Integer Programming and Cryptography

H. W. Lenstra, Jr.

Not long ago it was reported in the press that Adi Shamir, from the Weizmann Institute of Science in Israel, had broken one of the first public key cryptosystems, the Merkle–Hellman knapsack system *Scientific American* (August 1982, p 79) reported inaccurately that

He did so by proving that a mathematical problem called the knapsack problem, which had been considered exceedingly difficult, can be solved rapidly by a simple computer algorithm

Scientific American summarized the purported method of solution as follows

Shamir was able to solve the knapsack problem after he noted that it could be transformed into a mathematically equivalent problem in integer programming Shamir showed that the integer-programming problem to which the knapsack problem is equivalent can be solved by an algorithm recently invented by Hendrik W Lenstra of the University of Amsterdam Hence Lenstra s algorithm can also solve the knapsack problem

It is the purpose of this article to explain exactly what has and has not been proved and to outline the methods that were used Shamir did not find a way to solve the general knapsack problem, which appears to be computationally intractable What he did do was to find a way to use a recent integer programming algorithm to solve a special type of knapsack problem that occurs in cryptography

Useful background information about cryptology and linear programming can be found in the *Mathematical Intelligencer* articles by Simmons [1] and Lovász [2], respectively

Integer Programming

For our purposes, the *integer programming problem* is most conveniently formulated as follows Let n and m be positive integers and let real n-vectors a_i and real numbers b_i be given, for $i = 1, 2, \ldots, m$ The problem is to decide whether or not there exists an n-vector x with *integral* coordinates x_j satisfying the inequalities

$$a_i x \leq b_i \quad \text{for } i = 1, 2, \ldots, m \qquad (1)$$

We shall assume throughout that the b_i and the coordinates of the a_i are *integers* This is not a substantial restriction if they are rational, and allowing more general real numbers leads to the question of how to specify these exactly, which I do not want to discuss

Notice that we formulated the problem as a *decision problem*, which has the answer "yes" or "no " There exist other versions of the integer programming problem For example, if the answer is "yes," we may ask for an actual integer vector x satisfying (1) to be exhibited, or we may ask for such an x maximizing cx, where c is a given n-vector with integer coordinates But all these versions are equivalent in the sense that an efficient method for solving one of them easily leads to an efficient method for solving the others

Efficient Algorithms

We are interested in an algorithm for solving the integer programming problem that not only gives the correct answer but also does so within a reasonable time Here "reasonable" can be exactly defined The time should be bounded by a *polynomial function* of the *length* l of the problem This length should be thought of as the time it takes to write down the $(n + 1)m$ coordinates of the n-vectors a_i and the m-vector b Let A denote the maximum absolute value of these coordinates Then each coordinate has at most a constant times $\log(A + 2)$ binary digits, so for our purposes we can take $l = (n + 1)m \cdot \log(A + 2)$

If the integrality constraint on the coordinates of the solution vector x is dropped, then such a *polynomial algorithm* indeed exists For a discussion of this algorithm, discovered by L G Khachiyan, we refer the reader to the article by Lovasz [2]

For the integer programming problem, no polynomial algorithm is likely to exist, since the problem is *NP-complete* This means, roughly speaking, that it is at least as difficult as many other problems that are notorious for their computational intractability, such as the traveling salesman problem and the problem of decomposing an integer into prime factors—see [3] for a fuller discussion

The new result on integer programming that *Scientific American* refers to is the following For every *fixed* value of n, the number of variables, there *does* exist a polynomial algorithm for solving the integer program-

ming problem—see [4]. This does not contradict the previous paragraph: A running time $2^n \ell$, for example, would be polynomially bounded for fixed n, but not in general. (My algorithm is actually much slower.)

We shall now first describe the basic ideas behind this new algorithm and discuss its cryptographic significance later.

Integral Points in a Triangle

It is trivial to design an algorithm for the integer programming problem with *one* variable: It suffices to perform a series of divisions and comparisons. These can be done in polynomial time, just like the other arithmetic operations such as addition, subtraction, and multiplication.

The two-variable case is already nontrivial. Let K be the plane region described by (1):

$$K = \{x \in \mathbf{R}^2: a_i x \leqslant b_i \quad \text{for } i = 1, 2, \ldots, m\}$$

This is a convex set bounded by at most m straight line segments. We shall consider the special case that K is a *triangle*. Then the question becomes: *How does one decide, in polynomial time, whether a given triangle in the plane contains a point with integral coordinates?* It makes no difference whether the triangle is given by the equations $a_i x = b_i$ ($i = 1, 2, 3$) defining its edges, or by the rational coordinates of the three vertices, since it is easy to compute, in polynomial time, the vertices from the edges and the edges from the vertices.

The reader who thinks very briefly about the above problem will probably react: Draw the triangle and look. He will argue that either the triangle is "large," in which case it must obviously contain an integral point, or the triangle is "small," and then it is contained in a small rectangle, all integral points in which can be checked one by one.

If one tries to make this argument precise, one discovers that it works all right for the decent-looking triangles one finds in geometry textbooks, but that a problem is presented by triangles that are very *long* and very *thin*. They are too thin to obviously contain an integral point, and so long that there are more integral points *near* the triangle than can possibly be enumerated in polynomial time.

There are several ways to deal with such triangles. It can be done with the help of continued fractions, but I will avoid them in my discussion and describe a method that generalizes better to higher dimensions.

The solution essentially consists of denying that "special" triangles exist. If the triangle K looks a bit weird, why not apply a nonsingular linear transformation τ such that the triangle $\tau[K]$ looks better? To be specific, choose τ so that the latter triangle is *equilateral*.

Clearly, K contains an element of $\mathbf{Z}^2$ if and only if the new triangle $\tau[K]$ contains an element of the lattice $L = \tau[\mathbf{Z}^2]$. If e_1, e_2 are the standard basis vectors of $\mathbf{R}^2$, then $\tau(e_1)$, $\tau(e_2)$ form a basis for L in the sense that $L = \mathbf{Z}\tau(e_1) + \mathbf{Z}\tau(e_2)$.

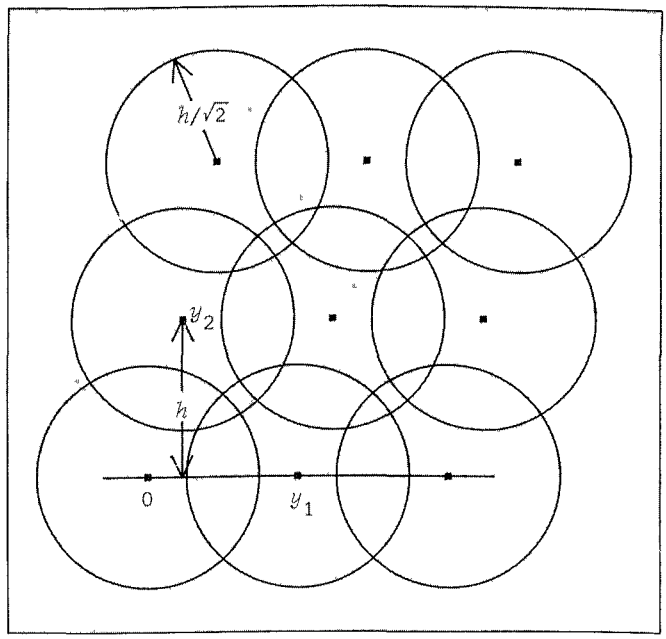

Figure 1

The problem has now been shifted from the triangle to the lattice. To describe the latter, it is notationally convenient to identify $\mathbf{R}^2$ in the usual way with the complex plane $\mathbf{C}$. It is a classical result that L, as every lattice in $\mathbf{C}$, has a basis y_1, y_2 with the property that $z = y_2/y_1$ belongs to the well-known fundamental domain for the modular group:

$$\operatorname{Im} z > 0 \qquad |\operatorname{Re} z| \leqslant 1/2 \qquad |z| \geqslant 1$$

Moreover, there exists a fairly straightforward algorithm that transforms a given basis for L into the basis y_1, y_2.

We need to know one more thing about L. Let h denote the distance of y_2 to the line $\mathbf{R}y_1$ (see Figure 1). It is an elementary exercise to prove that the *covering radius* of L is at most $h/\sqrt{2}$; i.e., closed discs of radius $h/\sqrt{2}$ centered at the points in L cover the whole complex plane:

$$\text{For every } u \in \mathbf{C} \text{ there exists } y \in L \text{ such that } |u - y| \leqslant h/\sqrt{2} \tag{2}$$

The remaining part of the solution is very much like the naive argument we started with. There are again two cases. Denote by e the edge length of the equilateral triangle $\tau[K]$. In the first case the triangle is large, i.e.,

$$e\sqrt{3}/6 \geqslant h/\sqrt{2}$$

Here $e\sqrt{3}/6$ is the radius of the inscribed circle of the triangle, so applying (2) with u equal to the center of

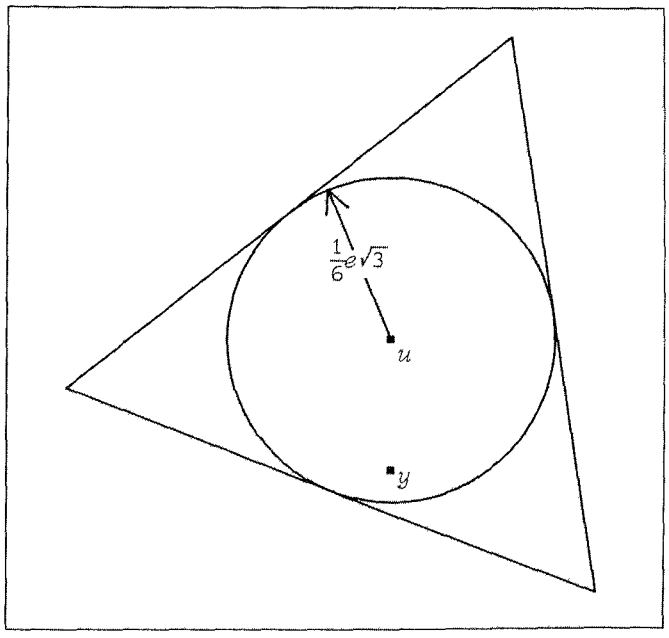

Figure 2

this circle we see that in this case there is indeed a lattice point y in the triangle (see Figure 2).

In the second case the triangle is small:

$$e\sqrt{3}/6 < h/\sqrt{2}$$

so $e/h < \sqrt{6}$. Observing that the parallel lines

$$ky_2 + \mathbf{R}y_1 \qquad k \in \mathbf{Z}$$

have successive distances h from each other (see Figure 3) one easily proves that in this case no more than $[\sqrt{6}] + 1 = 3$ of these lines intersect the triangle. Since every lattice point is on one of these lines, it now suffices to check these lines one by one, and this can be done without difficulty.

If all details in this decision procedure are made explicit, it turns out that the resulting algorithm runs indeed in polynomial time.

Higher Dimensions

The above algorithm can be extended to the general integer programming problem. We give only a brief sketch. Let again K be the closed convex set described by (1):

$$K = \{x \in \mathbf{R}^n: \ a_i x \leq b_i \qquad \text{for } i = 1, 2, \ldots, m\}$$

It can be shown that there is no loss of generality in assuming that K is *bounded* and has *positive volume.*

One begins by constructing a nonsingular linear transformation τ such that $\tau[K]$ has a "round" appearance in the sense that the ratio

$$\frac{\text{Outer radius of } \tau[K]}{\text{Inner radius of } \tau[K]}$$

is bounded above by a constant depending only on n. Here the *outer radius* of $\tau[K]$ is the radius of the smallest sphere containing $\tau[K]$, and the *inner radius* is the radius of the largest sphere contained in it. Using Khachiyan's linear programming algorithm, Lovász has shown that such a transformation τ can be found in polynomial time, even for varying n.

It is now to be decided whether $\tau[K]$ intersects the lattice $L = \tau[\mathbf{Z}^n]$. To this end one constructs a basis $y_1, y_2, \ldots, y_n$ for L that is *reduced* in the sense that

$$\frac{\text{Volume}\{\Sigma_{i=1}^n r_i y_i: \ r_i \in \mathbf{R} \qquad 0 \leq r_i \leq 1\}}{\Pi_{i=1}^n |y_i|}$$

is bounded below by a constant depending only on n. Notice that this ratio is always ≤ 1, with equality if and only if the y_i are pairwise orthogonal. Thus a reduced basis is "nearly orthogonal." There exists a polynomial algorithm for finding such a basis, even for varying n. This observation is again due to Lovász, and his *basis reduction algorithm* has several other applications, notably to the factorization of polynomials [5].

As with the triangle, there are now two cases. Let it be supposed that y_n is the longest of $y_1, y_2, \ldots, y_n$ and denote by h the distance of y_n to the hyperplane $\Sigma_{i=1}^{n-1} \mathbf{R}y_i$. In the first case, the inner radius of $\tau[K]$ is so much larger than h that the required lattice point in $\tau[K]$ exists by an analog of (2). In the other case one proves that the number of integers k for which the hyperplane

$$ky_n + \Sigma_{i=1}^{n-1} \mathbf{R}y_i$$

meets the convex set $\tau[K]$ is bounded by a constant depending only on n. Since every lattice point is on one of these hyperplanes, it suffices to investigate these values of k one by one. For a fixed value of k one obtains an integer programming problem with only $n - 1$ variables, and this problem can be solved by recursion.

This finishes the sketch of the algorithm. It can be shown that for fixed n the algorithm runs in polynomial time.

Lovász' two auxiliary algorithms mentioned above were in fact invented later. They replace, and were partly motivated by, earlier algorithms that were only polynomial for fixed n.

Applications

So far, I have not heard of an actual implementation of the algorithm just described. This seems to indicate that its practical value is rather limited. It is my understanding that the theoretical requirement that n be fixed implies the practical requirement that n be small, but that for small n older algorithms are adequate.

On the other hand, there is the application to cryptography explained below. But even here it may be

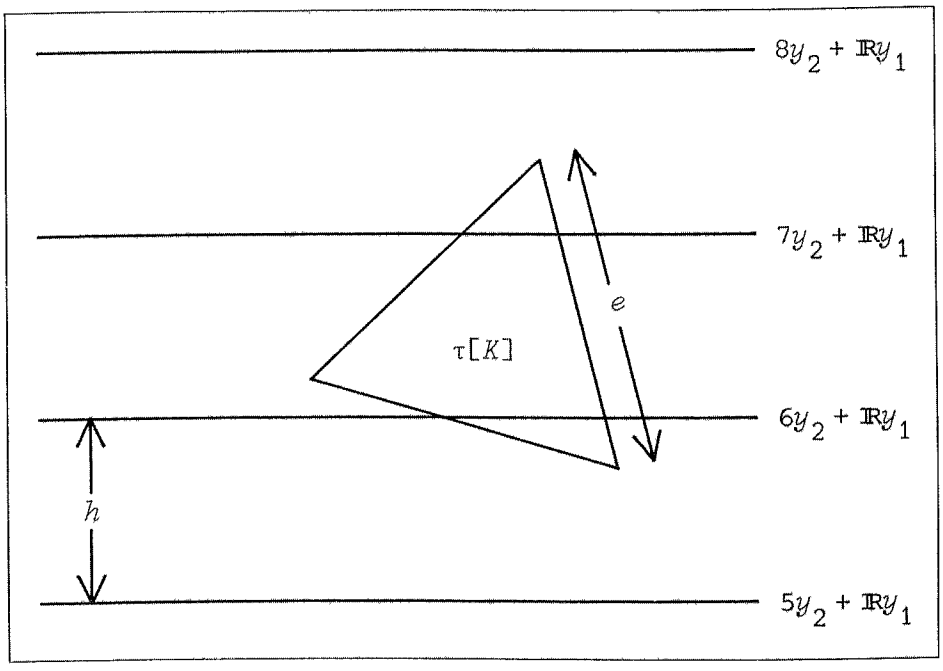

Figure 3

argued that this is an application not of the whole integer programming algorithm but of the basis reduction algorithm that was used as a subroutine.

The Knapsack Problem

The *knapsack problem* is formulated as follows. Given positive integers $a_1, a_2, \ldots, a_n, b$ it is to be decided whether there exists a subset $I \subset \{1, 2, \ldots, n\}$ such that

$$\Sigma_{i \in I} a_i = b$$

That is, given a knapsack of capacity b, and n items of sizes $a_1, a_2, \ldots, a_n$, it is to be decided whether the knapsack can be filled to capacity with a subset of these items.

If a denotes the n-vector with coordinates $a_1, a_2, \ldots, a_n$, it is clearly equivalent to ask whether there exists an n-vector x with *integral* coordinates x_j such that

$$ax = b$$
$$0 \leqslant x_j \leqslant 1 \qquad \text{for } j = 1, 2, \ldots, n$$

This is an instance of the integer programming problem, with $m = 2n + 2$. However, the new integer programming algorithm is of no use in solving the knapsack problem. It would, in fact, be faster to apply *complete enumeration*, i.e., to try the 2^n vectors $x \in \{0, 1\}^n$ one by one.

Below we shall encounter the knapsack problem in a slightly different formulation: If a set I as above exists, we also want to find it. But it is easy to see that both versions are equivalent, in the same sense as this was the case for the integer programming problem.

No polynomial algorithm for solving the knapsack problem is known, and since the knapsack problem is *NP*-complete no such algorithm is expected to exist; see [3].

Shamir has not found a way to solve the general knapsack problem. What he has solved is a special type of knapsack problem that occurs in cryptography, which we shall now describe.

Cryptographic Knapsacks

The knapsack problems that occur in cryptography are of a very special type. They have a hidden structure, knowledge of which enables one to solve them in a trivial manner. Before I describe how such knapsacks are constructed, let me briefly indicate their use in cryptography. For background, see Simmons' article [1].

Someone, to be called the *sender*, wishes to send a certain message to someone else, the *receiver*. It is supposed that the message is represented as a sequence $x = (x_j)_{j=1}^n \in \{0, 1\}^n$ of n "bits," for a suitable number n. The message is to be sent over a public channel in such a way that someone who listens in—the *eavesdropper*—is not able to reconstruct the message x.

To this end the sender proceeds as follows. He looks up the receiver's name in a public file, such as a telephone directory, and there he finds n numbers $a_1, a_2, \ldots, a_n$. Next he sends to the receiver, instead of the message x, the number b defined by

$$b = \Sigma_{j=1}^n a_j x_j$$

After reception of b, the receiver uses the hidden structure of $a_1, a_2, \ldots, a_n$ to solve the knapsack problem and to recover the original message $(x_1, x_2, \ldots, x_n)$.

The eavesdropper knows $a_1, a_2, \ldots, a_n$ from the public file, and he knows b by listening in to the public channel, but he does not know the hidden structure. Consequently, he is apparently faced with the task of solving a general knapsack problem, for which no good algorithm is known, and he will presumably be unable to reconstruct the message.

How did the receiver construct the numbers $a_1, a_2, \ldots, a_n$ that were put into the public file? Several methods to do this have been proposed by R. C. Merkle and M. E. Hellman [6], to whom the above idea is due, and it is only the simplest of these methods that has been proved insecure by A. Shamir. It is as follows.

A very easy knapsack problem to solve, not only for the receiver but also for the eavesdropper, is one in which the sequence $a_1, a_2, \ldots, a_n$ is *superincreasing*. This means that each a_i is greater than the sum of its predecessors:

$$a_i > \Sigma_{j=1}^{i-1} a_j \qquad \text{for } 1 \leqslant i \leqslant n$$

For such a knapsack problem, one must clearly have $x_n = 1$ if $b \geqslant a_n$, and $x_n = 0$ if $b < a_n$; in a similar way $x_{n-1}, x_{n-2}, \ldots, x_1$ are successively determined.

When constructing his knapsack, the receiver starts from such a superincreasing sequence $a_1', a_2', \ldots, a_n'$. To hide its obvious structure, he chooses two secret numbers u (the *multiplier*) and m (the *modulus*) satisfying

$$m > \Sigma_{j=1}^{n} a_j' \qquad gcd(u, m) = 1$$

He now defines a_j by

$$a_j \equiv ua_j' \bmod m \qquad 0 < a_j \leqslant m$$

and $a_1, a_2, \ldots, a_n$ are the numbers he makes publicly available; but $a_1', a_2', \ldots, a_n', u, m$ he keeps to himself.

To decode a received message $b = \Sigma_{j=1}^{n} a_j x_j$ the receiver proceeds as follows. Using the Euclidean algorithm he determines an integer w satisfying $wu \equiv 1 \bmod m$; this is the *inverse* multiplier. Next he calculates the number b' defined by

$$b' \equiv wb \bmod m \qquad 0 \leqslant b' < m$$

Using the fact that $a_j' \equiv wa_j \bmod m$ and the inequality $m > \Sigma_{j=1}^{n} a_j'$ one now easily proves that

$$b' = \Sigma_{j=1}^{n} a_j' x_j$$

Since the a_j' are superincreasing, the x_j can be solved from this. The eavesdropper does not know w or m, nor any of the a_j', and is therefore supposedly unable to carry out the required transformation.

Shamir's Attack

Shamir devised an algorithm for solving knapsack problems *known to have a hidden structure as described above*, but without the numbers u and m being known. His algorithm solves *most* such knapsack problems but is not guaranteed to solve them *all;* this is, however, as he writes, "not a severe handicap in the context of cryptography, since a cryptosystem becomes useless when most of its keys can be efficiently cryptanalyzed" [7].

The performance of Shamir's algorithm may be formulated as follows. Let a real number $d > 1$ be fixed, to be thought of as the ratio

$$\frac{\text{Number of bits of the encoded message}}{\text{Number of bits of the original message}}$$

which is about $(\log b)/(n \cdot \log 2)$. Let further an integer $m < 2^{dn}$ be fixed. By S we denote the set of cryptographic knapsacks with modulus m; so the elements of S correspond one to one with the sequences $(a_1', a_2', \ldots, a_n', u)$ of positive integers satisfying

$$a_1', a_2', \ldots, a_n' \text{ is superincreasing}$$
$$\Sigma_{j=1}^{n} a_j' < m$$
$$gcd(u, m) = 1 \qquad u \leqslant m$$

With this notation, it can be proved that a suitable version of Shamir's algorithm solves *almost all* problems in S, in the sense that the fraction it *cannot* solve tends rapidly to zero as n tends to infinity. Further, for fixed d the running time of Shamir's algorithm is bounded by a polynomial function of n.

Shamir claims a proof of this only for $d < 2$; the general case was proved by J. C. Lagarias (Bell Laboratories).

It is the purpose of Shamir's algorithm to calculate, given $a_1, a_2, \ldots, a_n$, new numbers w' and m' that can be used for exactly the same purpose as w and m; that is, there should exist numbers $a_1'', a_2'', \ldots, a_n''$ satisfying

$$a_j'' \equiv w'a_j \bmod m' \qquad \text{for } 1 \leqslant j \leqslant n$$
$$a_1'', a_2'', \ldots, a_n'' \text{ is a superincreasing sequence}$$
$$\Sigma_{j=1}^{n} a_j'' < m'$$

It turns out that all pairs (w', m') for which w'/m' is sufficiently close to w/m have this property. The object of Shamir's algorithm is thus to find a good enough Diophantine approximation w'/m' to w/m.

The main idea of Shamir's method and its relation to integer programming are as follows. We have $a_j' \equiv wa_j \bmod m$, so

$$a_j' = wa_j - y_j m \qquad 1 \leqslant j \leqslant n \qquad \textbf{(3)}$$

for certain integers $y_1, y_2, \ldots, y_n$. Here the cryptanalyst only knows the a_j, all the others are unknowns. But he also knows that the a_j' form a superincreasing sequence, and from this it can be deduced that for *small* j the numbers a_j' are quite small with respect to m. Dividing (3) by $a_j m$ we therefore see that, say, the numbers y_1/a_1, y_2/a_2, y_3/a_3, y_4/a_4 are close to w/m and therefore also close to each other. This leads to two-

sided inequalities for the three numbers

$$a_j y_1 - a_1 y_j \qquad j = 2, 3, 4$$

These inequalities, taken together with $0 < y_j < a_j$, give rise to a four-dimensional integer programming problems from which y_1, y_2, y_3, y_4 can be solved. At this point it must be shown that this four-dimensional integer programming problem is not likely to have many extraneous solutions for which y_1/a_1 is not close to w/m. This can be done under the assumption that $d < 2$.

Once y_1, y_2, y_3, y_4 have been found, one knows a nearly good enough approximation y_1/a_1 to w/m. Using Diophantine approximation techniques Shamir is then usually able to find the desired numbers w' and m'. This concludes my sketch of Shamir's method.

For higher values of d one must solve integer programming problems with more variables. According to J. C. Lagarias, $[d] + 2$ variables suffice for $d \geq 3$.

Current research is directed toward the problem of solving other, more complicated cryptographic knapsacks proposed by Merkle and Hellman and by others. Known attacks on these systems use special properties of cryptographic knapsacks which enable cryptanalysts to apply Diophantine approximation tools, especially Lovász' basis reduction algorithm, to solve them. None of these attacks, however, apply to general knapsack problems.

Acknowledgments. I am indebted to J. C. Lagarias and A. M. Odlyzko for commenting on earlier versions of this article, and to F. J. van der Linden for preparing the figures.

References

1. G. J. Simmons (1979) Cryptology: The mathematics of secure communication. *Math. Intelligencer 1*(4):233–246
2. L. Lovász (1980) A new linear programming algorithm—Better or worse than the simplex method? *Math. Intelligencer 2*(3):141–146
3. M. R. Garey, D. S. Johnson (1979) *Computers and Intractability: A Guide to the Theory of NP-completeness*. San Francisco: Freeman
4. H. W. Lenstra, Jr. (1983) Integer programming with a fixed number of variables. *Math. Oper. Res 8* (4) (in press)
5. A. K. Lenstra, H. W. Lenstra, Jr., L. Lovász (1982) Factoring polynomials with rational coefficients. *Math. Ann. 261*:515–534
6. R. C. Merkle, M. E. Hellman (1978) Hiding information and signatures in trap-door knapsacks. *IEEE Trans. Inf. Theory*, IT-24-5, pp. 525–530
7. A. Shamir (1982) A polynomial time algorithm for breaking the basic Merkle–Hellman cryptosystem, *Proc. 23rd IEEE Symp. Found. Computer Sci.* pp. 145–152

Mathematisch Instituut
Universiteit van Amsterdam
Roetersstraat 15
1018 WB Amsterdam
The Netherlands