve operation.'' Ironically, the classes of curves susceptible to the new attack include many of those that have been recommended for use, including curves suggested in [3], [22], [34], and [36].

6 CONCLUDING REMARKS

It is well known that the computation of discrete logarithms in GF(2^n) is easier than the factorization of n-bit integers N, using the best currently known algorithms for each problem. Hence, for example, the El Gamal cryptosystem in GF(2^n) is less secure than RSA using an n-bit modulus N. This can be compensated for by using larger bitlengths n for the GF(2^n) system. Indeed, fields GF(2^n) can be carefully chosen that are larger and yet admit efficient arithmetic resulting in high throughput; the price is somewhat larger key sizes, and greater bandwidth and storage requirements. Asymptotic running times for the best practical algorithms for integer factorization and discrete logarithms in GF(2^n) reveal that for similar levels of security, the relative bitlength by which the GF(2^n) system must exceed the bitlength in RSA increases with the bitlength of the RSA system. For extremely secure (by present measures) RSA systems, such as 1024 bits, the corresponding bitlength for El Gamal in GF(2^n) becomes problematic. To overcome problems due to larger key sizes, serious consideration should be given to elliptic curve systems over GF(2^n). However, in light of recent advances, one must now exercise caution in the choice of elliptic curves, and in the size of the underlying fields. Further experience and large-scale implementations for computing discrete logarithms in larger fields GF(2^n) will provide further empirical knowledge leading to a better understanding and more accurate practical estimates of the relative difficulty of integer factorization and the computation of discrete logarithms in GF(2^n).

ACKNOWLEDGMENTS

The author would like to thank Arjen Lenstra, Alfred Menezes, Gus Simmons, Scott Vanstone, and Michael Wiener for discussions and comments, and for bringing various research to his attention.

References

[1] E. Bach, "Intractable problems in number theory," in *Lecture Notes in Computer Science 403; Advances in Cryptology: Proc. Crypto'88,* S. Goldwasser, Ed., Santa Barbara, CA, Aug. 21–25, 1987, pp. 77–93. Berlin: Springer-Verlag, 1990.

[2] E. Bach, "Number-theoretic algorithms," *Ann. Rev. Comput. Sci.,* vol. 4, pp. 119–172, 1990.

[3] A. Bender and G. Castagnoli, "On the implementation of elliptic curve cryptosystems," in *Lecture Notes in Computer Science 435; Advances in Cryptology: Proc. Crypto'89,* G. Brassard, Ed., Santa Barbara, CA, Aug. 20–24, 1989, pp. 186–192. Berlin: Springer-Verlag, 1990.

[4] P. Bertin, D. Roncin, and J. Vuillemin, "Introduction to programmable active memories," in *Systolic Array Processors,* J. McCanny, J. McWhirter, and E. Swartzlander, Eds., Englewood Cliffs, NJ: Prentice Hall, pp. 301–309, 1989.

[5] I. F. Blake, R. Fuji-Hara, R. C. Mullin, and S. A. Vanstone, "Computing logarithms in finite fields of characteristic two," *SIAM J. Alg. Disc. Meth.* vol. 5, no. 2, pp. 276–285, June 1984.

[6] I. F. Blake, P. C. van Oorschot, and S. A. Vanstone, "Complexity issues for public key cryptography," in *Performance Limits in Communication Theory and Practice,* J. K. Skwirzynski, Ed., Amsterdam: Kluwer Academic Publishers, pp. 75–97, 1988.

[7] E. F. Brickell, "A survey of hardware implementations of RSA (abstract)," in *Lecture Notes in Computer Science 435; Advances in Cryptology: Proc. Crypto'89,* G. Brassard, Ed., Santa Barbara, CA, Aug. 20–24, 1989, pp. 368–370. Berlin: Springer-Verlag, 1990.

[8] J. Brillhart, D. H. Lehmer, J. L. Selfridge, B. Tuckerman, and S. S. Wagstaff, Jr., *Factorizations of* $b^n \pm 1$, $b = 2,3,5,6,7,10,11,12$ *Up to High Powers,* 2nd ed., *Contemporary Mathematics,* vol. 22, Amer. Math. Soc., 1988.

[8a] Update, dated March 12, 1991, by S. Wagstaff.

[9] E. R. Canfield, P. Erdös, and C. Pomerance, "On a problem of Oppenheim concerning 'Factorisatio Numerorum'," *J. Number Theory,* vol. 17, no. 1, pp. 1–28, Aug. 1983.

[10] D. G. Cantor and H. Zassenhaus, "A new algorithm for factoring polynomials over finite fields," *Math. Comp.,* vol. 36, pp. 587–592, 1981.

[11] T. T. Caron and R. D. Silverman, "Parallel implementation of the quadratic sieve," *J. Supercomput.,* vol. 1, pp. 273–290, 1988.

[12] D. Coppersmith, "Fast evaluation of logarithms in fields of characteristic two," *IEEE Trans. Inform. Theory,* vol. IT-30, no. 4, pp. 587–594, July 1984.

[12a] D. Coppersmith, "Modifications to the number field sieve," IBM Research Report #RC 16264 (Nov. 1990; updated Mar. 1991).

[13] D. Coppersmith, A. M. Odlyzko, and R. Schroeppel, "Discrete logarithms in GF(p)," *Algorithmica,* vol. 1, no. 1, pp. 1–15, 1986.

[14] J. A. Davis, D. B. Holdridge, and G. J. Simmons, "Status report on factoring," in *Lecture Notes in Computer Science 209; Advances in Cryptology: Proc. Eurocrypt'84,* T. Beth, N. Cot, and I. Ingermarsson, Eds., Paris, France, April 9–11, 1984, pp. 183–215. Berlin: Springer-Verlag, 1985.

[15] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Trans. Inform. Theory,* vol. IT-22, no. 6, pp. 644–654, Nov. 1976.

[16] S. R. Dusse and B. S. Kaliski, Jr., "A cryptographic library for the Motorola DSP56000," in *Lecture Notes in Computer Science 493; Advances in Cryptology, Proc. Eurocrypt '90,* Aarhus, Denmark, May 21–24, 1990; I. Damgård, Ed., pp. 230–244. Berlin: Springer-Verlag, 1991.

[17] T. El Gamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Trans. Inform. Theory,* vol. IT-31, no. 4, pp. 469–472, July 1985.

[18] D. M. Gordon, "Discrete logarithms in GF(p) using the number field sieve," presented at the *Workshop on Number Theory and Algorithms,* Mathematical Sciences Research Institute (Berkeley, CA), March 26–29, 1990.

[19] G. H. Hardy and E. M. Wright, *An Introduction to the Theory of Numbers,* Oxford: Oxford University Press, 5th ed. (reprinted with corrections), 1983.

[20] D. E. Knuth and L. Trabb Pardo, "Analysis of a simple factorization algorithm," *Theoretical Computer Science,* vol. 3, pp. 321–348, 1976.

[21] D. E. Knuth, *The Art of Computer Programming,* Vol. 2: *Semi-numerical Algorithms,* 2nd ed., Reading, MA: Addison-Wesley, 1981.

[22] N. Koblitz, "Elliptic curve cryptosystems," *Math. Comp.,* vol. 48, pp. 203–209, 1987.

[23] N. Koblitz, "Constructing elliptic curve cryptosystems in characteristic 2," paper presented at Crypto'90, Santa Barbara, CA, Aug. 11–15, 1990; to appear in *Advances in Cryptology,* S. Vanstone, Ed. Berlin: Springer-Verlag (in press).

[24] B. A. LaMacchia and A. M. Odlyzko, "Solving large sparse linear systems over finite fields," in *Lecture Notes in Computer Science 537; Advances in Cryptology*: Proc. Crypto'90. A. J. Menezes and S. A. Vanstone, Eds., Santa Barbara, CA, Aug. 11–15, 1990, pp. 109–133. Berlin: Springer-Verlag, 1991.

[25] B.Ä. LaMacchia and A. M. Odlyzko, "Computation of discrete logarithms in prime fields," in Designs, Codes, and Cryptography, vol. 1, pp. 46–62, 1991.

[26] A. K. Lenstra and H. W. Lenstra, Jr., "Algorithms in number theory," in *Handbook of Theoretical Computer Science,* A. Meyer, M. Nivat, M. Paterson, and D. Perrin, eds., North Holland, Amsterdam, volume A, chapter 12, pp. 673–715, 1990.

[27] A. K. Lenstra, H. W. Lenstra, Jr., M. S. Manasse, and J. M. Pollard, "The number field sieve," *Proc. 22nd ACM Symp. Theory of Computing,* pp. 464–572, 1990.

[28] A. K. Lenstra and M. S. Manasse, "Factoring by electronic mail," in *Lecture Notes in Computer Science 434; Advances in Cryptology; Proc. Eurocrypt'89,* J.-J. Quisquater and J. Vandewalle, Eds., Houthalen, Belgium, April 10–23, 1989, pp. 355–371. Berlin: Springer-Verlag, 1990.

[29] A. K. Lenstra and M. S. Manasse, "Factoring with two large primes," in *Lecture Notes in Computer Science 473; Advances in Cryptology, Proc. Eurocrypt'90,* Aarhus, Denmark, May 21–24, 1990; I. Damgård, Ed., pp. 72–82. Berlin: Springer-Verlag, 1991.

[30] H. W. Lenstra, Jr., "Factoring with elliptic curves," *Ann. Math.,* vol. 126, pp. 649–673, 1987.

[31] R. Lidl and H. Niederreiter, *Finite Fields, Encyclopedia of Mathematics and Its Applications,* Reading MA: Addison-Wesley, vol. 20, 1983.

[31a] M. Manasse, seminar given at Sandia National Laboratories, Albuquerque, NM, June 27, 1990, communicated by Gustavus J. Simmons.

[32] K. S. McCurley, "The discrete logarithm problem," in *Cryptology and Computational Number Theory; Proc. Symp Appl. Math.,* vol. 42, C. Pomerance, Ed.,

Boulder, CO, Aug. 6–7, 1989, pp. 145–166. Providence, RI: American Mathematical Society (1990).

[33] A. Menezes and S. Vanstone, ''Isomorphism classes of elliptic curves over finite fields,'' Research Report 90-91 (Jan. 1990), Faculty of Mathematics, University of Waterloo, revised manuscript (Aug 1990).

[34] A. Menezes and S. Vanstone, ''The implementation of elliptic curve cryptosystems,'' in *Lecture Notes in Computer Sciences 453; Advances in Cryptology*: Auscrypt'90. J. Seberry, J. Pieprzyk, Eds. Sydney, Australia, Jan. 1990, pp. 2–13. Berlin: Springer-Verlag, 1990.

[35] A. Menezes, T. Okamoto, and S. Vanstone, ''Reducing elliptic curve logarithms to logarithms in a finite field,'' (Unpublished manuscript, Sept. 1990).

[36] V. Miller, ''Uses of elliptic curves in cryptography,'' in *Lecture Notes in Computer Science 218; Advances in Cryptology: Proc. Crypto'85,* H. C. Williams, Ed., Santa Barbara, CA, Aug. 18–22, 1985, pp. 417–426. Berlin: Springer-Verlag, 1986.

[37] H. Morita, ''A fast modular-multiplication algorithm based on a radix 4 and its application,'' *Trans. IEICE,* vol. E 73, no. 7, special issue on Cryptography and Information Security, pp. 1081–1086, July 1990.

[38] M. A. Morrison and J. Brillhart, ''A method of factoring and the factorization of F_7,'' *Math. Comp.*, vol. 29, no. 129, pp. 183–205, Jan. 1975.

[39] R. Mullin, I. Onyszchuk, S. Vanstone, and R. Wilson, ''Optimal normal bases in GF(p^n),'' *Discrete Applied Mathematics,* vol. 22, pp. 149–161, 1988/1989.

[40] K. K. Norton, ''Numbers with small prime factors, and the least kth power non-residue,'' *Memoirs of the AMS,* no. 106, 1971.

[41] A. M. Odlyzko, ''Discrete logarithms in finite fields and their cryptographic significance,'' in *Lecture Notes in Computer Science 209; Advances in Cryptology: Proc. Eurocrypt'84,* T. Beth, N. Cot, and I. Ingemarsson, Eds., Paris, France, April 9–11, 1984, pp. 224–314. Berlin: Springer-Verlag, 1985.

[42] S. C. Pohlig and M. E. Hellman, ''An improved algorithm for computing logarithms over *GF*(p) and its cryptographic significance,'' *IEEE Trans. Inform. Theory,* vol. IT-24, no. 1, pp. 106–110, Jan. 1978.

[43] C. Pomerance, ''Analysis and comparison of some integer factoring algorithms,'' in *Computational Methods in Number Theory,* H. W. Lenstra, Jr., and R. Tijdeman, Eds., *Math. Centrum Tract*, vol. 154, pp. 89–139, 1982.

[44] C. Pomerance, ''The quadratic sieve factoring algorithm,'' in *Lecture Notes in Computer Science 209; Advances in Cryptology: Proc. Eurocrypt'84,* T. Beth, N. Cot, and I. Ingemarsson, Eds., Paris, France, April 9–11, 1984, pp. 169–182. Berlin: Springer-Verlag, 1985.

[45] C. Pomerance, J. W. Smith, and R. Tuler, ''A pipeline architecture for factoring large integers with the quadratic sieve algorithm,'' *SIAM J. Computing,* vol. 17, no. 2, pp. 387–403, April 1988.

[46] R. Rivest, A. Shamir, and L. Adleman, ''A method for obtaining digital signatures and public-key cryptosystems,'' *Commun. ACM,* vol. 21, pp. 120–126, 1978.

[47] T. Rosati, ''A high speed data encryption processor for public key cryptography,'' in *Proceedings of the IEEE Custom Integrated Circuits Conference,* San Diego, May 15–18, 1989, pp. 12.3.1–12.3.5.

[48] C. P. Schnorr, ''Efficient identification and signatures for smart cards,'' in *Lecture Notes in Computer Science 435; Advances in Cryptology: Proc. Crypto'89,* G. Brassard, Ed., Santa Barbara, CA, Aug. 20–24, 1989, pp. 239–351. Berlin: Springer-Verlag, 1990.

[49] R. Schoof, "Elliptic curves over finite fields and the computation of square roots mod p," *Math. Comp.*, vol. 44, pp. 483–494, 1985.

[50] M. Shand, P. Bertin, and J. Vuillemin, "Hardware speedups in long integer multiplication," *Proceedings of the 2nd ACM Symposium on Parallel Algorithms and Architectures,* Crete, July 2–6, 1990, (in press).

[51] R. D. Silverman, "The multiple polynomial quadratic sieve," *Math. Comp.*, vol. 48, pp. 329–339, 1987.

[52] D. H. Wiedemann, "Solving sparse linear equations over finite fields," *IEEE Trans. Inform. Theory,* vol. IT-32, no. 1, pp. 54–62, Jan. 1986.

[53] M. J. Wiener, "Cryptanalysis of short RSA secret exponents," *IEEE Trans. Inform. Theory,* vol. IT-36, no. 3, pp. 553–558, May 1990.

[54] M. C. Wunderlich, "Implementing the continued fraction factoring algorithm on parallel machines," *Math. Comp.*, vol. 44, no. 169, pp. 251–260, Jan. 1985.