THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 60, No. 5, May–June 1981
Printed in U.S.A.

B.S.T.J. BRIEF

Fast Decryption Algorithm for the Knapsack Cryptographic System

By P. S. HENRY

(Manuscript received December 5, 1980)

I. INTRODUCTION

Public-key cryptosystems offer a degree of flexibility not available with conventional (private-key) systems.[1,2] In particular, the key required for decryption in a public-key system can be changed at will, even in the middle of a message. This makes the task of the eavesdropper very difficult indeed. A frequently cited disadvantage of public-key systems is their relative slowness (typically a few kilobit/s) caused by the large amount of number-crunching they require.[3,4] This has led to the development of hybrid cryptosystems in which a key, exchanged via a slow public-key system, is subsequently used in a fast conventional system, such as the Data Encryption Standard (DES).[5] In this paper we present a fast algorithm for executing the knapsack cipher (a public-key cryptosystem).[6] When implemented with TTL integrated circuitry, this algorithm should permit data rates in the neighborhood of 10 Mbit/s. This speed is sufficient to provide security for a wide range of voice, data, and narrowband video traffic without the need for a hybrid cryptosystem.

Section II presents an elementary example of the knapsack cipher to show how it operates. In Section III we describe the fast algorithm, and in Section IV we discuss a more sophisticated knapsack cipher.

II. AN EXAMPLE OF THE KNAPSACK CIPHER

A very simple (and insecure) knapsack cipher begins with an "easy" knapsack vector generated by a party who wishes to receive encrypted data [eq. (29) of Ref. 6],

$$\mathbf{E} = (1, 2, 4, 8, 17, 35, 68, 142). \tag{1}$$

The eight components of **E** form a super-increasing sequence: Each term is larger than the sum of all those preceding it. Let the data to be encrypted be represented as a vector with eight binary components,

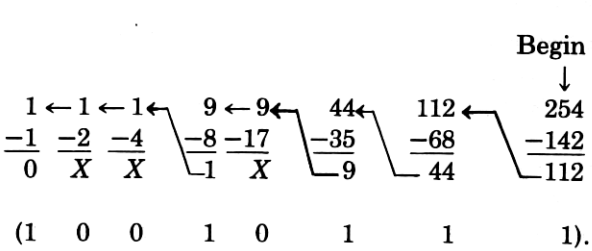
$$\mathbf{D} = (1, 0, 0, 1, 0, 1, 1, 1). \tag{2}$$

To encrypt **D** using **E**, form the dot product,

$$S_E = \mathbf{E} \cdot \mathbf{D} = 254. \tag{3}$$

The number S_E is an encrypted form of **D**.

The super-increasing property of **E** guarantees that **D** can be recovered from S_E by subtracting successive components of **E** (beginning with the largest) from S_E and keeping the residue. If a component of **E** is less than or equal to the residue at any stage in the subtraction, the corresponding component of **D** is 1. If a component of **E** is larger than the residue, the corresponding **D** component is 0 and we try the next (smaller) component of **E**. This process is illustrated below for $S_E = 254$ [eq. (3)],



D = (1 0 0 1 0 1 1 1).

Of course, **E** cannot be used for secure encryption, because if **E** were obtained by an eavesdropper he could use it to decrypt any transmitted message. The knapsack cipher provides security by transforming **E** into a "hard" knapsack vector **H** (the public key), which can be used for encryption, but which is useless for decryption. To generate this transformation, the receiver chooses two secret integers M and W such that: (*i*) M is larger than the sum of all the components in **E**, and (*ii*) W and M are relatively prime. (This condition means that W is invertible modulo M: $W^{-1} \cdot W \equiv 1 \bmod M$.) Following Ref. 6, we choose $M = 291$ and $W = 176$ (which implies $W^{-1} = 167$). **H** is generated from **E** by

$$H_j \equiv W \cdot E_j \bmod M, \tag{4}$$

yielding

$$\mathbf{H} = (176, 61, 122, 244, 82, 49, 37, 257).$$

In the ideal case **H** looks like a random sequence; the super-increasing

structure of the original **E** is completely obliterated. **H**, the public key, is sent to the transmitter and need not be kept secret.

To encrypt **D** using **H**, form the dot product as before,

$$S_H = \mathbf{H} \cdot \mathbf{D} = 763. \tag{5}$$

S_H is the encrypted data. If the number of components in **H** is large, say 100 or more, then an eavesdropper, even though he has **H** and S_H, cannot recover **D** in a reasonable time. The legitimate receiver, however, can recover **D** easily by using the inverse transformation,

$$S_E \equiv S_H \cdot W^{-1} \bmod M. \tag{6}$$

That is, by using his secret M and W^{-1}, the receiver can convert S_H into the number S_E [eq. (3)], the same number that would have been obtained if **D** had been encrypted with **E** instead of **H**. Once he has S_E, the receiver simply subtracts off successive components of **E** to recover **D**.

III. A FAST DECRYPTION ALGORITHM

The most time-consuming step of the knapsack cipher is the modular multiplication of eq. (6). In practice, the quantities S_H, W^{-1}, and M might be 100 to 200 bits long, making computation of S_E very slow. The calculation can be expedited by considering the n-bit binary expansion of S_H,

$$S_H = b_{n-1} \cdot 2^{n-1} + \cdots + b_0 \cdot 2^0. \tag{7}$$

Substituting eq. (7) into eq. (6), we have

$$S_E \equiv [b_{n-1}(2^{n-1}W^{-1} \bmod M) + \cdots + b_0(2^0 W^{-1} \bmod M)] \bmod M. \tag{8}$$

Each term in the square brackets is the product of a binary digit (0 or 1) and a fixed quantity (in parentheses), which can be computed ahead of time and stored in a memory. Evaluation of S_E thus reduces to a sequence of table lookups and accumulations, one lookup for each bit in S_H. After all the bits in S_H have been processed, the final reduction mod M is accomplished by an easy long division. [The division is "easy" because each term in eq. (8) can be no bigger than M, so the final sum can be no bigger than nM; division by M can therefore be accomplished with only approximately $\log_2 n$ substract-and-shift operations in binary arithmetic.]

Table I shows the contents of the lookup table required for decryption of the example in Section II, along with the binary representation of $S_H = 763$. The value of the sum within the square brackets of eq. (8) is seen to be 1127, which is equivalent to 254 in mod 291 arithmetic, as required.

Table I—Lookup table for decryption of the example in Section II

k	$2^k \cdot 167$ mod 291	b_k
9	241	1
8	266	0
7	133	1
6	212	1
5	106	1
4	53	1
3	172	1
2	86	0
1	43	1
0	167	1

Figure 1 shows a block diagram of the decryption process. The basic steps of lookup, accumulation, reduction mod M and successive subtraction are pipelined, and within each step most of the processing can be performed on all bits in parallel. This architecture results in very fast operation, the speed limitation being either the memory access time or the accumulator add time, whichever is greater. With Schottky TTL and carry-lookahead addition, these times are both in the neighborhood of 50 ns, so a throughput rate of 10 Mbit/s is reasonable.

Implementation of the decryption algorithm using very large scale integration appears attractive. Most of the circuitry is simply a large lookup table, as shown at the top of Fig. 1. Its capacity is determined primarily by the number of components in **E** and the allowed range (number of possible values) for each component of **E**. We can achieve reasonable security by using 100 and 2^{100}, respectively, for these two parameters; this leads to a value for the modulus M in the neighborhood of 2^{200}. Since each component of **H** is less than M, S_H [eq. (5)] will be less than 2^{207}. The lookup table must therefore contain 207 words, each 200 bits long, implying a memory size of approximately 41 kilobits. Additional memory (~15 kilobits) is required to store the components of **E**. Thus, approximately 56 kilobits of memory and some simple arithmetic logic to perform the steps of accumulation, long division, and successive subtraction are adequate to implement the decryption process. This level of complexity is within the range of current VLSI technology.[7,8]

Finally, we remark that a straightforward implementation of Fig. 1 may not be the best approach; several modifications of the basic decryption algorithm must be investigated. For example, the lookup table can be eliminated by calculating the numbers $2^k \cdot W^{-1}$ mod M one-by-one as they are needed for each incoming bit of S_H. Starting with W, successive numbers can be generated by a simple left shift (and subtraction of M if necessary).

IV. ITERATED KNAPSACK TRANSFORMATIONS

The security of the knapsack cipher is enhanced if iterated (multiple) knapsack transformations are employed.[9] For example, the "hard" vector **H** [eq. (4)] can be kept secret and used to generate a "harder" public vector **H'**,

$$H'_j \equiv W' \cdot H_j \bmod M'. \tag{9}$$

Data can be encrypted with **H'** in the usual fashion,

$$S_{H'} = \mathbf{H}' \cdot \mathbf{D}. \tag{10}$$

If M' is chosen to be greater than the sum of all the components of **H**, then data encrypted using **H'** may be decrypted using two successive inverse transformations having the form of eq. (6). The cost of this double-iteration technique in terms of the bandwidth efficiency of the cipher is modest. For a 100-component knapsack, the modulus M' will

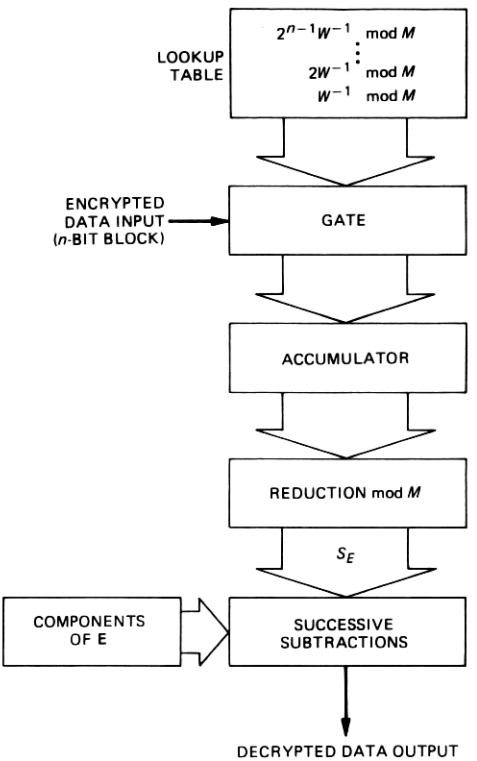


Fig. 1—The fast knapsack decryption algorithm. (Wide arrows signify parallel data transfer.) Pipeline architecture and parallel processing contribute to a high throughput rate. Hardware implementation would require approximately 56 kilobits of memory and a small amount of arithmetic logic.

be roughly 100 times bigger than M; thus $S_{H'}$ will require only about seven more bits than S_H would have required.

We illustrate the double-iteration technique by continuing with the example of Section II. Let $M' = 2001$ and $W' = 1984$, giving $(W')^{-1} = 1177$. From eq. (9) we have

$$\mathbf{H}' = (1010, 964, 1928, 1855, 607, 1168, 1372, 1634). \tag{11}$$

Encrypting **D** [eq. (2)] with **H'** yields $S_{H'} = 7039$.

Decryption requires two inverse transformations,

$$\begin{aligned} S_E &\equiv S_H \cdot W^{-1} \bmod M \\ &\equiv [S_{H'} \cdot (W')^{-1} \bmod M'] \cdot W^{-1} \bmod M \\ &\equiv [7039 \cdot 1177 \bmod 2001] \cdot 167 \bmod 291 \\ &\equiv 763 \cdot 167 \bmod 291 \\ &= 254. \end{aligned} \tag{12}$$

The cascaded inverse transformations in eq. (12) can be executed in tandem using the algorithm of Section III. Thus, the decryption process will entail a longer total delay (compared with the single-iteration case), but the net throughput rate will be essentially unchanged.

We mentioned earlier that straightforward use of the multiple iteration technique reduces the bandwidth efficiency of the cipher. It is possible, however, that for a given level of security, multiple interations may actually be more efficient than a single knapsack transformation. This is because the enhanced security associated with repeated transformations might permit a smaller range for the components of **E**, and hence smaller values for the moduli M, M', etc. The consequent reduction in the encrypted block length could offset the seven-bit increase normally associated with each iteration.

V. CONCLUSIONS

The existence of a fast algorithm for decryption of the knapsack cipher means that the advantages of public-key cryptosystems can be realized even in high-speed applications. Full integration of the decryption process onto a single chip appears feasible with current VLSI technology. The relationships among cipher security, bandwidth efficiency, and number of iterations need further investigation.

VI. ACKNOWLEDGMENT

F. H. Myers has been generous with his time and insights; I am grateful for his help.

REFERENCES

1. M. E. Hellman, "An Overview of Public Key Cryptography," IEEE Commun. Soc. Mag., *16* (November 1978), pp. 24–32.
2. W. Diffie and M. E. Hellman, "Privacy and Authentication: An Introduction to Cryptography," Proc. IEEE, *67* (March 1979), pp. 397–427.
3. B. P. Schanning, "Data Encryption with Public Key Distribution," EASCON Conf. Rec., Washington, D.C., October 9–11, 1979, pp. 653–60.
4. G. J. Simmons, "Symmetric and Asymmetric Encryption," Comput. Surv., *11* (December 1979), pp. 306–30.
5. F. H. Myers, "A Data Link Encryption System," Conf. Rec. Nat. Telecommun. Conf., Washington, D.C., November 27–29, 1979, Paper 43.5.
6. R. C. Merkle and M. E. Hellman, "Hiding Information and Signatures in Trapdoor Knapsacks," IEEE Trans. Inf. Theor., *IT-24* (September 1978), pp. 525–30.
7. S. Matsue et al., "A 256K Dynamic RAM," 1980 IEEE Int. Solid State Circuits Conf. Dig. Tech. Papers, February 13–15, 1980, pp. 232–3.
8. P. M. Russo, "VLSI Impact on Microprocessor Evolution, Usage, and System Design," IEEE J. Solid State Circuits, *SC-15* (August 1980), pp. 397–405.
9. A. Shamir and R. E. Zippel, "On the Security of the Merkle–Hellman Cryptographic Scheme," IEEE Trans. Inf. Theor., *IT-26* (May 1980), pp. 339–40.