Knapsack and Its Applications To Subset Cryptosystems

By: Gregory Ryslik

I. Background on Knapsack and Its Inventors

Whitfield Diffie and Martin E. Hellman first published the notion of a public key cryptography scheme in 1976 in their paper "New Directions in Cryptography". This scheme was the first to implement the idea of key agreement using the discrete log problem (2). Within the paper, Diffie and Hellman note the idea of trap-door one-way functions and the possibility of using a knapsack problem for future cryptography. They then go on to mention that great care must be taken in choosing the parameters of such a function or else solving the knapsack problem becomes computationally feasible (6). In 1978, (just months after Rivest, Shamir and Adleman published the first public-key cryptosystem, RSA (2)) Ralph C. Merkle and Martin E. Hellman, both members of the Electrical Engineering department at Stanford University discovered a few trapdoors in the knapsack function. With this discovery, they became the inventors of the first knapsack cryptography scheme (5). Since their discovery, Professor Hellman briefly taught at MIT from 1969-1971 at which point he returned to Stanford. There, he has served as the Associate Chair of the Electrical Engineering department, the Associate Dean for Graduate Studies of Minority Students and Chairman of Electrical Engineering Graduate Admissions. In addition, Hellman has also written more than sixty technical papers and holds a number of both domestic and foreign patents (8). Ralph C. Merkle on the other hand went to work for Xerox PARC in 1988 where he has been conducting research on computational nanotechnology (9). He is currently a professor at Georgia Tech's College of Computing, holds eight patents and was a co recipient of the 1998 Feynman Prize for Nanotechnology(8).

II. Introduction to the Knapsack Problem

The subset-sum problem (commonly known as the knapsack problem) is to determine whether or not some subset of positive integers $a_1,\ldots a_n$ (known as weights) adds up to a sum s. This is the same as whether or not there is a set of variables $x_1,\ldots x_n$ such that

$$S = \sum_{j=1}^{n} x_j a_j \qquad x_j \in \{0,1\} \tag{1.0}$$

The knapsack problem is considered to be quite difficult and is classified under the group of problems known as NP-Complete (5). One method to determine whether or not equation (1.0) has a solution is to compute all the

$$\sum_{j=1}^{n} x_j a_j \qquad x_j \in \{0,1\} . \tag{1.1}$$

However, this process takes $O(2^n)$ time. This is due to the fact that for every a_j a binary decision of whether or not this element is in the subset needs to be made. However, this running time has actually been reduced to $O(n2^{\wedge n/2})$ through the following algorithm. First compute:

$$S_1 = \{\sum_{j=1}^{\lfloor n/2 \rfloor} x_j a_j\} \quad \text{AND} \quad S_2 = \{s - \sum_{j>\lfloor n/2 \rfloor}^{n} x_j a_j\} \quad \text{for all} \quad x_j \in \{0,1\} \quad (1.2)$$

This process takes $O(2^{n/2})$. Then, proceed to sort the sets S1 and S2 (which is $O(n2^{n/2})$) and then scan them for an identical element (which is $O(2^{n/2})$). An identical element in both sets would occur if and only if the there is a solution to (1.0). (3)

III. Knapsack and its applications in cryptosystems

The idea behind all knapsack cryptosystems is to use a public set of weights to encode the message, send a sum S and then have the receiver use his private set of weights to decode the message. The cryptanalyst, having only the public set of weights available would be forced to solve a NP-Complete problem while the receiver would just have to solve a polynomial time problem. (3)

IV. Transmission of the message

One of the prime advantages of the knapsack system is that it lends itself exceptionally well to computer transmissions due to the fact that the x_j is either a 0 or a 1. Thus, when the computer wants to send a bit stream (which consists of a sequence of 0's and 1's), it can simply take the bit and use it as the x_j in the summation

$$S = \sum_{j=1}^{n} x_j a_j \quad . \quad (1.3)$$

For example, if the user wanted to send the bit stream 01011 and the set of weights was 5,14,25,50,95, the user would have to perform the following summation. S= 0*5 + 14*1 +25*0+50*1 + 95*1 = 64. This can be clearly seen by "overlaying" the bit stream with the set of weights as follows:

0	1	0	1	0
5	14	25	50	95

Then, the weights with the 1's on top would be added. Finally, the last part of the transmission would be to simply send the 64. Note that the actually bit stream is never

sent, just the sum S. It is the receiver's task to actually solve this knapsack problem and recreate the original bit stream. However, as mentioned earlier, the knapsack problem falls under the category of NP-Complete and thus is difficult to solve. Hence, a method must created in order to allow the receiver to recreate the original bit stream in a reasonable time frame.

V. Simple Knapsack Problems

Due to the fact that in general knapsack problems fall in the complexity category of NP-Complete, a simple knapsack problem must be devised in order for the receiver to be able to solve the problem. One such example arises if the set of weights used forms a super-increasing sequence, which is defined as a sequence satisfying for all $1 < j \leq n$:

$$a_j > \sum_{i=1}^{j-1} a_i \quad . \quad (1.4)$$

Thus, if the next element in the set is larger than the summation of all the previous elements the knapsack problem is easy to solve (3). This comes from the fact that the x_n^{th} term (where n is the total number of elements in the set) is 1 if and only if: $S \geq \sum_{j=1}^{n-1} a_j$.

(1.5)

For example, the previous set of weights used is super-increasing. Thus if the receiver wanted to recreate the bit stream (as in the previous example) and was sent the sum 64, the following algorithm would have to be executed in order to recreate the original bit stream. Note the highly iterative process which lends itself easily to computer code.

1) check if $S \geq 95$: no. Thus $x_4 = 0$;
2) check if $S \geq 50$: yes. Thus $x_3 = 1$. Subtract 50 from S to get S= 14.
3) check if $S \geq 25$: no. Thus $x_2 = 0$;
4) check if $S \geq 14$: yes. Thus $x_1 = 1$; Subtract 14 from S to get S = 0.
5) check if $S \geq 5$: no. Thus $x_0 = 0$

Hence, reading from step 5 back to step 1 (x_0 up to x_5) the original bit stream is reconstructed to be 01010.

VI. Making it hard for the cryptanalyst to decipher

While the above process is good from the point of view that it is easy for the receiver to obtain the original message, this protocol is invalid due to the fact that the cryptanalyst can just as easily follow the same steps and arrive at the original message.

Hence, the basic idea of all knapsack cryptosystems is to change secret weights $b_1,\ldots,b_n$ (which was the super-increasing sequence) into a public set $a_1,\ldots,a_n$ which would have no apparent structure to the cryptanalyst.

Merkle and Hellman created the first and most famous of these trap-door functions that transform the private set to the public set in 1978. The transformation follows the following algorithm:

1. Choose positive integers M and W where the greatest common divisor of M and W is 1 (abbreviated gcd(M,W) = 1) and $\mathrm{M} > \sum_{j=1}^{n} b_j$. (1.6)
2. Then the receiver computes $a'_j \equiv b_j W \pmod{M}$. (1.7)
    i. Note that: a'_j cannot be = 0 since gcd(M,W) = 1.
3. Then the user chooses a permutation π on the set {1,…,n}.
4. Set $a_j = a'_{\pi(j)}$. (1.8)

Hence the a_j's are used for the public weights while M and W and π are kept secret as well as the original super-increasing sequence of the b_j's. (5)

For example:

1. In order to form a public set of weights from the super-increasing sequence {5,14,25,50,95}[1], the following operations would have to be performed:
2. Sum up each element in the private set of weights to get a total of 189.
3. Thus since M must be greater than 189, let M be 225.
4. Now select a W where the gcd(W,M) is 1. Let W be 17.
5. First, the receiver modifies the weights using modular arithmetic as in step 2 of the algorithm to get: 85,13,200,175,40
6. Now a permutation must be chosen. Suppose the permutation is: 1-> 3, 2->1, 3->4,4->2, 5->5.
7. Thus the set of public weights becomes: 13,175,85,200,40.

Note that the sequence is no longer super-increasing and also due to the permutation any element does not necessarily match up to its original in the private set (i.e. the first element in the public set was not necessarily the smallest element in the private set).

Thus resulting a_j's would be difficult for the cryptanalyst to solve.

---

[1] This super-increasing sequence is technically invalid. In order to create a super-increasing sequence that when encrypted does not lend itself to be easily broken while at the same time does not need enormous storage potential, the following restrictions are imposed: $b_1 \approx 2^n, b_j > \sum_{t=1}^{j-1} b_t, 2 \le j \le n, b_n \approx 2^{2n}$ (where n usually ranges anywhere from 100 to 200). (5)

The above method is called a *singly-iterated Merkle-Hellman system*. A more complex variety of the above procedure is what is known as the *multiply-iterated Merkle-Hellman system*. This system iteratively chooses a M and W for each element in the private set of weights based upon the following guidelines:

$$M_k > \sum_{j=1}^{n} a_j^{(k-1)}, \quad a_j^{(k)} \equiv a_j^{(k-1)} W_k \ Mod(M_k), \text{ and } \gcd(M_k, W_k) = 1.$$

This method hides the original private set even more successfully than the singly iterated system (3). For the scope of this paper however, the remainder of the examples will be based upon the singly-iterated system in order to decrease the amount of computational steps needed to be shown.

VII. Sending the transmission on a modified set of public weights

Once a public set of weights is published, the sender sends his data based upon the public set which is difficult to decode. The sum S would be calculated based on the formula (1.1) where the a_j's form the public set and then sent to the receiver.

For example:
Suppose the sender wanted to encrypt the message 10100.
The sender would perform the following 13*1 + 175*0 + 85*1+200*0 + 40*0 = 98.
Thus the sender would send the sum S = 98. (3)

VIII. Deciphering the transmission on the receiving side.

The receiver, upon receiving S would then have to recreate the original bit stream sequence. However, due to the fact that S was calculated on the public set, the receiver would have to use the following transformations to successfully solve the problem.

The receiver computes $C \equiv SW^{-1} \pmod{M}$. (1.8)

Note: W^{-1} denotes the multiplicative inverse of W modulo M.

However, C must then equal the following:

$$C = \sum_{j=1}^{n} x_j a_j W^{-1} \pmod{M} = \sum_{j=1}^{n} x_j a'_{\pi(j)} W^{-1} \pmod{M} = \sum_{j=1}^{n} x_j b_{\pi(j)} \pmod{M} \quad (1.9)$$

Thus, the user would arrive back at the original knapsack problem that is based upon a super-increasing sequence and is easy to solve. (3)

For instance, if the receiver received the sum 98 based upon our previous set of public weights (section V), the receiver would proceed to do the following operations.

1) Calculate C = 98*53 = 19 as per formula (1.8)
2) Then he would solve the knapsack problem based upon the secret b_j's and the C calculated in step 1.
    1) check if $S \geq 95$: no. Thus $x_4 = 0$;
    2) check if $S \geq 50$: no. Thus $x_3 = 0$.
    3) check if $S \geq 25$: no. Thus $x_2 = 0$;
    4) check if $S \geq 14$: yes. Thus $x_1 = 1$; Subtract 14 from S to get S = 5.
    5) check if $S \geq 5$: yes. Thus $x_0 = 1$.

The result is 11000. However, since the sum was created on a permuted set of weights, the receiver would have to permute his result to end up with the original sequence. Using the permutation defined in section V on 11000, the receiver arrives at 10100, which was the original message (as sent by the sender in section VI).

IX. Benefits of the knapsack system

The main benefit of an encryption scheme utilizing the knapsack protocol is due to its speed. RSA which is currently the most widely used system today can only encrypt at the rate of 10^4 bits per second (using the original modulus of 430 bits and special purpose chips). With software restrictions the encryption rate drops down further to the rate of 100 bits per second. Thus at its best, the RSA system works about 100 to 1000 times slower than the classical key cryptosystems such as DES(Data Encryption Standard) which with software restrictions runs at about 10^5 bits per second (on special purpose chips this figure jumps up to tens of millions of bits per second).

However, using the singly-iterated Merkle-Hellman scheme with an n roughly equal to 100 (this n is large enough due to the fact that even the best known way to solve a knapsack problem is $O(2^{n/2})$), allows the encryption rate to be 100 times faster than RSA. (3)

X. The downfall of Knapsack

Unfortunately, even though knapsack is significantly faster than RSA, the singly iterated Merkle-Hellman system was broken by Shamir in 1982 (1) (for which Shamir received a grand total of $100 (2)). As mentioned earlier, the singly-iterated Merkle-Hellman system used the fact that using the series of transformations (as described in section VI), a trap-door function would be created that would be difficult for the cryptanalyst to solve. Shamir was able to show that through a few basic assumptions (that are inherent to choosing a valid knapsack as discussed in the footnote on page 4), a cryptanalyst could arrive at the original W and M used in the private set of weights by solving a system of equations using Lenstra's integer programming algorithm. Due to the

fact that this algorithm runs in polynomial time, the derivation of W and M becomes computationally feasible. This rendered the original knapsack scheme insecure and useless from the point of view of secure communication (1). To counter this, Merkle and Hellman came up with the multiply-iterated scheme. In response, Ernest F. Brickell, using roughly an hour of Cray-1 time was able to break a multiply-iterated Merkle-Hellman system that used a series of 40 iterations (which was well inside the set parameters as to the number of iterations). Due to the fact that the knapsack scheme was broken, the communication industry chose to use the RSA protocol, which is unfortunately significantly slower.

However, there are a number of knapsack systems which have not been broken. For example, the system designed by Masakatu Morrii and Masao Kasahara in 1988 which involves a discrete log with a hard multiplicative knapsack has not yet been successfully attacked. (2) It remains to be seen whether the communications industry will decide that this type of a knapsack encryption scheme is in fact secure enough to use as a secret communication protocol.

Bibliography:

(1) **A polynomial-time algorithm for breaking the basic Merkle - Hellman cryptosystem**
Shamir, A.;
Information Theory, IEEE Transactions on , Volume: 30 Issue: 5 , Sep 1984
Page(s): 699 –704

(2)http://www.ics.uci.edu/~mingl/knapsack.html
Ming Kin Lai

(3)http://www.dtc.umn.edu/~odlyzko/doc/arch/knapsack.survey.pdf
A.M. Odlyzko

(5) **Hiding information and signatures in trapdoor knapsacks**
Merkle, R.; Hellman, M.;
Information Theory, IEEE Transactions on , Volume: 24 Issue: 5 , Sep 1978
Page(s): 525 –530

(6)New directions in cryptography
Diffie, W.; Hellman, M.;
Information Theory, IEEE Transactions on , Volume: 22 Issue: 6 , Nov 1976
Page(s): 644 –654

(7) http://www.house.gov/science/merkle_062299.htm
Ralph C. Merkle

(8) http://gtisc.gatech.edu/hellman_bio.htm

(9) http://www.merkle.com/
Ralph C. Merkle