RJ 6433 (62551) 9/14/88
Mathematics

CRYPTOGRAPHIC KEY DISTRIBUTION AND COMPUTATION IN CLASS GROUPS

Kevin S. McCurley *
IBM Research Division
Almaden Research Center, K53
650 Harry Road
San Jose, CA 95120-6099

ABSTRACT: There are numerous cryptosystems that have their security based on the presumed difficulty of computational problems in number theory. In particular, variations of the Diffie and Hellman key distribution scheme have been proposed that rely on the difficulty of discrete logs in various groups, on the difficulty of factoring integers, and on the difficulty of computing the class number of an imaginary quadratic number field. We begin by surveying the development of key distribution schemes, and conclude with a discussion of computational problems in class groups. We present new probabilistic algorithms for calculating class numbers of imaginary quadratic number fields, and for calculating discrete logarithms in class groups. We also prove under the assumption of the extended Riemann hypothesis that the problem of calculating the class number of an imaginary quadratic number field belongs to the complexity class NP.

*Much of this research was carried out while the author was at the University of Southern California, with support by a grant from the National Security Agency.

1 The Diffie and Hellman Key Distribution Scheme

In 1976, Diffie and Hellman wrote a paper [12] that began a revolution in the design of secure communication systems, a field known as *cryptography.* A central idea in their paper is that secure communications might be conducted between two parties without any prior exchange of secret information. They discussed many different types of cryptosystems, including key distribution systems, public-key cryptosystems, and authentication schemes via digital signatures. These developments have opened a new area of applications of number theory to cryptography, and sparked a renewed interest in computational number theory.

In the first part of this paper we shall describe several different variations of the original Diffie-Hellman key distribution scheme, with emphasis on the computational problems in number theory that are involved. This shall lead us to consider computational problems in class groups of imaginary quadratic number fields, and in the second part of the paper we shall describe several new algorithms for such problems.

The purpose of a key distribution scheme is to allow two parties to conduct a conversation over an insecure communication channel, after which they are both able to construct a cryptographic key (string of bits) that is infeasible for an eavesdropper to construct. In the original paper of Diffie and Hellman, they proposed a very simple scheme that has now found widespread use. Since their original proposal, several variations have been proposed that have the potential to offer greater security. Of these, several can be viewed within the following framework. Let G be a finite group (written multiplicatively), and let $g \in G$. Assume that the following information is publicly known:

- the group element g,
- a fast algorithm for computing $a \cdot b$ when given $a \in G$, $b \in G$.
- an easily computable injective function $f : G \longrightarrow \{0, 1, \ldots, M\}$.

The purpose of the function f is to provide a way to identify group elements with keys. The set $\{0, 1, \ldots, M\}$ is chosen only because it is a convenient set of keys. Note that M is an upper bound for the cardinality of the group G.

A Diffie and Hellman key distribution scheme can now be described as follows: Two parties A and B who wish to agree on a common key from the set $\{0, 1, \ldots, M\}$ proceed as follows:

- A chooses a random number x with $0 < x < M$, computes g^x, and sends the result to B , keeping x secret.
- B chooses a random number y with $0 < y < M$, computes g^y, and sends the result to A , keeping y secret.
- Both A and B use the key $f(g^{xy})$, which A computes as $f((g^y)^x)$, and B computes as $f((g^x)^y)$.

The well known binary method of exponentiation (see [21, p. 442]) can be used to compute g^{xy} in $O(\log M)$ group operations.

If this scheme is to be secure, then the problem of computing g^{xy} from knowledge of g, g^x and g^y should be intractable. We shall refer to this problem as the *Diffie-Hellman problem*. A related problem is the so-called *discrete logarithm problem*: given g and $a \in G$, find x with $g^x = a$, provided one exists. Clearly if one can solve a generic instance of the discrete logarithm problem, then one can also solve the Diffie and Hellman problem. It has further been conjectured that if one can solve the Diffie and Hellman problem, then one can also solve the discrete logarithm problem. This remains one of the interesting unsolved problems in the field, but progress was very recently made in this direction [7]. At least we can say that no one has found any other approach to the Diffie and Hellman problem other than by solving the discrete logarithm problem.

1.1 The Choice of a Group

The history of algorithms for discrete logarithms is somewhat obscure. Apparently the earliest efforts were concerned with the case where $G = (\mathbb{Z}/p\mathbb{Z})^*$ for a prime p, and g is a primitive root modulo p. These early efforts were directed toward compilation of tables of discrete logarithms rather than working on algorithms for individual logarithms. One such table appeared in Gauss' *Disquisitiones Arithmeticæ* [13] of 1801, in which he published a table of primitive roots and indices for the primes less than 100. Another notable example is Jacobi's *Canon Arithmeticus* [19] of 1839, which contains a table for the primes less than 10000 (a more complete history of tables appears in Lehmer's guide [25]). The techniques used by these early researchers were extremely tedious, and their efforts were probably focused more on managing the size of the tables rather than on producing the logarithms efficiently. Still, it does at least show that eminent mathematicians started thinking about the difficulty of the problem at a very early stage.

The discovery of algorithms with nontrivial complexity was delayed until well into the computer age [35], [34], [21, exercise 4.5.3]. The most efficient algorithm that is known for the Diffie and Hellman problem (and the discrete logarithm problem) in a general finite group uses $O(|G|^{1/2+\epsilon})$ group operations, where $|G|$ denotes the cardinality of the group G. For some specific choices of G however, extra knowledge concerning the group structure sometimes provide us with more efficient algorithms. As an extreme example, consider the case where a polynomial time computable isomorphism $\phi : G \longrightarrow \mathbb{Z}/n\mathbb{Z}$ is known for some integer n. In this case, the extended Euclidean algorithm provides a polynomial time algorithm for the discrete logarithm problem! To see this, notice that $\phi(g^x) = x\phi(g)$, so that solving $g^x = y$ can be accomplished by computing x so that $x\phi(g) \equiv \phi(y) \pmod{n}$.

Between these two extremes one might wonder if there can be any good choices for a group to use in the Diffie and Hellman scheme. The original Diffie and Hellman proposal used the multiplicative group of invertible elements in a finite field $\mathbb{Z}/p\mathbb{Z}$ with a prime number of elements. In this case the best known algorithm for the Diffie and Hellman problem uses $L(p)^{1+o(1)}$ group operations, where $L(p)$ is defined as

$$L(p) = \exp(\sqrt{\log p \log \log p}).$$

A completely different algorithm due to Silver, Pohlig, and Hellman [34] has running time $O(q^{1/2} \log^C p)$, where C is a constant and q is the largest prime factor of $p - 1$. In the case where q is small this provides a far more efficient algorithm.

Numerous other groups have been suggested for a Diffie and Hellman scheme. One possibility is to use a finite field of characteristic 2. This gives the advantage that the group arithmetic is easier to implement than in fields of prime order about the same size. Unfortunately this same structure that allows us to do the arithmetic faster also allows us to devise an asymptotically faster algorithm for the discrete logarithm problem [10]. For more information concerning modern algorithms for the discrete logarithm problem in finite fields, consult the survey by Odlyzko [32].

Another proposal for a group was made by Odoni, Varadharajan, and Sanders, [33], who suggested using $GL_n(\mathbb{Z}/p\mathbb{Z})$, the group of invertible $n \times n$ matrices over the finite field $\mathbb{Z}/p\mathbb{Z}$. While this may appear to add something to the problem, it turns out that discrete logarithms in this group can be computed by computing discrete logarithms in extension fields of $\mathbb{Z}/p\mathbb{Z}$ (see [32, p. 230]).

Yet another proposal made independently by Miller [31] and Koblitz [22] was to use the group of points on an elliptic curve over a finite field. The best known algorithms for such groups are as yet only the algorithms that work for arbitrary finite groups, so that one might be able to use a smaller group. A pessimistic view of these groups would say that since elliptic curves are endowed with such a rich algebraic structure, perhaps there is a way to exploit this to find a very efficient algorithm for discrete logarithms. This remains only speculation, and in the meantime the choice appears to be secure. More recently, Koblitz [23] has suggested the use of groups associated with higher dimensional varieties, since they seem to lack the structure built into elliptic curve groups.

1.2 A Scheme Based on Factoring

Since we have seen that the specific structure of the group may provide faster algorithms for discrete logarithms, there is some reason for hesitation in the choice of a group. Ideally, the goal of cryptography should be to design schemes for which there exists a proof of security. In the case of the Diffie and Hellman key distribution scheme, we would like to prove that the computation of the secret key is infeasible for an eavesdropper, while preserving the ability of the two parties A and B to easily construct the key. Unfortunately, the state of computational complexity theory does not allow us to do this. In fact, the type of result that we would like to prove is very similar to what would be required to prove that NP$\neq$P, which is probably the biggest unsolved problem in complexity theory. Because of such difficulties, the popular approach has been to devise cryptosystems whose security is based on computational problems that are *believed* to be difficult. With this approach, the best argument that can be made is one based on history, namely that a computational problem has been studied over a long period of time without an efficient algorithm being found.

The problem in computational number theory that has probably been studied most extensively is that of finding the prime factors of an integer. In particular, factoring was studied at length by Gauss [13], who regarded it as one of the most fundamental computational problems. The problem has continued to occupy the imagination of mathematicians in this century. Some of the earliest computational work done by machines was done by D. N. Lehmer for the purpose of factoring. Since the discovery of the RSA public key cryptosystem, there has been a flurry of activity involving new algorithms and the application of the most powerful computers yet developed. In spite of all this attention and the advances that have been made, factoring has remained a difficult problem.

Given the amount of attention that has been directed to factoring, it would seem desirable to construct a Diffie and Hellman key distribution scheme that makes use of the difficulty of factoring. This was in fact suggested independently by Z. Shmuely [44] and the author [30]. The basic idea is to construct an integer $n = pq$ that is the product of two large primes, and use the multiplicative group $(\mathbb{Z}/n\mathbb{Z})^*$. In the variation of [30], it is proved that if we use $g = 16$ and choose $n = pq$, where p and q are primes such that $p \equiv 3 \pmod 8$, $q \equiv 7 \pmod 8$, $(p-1)/2$ and $(q-1)/2$ are prime, and $(p+1)/4$ and $(q+1)/8$ each have a large prime factor, then if one can solve the Diffie and Hellman problem over $\mathbb{Z}/n\mathbb{Z}$, one can also recover the factors of n. Hence it is at least as difficult to break the scheme as it is to factor such an integer n.

The most important feature of this variation is that this scheme will remain secure if *either* factoring or the original Diffie-Hellman problem in $\mathbb{Z}/p\mathbb{Z}$ remains intractable. It is quite obvious that if one can solve the

Diffie and Hellman problem in $\mathbb{Z}/n\mathbb{Z}$, then one can also solve it in $\mathbb{Z}/p\mathbb{Z}$. Hence the new scheme is at least as difficult to break as the original proposal of Diffie and Hellman, and it is also as difficult to break as it is to factor n. This provides an extra measure of security which could prove to be crucial, since the scheme will still be secure even if someone discovers a fast algorithm for one of the two problems.

1.3 Class Groups

The real attraction of using the problem of factoring as a basis for security is that the problem has been studied over a long period of time. A natural question to ask is whether there are any other natural problems aside from factoring that are apparently difficult and have been studied from a computational viewpoint over a long period of time. One such problem is clearly the ***class number problem***. This is a problem that was studied at length by Gauss, who made it the centerpiece of his monumental *Disquisitiones Arithmeticæ*. Since then the theory of class groups has been the object of intense scrutiny by many eminent number theorists.

In order to describe the class number problem, we shall need a few definitions. Let $d > 4$ satisfy $d \equiv 0$ or $3 \pmod 4$, and let $C(-d)$ denote the group of $SL_2(\mathbb{Z})$ equivalence classes of primitive binary quadratic forms of discriminant $-d$ (see section 2 for definitions). Furthermore let $h(-d)$ denote the cardinality of $C(-d)$. From Gauss' point of view, the class number problem would be stated as follows: given an integer k, determine all integers d for which $h(-d) = k$. The form of the class number that is more suitable for our purposes is: Given a value of d, calculate $h(-d)$.

The possibility of using a class group for Diffie-Hellman key distribution was mentioned in [30], but Buchmann and Williams [8] also had the same idea and were the first to recognize the significance of such a choice. They proposed a scheme with a property that is similar to the one of [30], namely that it will remain secure if either of two presumably hard computational problems remains secure. This can be done by choosing a discriminant $d = pq$ that is the product of two large primes. They prove that if an eavesdropper can break the Diffie and Hellman problem in $C(-d)$ with randomly chosen bases and exponents, then the eavesdropper can also factor d. Hence their scheme provably combines the difficulty of factoring with the difficulty of breaking the Diffie and Hellman scheme in a class group.

All of the known approaches to finding discrete logarithms in a group G require the knowledge of $|G|$. In fact, an argument similar to that in [4] will show that this is to be expected, since any algorithm that will compute discrete logarithms for arbitrary powers of an element in a finite abelian group can be used to compute the order of that element. As a consequence of this situation, the known approaches to breaking the Buchmann-Williams scheme require the computation of $h(-d)$ in addition to the ability to factor d.

1.4 Outline of the Rest of the Paper

At the time that Buchmann and Williams wrote their paper, the fastest published algorithm for computing the class number $h(-d)$ was due to Shanks, and had running time of $O(d^{1/4+\epsilon})$, or $O(d^{1/5+\epsilon})$ assuming the extended Riemann hypothesis (ERH). These algorithms are deterministic, but have running times that are exponential in the size of the input d, and are clearly hopeless if the discriminant exceeds 10^{100}. In section 3 I shall sketch new two probabilistic algorithms for the computation of $h(-d)$. These algorithms have running times that are of the form $L(d)^c$ for a constant c, but the analysis of the running time depends on a heuristic assumption. In section 5 I shall sketch a related algorithm for the computation of discrete logarithms in $C(-d)$, when $C(-d)$ is cyclic and $h(-d)$ is provided. The expected running time of the discrete logarithm algorithm can be proved to be $L(d)^{1+o(1)}$ under the assumption of ERH.

The precise complexity of computing class numbers is unknown, but it is at least as difficult to compute $h(-d)$ as it is to factor d. In fact, some of the most sophisticated factoring methods are based on the theory of class groups (see [41] and [26]). To show just how little we know about computing $h(-d)$, it was pointed out in [6] that it is unknown whether the problem of computing $h(-d)$ belongs to the complexity class NP. In section 4 I shall give a partial answer to this question by sketching a proof that the problem of computing $h(-d)$ is in NP if the extended Riemann hypothesis is true.

Even though these new algorithms have conjectured asymptotically smaller running times than the previous best algorithms, it remains to be seen whether the crossover point between their running times is small enough to have any influence on the choice of d in the Buchmann-Williams scheme. Even under the most optimistic circumstances these new algorithms would run only as fast for a discriminant d as the fastest known factoring algorithms for a number of size d. Hence it should be easy to choose the size of d in such a way as to make the Buchmann-Williams scheme immune to such an attack.

2 Background Information on Class Groups

There are two possible points of view in any discussion of class groups; namely that of ideals or quadratic forms. The former has the advantage of allowing easy generalizations, while the latter has the advantage that it is slightly easier to describe. We shall use the language of forms; for a good exposition of the connection between the two languages, see the paper of Lenstra [28], or the book of Hua [18]. Let $d > 4$ be such that $d \equiv 0$ or $3 \pmod 4$. A binary quadratic form of discriminant $-d$ is a polynomial $ax^2 + bxy + cy^2$ with $b^2 - 4ac = -d$. We shall always assume that $a > 0$, so that our forms are positive definite. For convenience we shall write such a form as (a, b, c), or simply $(a, b, \cdot)$ when the discriminant is understood. A form (a, b, c) is called *primitive* if and only if $\gcd(a, b, c) = 1$, and a primitive positive definite form (a, b, c) is called *reduced* if and only if

$$-a < b \leq a < c \quad \text{or} \quad 0 \leq b \leq a = c.$$

Primitive positive definite forms $f_1 = (a_1, b_1, c_1)$ and $f_2 = (a_2, b_2, c_2)$ are called *equivalent* if and only if $a_1x^2 + b_1xy + c_1y^2 = a_2X^2 + b_2XY + c_2Y^2$, with

$$\begin{bmatrix} X \\ Y \end{bmatrix} = D \begin{bmatrix} x \\ y \end{bmatrix}.$$

for some matrix D such that $\det(D) = 1$. It is easy to show that this is indeed an equivalence relation on the set of primitive positive definite forms of a given negative discriminant, and we shall denote the equivalence class of a form f as $[f]$. The number of such equivalence classes is finite, and is called the class number $h(-d)$. It is known that the set of equivalence classes form a finite abelian group under the operation of "composition". This operation can be given explicitly as

$$[(a_3, b_3, c_3)] = [(a_1, b_1, c_1)] \cdot [(a_2, b_2, c_2)],$$

where

$$\begin{aligned}
a_3 &= a_1 a_2 / g^2 \\
b_3 &= b_2 + 2a_2 r / g \\
c_3 &= \frac{b_3^2 + d}{4a_3} \\
g &= \gcd(a_1, a_2, (b_1 + b_2)/2) \\
g &= \alpha a_1 + \beta a_2 + \gamma(b_1 + b_2)/2 \\
r &= \beta(b_1 - b_2)/2 - \gamma c_2.
\end{aligned}$$

So far the arithmetic of the group has been described in terms of the equivalence classes themselves, but fortunately each equivalence class contains a unique reduced form, and there is an efficient algorithm that will produce this reduced form. One such algorithm is as follows:

Reduction algorithm. Input a form f of discriminant $-d$, and output an equivalent reduced form (a, b, c).

Step 1 Set $(a, b, c) := f$.

Step 2 Find $\lambda \in \mathbb{Z}$ with $-a \leq b + 2\lambda a < a$.

Step 3 Set $(a, b, c) := (c + \lambda b + \lambda^2 a, -b - 2\lambda a, a)$.

Step 4 If (a, b, c) is not reduced, then return to Step 2. Otherwise output (a, b, c).

Henceforth we shall assume that the group operation consists of composition followed by reduction. Using this form of the group law, it is easy to test equality between group elements, and a single group operation requires $O(\log^2 d)$ bit operations.

The following result of Schoof [40, Corollary 6.2] shows that there is an easily computable set of generators for the class group $C(-d)$.

Theorem 2.1. *Let p_i be the ith prime with $\left(\frac{-d}{p_i}\right) = 1$, and let*

$$b_i = \min\{b \in \mathbb{Z}^+ : b^2 \equiv -d \pmod{4p_i}\}. \tag{2.1}$$

If ERH is true, then there exists an effectively computable constant c_0 such that the classes $[(p_i, b_i, \cdot)]$, $1 \leq i \leq m$ generate $C(-d)$ provided $p_m > c_0 \log^2 d$.

We remark that the precise value of c_0 can be deduced from the work of Bach [5].

3 Algorithms for Computing Class Numbers

The main goal of this section is to describe an algorithm for the computation of $h(-d)$ whose heuristic running time is $L(d)^{\sqrt{2}+o(1)}$. We also sketch the ideas required to produce an algorithm with running time of $L(d)^{3\sqrt{2}/4+o(1)}$ (note that $3\sqrt{2}/4 \approx 1.06$). The primary inspiration for these algorithms came from reading Seysen's paper [41], and in fact many of the arguments used here follow very closely those of [41] and Lenstra[26]. Seysen's paper deals with an algorithm to factor N by computing a very large multiple of the class number $h(-N)$ (or $h(-3N)$ if $N \equiv 1 \pmod 4$). The observation that led me to the algorithms described here is that if we can generate *random* multiples of the class number, then by generating several of these and taking gcd$'s$, we can obtain the class number itself rather quickly. It should be noted that H. W. Lenstra Jr. and A. K. Lenstra [27] have described similar ideas for computing class numbers and discrete logarithms in class groups.

The algorithms for computing $h(-d)$ that we shall describe are probabilistic, but may be classified as Las Vegas algorithms under the assumption of the extended Riemann hypothesis. This means that if the algorithm produces an output, then the output will be correct if the extended Riemann hypothesis is true. This is not a trivial point, since even if a suspected value of $h(-d)$ is known, it is not obvious how to verify its correctness. We shall return to this point later in section 4.

In order to prove that the algorithm will produce an output with non-negligible probability, we shall rely on a heuristic assumption. The precise statement of this assumption will be stated later, but it involves the assumption that the multiples of the class number generated behave as true random multiples. From a mathematical standpoint it is disappointing not to have a complete proof for the running time of the algorithm, but this is less important from a cryptographic standpoint. After all, if there is reason to believe that an algorithm may break the system, then this is sufficient justification to discard the system (or possibly increase the key size) without waiting for a proof that it is vulnerable.

3.1 Shanks' Algorithm

Before describing the new algorithm, it is instructive to consider the algorithm of Shanks [42], [43] (see also [40] or [28]). This algorithm combines three major ideas. The first is based on Dirichlet's class number formula, which for a discriminant $d > 4$ takes the form

$$h(-d) = \frac{\sqrt{d}}{\pi} \prod_{p \text{ prime}} \left(1 - \left(\frac{-d}{p}\right) p^{-1}\right)^{-1}.$$

The first idea used in Shanks' algorithm is to use a partial product to obtain an interval containing the class number. Unfortunately the product converges very slowly, so that many terms are required to get a short interval. The second idea is to use Lagrange's theorem

$$f \in C(-d) \Rightarrow f^{h(-d)} = 1. \tag{3.1}$$

in order to eliminate integers from the interval. The last idea that is used in Shanks' algorithm is the so-called baby steps-giant steps method, which allows us to search the interval faster.

3.2 The Basic Subexponential Algorithm

I will now outline a new subexponential algorithm for computing $h(-d)$ that may be viewed as an extension of Shanks' algorithm. The key observation is that we can replace the trivial group relation (3.1) by a set of random relations on a set of generators of the class group. Once we do this, we are able to derive significantly more information concerning the class number.

A rough outline of the most basic form of the new algorithm is as follows: (for later convenience, we leave the parameter z unspecified)

Algorithm CN1 Input $d > 4$ with $d \equiv 0$ or $3 \pmod 4$, and output $h(-d)$.

```
    Step 1 Compute a rational number $B$ with $B \le h(-d) < 2B$.
    Step 2 Set $M = \lfloor \max\{c_0 \log^2 d, L(d)^z\} \rfloor$ and $H := 0$.
    Step 3 Compute the prime forms $(p_i, b_i, \cdot)$ with $p_i \le M$, and let $N$
        be the number of such forms.
    Step 4 Generate "random relations" of the form
        $$\prod_{j=1}^{N} [(p_j, b_j, \cdot)]^{a_{ij}} = 1, \quad 1 \le i \le N. \qquad (3.2)$$
    Step 5 Set $H := \gcd(H, |\det(a_{ij})|)$. If $B \le H < 2B$, then output
        $h(-d) = H$. Otherwise return to step 4.
```

It remains to determine the probability that such an algorithm will produce an output, and to give details on how to carry out each step, particularly the mysterious step 4. Before doing so, let us first see why any output that is produced from such an algorithm will in fact be the class number $h(-d)$, at least assuming ERH.

From Theorem 2.1 it follows that the classes $[(p_i, b_i, \cdot)]$, $1 \le i \le N$ form a generating set for $C(-d)$, assuming ERH. Let $\mathcal{L}$ be the additive subgroup of $\mathbb{Z}^N$ consisting of all $(x_1, \ldots, x_N)$ with $\prod_{j=1}^{N} [(p_j, b_j, \cdot)]^{x_j} = 1$. As was pointed out in [41], we have $\mathbb{Z}^N/\mathcal{L} \cong C(-d)$, and the subgroup $\mathcal{K}$ of $\mathbb{Z}^N$ generated by the rows of the matrix $A = (a_{ij})$ is a subgroup of $\mathcal{L}$, so that $|\det(A)| = |\mathbb{Z}^N/\mathcal{K}|$ is a multiple of $h(-d)$. Hence any output of the algorithm is a multiple of $h(-d)$, but then it must be the actual class number since that is the only multiple of $h(-d)$ in the interval $[B, 2B)$.

We next explain how to carry out step 1. The idea here is to use the class number formula in essentially the same manner as in Shanks' algorithm, except that we need far fewer terms since the interval is longer than what is required in Shanks' algorithm. A result of Schoof [40, Theorem 6.3] states that assuming ERH, there exist constants c_1 and c_2 such that if $x > c_1 \log^2 d$, then

$$\left|1 - \prod_{p>x}\left(1 - \left(\frac{-d}{p}\right)p^{-1}\right)\right| < c_2 x^{-1/2}\log(dx).$$

Define

$$h^*(x) = \prod_{p\leq x}\left(1 - \left(\frac{-d}{p}\right)p^{-1}\right)^{-1}.$$

From Schoof's result we can easily derive a constant c_3 such that if $x = c_3 \log^2 d$, then

$$\frac{3}{4} < \frac{\overline{\pi}h(-d)}{\lfloor\sqrt{d}\rfloor h^*(x)} < \frac{3}{2}$$

where $\overline{\pi} = 22/7$ is an approximation to π. We then take $B = 3\lfloor\sqrt{d}\rfloor h^*(x)/(4\overline{\pi})$. Note that each of the factors in the product for $h^*(x)$ can be computed in polynomial time using quadratic reciprocity, and that the number of bits in the numerator and denominator of the rational number $h^*(x)$ is $O(\sum_{p\leq x}\log p)$, or $O(x)$ by the prime number theorem. Thus B can be computed in polynomial time.

Step 3 requires us to find the primes $p \leq M$ for which $\left(\frac{-d}{p}\right) = 1$, and for each of these primes to find b with $b^2 \equiv -d \pmod{p}$. It follows from the Čebotarev density theorem that the number N of such primes satisfies $N = L(d)^{z+o(1)}$. This computation can be done by generating the primes by a sieve, evaluating the Jacobi symbol via quadratic reciprocity, and for each of the acceptable primes solving the quadratic congruence by Berlekamp's algorithm. All of these computations can be done in $O(M^{1+\epsilon})$ bit operations.

Step 4 is particularly vague; what do we mean by a random relation, and how do we generate such things ? We begin by stating the following result of Schnorr [38]:

Theorem 3.1. *If $[(a,b,c)] \in C(-d)$, then for every prime p with $p|a$ we have $\left(\frac{-d}{p}\right) = 1$. Furthermore, if $a = \prod_{i=1}^{m} p_i^{e_i}$, then $b \equiv \epsilon_i b_i \pmod{2p_i}$*

and

$$[(a,b,c)] = \prod_{i=1}^{m} [(p_i, b_i, \cdot)]^{\epsilon_i e_i},$$

where $\epsilon_i = \pm 1$ *and* b_i *is defined by (2.1).*

It follows from this result that a form $[(a,b,\cdot)]$ can be written as a product of small prime forms if the lead coefficient a can be factored as a product of the corresponding small primes. Using this fact, we can produce relations of the form (3.2) by the following simple procedure.

For $i = 1, \dots, N$, do:

> **Step 1** Choose x_j, $1 \leq j \leq N$ randomly and uniformly from $\{1, \dots, d\}$.
>
> **Step 2** Compute a reduced form (a,b,c) with
>
> $$[(a,b,c)] = \prod_{j=1}^{N} [(p_j, b_j, \cdot)]^{x_j}. \tag{3.3}$$
>
> **Step 3** If a admits a factorization $a = \prod_{j=1}^{N} p_j^{e_j}$, then set $a_{ij} = x_j - \epsilon_j e_j, 1 \leq j \leq N$. If not, then return to step 1.

The most straightforward way to test the lead coefficients a is to use trial division by the primes $p_1, \dots, p_N$, but a faster method is to use the rigorous version of the elliptic curve factorization method as described by Pomerance [36]. Using this technique, we essentially attempt to factor the lead coefficients over a slightly smaller set of "good primes" using the elliptic curve method of H. W. Lenstra, Jr. In the notation of [36], let $S(y)$ denote the set of primes $3 < p \leq y$ for which there are at least $\sqrt{p}\exp(-\frac{1}{6}(\log y)^{1/7}\log\log y)$ numbers in the interval $(p - \sqrt{p}, p + \sqrt{p})$ that have all of their prime factors less than $\exp((\log p)^{6/7})$. Pomerance proved that there is a random algorithm with the following properties. It receives as input an integer a and a parameter $M \geq 2$, and produces as output integers F and R with $a = FR$ and the complete prime factorization of F. Moreover, with probability at least $1 - (\log a)/a$, no prime in $S(M)$ divides R. The running time of Pomerance's algorithm is $O\left((\log a)^4 \exp\left(2(\log M)^{6/7}\right)\right)$.

Pomerance's rigorous elliptic curve algorithm provides an efficient procedure for testing the lead coefficients a. We first use trial division to find all prime factors of a less than $\exp(64(\log\log d)^6)$. If this does not completely factor a, then we apply Pomerance's rigorous elliptic curve algorithm with $M = L(d)^z$ to the residual. By the manner in which the forms (a, b, c) are generated in (3.3), it can be shown that the forms that are generated are essentially uniformly distributed among the different equivalence classes (see [41, Proposition 4.3]). Furthermore, a slight modification of the argument leading to Proposition 4.4 of [41] shows that the probability is at least $L(d)^{-1/(4z)+o(1)}$ that the procedure just described will yield a factorization of a and a relation of the form (3.2) . Since the calculation in (3.3) takes $O(N \log^3 d)$ bit operations, we expect to generate a relation with $L(d)^{z+1/(4z)+o(1)}$ operations, and to generate N such relations takes expected time $L(d)^{2z+1/(4z)+o(1)}$. By choosing $z = 1/\sqrt{8}$, we obtain a relation matrix in $L(d)^{\sqrt{2}+o(1)}$ operations.

We now discuss the work required to carry out step 5 of algorithm CN1. An upper bound on the size of $\det(A)$ is provided by Hadamard's inequality, using the fact that the entries are bounded in size by $2d$:

$$|\det(A)| \leq \left(\sqrt{4Nd^2}\right)^N = \exp(L(d)^{z+o(1)}).$$

Using Gaussian elimination and the Chinese remainder theorem as described in [41, section 3], we can compute $\det(A)$ in $L(d)^{4z+o(1)}$ operations. The gcd$'s$ can be computed via the Euclidean algorithm in $L(d)^{2z+o(1)}$ operations [21, p. 338].

Up until this time all of the arguments can be made completely rigorous under the assumption of the extended Riemann hypothesis. The number of operations used for steps 1–4 is $L(d)^{\sqrt{2}+o(1)}$. The only issue remaining to be addressed is the probability that an output will actually be produced, and it is here that I will resort to a heuristic argument. The argument is based on the principle that "two randomly chosen integers are relatively prime with probability $6/\pi^2$". A more precise and more general mathematical basis for this heuristic is the statement that

$$\lim_{x\to\infty} \frac{\#\{(m_1,\ldots,m_t) : 1 \le m_i \le x,\ \gcd(m_1,\ldots,m_t) = 1\}}{x^t} = \frac{1}{\varsigma(t)}, \tag{3.4}$$

where $\varsigma(t)$ is the Riemann zeta function. Imagine now a situation in which we are attempting to compute some unknown integer k, but all that we are able to do is produce random multiples of k from the interval $[0, x]$. We can then devise a probabilistic algorithm to compute k by simply computing several such random multiples $km_1, \ldots, km_t$, and computing their greatest common divisor. If x is large, then with probability close to $\frac{1}{\varsigma(t)}$, we will have $\gcd(km_1, \ldots, km_t) = k$. Since $1/\varsigma(t) \approx 1 - 2^{-t}$ as t tends to infinity, the probability that k will be computed by the process tends very rapidly to 1.

In the case of the class number algorithm, the random multiples that are generated are not chosen exactly from a uniform distribution since they are determinants. As before, let $\mathcal{L}$ denote the lattice of all relations on the generators $[(p_i, b_i, \cdot)]$, $1 \le i \le N$, and let L be a matrix whose rows are a basis for the lattice $\mathcal{L}$. It follows from the theory of lattices that there exists a matrix B such that $A = BL$, where A is the matrix of relations generated in the algorithm. Hence $\det(A) = \det(B)\det(L) = \det(B)h(-d)$. What is required is to prove that the matrices B have the property that they have a nonneglible probability of having relatively prime determinants. It can be proved that each relation vector x in the box $\|x\|_\infty \le d$ is nearly equally likely to occur as a row of the matrix A, so that a representative row b of B is essentially chosen uniformly from a parallelepiped of the form $\|bL\|_\infty \le d$. While I am not able to rigorously prove that matrices chosen in such a way are likely to have relatively prime determinants, it seems very likely to be true. In fact, there is a heuristic argument suggesting that two such determinants are relatively prime with probability

$$\prod_p \left(1 - \left(1 - \prod_{i=1}^{N}(1 - p^{-i})\right)^2\right).$$

If the probability can only be shown to be bounded away from zero, then a bounded number of passes through step 4 will produce the correct output.

3.3 A Faster Version

In this section we shall describe the changes that are necessary to achieve a heuristic running time estimate of $L(d)^{3\sqrt{2}/4+o(1)}$ (Note that $3\sqrt{2}/4 \approx 1.06$). The major idea is to use a sparse relation matrix. Again leaving the parameter z to be chosen later, an outline of the algorithm is as follows.

Algorithm CN2 Input $d > 4$ with $d \equiv 0$ or $3 \pmod 4$, and output $h(-d)$.

Step 1 Compute an integer B with $B \leq h(-d) < 2B$.

Step 2 Set $H := 0$ and $M = \lfloor \max\{L(d)^z, c_0 \log^2 d\} \rfloor$.

Step 3 Compute the prime forms $(p_i, b_i, \cdot)$ with $p_i \leq M$. Let m be the number with $p_i \leq c_0 \log^2 d$, and let N be the number with $p_i \leq M$.

Step 4 Set i:=0.

Step 5 Choose x_j, $1 \leq j \leq m$ randomly and uniformly from $\{1, \ldots, d\}$.

Step 6 Compute a reduced form (a, b, c) with

$$[(a,b,c)] \sim \prod_{j=1}^{m} [(p_j, b_j, \cdot)]^{x_j}.$$

Step 7 Apply trial division by the primes $p_i \leq \exp(64(\log\log d)^6)$, followed by the rigorous elliptic curve method to determine if a admits a factorization

$$a = \prod_{j=1}^{N} p_j^{e_j}.$$

If not, then return to step 5. If a does admit such a factorization, then set $i := i + 1$ and

$$a_{ij} = \begin{cases} x_j - \epsilon_j e_j, & 1 \leq j \leq m \\ -\epsilon_j e_j, & m < j \leq N \text{ and } e_j \neq 0 \\ 0, & \text{otherwise.} \end{cases}$$

Step 8 If $i < N$, then return to step 5. Otherwise set $H := \gcd(H, |\det(a_{ij})|)$. If $B \leq H < 2B$, then output $h(-d) = H$. Otherwise return to step 4.

The number of operations required by each of steps 1–7 is $L(d)^{o(1)}$, and the probability that a will be completely factored in step 7 is $L(d)^{-1/(4z)+o(1)}$. Hence the generation of the relation matrix can be expected to take $L(d)^{z+1/(4z)+o(1)}$ bit operations. Note that the relation matrix (a_{ij}) has $O(\log^2 d)$ nonzero entries in each row, so at most $L(d)^{z+o(1)}$ nonzero entries in all.

The evaluation of $\det(A)$ can be speeded up if we use the Chinese remainder theorem and Wiedemann's coordinate recurrence method [45]. We first choose $L(d)^{z+0(1)}$ primes of size $O(d)$ in such a way that their product exceeds $\left(\sqrt{4Nd^2}\right)^N$ (our upper bound for $|\det(A)|$). This can be done probabilistically in $L(d)^{z+o(1)}$ operations by choosing random integers from the interval $[d, 2d]$ and applying the Adleman-Pomerance-Rumely primality test [2]. For each of these primes p, we then use Wiedemann's algorithm to compute $|\det(A)| \pmod p$. For each prime this can be done in $L(d)^{2z+o(1)}$ operations. We then use the Chinese remainder theorem to compute $|\det(A)|$ modulo the product of the primes, and in so doing we get $|\det(A)|$ exactly. The work for the Chinese remainder theorem can be done in $L(d)^{z+0(1)}$ bit operations (see [3, p. 311], so the determinant can be done in $L(d)^{3z+o(1)}$ operations. The $\gcd's$ can be computed in $L(d)^{2z+o(1)}$ operations.

Putting all of these estimates together, we obtain a running time of $L(d)^{\max\{z+1/(4z),3z\}+o(1)}$ operations. By choosing $z = 1/\sqrt{8}$, we obtain a heuristic running time estimate of $L(d)^{3\sqrt{2}/4+o(1)}$ bit operations. Again, with the exception of the argument concerning the determinants being relatively prime, we can rigorously prove all of the estimates assuming ERH.

The bottleneck in the previous algorithm is in computing the huge determinants, but this part might be avoided altogether. One approach to this is provided in [27], where they generate an oversquare matrix with the hope that the extra relations will be enough to generate the entire lattice. From this set of generators, one can produce a basis for the lattice using an algorithm as in [29] of [20], and compute $h(-d)$ from this much smaller determinant.

3.4 Practical Versions ?

I make no claim that the subexponential algorithms I have described here are presented in a manner suitable for implementation. This is analogous to the situation for factoring algorithms, where the algorithm that seems to be most practical for factoring integers with two large prime factors is the quadratic sieve algorithm, but the running time of this algorithm is asymptotically bigger than that of the best known algorithms. In any actual implementation of an algorithm for computing $h(-d)$ based on ideas from this paper, a great deal of attention should be payed to minimizing the number of operations such as composition and reduction, even though in the theoretical versions presented here they contribute essentially nothing to the running time. Probably the most significant improvement in a practical version is to use the idea of generating a slightly oversquare matrix, perhaps calculating determinants of submatrices. This allows us to gain as much information as possible from the relations that are generated. Finally, anyone who intends to build an implementation should probably take the following advice.

> Skillful mathematicians know how to reduce tedious calculations by a variety of devices, and here experience is a better teacher than precept.
>
> — K. F. Gauss (1801), *Disquisitiones Arithmeticæ, Art. 73*

4 Computation of $h(-d)$ in NP

The purpose of this section is to prove under the assumption of the extended Riemann hypothesis that the problem of computing $h(-d)$ belongs to the complexity class NP. One interpretation of this statement is that there exist short proofs of correctness for the value of $h(-d)$. A complete proof that the problem of computing $h(-d)$ would use the terminology of language recognition problems, but I shall instead use an informal description of the argument. A formal definition of the complexity class NP can be found in [3]. Informally we may think of the class NP as the set of languages for which there exists a succinct certificate of membership, or as the set of problems that are solvable in polynomial time by a nondeterministic Turing machine. We can think of such a machine as making a polynomial length sequence of binary guesses for the certificate, and then checking the validity of the certificate. The trick to showing that a problem belongs to NP is to come up with a polynomial length certificate.

The basic idea of the algorithm is to first compute a small generating set for $C(-d)$ as in Theorem 2.1, and a rational number B such that $B < h(-d) < 2B$, using the class number formula just as in Section 3.2. Assuming ERH, we have already seen that this can be done in polynomial time. The algorithm then guesses a basis for the lattice of relations on the set of generators, computes the determinant of the basis matrix A, and verifies that $B < |\det(A)| < 2B$. According to the theory of finitely generated abelian groups (see [17] for the theory), such a basis must exist, and the determinant is equal to $h(-d)$. The matrix A has a number of entries that is polynomial in $\log d$, and the determinant can be computed in a polynomial of steps. All of these things together then constitute the certificate of validity for $h(-d)$.

In fact we might as well assume that the algorithm gives a *triangular* basis for the lattice of relations, since the Hermite reduction algorithm as described in [20] gives a polynomial time algorithm to produce such a basis from an arbitrary basis. Given a triangular basis, we can recover the invariant factors of $C(-d)$ from the diagonal entries of the triangular basis, and in this way produce the structure of the group in addition to its cardinality.

5 Algorithms for Discrete Logarithms in Class Groups

At the time that Buchmann and Williams proposed their scheme, the best known algorithm for computing discrete logarithms in the class group was the algorithm that works over an arbitrary finite group, and has complexity $O(d^{1/4+\epsilon})$. In this section I shall explain how the methods of the previous section can be adapted to give a probabilistic algorithm with expected running time $L(d)^{1+o(1)}$, assuming that $h(-d)$ is known.

Before we discuss the algorithm there are a few other matters to clear up. First of all, as we mentioned previously, any algorithm that will solve $g^x = f$ for random choices of x can be used to compute the order of g. Hence we may as well assume that $h(-d)$ is known, particularly since we can calculate it by applying the algorithm of Section 3.2 (although the running time for that algorithm is longer).

The second point is that the class group $C(-d)$ is not always cyclic, and even if it was, the base for our discrete logarithms need not be a generator of the entire group. For the purpose of the discussion here we shall ignore this problem, and assume that the class group is cyclic and generated by the element g. In fact, heuristic evidence of Cohen and Lenstra [9] suggests that the odd part of the class group is cyclic about 97.75% of the time anyway, and the cyclic case is probably the most interesting for the cryptographic application.

The discrete logarithm algorithm that we shall describe uses what is known as the index calculus method. This technique has a precomputation phase in which the discrete logarithms of the elements of a small set of generators are computed. In the case of $G = \mathbb{Z}/p\mathbb{Z}$, this small set of generators consists of an initial segment of small primes. As a historical note, it is perhaps interesting to record that a similar technique was already known to Kraitchik [24] in 1924, although he seemed to view it more as a way to devise compact tables of discrete logarithms. In the case of a class group, we use the small prime forms for the set of generators. It should be noted that the idea of using the index calculus method in class groups was also arrived at independently by A. K. Lenstra and H. W. Lenstra, Jr. [27], and by C.-P. Schnorr [39].

We shall begin by describing a simplified version of the algorithm. Let $M = \lfloor L(d)^z \rfloor$, let $p_1, \ldots, p_N$ be the primes for which $p_i \leq M$ and $\left(\frac{-d}{p_i}\right) = 1$, and let $f_j = [(p_j, b_j, \cdot)]$, $1 \leq j \leq N$. The index calculus method has three stages. In the first stage, we gather equations of the form

$$\prod_{j=1}^{N} f_j^{a_{ij}} = g^{x_i}, \quad 1 \leq i \leq N. \tag{5.1}$$

In stage two, we interpret these equations as a system of linear equations of the form

$$\sum_{j=1}^{N} a_{ij} \log_g f_j \equiv x_i \pmod{h(-d)}, \quad 1 \le i \le N, \tag{5.2}$$

and solve this system for the unknown values $\log_g f_j$. Finally in stage three we use the discrete logarithms of the small prime forms to solve for $\log_g f$. Subsequent calculations of $\log_g y$ for other classes y require only the third stage to be carried out.

Stage one can be carried out in much the same way that relations were generated in the class number algorithms. We simply choose integers x_i randomly and uniformly from the interval $[1, h(-d)]$, compute a reduced form (a, b, c) in the class g^{x_i}, and attempt to write it in terms of the f_j by factoring the leading coefficient a. The factorization of a can be attempted with the rigorous elliptic curve method, using $L(d)^{o(1)}$ operations. If g generates all of $C(-d)$, then this procedure will succeed with probability $L(d)^{-1/(4z)+o(1)}$, so the generation of N such equations can be done in expected time $L(d)^{z+1/(4z)+o(1)}$.

Let us assume that the matrix A has full rank, so that we can solve this system of linear equations for the unknown values $\log_g f_i$. The solution can be computed using Gaussian elimination or the coordinate recurrence method of Wiedemann. Using Gaussian elimination takes $L(d)^{3z+0(1)}$ operations in the ring $\mathbb{Z}/h(-d)\mathbb{Z}$, so also $L(d)^{3z+0(1)}$ bit operations. If we instead use Wiedemann's method, then it takes only $L(d)^{2z+o(1)}$ operations since the matrix A has only $L(d)^{z+o(1)}$ nonzero entries. Here we should take note of the fact that Wiedemann's method is intended to work over a finite field, whereas we are really working over the ring $\mathbb{Z}/h(-d)\mathbb{Z}$, which is not a field. Pomerance [36] has pointed out two ways of getting around this. In the first method, we first factor $h(-d)$, using for instance the algorithm REC of [36], which takes $L(h(-d))^{\sqrt{2}+o(1)}$ operations. Since it is known that $h(-d) \ll \sqrt{d} \log d$, this can be done in $L(d)^{1+o(1)}$ operations. Once $h(-d)$ is factored, we can use Wiedemann's method to solve the system of equations modulo the prime factors of $h(-d)$, and use the Chinese remainder theorem and Hensel's lifting technique to construct the solution modulo $h(-d)$. Hence if A has full rank, then we can carry out stages one and two in $L(d)^{\max\{2z, z+1/(4z)\}+o(1)}$ operations. If we choose $z = 1/2$, then we get a running time for stages one and two of $L(d)^{1+o(1)}$, and a storage requirement of $L(d)^{1+o(1)}$ as well.

It may be that the matrix A that is generated in stage one does not have full rank. In practice this should happen with low probability, but it appears difficult to prove this directly. In order to obtain a rigorous proof for this, we modify stage one in the manner used by Pomerance [36]. Let $\ell = \lfloor 2\log_2 N \rfloor + 3$ (that's base 2 logarithm!). Proceeding as in (5.1), we choose random values of $x \in [1, h(-d)]$ and generate $N\ell$ equations of the form

$$\prod_{j=1}^{N} f_j^{e_j} = g^x. \tag{5.3}$$

Then for $k = 1, \ldots, N$ we generate ℓ equations of the form

$$\prod_{j=1}^{N} f_j^{e_j} = f_k g^x. \tag{5.4}$$

All together this gives us $2N\ell$ linear equations, and it follows from a result of Pomerance [36, Lemma 4.1] that this system will have full rank with probability at least $1 - 1/(2N)$.

We now describe stage three of the algorithm in which we calculate $\log_g f$. If the leading coefficient of f can be factored as a product of the primes less than M, then this gives $\log_g f$ as a linear combination of $\log_g f_i$, $1 \le i \le N$, since

$$f = \prod_{j=1}^{N} f_j^{e_j} \Rightarrow \log_g f = \sum_{j=1}^{N} e_j \log_g f_j.$$

If the leading coefficient of f does not admit such a factorization (or if we are unable to calculate it), then we simply choose random value $x \in [1, h(-d)]$, and compute a reduced form in the class of $g^x f$. The probability is at least $L(d)^{-1/(4z)+o(1)}$ that we can find a factorization of the leading coefficient of this form, and we can compute $\log_g f$ from this. Hence the expected running time of stage three is $L(d)^{1/2+o(1)}$.

6 Epilogue

In case the reader doubts the practicality of these applications, they should be aware that at Eurocrypt '88 (an annual conference on cryptography), there were four private companies displaying cryptographic equipment using some variation of the Diffie and Hellman key distribution scheme, and a company called CYLINK of Sunnyvale, California has recently produced a special purpose chip that is capable of performing a 1024-bit modular exponentiation in 640 milliseconds. Diffie and Hellman schemes are apparently very practical, and are becoming widely accepted.

Number theory is one field of mathematics that up until recently has seemed most remote from applications, as attested to by the famous number theorist G. H. Hardy [16]

> The Theory of Numbers has always been regarded as one of the most obviously useless branches of Pure Mathematics.

A similar view was expressed by Paul R. Halmos [14, p. 18]:

> Whether contact with applications can prevent or cure the disease of elaboration and attenuation in mathematics is not really known; what is known is that many of the vigorous and definitely non-cancerous parts have no such contact (and probably, because of their level of abstraction, cannot have any). Current examples: analytic number theory and algebraic geometry.

It is amusing to note that both analytic number theory and algebraic geometry have figured prominently in the recent cryptographic literature.

There is a tendency among many "pure" mathematicians to regard any contact of their mathematics with applications as degrading to its quality as mathematics. This view was not shared by Hardy, who later in life wrote the following [16, p. 120].

> If the theory of numbers could be employed for any practical and obviously honorable purpose ... then surely neither Gauss nor any other mathematician would have been so foolish as to decry or regret such applications.

In my view, the recent discovery that very old and elegant parts of number theory have important and practical applications should not be viewed with alarm by mathematicians. Many areas of mathematics have derived benefit from stimulation provided by "the outside world," and there is no reason to think that number theory should be any different. The field of cryptography provides a powerful motivation to investigate computational questions in number theory and algebraic geometry, with the goal of proving either that the systems discussed here are secure because the problems are truly difficult, or the systems are insecure because of some algorithms yet to be discovered. The interplay between the pure (number theory) and the applied (cryptography) is invigorating to both, since it has the potential to stimulate much fruitful research in the future, and it also has the potential to dramatically improve the methods used to protect the integrity of information.

References

[1] L. M. Adleman and Kevin S. McCurley, "Open Problems in Number Theoretic Complexity," pp. 237–262 in *Discrete Algorithms and Complexity; Proceedings of the Japan–U.S. Joint Seminar*, Academic Press, Orlando, Florida, 1987.

[2] L. M. Adleman, C. Pomerance, and R. Rumely, "On Distinguishing Prime Numbers from Composite Numbers," *Annals of Mathematics* **117** (1983), 173–206.

[3] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[4] Eric Bach, *Discrete Logarithms and Factoring*, Technical Report UCB/CSD 84/186, Computer Science Division (EECS), University of California, Berkeley, California, June, 1984.

[5] Eric Bach, "What to do Until the Witness Comes: Explicit Bounds for Primality Testing and Related Problems," preprint, March 4, 1988.

[6] Eric Bach, Gary Miller, and Jeffrey Shallit, "Sums of divisors, perfect numbers, and factoring," *Proc. 16th Annual ACM Symposium on Theory of Computing* (1984), Association for Computing Machinery, New York, 1984.

[7] Bert den Boer, "Diffie Hellman equivalent to discrete log for certain primes," preprint, 1988. (presented at rump session of Crypto 88).

[8] Johannes Buchmann and H. C. Williams, "A Secure Key Exchange System Based on Imaginary Quadratic Fields," preprint, 1988.

[9] H. Cohen and H. W. Lenstra, Jr., "Heuristics on Class Groups of Number Fields," *Number Theory* (Noordwijkerhout, 1983), *Lecture Notes in Mathematics*, vol. 1068, pp. 33–62, Springer-Verlag, New York, 1984.

[10] D. Coppersmith, "Fast evaluation of discrete logarithms in fields of characteristic 2," *IEEE Transactions on Information Theory* **30** (1984), 587–594.

[11] D. Coppersmith, A. Odlyzko, and R. Schroeppel, "Discrete logarithms in GF(p)," *Algorithmica* **1** (1986), 1-15.

[12] W. Diffie and M. Hellman, "New Directions in Cryptography," *IEEE Transactions on Information Theory* **22** (1976), 472-492.

[13] K. F. Gauss, *Disquisitiones Arithmeticæ*, Leipzig, Fleischer, 1801. Translation into English by Arthur A. Clarke, S.J. reprinted by Springer-Verlag, New York, 1985.

[14] Paul R. Halmos, "Applied Mathematics is Bad Mathematics," pp. 9–20 in *Mathematics Tomorrow*, L. A. Steen, (ed.), Springer-Verlag, New York, 1981.

[15] G. H. Hardy, *A Mathematician's Apology*, Cambridge University Press, London, 1940.

[16] G. H. Hardy, "Prime Numbers," British Association Report (1915), 350–354.

[17] B. Hartley and T. O. Hawkes, *Rings, Modules and Linear Algebra*, Chapman and Hall, London, 1970.

[18] Hua Loo Keng, *Introduction to Number Theory*, translation into English by Peter Shiu, Springer-Verlag, New York, 1982.

[19] C. G. J. Jacobi, *Canon Arithmeticus*, Typis Academicis, Berlin, 1839.

[20] Ravindran Kannan and Achim Bachem, "Polynomial Algorithms for Computing the Smith and Hermite Normal Forms of an Integer Matrix," *SIAM Journal of Computing* 8 (1979), 499–507.

[21] D. E. Knuth, *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*, second edition, Addison-Wesley, Reading, MA, 1981.

[22] Neal Koblitz, "Elliptic curve cryptosystems," *Mathematics of Computation* 48 (1987), 203–209.

[23] Neal Koblitz, "A Family of Jacobians Suitable for Discrete Log Cryptosystems," to appear in *Proceedings of Crypto '88, Lecture Notes in Computer Science*, Springer-Verlag.

[24] M. Kraitchik, *Recherches sur la Théorie des Nombres* Paris, 1924.

[25] D. H. Lehmer, *Guide to Tables in the Theory of Numbers*, National Academy of Science, Washington, 1941.

[26] A. K. Lenstra, "Fast and rigorous factorization under the generalized Riemann hypothesis," preprint, 1987.

[27] A. K. Lenstra and H. W. Lenstra, Jr., "Algorithms in number theory," Technical Report 87-008, Department of Computer Science, University of Chicago, May, 1987.

[28] H. W. Lenstra, Jr. , "On the calculation of regulators and class numbers of quadratic fields," *Journées Arithmétiques 1980* (J. V. Armitage, ed.), Cambridge University Press, New York, 1982.

[29] L. Lovász, "Some Algorithmic Problems on Lattices," *Theory of Algorithms, Colloquia Mathematica Societatis János Bolyai* 44 (1984), 323–337.

[30] Kevin S. McCurley, A Key Distribution Scheme Based on Factoring, to appear in *J. Cryptology.*

[31] V. Miller, "Use of Elliptic curves in cryptography," *Advances in Cryptology* (Proceedings of Crypto '85), *Lecture Notes in Computer Science* **218** (1986), Springer-Verlag, NY, 417–426.

[32] A. M. Odlyzko, "Discrete logarithms in finite fields and their cryptographic significance," *Advances in Cryptology* (Proceedings of Eurocrypt 84), *Lecture Notes in Computer Science* **209**, Springer-Verlag, NY, 224–314.

[33] R. W. K. Odoni, V. Varadharajan, and P. W. Sanders, "Public Key Distribution in Matrix Rings," *Electronics Letters* **20** (1984), 386– 387.

[34] S. Pohlig and M. Hellman, "An improved algorithm for computing discrete logarithms over GF(p) and its cryptographic significance," *IEEE Transactions on Information Theory* **24** (1978), 106–110.

[35] J. M. Pollard, "Monte Carlo methods for index computation (mod p)," *Mathematics of Computation* **32** (1978), 918–924.

[36] Carl Pomerance, "Fast, rigorous factorization and discrete logarithm algorithms," *Discrete Algorithms and Complexity; Proceedings of the Japan-US Joint Seminar, June 4, 1986, Kyoto, Japan*, Academic Press, Orlando, 1987, 119–143.

[37] R. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public key cryptosystems", *Communications of the ACM* **21** (1978), 120–126.

[38] C.-P. Schnorr, "Refined analysis and improvements of some factoring algorithms," *Journal of Algorithms* **2** (1982), 101–127.

[39] C.-P. Schnorr, personal communication.

[40] R. Schoof, "Quadratic fields and factorization," *Computational Methods in Number Theory* (R. Tijdeman and H. W. Lenstra, Jr, eds.), Mathematisch Centrum Tract 154, Amsterdam, 1982, 235–286.

[41] Martin Seysen, "A probabilistic factorization algorithm with quadratic forms of negative discriminant," *Mathematics of Computation* **48** (1987), 757–780.

[42] Daniel Shanks, "Class number, a theory of factorization, and genera, " *Proceedings Symposium Pure Mathematics*, American Mathematical Society, 1972.

[43] Daniel Shanks, "Five Number-theoretic Algorithms," *Proc. Second Manitoba Conference on Numerical Mathematics* (1972), 51–70.

[44] Z. Shmuely, *Composite Diffie-Hellman public-key generating systems are hard to break*, Technical report # 356, Computer Science Department, Technion-Israel Institute of Technology, February, 1985.

[45] Douglas H. Wiedemann, "Solving Sparse Linear Equations Over Finite Fields," *IEEE Transactions on Information Theory* **32** (1986), 54–62.