THE REDOC II CRYPTOSYSTEM

Thomas W. Cusick
Department of Mathematics
State University of New York at Buffalo
Buffalo, New York 14214
York 14701

Michael C. Wood
Cryptech, Inc.
508 Lafayette Street
Jamestown, New

ABSTRACT. *A new cryptosystem called REDOC II is introduced. Analysis of a miniature (one–round) version of this system is given.*

INTRODUCTION.

This report discusses the REDOC II cryptosystem developed by Michael C. Wood and the one round attack performed on the cryptosystem by Dr. Thomas Cusick.

REDOC II is a high–speed Shannon confusion diffusion arithmetic cryptosystem capable of enciphering 800 kilobits per second on a 20 MHz clock. The current implementation involves a 10–round procedure performed on a 10–byte (80 bit) data block.

The REDOC II cryptosystem possesses exceptional cryptographic strength. The most direct attack developed to date against a single round requires approximately 2^{30} operations for the original REDOC II implementation. This attack is discussed in full detail below.

Recently, design modifications have been made to the REDOC II cryptosystem. While these changes do not affect the encryption speed, the work factor for a single round attack appears to be greatly increased. Presently, an upper bound for the work factor for a single round modified REDOC II stands at 2^{44} operations.

As with the DES, the work factor for REDOC II appears to multiply with each successive round. Achieving such an unusually high work factor after one round gives strong evidence for extraordinary cryptographic strength to be possessed by the complete 10 round system.

With the promise of great strength and efficiency, REDOC II cryptosystems warrant further investigation.

BRIEF DESCRIPTION OF REDOC II.

Figure 1 summarizes the REDOC II system. As shown there, REDOC II is a block cipher which operates on ten–byte plaintext blocks. In Figure 1, R is the round number. Carrying out ten rounds (as shown in Figure 1) on a given plaintext block produces the ciphertext. The permutation, substitution, and enclave tables are all chosen prior to implementing the cryptosystem. The key table and mask table are generated after the cryptosystem is implemented.

The original REDOC II contains 128 permutation table entries [see Sample Table 1]. Each entry specifies a position change for each of the 10 bytes to be enciphered. For example, if perm 0 were performed on data block : **90 27 11 41 114 117 56 33 72 122** : the result would be : **90 117 72 122 56 27 11 33 41 114** : . Thus, with REDOC II, permutations are performed on the byte, rather than the bit, level.

Sixteen substitution table entries were included in the original REDOC II [see Sample Table 2]. Each entry specifies a new value for every possible value in the alphabet space.

The enclave function is a procedure unique to the REDOC family of cryptosystems. The original REDOC II contained 32 enclave table entries, each with four subsections (a, b, c, & d) [see Sample Table 3]. Each subsection specifies how the enclave function will be performed on a half block (five bytes). Subsections 'a' and 'b' alter the right half block while 'c' and 'd' direct the procedure for the left block.

Figure 2 illustrates how the enclave procedure transforms an entire 10 byte data block. First, the block is divided in half (left block and right block). The right block is then transformed by subsection 'a' of the enclave entry chosen. The right block is then transformed again by subsection 'b'. The resulting right block is then exclusive or–ed with the original left block to produce a new left block. This left block is then transformed by subsection 'c' , which in turn is altered again by subsection 'd' . The resulting left block is then exclusive or–ed with the right block to produce a new right block. The two blocks are then combined yielding the new 10 byte data block.

Figure 3 illustrates how each subsection functions to alter a half block. Each subsection consists of five rows and three columns. For encryption, the tables are read row by row from top to bottom. The first value in each row specifies the position of the byte to be changed. The second and third values specify the postions of the bytes used to transform the byte specified by the first value. Transformation is accomplished by replacing the first byte specified by the sum of the three bytes (mod 128). The original REDOC II cryptosystem only encrypted the first 7 bits of each byte. Thus the alphabet space for each byte ranges from 0 to 127. Therefore, all addition is performed by modulo 128.

After the permutation, substitution, and enclave tables are chosen, a key must be selected. The original key length for REDOC II was 10 bytes. However, since only the first seven bits were used, the effective key size was 70 bits. However, REDOC II can support a key size ranging from 70 to 17,920 bits. Current implementations use a 140 bit key.

INITIALIZATION

The REDOC II cryptosystem must be initialized every time a new key is selected. The initialization process consists of the creation of the key table and the Mask table.

In the original REDOC II, 128 ten byte keys are generated from the installed ten byte key [see Sample Table 4]. These keys are created from a series of one way functions performed on the installed and previously generated keys. Since the attack on a one round system does not rely on the key generation methodology, a detailed description is omitted from this paper.

The Mask table consists of 4 ten byte blocks [see Sample Table 5]. The Masks are generated by exclusive or–ing a large number of values from the key table. The first Mask is created from the first thirty–two keys in the key table. The second Mask is created from the next thirty–two, and so on.

The Masks are used to choose which entry in the function and key tables will be chosen. The first Mask is used to choose the permutation entry. The second Mask chooses the keys from the key table. Mask three selects an enclave entry, and Mask four selects a substitution entry.

Each of the ten bytes for every Mask corresponds to the round of encryption. The first byte of each Mask is used during round one. The second byte is used during round two. For each round, the corresponding byte is used.

After the key table and Mask table are created, encryption and decryption can begin.

REDOC II ENCRYPTION

Figure 1 contains an overview of the REDOC II encryption process. For each round, entries from the function and key tables are determined by the values of at least one byte in the internal ciphertext and one byte from the Mask table . For each round:

1. An entry from the permutation table is selected.
2. The internal ciphertext is permuted according to the selected entry.
3. A key is selected from the key table.
4. Each byte in the internal ciphertext is exclusive or–ed with the corresponding byte in the key except the byte used in selecting the key.
5. Another key is selected from the key table.
6. Each byte in the internal ciphertext is exclusive or–ed with the corresponding byte in the key except the byte used in selecting the key.
7. An enclave entry is selected.
8. The internal ciphertext is transformed by the selected enclave procedure.
9. An entry from the substitution table is selected.
10. Every byte in the internal ciphertext is substituted except the byte used in choosing the entry
11. Another entry from the substitution table is selected.
12. Every byte in the internal ciphertext is substituted except the byte used in choosing the entry.

NOMENCLATURE

A : Block 'A'

A_w = Byte 'w' of Block 'A' $\{0 \leq w \leq 10\}$

P_w : Permutation entry 'w'

P_w (A) = B : Perform Permutation 'w' on 'A' to produce 'B' $\{0 \leq w \leq 127\}$

K_w : Denotes KEY 'w' $\{0 \leq w \leq 127\}$

$K_{w,y}$: Denotes byte 'y' of Key 'w' e.g. fourth byte of Key 32 = $K_{32,4}$

$K_{w,y}$ xor A_y where $\{y=z$ and $0 \leq y \leq 10\}$ = B : XOR every byte in block 'A' with corresponding byte in Key 'w' except for byte 'z' to produce block 'B'.

E_w : Denotes Enclave entry 'w'

E_w (A) = B : Perform Enclave as directed by table 'w' on Block A producing output B

S_w : Denotes Substitution table 'w'

S_w (A_y) where $\{y=z$ and $0 \leq y \leq 10\}$ = B : Substitute the corresponding value in table 'w' for the corresponding byte value of every byte in the block except byte 'z' .

SUM (A) = w : Sum all the values of block 'A' (mod 128) to obtain value 'w' .

MATHEMATICAL DESCRIPTION OF ROUNDS

Input block 'A' for round 'r'

1. SUM (A) xor $Mask_{1,r}$ = w
2. P_w (A) = B
3. B_r xor $Mask_{2,r}$ = W
4. $K_{w,y}$ xor B_y where $\{y=r$ and $0 \leq y \leq 10\}$ = C
5. X = (r mod 10) + 1
6. C_x xor $Mask_{2,x}$ = W
7. $K_{w,y}$ xor C_y where $\{y=r$ and $0 \leq y \leq 10\}$ = D
8. D_r xor $Mask_{3,r}$ = W
9. E_w (D) = E
10. E_r xor $Mask_{4,r}$ = W

11. $S_w(E_y)$ where $\{y=z \text{ and } 0 \leq y \leq 10\} = F$
12. $(r \bmod 10) + 1 = x$
13. F_x xor $Mask_{4,x} = W$
14. $S_w(F_y)$ where $\{y=x \text{ and } 0 \leq y \leq 10\} = G$

Output Block 'G';

[Output Block 'G' becomes the Input Block 'A' of the next round.]

ATTACK ON ONE–ROUND VERSION OF REDOC II

For convenience of reference, we divide up the one–round REDOC algorithm into "stages", which we label as follows:

Stage 1 – Variable permutation
Stage 2 – First variable key add
Stage 3 – Second variable key add
Stage 4 – Enclave
Stage 5 – First variable substitution
Stage 6 – Second variable substitution

We let $B = (B_1, B_2, ..., B_{10})$ denote a typical plaintext block of ten 7–bit bytes B_i and we let $C = (C_1, C_2, ..., C_{10})$ denote a typical ciphertext block of ten 7–bit bytes C_i.

We begin our attack by assuming that we have enough plaintext blocks and corresponding ciphertext so that we can choose 256 plaintext blocks B_i $(1 \leq i \leq 256)$ with

(i) values of B_1 and B_2 identical in all 256 blocks

(ii) values of $B_1 + ... + B_{10}$ mod 128 give each of the 128 possibilities exactly twice.

The purpose of (i) is to guarantee that the two keys used in stages 2 and 3 of the REDOC encryption are the same for all 256 blocks B_i (by definition of the

REDOC algorithm). The purpose of (ii) is to guarantee that at least one of the variable permutations which keep B_1 and B_2 fixed is used twice in the set of 256 encryptions of blocks B_i. We assume for definiteness that there are exactly 2 permutations which fix B_1 and B_2, namely permutations #8 and #29. It will be clear from what follows that our attack does not depend on this specific feature of the particular permutation table which we happen to be using. Our use here of a particular property of permutations #8 and #29 merely makes it easier to describe the attack.

We let C_i $(1 \le i \le 256)$ denote the ciphertext blocks corresponding to the plaintext blocks B_i. We let

$$\mathbf{W}_j = (W_{j,1}, W_{j,2}, \ldots, W_{j,10}) \qquad (j = 1 \text{ or } 2)$$

denote the two keys which are used at stages 2 and 3, respectively, of the encryptions of the blocks B_i. *If* we have chosen a block B_i such that permutation #8 or #29 is used in stage 1 of the encryption, then the result at each of the first 3 stages is as follows:

(1)
$$\begin{aligned}
&\text{Stage 1:} && B_1, B_2, B_{3,p}, B_{4,p}, \ldots, B_{10,p} \qquad (p = 8 \text{ or } 29)\\
&\text{Stage 2:} && B_1, B_2 \oplus W_{1,2}, B_{3,p} \oplus W_{1,3}, \ldots, B_{10,p} \oplus W_{1,10}\\
&\text{Stage 3:} && B_1 \oplus W_{2,1}, B_2 \oplus W_{1,2}, B_{3,p} \oplus W_{1,3} \oplus W_{2,3}, \ldots, B_{10,p} \oplus W_{1,10} \oplus W_{2,10}
\end{aligned}$$

(Here $\oplus$ is component–wise addition mod 2, i.e. xor.) Here the value of p indicates which of the permutations #8 or #29 is actually used at stage 1.

The object of our attack is to locate, among the 256 blocks B_i which we shall denote by

$$\mathbf{B}_i = (B_1^{(i)}, B_2^{(i)}, \ldots, B_{10}^{(i)}) \qquad (1 \le i \le 256),$$

two blocks which encipher using the same key values $W_{1,2}$ and $W_{2,1}$ and the

same variable permutation (which will be #8 or #29). The attack proceeds by applying a process which we shall call "reversal" to each of the 256 blocks of ciphertext C_i. Reversal consists in taking a ciphertext block C_i after stage 6 of REDOC, then inverting the two variable substitutions of stages 5 and 6 in all $16^2 = 2^8 = 256$ possible ways, and finally applying to each of these blocks the inverse of each of the 32 possible enclaves of stage 4. Thus applying reversal to a block C_i gives $2^{13} = 8{,}192$ blocks which are candidates for the actual block which results at Stage 3 in (1) above when the corresponding plaintext block B_i is enciphered. Only the unique candidate which corresponds to the actual choice of the enclave and the two variable substitutions at Stages 4, 5 and 6 of the encryption is the correct one.

It is clear from the form of the block B_i after Stage 3 of the encryption process (see (1) above) that each candidate block arising from reversal of block C_i uniquely determines values for $W_{2,1}$ and $W_{1,2}$ (of course these values may be incorrect if the candidate is not the right one). Thus we can associate with each candidate block arising from reversal of C_i a triple $(W_{1,2}(i,j,n), W_{2,1}(i,j,n), n)$, where n $(1 \leq n \leq 32)$ is the number of the enclave whose inverse is used to obtain the candidate and j $(1 \leq j \leq 256)$ is an index which identifies the pair of substitutions whose inverses are used to obtain the candidate. The grouping of the triples according to the enclave used is important for the analysis which follows. Using the above notation for the triples, we can schematically show the reversal process on the blocks of ciphertext as follows:

$$
\begin{array}{ccccc}
(W_{1,2}(1,j,n)\,, W_{2,1}(1,j,n),\, n) & & & & (W_{1,2}(256,j\,,\, n)\,, W_{2,1}(256,j,n),\, n) \\
\uparrow & & \cdots & & \uparrow \\
C_1 & & & & C_{256}
\end{array}
\qquad (2)
$$

Given two identical triples in the list of $256 \times 2^{13} = 2^{21}$ triples generated by the reversal (2), we set the corresponding 20 bytes in the two blocks associated with the two triples equal to the respective 20 bytes which arise at stage 3 (see (1) above) in the encryption of the corresponding two plaintext blocks B_i. To do this

we must guess the needed two values of p in (1), which is possible in 4 different ways. Thus we obtain 4 systems of 20 linear equations in the 20 unknowns $W_{1,j}$ and $W_{2,j}$ $(1 \leq j \leq 10)$. Suppose that the two identical triples in (2) correspond to ciphertexts C_i and C_j, and suppose that the reversal process which leads to the two identical triples gives the blocks

$$(d_{i,1}, d_{i,2}, \ldots, d_{i,10})$$

and

$$(d_{j,1}, d_{j,2}, \ldots, d_{j,10}),$$

respectively. Now each of the 4 systems of 20 linear equations has the form

$$\begin{aligned}
B_1^{(i)} \oplus W_{2,1} &= d_{i,1} \\
B_2^{(i)} \oplus W_{1,2} &= d_{i,2} \\
B_{3,p}^{(i)} \oplus W_{1,3} \oplus W_{2,3} &= d_{i,3} \\
&\ldots\ldots\ldots\ldots \\
B_{10,p}^{(i)} \oplus W_{1,10} \oplus W_{2,10} &= d_{i,10} \\
B_1^{(j)} \oplus W_{2,1} &= d_{j,1} \\
B_2^{(j)} \oplus W_{1,2} &= d_{j,2} \\
B_{3,p}^{(j)} \oplus W_{1,3} \oplus W_{2,3} &= d_{j,3} \\
&\ldots\ldots\ldots\ldots \\
B_{10,p}^{(j)} \oplus W_{1,10} \oplus W_{2,10} &= d_{j,10}
\end{aligned} \tag{3}$$

We shall analyze a typical system of equations (3) later, but first we show that we can reduce the number of candidate permutation pairs to 2 instead of 4 for each pair of identical triples in (2) and also reduce the number of identical triples that need to be considered, by a closer analysis. In particular, suppose we have two identical triples $(W_{1,2}, W_{2,1}, n)$ corresponding to ciphertext blocks C_i and C_j. If the encipherments of the corresponding plaintext blocks B_i and B_j actually both use permutation #8 or both use permutation #29 at stage 1, then we have

$$(4) \qquad B_1^{(i)} + \ldots + B_{10}^{(i)} \equiv B_1^{(j)} + \ldots + B_{10}^{(j)} \bmod 128$$

If, on the other hand, when B_i and B_j are enciphered permutation #8 is used at stage 1 for one of the encipherments and permutation #29 is used for the other, then we have

$$(5) \qquad 8 \oplus 29 = (B_1^{(i)} + \ldots + B_{10}^{(i)} \bmod 128) \oplus (B_1^{(j)} + \ldots + B_{10}^{(j)} \bmod 128) = 21 .$$

Thus a pair of identical triples in (2) can only lead to a genuine system of equations (3) if either (4) or (5) is true; this will allow us to eliminate some pairs of identical triples from further consideration. If (4) is true for a given pair of identical triples, then we need only consider the cases where $p = 8$ for both of the corresponding encipherments or where $p = 29$ for both of the encipherments. Similarly, if (5) is true for the given pair of triples, we need only consider the two cases where $p = 8$ for one of the encipherments and $p = 29$ for the other. Thus each pair of identical triples in (2) leads to at most two systems of equations (3).

Now we need a count of the number of pairs of identical triples in (2). For each fixed n, there are $256 \times 2^8 = 2^{16}$ triples in (2) but there are only 2^{14} distinct triples for any given n, so there will be many duplicated triples. However, the maximum possible number of pairs of distinct plaintext blocks B_i and B_j which satisfy one of the necessary conditions (4) or (5) is 768. This follows since, for each B_i $(1 \le i \le 256)$, if we have

$$(6) \qquad B_1^{(i)} + \ldots + B_{10}^{(i)} \equiv m \bmod 128 \quad (0 \le m \le 127)$$

then there is a unique B_j such that (6) holds with $i = j$ and the same m; and there is a unique pair B_h, B_k such that (5) holds with j equal to h or k. (Here we are using our initial assumption (ii) about the 256 plaintext blocks B_i.) Now any of the 6 possible pairs chosen from the set $\{B_h, B_i, B_j, B_k\}$ will satisfy either (4) or (5) and this gives ≤ 768 distinct pairs as i varies, $1 \leq i \leq 256$.

Let us consider one of the 768 pairs – say B_i and B_j – of the plaintext blocks for which (4) or (5) is satisfied. In the corresponding system of equations (3), we must have $B_1^{(i)} = B_1^{(j)}$ and $B_2^{(i)} = B_2^{(j)}$ by our assumption (i), so the system is inconsistent unless we also have $d_{i,1} = d_{j,1}$ and $d_{i,2} = d_{j,2}$. If the pairs $d_{j,1}$, $d_{j,2}$ were randomly distributed in our 768 pairs, the probability of matching the pair $d_{i,1}, d_{i,2}$ would be $768/128^2$, i.e. we would not expect to find any matches.

However, by our assumptions (i) and (ii) we know that exactly 4 of the encryptions of our 256 blocks B use either permutation #8 or permutation #29. Suppose the set $\{B_h, B_i, B_j, B_k\}$ of the previous paragraph corresponds to these blocks B. Then by the discussion in the previous paragraph this set gives 6 pairs of distinct blocks B which must give consistent systems of equations of form (3). Thus it is very likely that exactly 6 of the 768 pairs give consistent systems of equations of form (3), and these systems will involve just 4 of the 256 blocks B. Thus we have determined the values of

(7) $$W_{1,2}, W_{2,1}, W_{1,3} \oplus W_{2,3}, W_{1,4} \oplus W_{2,4}, \cdots, W_{1,10} \oplus W_{2,10}.$$

By the definition of REDOC decryption, the number of the variable substitution at stage 6 is determined by the ciphertext and the mask byte $\text{Mask}_{4,1}$; the number of the variable substitution at stage 5 is determined by the ciphertext and $\text{Mask}_{4,1}$; and the number of the enclave at stage 4 is simply $\text{Mask}_{3,1}$. Since we know the enclave number and the variable substitution numbers corresponding to the systems (3) which determine the numbers in (7), we can also determine

(8) $$\text{Mask}_{3,1} \text{ and } \text{Mask}_{4,1}.$$

Since the number of the variable permutation at stage 1 is given by

$$(B_1 + \ldots + B_{10} \bmod 128) \oplus \text{Mask}_{1,1},$$

we can also determine $\text{Mask}_{1,1}$ from the known fact that the encryptions of the blocks B_h, B_i, B_j, B_k above use either permutation #8 or permutation #29.

An upper bound for the number of operations needed to reach this point is $2^{29} \approx 5 \times 10^8$. We arrive at this number as follows: We estimate that 150 is an upper bound for the number of operations needed to carry out an inverse enclave and two inverse substitutions. Thus at most $150 \times 2^{21} < 3.2 \times 10^8$ operations are needed to produce the list of 2^{21} triples needed for the reversals in (2). By our work above, we must examine at most 768 ciphertext pairs C_i, C_j in order to find identical triples in the list of 2^{21} triples. For each such ciphertext pair, we must look for matches in 32 pairs of sets of 2^8 triples; these 32 pairs arise because we need only try to match triples with the same enclave number in (2), and there are 2^{13} triples associated with each C_i in (2). We can estimate that checking one such pair of sets takes $\leq 2^{12}$ operations. Thus checking all 32 pairs of sets for the ≤ 768 ciphertext pairs can be done in $\leq 768 \times 2^{17} \approx 10^8$ operations. Finally, testing for consistency of the systems (3) will require few operations. This gives our bound of 2^{29} operations.

We note that 2^{29} operations could be done in about 30 seconds on an inexpensive 20 megahertz personal computer. The attack to this point could be done in parallel with many processors (see (2) for a natural use of 256 processors) and so could be greatly speeded up by a large investment in hardware.

Stage 1 – Variable permutation number is $(B_1 + \ldots + B_{10} \bmod 128) \oplus \text{Mask}_{1,1}$

Stage 2 – Key number is $B_1 \oplus \text{Mask}_{2,1}$

Stage 3 – Key number is $B_2 \oplus \text{Mask}_{2,1}$

Stage 4 – Enclave number is $\text{Mask}_{3,1}$

Stage 5 – Variable substitution number is (Byte 1 $\oplus$ $\text{Mask}_{4,1}$) mod 32

Stage 6 – Variable substitution number is (Byte 2 $\oplus$ $\text{Mask}_{4,1}$) mod 32

Table 1. Use of mask values in one–round REDOC

Table 1 shows that, since we know the mask values in (8), we can apply the reversal process to any ciphertext block $(C_1, ..., C_{10})$ and obtain the numerical values of the ten bytes in the stage 3 block of the encipherment of the corresponding plaintext block $(B_1, ..., B_{10})$. Since we also know $Mask_{1,1}$, the block at stage 3 will have the form (see (1) above)

$$(9)\quad B_{p(1)} \oplus W_{j,1},\ B_{p(2)} \oplus W_{i,2},\ B_{p(3)} \oplus W_{i,3} \oplus W_{j,3},\ \cdots,\ B_{p(10)} \oplus W_{i,10} \oplus W_{j,10}$$

where $(p(1), ..., p(10))$ is a certain permutation of $(1, ..., 10)$, and i and j are the numbers of the keys used at stages 2 and 3, respectively. Here and in all that follows we slightly change our earlier notation and let

$$\mathbf{W}_j = (W_{j,1}, W_{j,2}, \cdots, W_{j,10}) \quad (0 \le j \le 127)$$

denote the j–th key in the REDOC key table. Thus when we convert (7) to our new notation, we see that our work above has determined the values of

$$(10)\qquad W_{j,1},\ W_{i,2},\ W_{i,3} \oplus W_{j,3},\ \cdots,\ W_{i,10} \oplus W_{j,10}$$

with $i = I$ and $j = J$, where I and J are certain key numbers which we do not know (of course we do know that

$$(11)\qquad I = B_1 \oplus Mask_{2,1} \text{ and } J = B_2 \oplus Mask_{2,1},$$

where B_1 and B_2 are the special values from the assumption (i) made at the beginning of our attack).

It is evident from (9) that if we know the key numbers i and j being used, and if we know the values for the ten bytes in (10) above, then we can immediately recover the plaintext block $(B_1, ..., B_{10})$ from knowledge of the values of the 10 bytes in (9), as follows: We xor the known values in (10) with the 10 bytes in (9) and thus recover the values of $B_{p(1)}, ..., B_{p(10)}$. Then we compute the number

$$(B_{p(1)} + \cdots + B_{p(10)} \bmod 128) \oplus Mask_{1,1}$$

of the permutation $(p(1), ..., p(10))$ (since we know the value of $Mask_{1,1}$); now by consulting the REDOC permutation table we invert this permutation and recover $B_1, ..., B_{10}$.

It is now clear that any ciphertext block whose corresponding plaintext happens to begin with the same bytes B_1 and B_2 that appear in (11) can be immediately decrypted by the above procedure. Furthermore, if we can produce a table of values of the 10 bytes in (10) for a large number of pairs (i, j) of key numbers, then we can very quickly decrypt a large number of blocks of ciphertext. We simply apply the reversal process to the ciphertext blocks and so obtain the values of the bytes in (9). Then we xor the bytes in (9) with values of the bytes in (10) for various pairs (i, j). Each xor gives a candidate for the bytes $B_{p(1)}, ..., B_{p(10)}$ and also determines the permutation number in stage 1, so we can immediately tell if we have sensible plaintext. If we do not get sensible plaintext, we continue trying entries from the sum table until the plaintext emerges or until we run out of pairs (i,j) to try; the larger our table is, the less likely it is that the latter possibility occurs. Even if the latter possibility does occur, we are defeated only for the single block under consideration.

Since most of the bytes in (10) are sums of key bytes rather than actual key bytes, let us call a table of values of the 10 bytes in (10) for a large number of pairs (i, j) of key numbers, a *sum table*. Using plaintext blocks **B** and corresponding ciphertext blocks **C**, we can rapidly produce a large sum table, as follows: We apply the reversal process to all of the ciphertext blocks in our list of known plaintext and ciphertext. Each plaintext block

$$B^{(1)} = (B_1^{(1)}, B_2^{(1)}, ..., B_{10}^{(1)})$$

will give values of the 10 bytes in (10) for key numbers

$$(12) \qquad i = B_1^{(1)} \oplus Mask_{2,1} \text{ and } j = B_2^{(1)} \oplus Mask_{2,1}.$$

It is easy to make a large sum table even from modest amounts of known plaintext and ciphertext because, for example, if we know the values for $W_{i,3} \oplus W_{j,3}$ and $W_{i,3} \oplus W_{k,3}$, then we immediately have the value of $W_{j,3} \oplus W_{k,3}$ by xor.

We do not know the value of $Mask_{2,1}$, but we do not need this in order to produce our sum table, or to use the sum table in decrypting REDOC ciphertext. We simply arrange the entries in the sum table according to the blocks $B_1^{(1)}$ and $B_2^{(1)}$ instead of the actual key numbers (12).

The creation of a large sum table in the last part of the attack certainly takes less time than the first part of the attack in which the mask values in (8) were found. A generous upper bound on the extra time involved is to say that the total time is doubled. It is clear that once the sum table has been created, decryption of given ciphertext can be done very rapidly (i.e., the vast majority of the effort in the attack is setting up the sum table).

It is interesting to note that in attacking the one–round REDOC we do ***not*** need to reconstruct the key table, and indeed we do not need to find any of the keys in the key table; we only need a large sum table as described above. It follows that complicating the process by which the key table is generated will not strengthen one–round REDOC.

Sample Table 1
Permutation Table

Original	=	1	2	3	4	5	6	7	8	9	10
Perm 0	=	1	6	7	9	10	2	5	8	3	4
Perm 1	=	10	4	8	3	1	7	2	9	5	6
Perm 2	=	1	6	4	9	8	5	10	2	3	7
Perm 3	=	9	8	3	4	5	10	6	1	7	2
. . .											
Perm 86	=	9	7	2	6	5	8	3	10	1	4
Perm 87	=	5	3	8	1	9	7	10	2	4	6
. . .											
Perm 126	=	9	8	3	7	1	10	5	6	2	4
Perm 127	=	7	8	5	10	9	3	4	2	1	6

Sample Table 2
Substitution Table

Original Value	Sub 0	Sub 1	Sub 4	Sub 10	Sub 14	Sub 15
0	90	47	25	66	73	0
1	46	89	51	13	36	52
2	66	87	103	31	107	44
3	21	20	116	7	43	83
. . .	. . .	. . .	. . .	. . .	. . .	. . .
126	24	14	105	114	77	6
127	122	62	11	63	49	79

Sample Table 3
Enclave Table

	a	b	c	d
Entry 0:	5 2 3	3 5 2	5 4 2	5 4 2
	4 3 1	1 3 5	4 3 1	2 5 1
	2 5 4	2 4 1	1 5 3	1 3 5
	1 4 5	5 1 4	3 2 5	3 2 4
	3 1 2	4 2 3	2 1 4	4 1 3
Entry 1:	3 1 2	3 2 5	4 2 1	4 2 3
	4 3 1	5 1 4	3 4 5	5 3 1
	2 5 4	2 4 3	5 1 4	2 1 5
	5 2 3	4 3 1	1 3 2	3 5 4
	1 4 5	1 5 2	2 5 3	1 4 2
...	...	...	...	...
Entry 31:	2 4 1	2 4 3	1 5 3	4 1 5
	3 5 4	4 1 2	2 4 1	3 5 2
	5 1 3	3 5 4	4 3 2	1 4 3
	1 2 5	5 2 1	5 2 4	2 3 4
	4 3 2	1 3 5	3 1 5	5 2 1

Sample Table 4
Key Table

KEY 0	=	0	34	5	63	9	73	74	107	109	33
KEY 1	=	10	62	48	85	32	101	8	0	63	56
KEY 2	=	26	59	75	97	33	80	8	6	73	26
. . .											
KEY 107	=	36	123	45	10	55	59	109	45	98	24
. . .											
KEY 118	=	95	25	48	47	1	20	117	55	19	67
. . .											
KEY 126	=	62	110	70	27	124	31	119	97	9	2
KEY 127	=	11	54	25	87	107	73	4	118	62	34

Sample Table 5
Mask Table

Mask 1	=	48	2	121	18	60	105	33	50	11	60
Mask 2	=	26	78	24	72	69	13	77	43	9	99
Mask 3	=	64	113	72	61	37	13	49	71	24	60
Mask 4	=	104	62	69	87	18	31	102	101	32	125

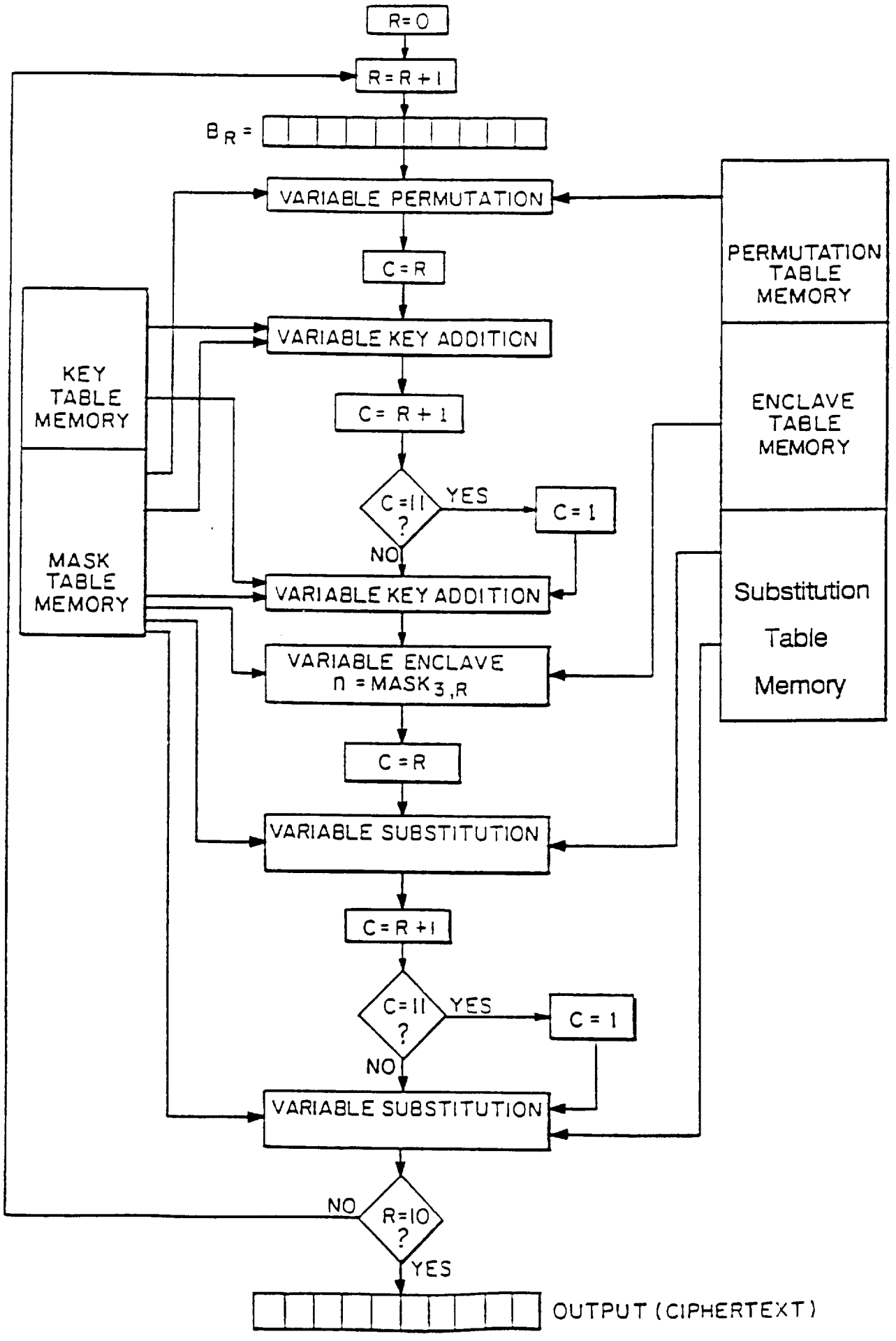

Figure 1

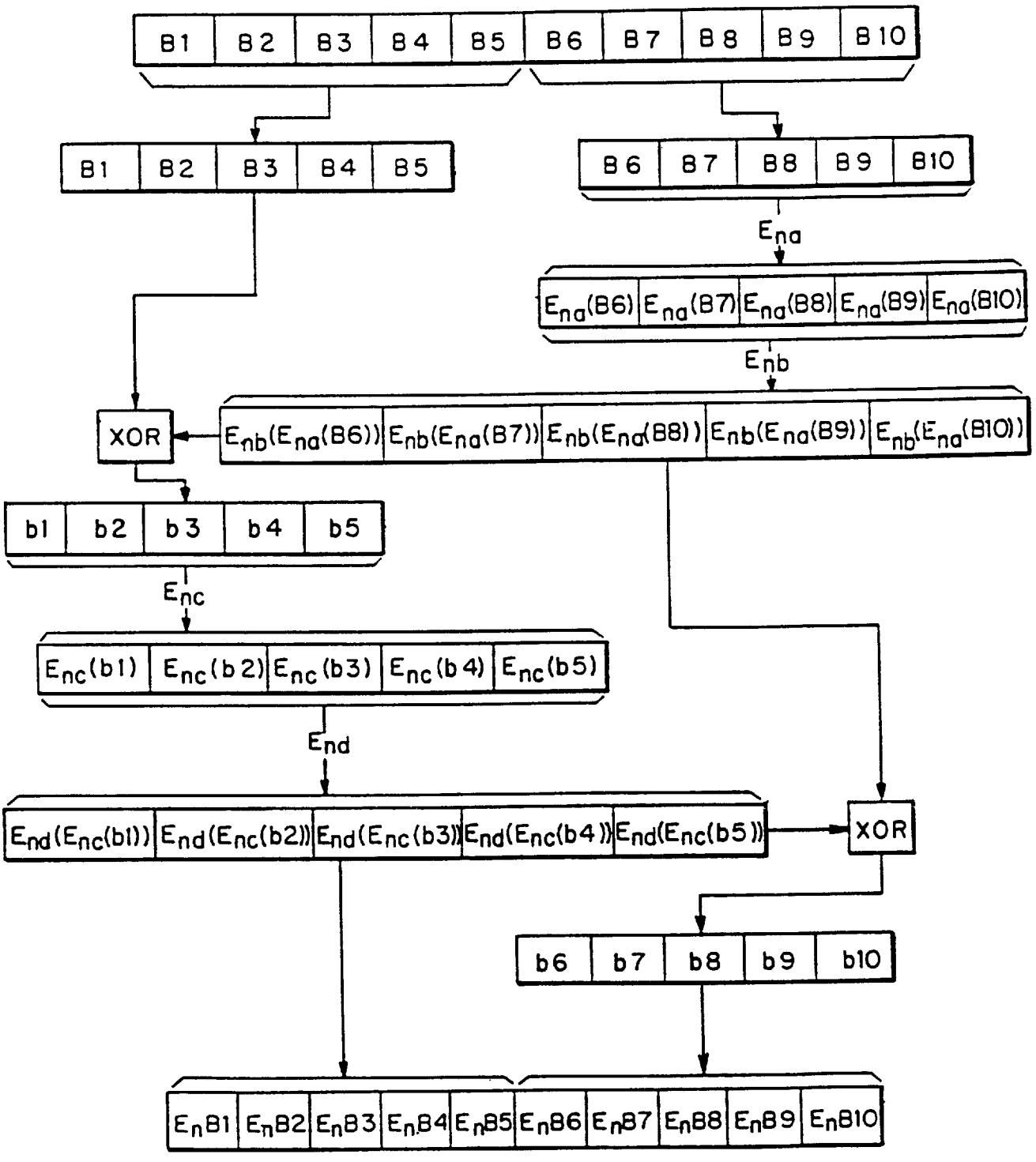

Figure 2

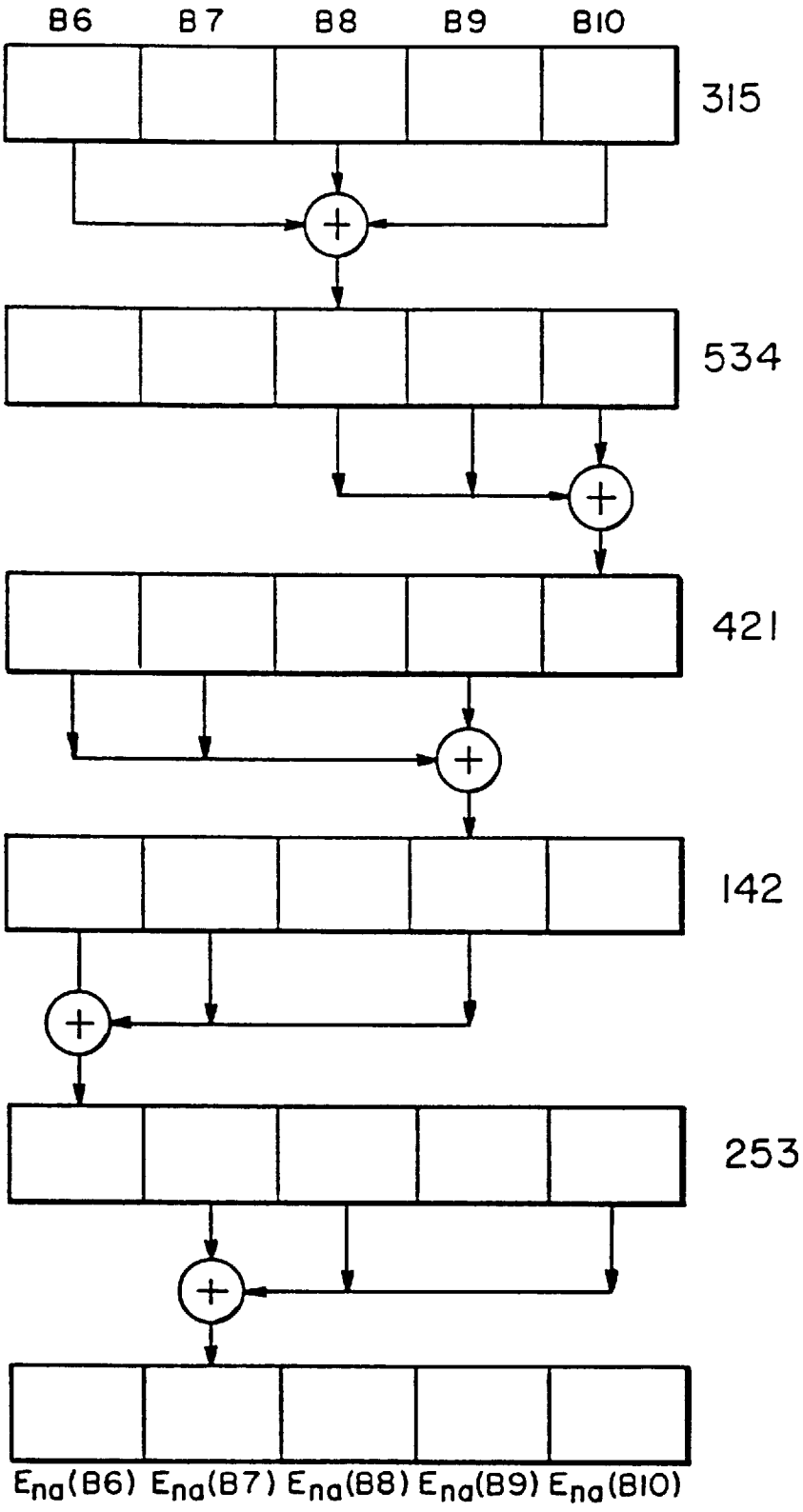

Figure 3