Hiding Information and Signatures in Trapdoor Knapsacks

RALPH C. MERKLE, STUDENT MEMBER, IEEE AND MARTIN E. HELLMAN, SENIOR MEMBER, IEEE

***Abstract*—The knapsack problem is an NP-complete combinatorial problem that is strongly believed to be computationally difficult to solve in general. Specific instances of this problem that appear very difficult to solve unless one possesses "trapdoor information" used in the design of the problem are demonstrated. Because only the designer can easily solve problems, others can send him information hidden in the solution to the problems without fear that an eavesdropper will be able to extract the information. This approach differs from usual cryptographic systems in that a secret key is not needed. Conversely, only the designer can generate signatures for messages, but anyone can easily check their authenticity.**

I. Introduction

GIVEN A one-dimensional knapsack of length S and n rods of lengths $a_1, a_2, \cdots, a_n$, the knapsack problem is to find a subset of the rods that exactly fill the knapsack, if such a subset exists. Equivalently, find a binary n-vector $\boldsymbol{x}$ such that $S = \boldsymbol{a} * \boldsymbol{x}$, if such an $\boldsymbol{x}$ exists ($*$ applied to vectors denotes dot product, otherwise normal multiplication).

A supposed solution $\boldsymbol{x}$ is easily checked in at most n additions, but finding a solution is believed to require a number of operations that grows exponentially in n. Exhaustive trial and error search over all 2^n possible $\boldsymbol{x}$ is computationally infeasible if n is larger than 100 or 200. The best published method for solving knapsacks of the form considered here requires $2^{n/2}$ complexity both in time and memory [1]. In addition, Schroeppel [2] has devised an algorithm that takes $O(2^{n/2})$ time and $O(2^{n/4})$ space. Theory supports these beliefs because the knapsack problem is known to be an NP-complete problem,[1] and is therefore one of the most difficult computational problems of a cryptographic nature [3, pp. 363–404], [4]. Its degree of difficulty, however, is crucially dependent on the choice of $\boldsymbol{a}$. If $\boldsymbol{a} = (1,2,4,\cdots,2^{n-1})$, then solving for $\boldsymbol{x}$ is equivalent to finding the binary representation of S. Somewhat less trivially, if for all i

$$a_i > \sum_{j=1}^{i-1} a_j, \tag{1}$$

then $\boldsymbol{x}$ is also easily found. $x_n = 1$ if and only if $S \geqslant a_n$, and for $i = n-1, n-2, \cdots, 1$, $x_i = 1$ if and only if

$$S - \sum_{j=i+1}^{n} x_j * a_j \geqslant a_i. \tag{2}$$

While the theory of NP-complete problems and these examples demonstrate that the knapsack problem is only difficult from a worst case point of view, it is probably true that choosing the a_i independently and uniformly from the integers between 1 and 2^n generates a difficult problem with probability tending to one as n tends to infinity. While several efficient algorithms exist for solving the knapsack problem under special conditions [1], [5], [6], none of these special conditions is applicable to trapdoor knapsacks generated as suggested in this paper.

A trapdoor knapsack [4] is one in which careful choice of $\boldsymbol{a}$ allows the designer to easily solve for any $\boldsymbol{x}$ but prevents anyone else from finding the solution. We will describe two methods for constructing trapdoor knapsacks and indicate how they can be used to hide information. Each user I in a system generates a trapdoor knapsack vector $\boldsymbol{a}(I)$ and places it in a public file with his name and address. When someone wishes to send the information $\boldsymbol{x}$ to the Ith user, he sends $S = \boldsymbol{x} * \boldsymbol{a}(I)$. The intended recipient can recover $\boldsymbol{x}$ from S, but no one else can. Section VI shows how trapdoor knapsacks can be used to generate electronic signatures and receipts [4].

Before proceeding, a word of caution is in order. First, as is usually the case in cryptography, we cannot yet prove that the systems described in this paper are secure. For brevity, however, we will not continue to repeat this. Second, the trapdoor knapsacks described in this paper form a proper subset of all possible knapsacks, and their solutions are therefore not necessarily as difficult as for the hardest knapsacks. It is the hardest knapsacks with which NP theory is concerned.

Manuscript received August 5, 1977; revised February 28, 1978. This work was supported by the National Science Foundation under Grant ENG 10173.

The authors are with the Department of Electrical Engineering, Stanford University, Stanford, CA 94305.

[1]Other definitions of the knapsack problem exist in the literature [1], [5]. The definition used here is adapted from Karp [14]. To be precise, Karp's knapsack problem is to determine whether or not a solution $\boldsymbol{x}$ exists, while the corresponding cryptographic problem is to determine $\boldsymbol{x}$, given that it exists. The cryptographic problem is not NP-complete, but it is just as hard as the corresponding NP-complete problem. If there is an algorithm for solving the cryptographic problem in time $T(n)$, i.e., for determining $\boldsymbol{x}$ given that it exists, then we can determine whether or not an $\boldsymbol{x}$ exists in time $T(n)$, i.e., solve the corresponding NP-complete problem. If the algorithm determines $\boldsymbol{x}$ in time $T(n)$, then some $\boldsymbol{x}$ exists. If the algorithm does not determine $\boldsymbol{x}$ in time $T(n)$ or determines an incorrect $\boldsymbol{x}$—which is easily checked—then no such $\boldsymbol{x}$ exists.

0018-9448/78/0900-0525$00.75 ©1978 IEEE

II. A Method for Constructing Trapdoor Knapsacks

The designer chooses two large numbers m and w such that w is invertible modulo m (equivalently $\gcd(w,m)=1$). He selects a knapsack vector $\boldsymbol{a}'$ which satisfies (1) and therefore allows easy solution of $S'=\boldsymbol{a}'*\boldsymbol{x}$. He then transforms the easily solved knapsack vector $\boldsymbol{a}'$ into a trapdoor knapsack vector $\boldsymbol{a}$ via the relation

$$a_i = w*a_i' \bmod m. \tag{3}$$

The a_i are pseudo-randomly distributed, and it therefore appears that anyone who knows $\boldsymbol{a}$, but not w and m, would have great difficulty in solving a knapsack problem involving $\boldsymbol{a}$. The designer then can easily compute

$$\begin{aligned} S' &= w^{-1}*S \bmod m && (4)\\ &= w^{-1}*\sum x_i*a_i \bmod m && (5)\\ &= w^{-1}*\sum x_i*w*a_i' \bmod m && (6)\\ &= \sum x_i*a_i' \bmod m. && (7)\end{aligned}$$

If m is chosen so that

$$m > \sum a_i', \tag{8}$$

then (7) implies that S' is equal to $\Sigma x_i*a_i'$ in integer arithmetic as well as mod m. This knapsack is easily solved for $\boldsymbol{x}$, which is also the solution to the apparently difficult, but trapdoor knapsack problem $S=\boldsymbol{a}*\boldsymbol{x}$.

To help make these ideas clearer, we give a small example with $n=5$. Taking $m=8443$, $\boldsymbol{a}'=(171,196,457,1191,2410)$, and $w=2550$ (so $w^{-1}=3950$), then $\boldsymbol{a}=(5457,1663,216,6013,7439)$. Given $S=1663+6013+7439=15115$, the designer computes

$$\begin{aligned} S' &= w^{-1}*S \bmod m && (9)\\ &= 3950*15115 \bmod 8443 && (10)\\ &= 3797. && (11)\end{aligned}$$

Because $S'>a_5'$, he determines that $x_5=1$. Then using (2) for the $\boldsymbol{a}'$ vector, he determines that $x_4=1$, $x_3=0$, $x_2=1$, $x_1=0$, which is also the correct solution to $S=\boldsymbol{a}*\boldsymbol{x}$.

Anyone who does not know m, $\boldsymbol{a}'$, and w has great difficulty in solving for $\boldsymbol{x}$ in $S=\boldsymbol{a}*\boldsymbol{x}$ even though the general method used for generating the trapdoor knapsack vector $\boldsymbol{a}$ is known by the public. The code breaker's task can be further complicated by scrambling the order of the a_i and by adding different random multiples of m to each of the a_i.

The example given was extremely small in size and was only intended to illustrate the technique. Using $n=100$, which is the bottom end of the usable range for secure systems, we would suggest that m be chosen uniformly from the numbers between $2^{201}+1$ and $2^{202}-1$, that a_1' be chosen uniformly from the range $[1, 2^{100}]$, that a_2' be chosen uniformly from $[2^{100}+1, 2*2^{100}]$, that a_3' be chosen uniformly from $[3*2^{100}+1, 4*2^{100}]$, that a_i' be chosen uniformly from $[(2^{i-1}-1)*2^{100}+1, 2^{i-1}*2^{100})]$, that a_{100}' be chosen uniformly from $[(2^{99}-1)*2^{100}+1, 2^{99}*2^{100}]$, and that w' be chosen uniformly from $[2, m-2]$ and then divided by the greatest common divisor of w' and m to yield w.

These choices ensure that (8) holds and that an opponent has at least 2^{100} possibilities for each parameter and hence cannot even search over one of them. Note that each a_i will be pseudo-randomly distributed between 1 and $m-1$ and hence will require a 202-bit representation. Since S requires a 209-bit representation, there is a 2.09:1 data expansion from $\boldsymbol{x}$ to S.

III. Multiplicative Trapdoor Knapsacks

A multiplicative knapsack is easily solved if the vector entries are relatively prime. Given $\boldsymbol{a}'=(6,11,35,43,169)$ and $P=2838$, it is easily determined that $P=6*11*43$ because 6, 11, and 43 evenly divide P but 35 and 169 do not. A multiplicative knapsack is transformed into an additive knapsack by taking logarithms. To make both vectors have reasonable values, the logarithms are taken over GF(m) where m is a prime number [7].

A small example is again helpful. Taking $n=4$, $m=257$, $\boldsymbol{a}'=(2,3,5,7)$, and the base of the logarithms to be $b=131$ results in $\boldsymbol{a}=(80,183,81,195)$. That is, $131^{80}=2 \bmod 257$, $131^{183}=3 \bmod 257$, etc. Finding logarithms over GF(m) is relatively easy if $m-1$ has only small prime factors [7]. (On a computer, the current upper limit on small is in the range 10^6 to 10^{12}.)

Now suppose we are given $S=183+81=264$ and are asked to find the solution to $S=\boldsymbol{a}*\boldsymbol{x}$. Knowing the trapdoor information m, $\boldsymbol{a}'$, and b, we are able to compute

$$\begin{aligned} S' &= b^S \bmod m\\ &= 131^{264} \bmod 257\\ &= 15\\ &= (2^0)*(3^1)*(5^1)*(7^0) && (12)\end{aligned}$$

which implies that $\boldsymbol{x}=(0,1,1,0)$. This is because

$$\begin{aligned} b^S &= b^{(\Sigma a_i * x_i)}\\ &= \prod b^{(a_i * x_i)}\\ &= \prod a_i'^{x_i} \bmod m. && (13)\end{aligned}$$

It is now necessary that

$$\prod_{i=1}^{n} a_i' < m \tag{14}$$

to ensure that $\Pi a_i' x_i \bmod m$ equals $\Pi a_i' x_i$ in arithmetic over the integers.

An opponent who knows the public information $\boldsymbol{a}$, but who does not know the trapdoor information m, $\boldsymbol{a}'$, and b, again appears to face an impossible computational problem.

The example given was again small and only intended to illustrate the technique. Taking $n=100$, if each a_i' is a random 100-bit prime number, then m would have to be approximately 10 000 bits long to ensure that (14) is met. While a 100:1 data expansion is acceptable in certain applications, such as secure key distribution over an insecure channel [4], [8], it is probably not necessary for an opponent to be so uncertain of the a_i'. It may even be possible to use the first n primes for the a_i', in which case m could be as small as 730 bits long when $n=100$ and still meet condition (14). There is a possible trade-off between security and data expansion.

IV. An Iterative Method

This section discusses techniques for improving the security and utility of the basic methods.

In the first method we transformed a hard and apparently very difficult knapsack problem $\boldsymbol{a}$ into a very simple and easily solved knapsack problem $\boldsymbol{a}'$ by means of the transformation

$$a_i' = w^{-1} * a_i \bmod m. \tag{15}$$

We could solve a knapsack involving $\boldsymbol{a}$ because we could solve a knapsack involving $\boldsymbol{a}'$. Notice though that it does not matter *why* we are able to solve knapsacks involving $\boldsymbol{a}'$; all that matters is that we *can* solve them. Rather than requiring that $\boldsymbol{a}'$ satisfy (1), we could require that $\boldsymbol{a}'$ be transformable into a new problem $\boldsymbol{a}''$ by the transformation

$$a_i'' = w'^{-1} * a_i' \bmod m' \tag{16}$$

where the new problem $\boldsymbol{a}''$ satisfies (1) or is otherwise easy to solve. Having done the transformation twice, there is no problem in doing it a third time. That is, we select an $\boldsymbol{a}''$ that is easy to solve, not because it satisfies (1), but because it can be transformed into $\boldsymbol{a}'''$, which *is* easy to solve, by

$$\boldsymbol{a}_i''' = w''^{-1} * \boldsymbol{a}_i'' \bmod m''. \tag{17}$$

It is clear that we can repeat this process as often as we wish.

With each successive transformation, the structure in the publicly known vector $\boldsymbol{a}$ becomes more and more obscure. In essence, we are encrypting the simple knapsack vector by the repeated application of a transformation that preserves the basic structure of the problem. The final result $\boldsymbol{a}$ appears to be a collection of random numbers. The fact that the problem can be easily solved has been totally obscured.

The effect of repeating the process several times is very different from that obtained with certain ciphers, such as a simple substitution. A simple substitution cipher is not strengthened by repetition because the composition of two substitution ciphers is yet another substitution cipher. The (w,m) transformations do not have this closure property. The following example shows that the repetition of two (w,m) transforms need not be equivalent to a single (w,m) transform.

If $w=3$, $m=89$, $w'=17$, $m'=47$, and $\boldsymbol{a}''=(5,10,20)$, then $\boldsymbol{a}'=(38,29,11)$ and $\boldsymbol{a}=(25,87,33)$. Assume there exists $\hat{w}$ and $\hat{m}$ such that

$$\boldsymbol{a} = \hat{w} * \boldsymbol{a}'' \bmod \hat{m}. \tag{18}$$

Then $a_1=25$ and $a_1''=5$ imply that

$$25 = \hat{w} * 5 \bmod \hat{m}. \tag{19}$$

From this we have

$$2*25 = \hat{w}*2*5 \bmod \hat{m} \tag{20}$$

or

$$50 = \hat{w}*10 \bmod \hat{m}. \tag{21}$$

But now the relation between $a_2=87$ and $a_2''=10$ implies that

$$87 = \hat{w}*10 \bmod \hat{m} \tag{22}$$

so $87=50 \bmod \hat{m}$ or $37=0 \bmod \hat{m}$, which implies that $\hat{m}=37$. Equation (19) then becomes

$$25 = \hat{w}*5 \bmod 37 \tag{23}$$

so $\hat{w}=5$. However, if $\hat{w}=5$ and $\hat{m}=37$, then (18) for $a_3=33$ and $a_3''=20$ becomes

$$33 = 5*20 \bmod 37 \tag{24}$$

or $33=26 \bmod 37$, a contradiction. We conclude that no such $\hat{w}$ and $\hat{m}$ can exist.

The original easy-to-solve knapsack vector can meet any condition, such as (1), that guarantees it is easy to solve. For example it could be a multiplicative trapdoor knapsack. In this way it is possible to combine both of the trapdoor knapsack methods into a single method, which is presumably harder to break.

It is important to consider the rate of growth of $\boldsymbol{a}$ because this rate determines the data expansion involved in transmitting the n-bit vector $\boldsymbol{x}$ as the larger quantity S. The rate of growth depends on the method of selecting the numbers, but with $n=100$, each a_i need be at most seven bits larger than the corresponding a_i', each a_i' need be at most seven bits larger than a_i'', and so on. Each successive stage of the transformation needs to increase the size of the problem by only a small fixed amount. Repeating the transformation 20 times will add at most 140 bits to each a_i. If each a_i is 200 bits long to begin with, then they need only be 340 bits long after 20 stages, and S is representable in 347 bits. The data expansion is then only 3.47:1.

V. Compressing the Public File

As described above, the Ith user must place his trapdoor knapsack vector $\boldsymbol{a}(I)$ in a public file. The Jth user can then look up $\boldsymbol{a}(I)$ and send a message $\boldsymbol{x}$ to I, hidden as $S=\boldsymbol{a}(I)*\boldsymbol{x}$. To avoid storing the rather large vector $\boldsymbol{a}(I)$, J could ask I to transmit $\boldsymbol{a}(I)$ to him. But, unless J has some method for testing $\boldsymbol{a}(I)$, user K might fool J by sending him $\boldsymbol{a}(K)$ and saying it was $\boldsymbol{a}(I)$. J would then mistakenly tell all his secrets to K. A method is needed for J to convince himself that he was really sent $\boldsymbol{a}(I)$. With a public file, each user can make one personal appearance when depositing his vector, and after identifying himself to the system, he could identify (authenticate) himself to any user by his ability to decipher messages hidden with his vector. The file itself must be protected, but this is relatively easy because only write protection is needed.

To preserve this authentication benefit of the public file, but to reduce its size (potentially 20 or more kilobits per user), we suggest storing a 100-bit one-way hash total $h[\boldsymbol{a}(I)]$ instead of $\boldsymbol{a}(I)$ itself. When J receives $\boldsymbol{a}(I)$ from I, he computes $h[\boldsymbol{a}(I)]$ and checks this against the value I stored in the public file. The hash function h must be a one-way function [4,9,10,11], so that K cannot generate a new vector $\boldsymbol{a}(K)$ such that $h[\boldsymbol{a}(K)]=h[(\boldsymbol{a}(I)]$, without having to perform a computationally impossible feat.

Allowing 100 bits for storing the user's name and address, (or phone number) the public file now contains 200 bits, instead of over 20 kbit/user. A system with a million users requires a 200 million bit, instead of a 20 billion bit, public file. Transmission costs are comparable for both implementations.

A 100-bit number can be coded as 20 alphanumeric characters, which is small enough to fit in a telephone book. A typical entry would look like this:

Joe Smith......497-1573
KSDJR E6K65 3GFVM OMK4K

The second line is the one-way hash total of Smith's trapdoor knapsack vector $\boldsymbol{a}$(Smith). With this information we can call up Smith and hold a secure conversation with him that no one else can understand. We do not need to have met Smith previously to know we are talking with him or for him to know he is talking with us.

Transmitting 20 kbits on a high-speed 50 kbit/s link takes 0.4 s, but on a low-speed 300 bit/s link, it takes more than a minute. The transmitted data can be reduced by a factor of five to about 4 kbits, which takes less than 15 s to transmit at 300 bit/s, by cutting the number of a_i to $n=20$. The vector $\boldsymbol{x}$, however, now has only 20 binary elements, which is small enough to allow solution by exhaustive search. To maintain security, the information in the $\boldsymbol{x}$ vector must be increased to about 100 bits while keeping $n=20$. This can be done by allowing each element x_i to take on values in the set $\{0,1,2,3,\cdots,31\}$ instead of just in $\{0,1\}$. Specifying each x_i takes 5 bits and specifying the whole vector $\boldsymbol{x}$ takes 100 bits. Equation (1) must now be modified to

$$a_i > 31 * \sum_{j=1}^{i-1} a_j. \tag{25}$$

If n is reduced to 1 and the single element of the $\boldsymbol{x}$ vector assumes a value in $\{0,1,2,\cdots,2^{100}-1\}$, then the system is easily broken because

$$x = S/a. \tag{26}$$

When $n=2$, the system can also be broken easily by an algorithm similar in spirit to the greatest common divisor algorithm. It seems that small values of n weaken the system, and further research is needed to determine how small n can be while still preserving security.

VI. Signatures

As discussed in [4], the need for a digital equivalent of a written signature is a major barrier to the replacement of physical mail by teleprocessing systems. Usual digital authenticators protect against third party forgeries but cannot be used to settle disputes between the transmitter and receiver as to what message, if any, was sent. A true digital signature allows the recipient to prove that a particular message was sent to him by a particular person. Obviously it must be impossible for the recipient to alter the contents of the message and generate the corresponding signature, but it must be easy for him to check the validity of a signature for any message from any user. A digital signature can also be used to generate receipts. The recipient signs a message saying, "I have received the following message: TEXT." This section describes how trapdoor knapsacks can be used to generate such signatures and receipts.

If every S in some large fixed range had an inverse image $\boldsymbol{x}$, then it could be used to provide signatures. When the Ith user wanted to send the message m, he would compute and transmit $\boldsymbol{x}$ such that $\boldsymbol{a}(I)*\boldsymbol{x}=m$. The recipient could easily compute m from $\boldsymbol{x}$ and by checking a date/time field (or some other redundancy in m) determine that the message was authentic. Because the recipient could not generate such an $\boldsymbol{x}$, he saves $\boldsymbol{x}$ as proof that the Ith user sent him the message m.

This method of generating signatures can be modified to work when the density of solutions (the fraction of S between 0 and Σa_i that have solutions to $\boldsymbol{x}*\boldsymbol{a}=S$) is less than 1, provided it is not too small. The message m is sent in plain-text form or encrypted if eavesdropping is a threat, and a sequence of one-way functions [4], [9], [10], [11] $y_1=F_1(m)$, $y_2=F_2(m),\cdots$ are computed. The transmitter then seeks inverse images for $y_1, y_2,\cdots$ until one is found and appends the corresponding $\boldsymbol{x}$ to m as a signature. The receiver computes $y=\boldsymbol{a}*\boldsymbol{x}$ and checks that y is equal to y_k with k not too large, for example, at most 10 times the expected value of k.

The sequence of functions $F_i(*)$ can be as simple as

$$F_i(m)=F(m)+i \tag{20a}$$

or

$$F_i(m)=F(m+i) \tag{20b}$$

where $F(*)$ is a one-way function. It is necessary that the range of $F(*)$ have at least 2^{100} values to foil trial and error attempts at forgery. If the message is much longer than 100 bits, the expansion caused by the addition of a 100-bit authentication field is unimportant.

If the trapdoor knapsack vector were generated as suggested at the end of Section II, the solution density would be less than $1/2^{100}$, and more than 2^{100} y_k would have to be tried on the average before one with a solution is found. The multiplicative method of Section III has an even smaller solution density. It is possible, however, to use the iterative method of Section IV to obtain a solution density of approximately $1/10^4$ with two iterations or $1/10^6$ with three iterations when $n=100$. First, a knapsack vector $\boldsymbol{a}''$ with a solution density near 1 is selected. If $\boldsymbol{a}''=(1,2,4,8,\cdots,2^{99})$ then the solution density is 1, but increasing some of the larger a_i'' need not greatly reduce the solution density. For example, $(1,2,4,8,17,35,68,142)$ has a solution density of 0.92 and still satisfies (1). Such choices may not be necessary, but they provide an additional margin of safety at almost no additional cost.

After selecting $\boldsymbol{a}''$, parameters m' and w' are chosen such that $m'>\Sigma a_i''$ and $w^{-1\prime}$ exists modulo m'. The weak trapdoor knapsack vector

$$\boldsymbol{a}'=w'*\boldsymbol{a}'' \qquad \bmod m' \tag{27}$$

is then computed. New parameters $m>\Sigma \boldsymbol{a}_i'$ and w (with w^{-1} existing mod m) are chosen, and the more secure trapdoor knapsack vector

$$\boldsymbol{a}=w*\boldsymbol{a}' \qquad \bmod m \tag{28}$$

is computed. The process can be iterated more than twice to obtain the final vector $\boldsymbol{a}$, but the solution density typically decreases by a factor of $n/2$ with each iteration. When used for hiding information this decrease is of little importance, but when used for signatures several iterations are all that can be afforded because of the need for a high solution density. With so few iterations, it is possible for two adjacent a_i to be in the same ratio (usually 2 : 1) as they were in the $\boldsymbol{a}$ vector. This weakness can be overcome by adding multiples of m' (or m) to a subset of the a_i' (or a_i) that suffer from this problem. This decreases the solution density somewhat and accounts for our $1/10^4$ and $1/10^6$ estimates for two and three iterations when $n=100$.

A small example is again helpful in illustrating the method. Starting with

$$\boldsymbol{a}''=(1,2,4,8,17,35,68,142) \tag{29}$$

whose components sum to 277, we choose $m'=291$ and $w'=176$ $(w'^{-1}=167)$ resulting in

$$\boldsymbol{a}'=(176,61,122,244,82,49,37,257). \tag{30}$$

The second, third, and fourth components are in the ratio of 2 : 1, which can be hidden by adding m' to the third component to obtain the new vector

$$\boldsymbol{a}'=(176,61,413,244,82,49,37,257) \tag{31}$$

whose components sum to 1319. Choosing $m=1343$, $w=498$ $(w^{-1}=925)$ yields

$$\boldsymbol{a}=(353,832,195,642,546,228,967,401) \tag{32}$$

whose components sum to 4164. The density of solutions using $\boldsymbol{a}$ is $256/4164=0.061$ so approximately 16 attempts are needed on the average to obtain a signature. This agrees well with the estimated range of $n^2/4=16$ to $n^2=64$.

The density of solutions can be increased by restricting the y_k to lie near the middle of the range $(0,\Sigma \boldsymbol{a}_i)$, say between 1000 and 3000 in this example. The law of large numbers indicates that for most $\boldsymbol{x}$ the sum $\boldsymbol{a}*\boldsymbol{x}$ will lie in this range.

Merkle and Reeds [12] have developed another approach to obtaining high-density knapsacks. Empirical results indicate densities of approximately 20 percent when $n=100$.

VII. Discussion

We have shown that it is possible to construct trapdoor knapsack problems and that information and signatures can be hidden in them for transmission over an insecure channel. Conventional cryptographic systems also can hide information and authenticators during transmission over an insecure channel but have the disadvantage that first a "key" must be exchanged via courier service or some other secure means. Also in conventional cryptography, the authenticator only prevents third party forgeries and cannot be used to settle disputes between the transmitter and receiver as to whether a message was actually sent.

We have not proved that it is computationally difficult for an opponent who does not know the trap information to solve the problem. Indeed, proofs of security are not yet available for normal cryptographic systems, and even the general knapsack problem has not been *proved* difficult to solve. The theory of computational complexity has not yet reached the level of development where such proofs are feasible. The best published algorithm for solving the knapsack problem is exponential, taking $O(2^{n/2})$ time and space [1]. Schroeppel [2] has devised an algorithm that takes $O(2^{n/2})$ time and $O(2^{n/4})$ space. Faith in the security of these systems must therefore rest on intuition and on the failure of concerted attempts to break them.

Attempts to break the system can start with simplified problems (e.g., assuming m is known). If even the most favored of certificational attacks is unsuccessful, then there is a margin of safety against cleverer, wealthier, or luckier opponents. Or, if the favored attack is successful, it helps establish where the security really must reside. For example, if knowledge of m allows solution, then an opponent's uncertainty about m must be large.

As noted, the techniques suggested in this paper generalize to x_i in the set $\{0,1,2,3,\cdots,N\}$. The advantages and weaknesses of such systems deserve further study. Further work with knapsack-based methods is in progress, and research oriented toward placing trapdoors in other combinatorial problems also appears promising.

Other techniques for securely communicating over an insecure channel have been proposed in [4], [8], and [13]. The method described in [4] involved exponentiation mod q. The techniques proposed in this paper appear to be significantly more secure and allow the direct transfer of information $\boldsymbol{x}$ generated by the transmitter. The technique proposed in [4] allows the transmitter and receiver to generate a common piece of information K which they then use as the key in a normal cryptographic system, but K cannot be predetermined by either party. Merkle's technique [8] was generalized to the first public key cryptosystem and is quite secure but computationally expensive. The current work describes a computationally efficient public key cryptosystem.

Recently, Rivest, Shamir, and Adleman [13] have proposed another public key cryptosystem that yields signatures more directly because the density of solutions in their problem is one. Their system also requires a smaller key (apparently 600 bits versus 20 kbits). Neither system's security has been adequately established, but when iterated, the trapdoor knapsack appears less likely to possess a chink in its armor. When used for obtaining signatures the trapdoor knapsack appears to be the weaker of the two. Both public key systems clearly need further certification and study.

References

[1] E. Horowitz and S. Sahni, "Computing partitions with applications to the knapsack problem," *JACM*, vol 21, no. 2, pp. 277–292, Apr. 1974.
[2] R. Schroeppel, unpublished work.
[3] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*. Reading, MA: Addison-Wesley, 1974.
[4] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Trans. Inform. Theory*, vol. IT-22, Nov. 1976, pp. 644–654.
[5] O. H. Ibarra and C. E. Kim, "Fast approximation algorithms for the knapsack and sum of subset problems," *JACM*, vol. 22, no. 4, pp. 463–468, Oct. 1975.
[6] E. L. Lawler, "Fast approximation algorithms for knapsack problems," Electronics Research Laboratory, College of Eng. U.C. Berkeley Memorandum UCB/ERL M77/45 21, June 1977.
[7] 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, pp. 106–110, Jan. 1978.
[8] R. Merkle, "Secure communications over insecure channels," *Commun. ACM*, vol. 21, no. 4, pp. 294–299.
[9] M. V. Wilkes, *Time-Sharing Computer Systems*. New York: Elsevier, 1972.
[10] A. Evans, Jr., W. Kantrowitz, and E. Weiss, "A user authentication system not requiring secrecy in the computer," *Commun. ACM*, vol. 17, no. 8, Aug. 1974, pp. 437–442.
[11] G. B. Purdy, "A high security log-in procedure," *Commun. ACM*, vol. 17, no. 8, Aug. 1974, pp. 442–445.
[12] R. Merkle and J. Reeds, unpublished work.
[13] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978.
[14] R. M. Karp "Reducibility among combinatorial problems," in *Complexity of Computer Computations*, R. E. Miller and J. W. Thatcher, Eds. New York: Plenum, (1972), pp. 85–104.

Compression of Individual Sequences via Variable-Rate Coding

JACOB ZIV, FELLOW, IEEE, AND ABRAHAM LEMPEL, MEMBER, IEEE

***Abstract*—Compressibility of individual sequences by the class of generalized finite-state information-lossless encoders is investigated. These encoders can operate in a variable-rate mode as well as a fixed-rate one, and they allow for any finite-state scheme of variable-length-to-variable-length coding. For every individual infinite sequence x a quantity $\rho(x)$ is defined, called the compressibility of x, which is shown to be the asymptotically attainable lower bound on the compression ratio that can be achieved for x by any finite-state encoder. This is demonstrated by means of a constructive coding theorem and its converse that, apart from their asymptotic significance, also provide useful performance criteria for finite and practical data-compression tasks. The proposed concept of compressibility is also shown to play a role analogous to that of entropy in classical information theory where one deals with probabilistic ensembles of sequences rather than with individual sequences. While the definition of $\rho(x)$ allows a different machine for each different sequence to be compressed, the constructive coding theorem leads to a universal algorithm that is asymptotically optimal for all sequences.**

Manuscript received June 10, 1977; revised February 20, 1978.

J. Ziv is with Bell Laboratories, Murray Hill, NJ 07974, on leave from the Department of Electrical Engineering, Technion—Israel Institute of Technology, Haifa, Israel.

A. Lempel is with Sperry Research Center, Sudbury, MA 01776, on leave from the Department of Electrical Engineering, Technion—Israel Institute of Technology, Haifa, Israel.

I. Introduction

In a recent paper [1], data-compression coding theorems and their converses were derived for the class of finite-state encoders that map at a fixed rate input strings drawn from a source of α letters into equally long strings over an alphabet of $\beta \leqslant \alpha$ letters. In the context of data-compression, the aim is to minimize the number of bits/symbol $\log_2 \beta$, while securing zero or negligibly small distortion. For every individual infinite sequence x, this minimal bit/symbol rate was shown in [1] to be equal to a quantity $H(x)$ that, in analogy with the Shannon entropy (which is defined for probabilistic ensembles rather than