IBM Research

THE DESIGN OF LUCIFER, A CRYPTOGRAPHIC DEVICE FOR DATA COMMUNICATIONS

J. L. Smith

April 15, 1971

RC 3326

RC 3326 J. L. Smith April 15, 1971

THE DESIGN OF LUCIFER, A CRYPTOPGRAPHIC DEVICE FOR DATA COMMUNICATIONS

NOTICE OF DECLASSIFICATION

This Research Report may be released for unrestricted distribution as of February 2, 1972. Please attach this notice to your copy of the Report.

Phyllis G. Stigall
Research Publications

Yorktown Heights, New York

San Jose, California

Zurich, Switzerland

THE DESIGN OF LUCIFER, A CRYPTOGRAPHIC DEVICE FOR DATA COMMUNICATIONS

J. L. Smith
IBM Thomas J. Watson Research Center
Yorktown Heights, New York

ABSTRACT: Lucifer embodies a block-cipher cryptographic system by which a data stream of any length is enciphered (or deciphered) on-line in groups of 16 bytes under control of a 128-bit cipher key. The system includes a method of message verification in which successive byte groups are linked together by redundant fields under cipher protection; validity is checked by the matching of these redundant fields after decipherment. Cipher keys, which may be private to each user, are entered either from a plug-in read-only-store, or from a magnetic-stripe card.

The organization of Lucifer is serial-parallel, one byte wide, and processing of a 16-character group requires 200 microseconds. The design includes the interface to a 2770 Data Communications System.

RC 3326 (#15211)
April 15, 1971

[redacted]

[redacted]

TABLE OF CONTENTS

1. INTRODUCTION

The design of Lucifer, a cryptographic device for data communications, incorporates many of the notions discussed by Feistel,[1] and is a distillation of those features, some with modifications introduced to permit a simple implementation. The objective was to create a hardware device of considerable cryptographic strength,* capable of being (ultimately) realized in LSI technology at a cost small compared with the cost of a terminal to which it might be attached. Because of the nature of the cryptographic system, ciphertexts generated by Lucifer resemble random bit strings, and consequently all possible bit combinations can occur. For this reason, a terminal having "transparency" capability was needed for Lucifer to work with so that the communication subsystem would be transparent to fortuitous line-control characters. Of the terminals having this feature, the 2770 Data Communications System (EBCDIC) was chosen because of its flexibility in the attachment of a variety of input-output devices.

The 2770-Lucifer combination is being used as part of an on-going experiment in system security in which the terminal can be in communication with a data processor via the public dial-up telephone system. An analogous cryptographic function for the data processor is provided by a software program. For the conduct of the experiment, there are programs which establish

---

* Work in cryptanalysis of the cipher system upon which the design of Lucifer is based is being carried out under the direction of A. J. Hoffman in the Mathematical Sciences Department of IBM Research.

an interactive mode of operation between the 2770 and the data processor, with a limited file-access and update capability.

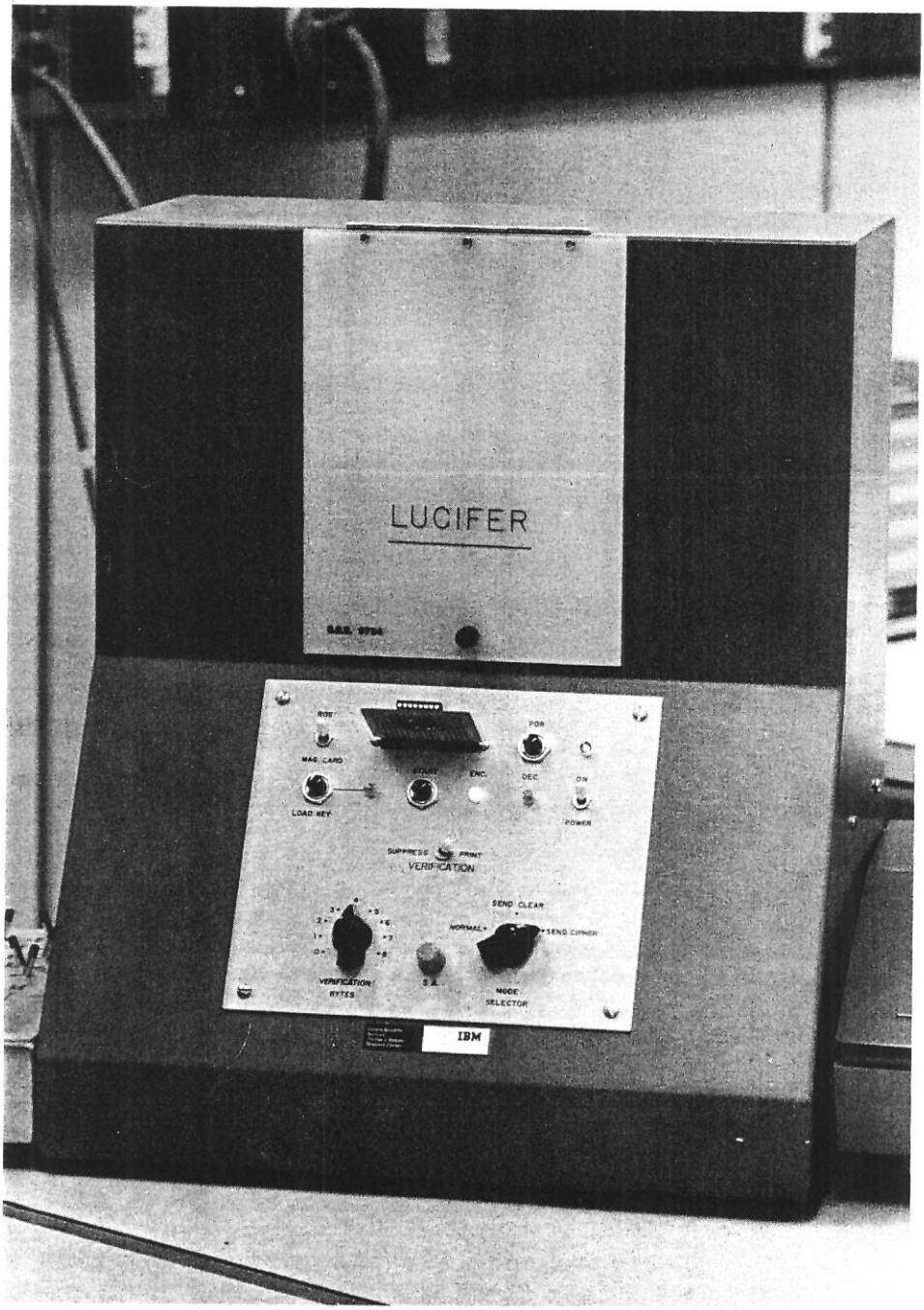
A discussion of the experiment as such and of the software aspects is beyond the scope of this report.

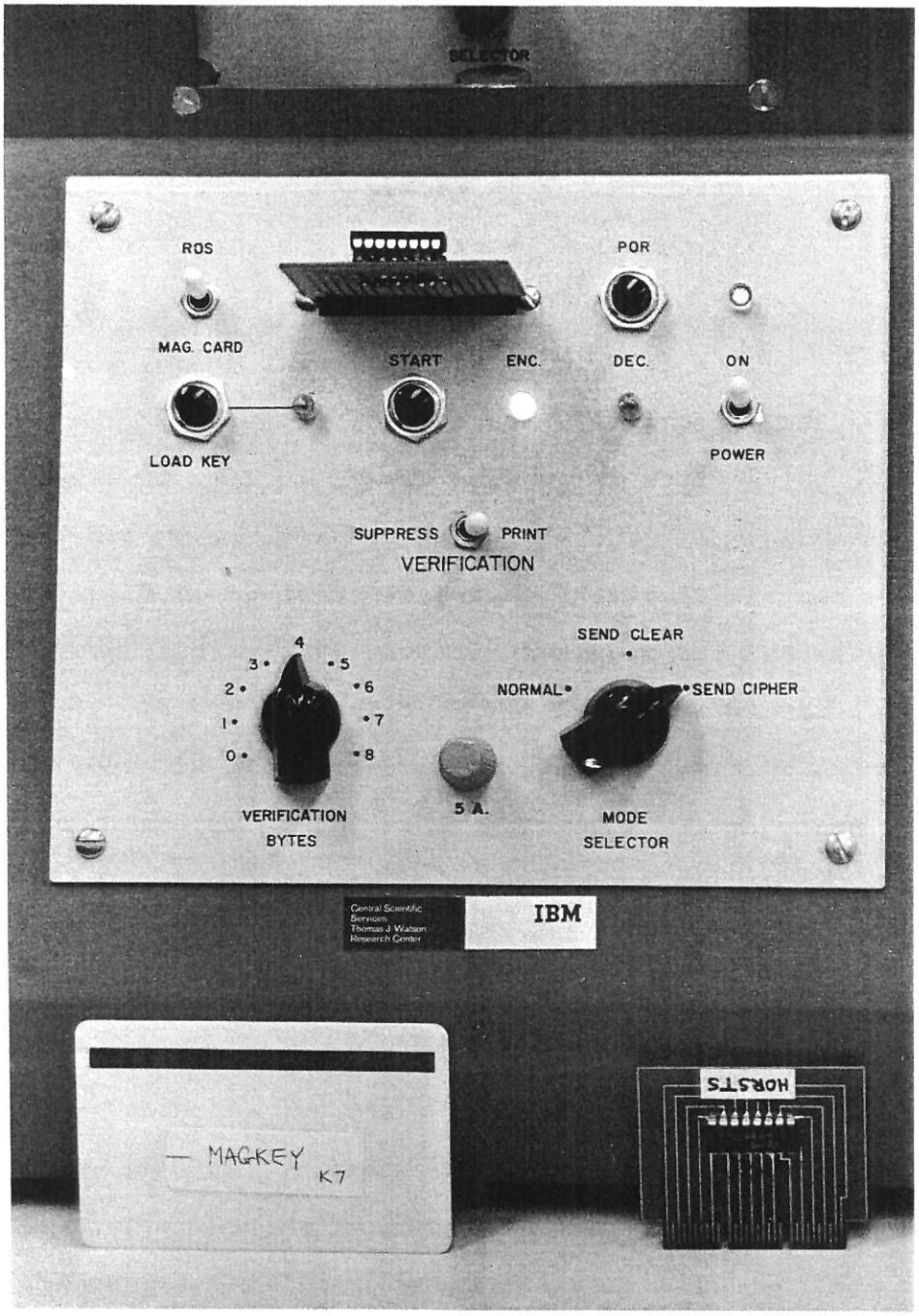
For the experiment, a pre-production EBCDIC model of the 2770 with keyboard and matrix printer (2213 model 1) was furnished on loan by SDD Raleigh. It has been determined that the data-processing speed of Lucifer (16 characters in 200 microseconds) is quite adequate, and in no way affects the throughput of the 2770 operating at 2000 bits per second. It could, in fact, with no modification, work with a new 2770 (having a maximum buffer-transfer rate twice as great) operating at speeds up to 9600 bits per second. With additional buffering, Lucifer could be made to operate at a line speed of 500,000 bits per second.

2. PHYSICAL DESCRIPTION

Lucifer is housed in a metal cabinet which is mounted atop the 2770 Control Unit, located for the convenience of the operator, and connection to the 2770 is by three ribbon cables. The logic circuits are mounted on a rear hinged panel for easy access. Plates 1, 2, and 3 show the 2770 and Lucifer in different aspects.

All the controls and indicator lamps are located on two front panels. The upper one, which is the "CE" Panel, is intended to be used only for testing and will be described in Section 12. The lower one, which is the Operator's Panel, contains all the controls ever needed for ordinary operation.


LUCIFER
MAG. CARD
LOAD KEY
ENC.
DEC.
ON
POWER
SUPPRESS
PRINT
VERIFICATION
SEND CLEAR
NORMAL
SEND CIPHER
VERIFICATION BYTES
S.A.
MODE SELECTOR
IBM


SELECTOR
ROS
MAG. CARD
LOAD KEY
START
ENC.
POR
DEC.
ON
POWER
SUPPRESS PRINT
VERIFICATION
4
3
5
2
6
1
7
0
8
VERIFICATION BYTES
5 A.
SEND CLEAR
NORMAL
SEND CIPHER
MODE SELECTOR
Central Scientific Services Thomas J. Watson Research Center
IBM
— MAGKEY K7
HORSTS

[redacted]

Features provided on the Operator's Panel are:

(1) An On-off power switch and indicator lamp.

(2) A Power-on Reset (POR) push button, whose function is to clear all storage registers and initialize all flip-flops, leaving Lucifer in the quiescent phase.

(3) A Mode selector switch having three positions: Normal, Send Clear, and Send Cipher. When this switch is on Normal and Lucifer is in the quiescent phase, Lucifer is disabled and the 2770 operates in its normal fashion. When this switch is on Send Clear, Lucifer is enabled for receiving only and, while all transmissions from the 2770 are sent in cleartext, messages received in ciphertext are deciphered but messages received in cleartext are unaltered. (Lucifer is activated for deciphering when the 2770 receives a "transparent" message.) When the Mode selector switch is on Send Cipher, Lucifer is enabled for transmitting as well, and all messages from the 2770 are in (transparent) ciphertext, with automatic activation for received ciphertext messages as before.

(4) Two state-indicator lamps, Encipher and Decipher. Lucifer is normally in the encipher state and switches to decipher when the 2770 is receiving a message.

(5) A cipher-key selector switch by which the operator can choose one of two alternative means of providing the cipher key to control the enciphering and deciphering. When the switch is on

ROS, the cipher key is gotten directly from a read-only store plugged into a receptacle on the operator's panel; when the switch is on Mag Card, the key is from internal key registers which have been preloaded from a magnetic card. Controls for this function are also on the operator's panel, and a description of the procedure is given in Section 7.

(6) A rotary switch for specifying the number of redundancy bytes to be used for the verification of contextual coherence in exchanges of messages, and a switch to Print or Suppress these verification bytes on an output device. These matters are explained in Section 8.

3. THE CIPHER SYSTEM

Lucifer embodies a block-cipher system in which a message of any length is processed in groups of 16 eight-bit characters (128 bits) to produce substitute groups also of 128 bits. (An incomplete last message group is padded with null characters to make 16.) From one viewpoint, this is a simple-substitution cipher for an "alphabet" consisting of 2^{128} (about 10^{39}) "characters" of 128 bits each. There are, of course, $2^{128}!$ possible ways of specifying substitutions within this "alphabet." The substitutions actually produced are generated systematically under the control of a cipher key which also consists of 128 bits. Hence, of all the possible ways of substituting, Lucifer is limited to (only) 2^{128} ways, and, of course, at any one time the substitutions are done in one particular way determined by the bit pattern of the cipher

key, which may be private to each user.

The method of generating the 128-bit substitute group (cryptogram) for any group of 128 message bits is basically nonlinear in nature, and it has a fundamental characteristic that every one of the resultant 128 bits is obtained as a rather complicated function of every one of the original 128 message bits and of every one of the 128 bits of the key. A change of a single bit in either message or key potentially affects every bit of the cryptogram, each with a seemingly independent probability very close to one-half. This is illustrated in the appendix.

The cryptographic strength of this cipher system need not depend on secrecy of any organizational details of either the abstract system or the hardware, but only on secrecy of the cipher-key bit pattern in use. The system itself, based on the work of Feistel, is a product cipher (meaning that the result of one operation is further treated by a second operation, then by a third, etc.) employing keyed nonlinear byte transformations and a mixing transformation (confusions and diffusions) with many entrances of the key, as suggested by Shannon.[2]

The system will be treated here as an abstraction, with descriptions of the general principles that are involved, without burdening the reader with unnecessary details. It is helpful to understand the abstract system before attempting to understand any hardware implementation. In Section 4 will be described the organization of Lucifer, and explanations will be offered of how

the various functions of the cipher system are performed in specific ways by various portions of the hardware. At that juncture, the reader will be prepared for a precise statement of the Lucifer algorithm (in Section 5), and a treatment of the key accessing regimen (in Section 6).

Let there be given a group of 16 message characters (bytes), and a cipher key consisting of 128 arbitrarily chosen bits. The objective is to transform the message into a cryptogram of 16 bytes under control of the cipher key, in such a way that it is possible to transform the cryptogram back again to the original message, using the same cipher key.

For the following description, it may be helpful to refer to Fig. 1. The message is divided into halves, shown at top and bottom in the figure. The functions in the system are:

(1) Confusion: Eight bits of the key are selected, to correspond to the eight bytes in the top half of the message (the source bytes). According as each of these key bits is a 0 or a 1, one of two distinct nonlinear transformations is performed using a copy of each corresponding source byte as argument. The source bytes themselves are retained unaltered. The eight bytes resulting from the keyed nonlinear transformations contain the elements of confusion, and are referred to as "confused" bytes.

(2) Key Interruption: Sixty-four selected bits of the key are combined with the eight "confused" bytes by mod-2 addition. This function is so called because it throws up a barrier and, by its presence, effectively interrupts the activities of a cryptanalyst.

[redacted]

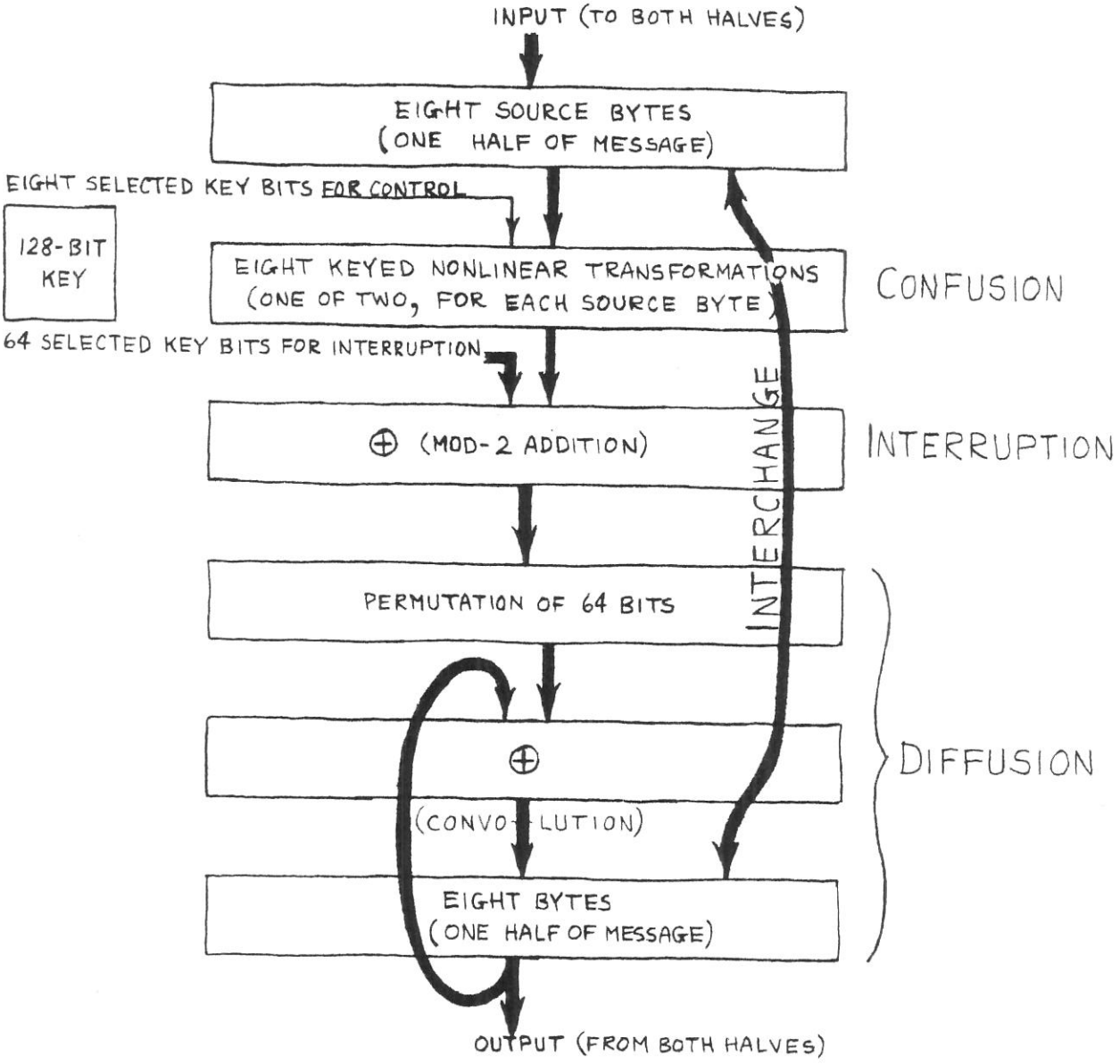


FIG. 1. FUNCTIONAL BLOCK DIAGRAM OF THE CIPHER SYSTEM

(3) Diffusion: The 64 bits of the mod-2 sum so obtained are permuted in some quasi-random fashion, and the resultant 64 bits are convolved with the eight bytes in the bottom half of the message by mod-2 addition. This permutation and convolution together make up the diffusion function, by which the totality of confusion from the one half, plus key, is diffused throughout the other half.

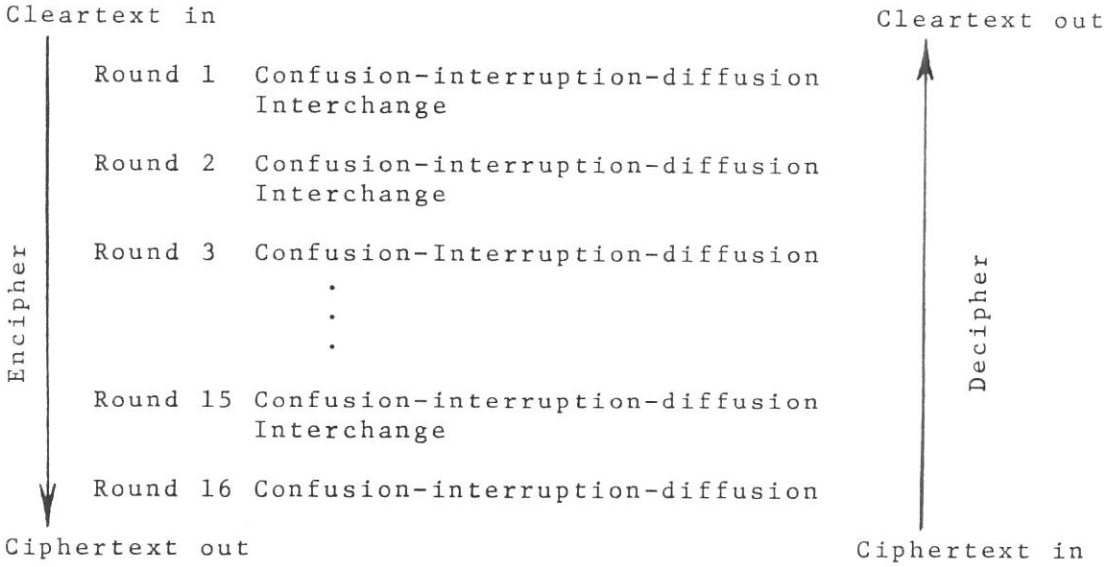
(4) Interchange: The top and bottom halves are interchanged, so that the previous source bytes are in the bottom half and the modified bytes are in the top half.

The same functions are performed again, but this time with different selected key bits. By this action, new confusion is generated from the present top half (already containing confusion) and, with key interruption, is diffused into the present bottom half. Succeeding confusion-interruption-diffusions, each preceded by an interchange of the two halves, cause the obfuscation to be compounded and recompounded in both halves of the message. After a total of 16 such confusion-interruption-diffusion functions have been performed, separated by 15 interchanges of the two halves, the 16 bytes then present constitute the cryptogram for the original 16 message bytes.

In performing these 16 rounds, the selection of the key bits (to be treated in detail in Section 6) is such that each of the 128 bits is used nine times--once as the control bit to govern one byte transformation for confusion, and eight times as a key-

interruption bit in eight different bit positions. For every possible message group, there is a uniquely corresponding cryptogram group, and the correspondences are determined by the cipher key.

It is necessary that the source bytes be retained unaltered during any confusion-interruption-diffusion operation so that the entire process can be reversed for deciphering. Because mod-2 addition and subtraction are identical, each act of convolving can be undone by redoing it, using the same source bytes and the same key bits for transformation control and interruption. The only requirement is that the order be reversed--that is, the last confusion-interruption-diffusion done for enciphering must be done first for deciphering, etc.--under control of the same cipher key working backwards. This is illustrated in the following chart.



4. LOGICAL ORGANIZATION OF LUCIFER

The data-flow portion of Lucifer (that is, those parts other than process-control and interface logic) is serial-parallel, one byte wide, utilizing shift-register storage. This choice was made for a number of reasons.

(1) The amount of hardware required for data processing (but not storage) is one-eighth of the amount required in a parallel organization.

(2) The schedule for accessing the key bits can be arranged so that it is unnecessary to shift the key backwards (or to execute long forward shifts) in order to decipher.

(3) A one-byte-wide organization is easier to implement in LSI technology because of the inherently fewer interconnections among the component parts.

This organization is a good choice insofar as the 2770 is concerned because internally the 2770 data flow is also one byte wide, and no special buffering is required to mate the data buses. The processing speed afforded by this choice is quite adequate for the data rates of present-day terminals.

Figure 2 shows all the data-flow paths along which the data in the shift registers circulate and flow from one place to another. Each shift register is composed of a series of flip-flops, the number of which is indicated by a numeral within each rectangle in the figure.

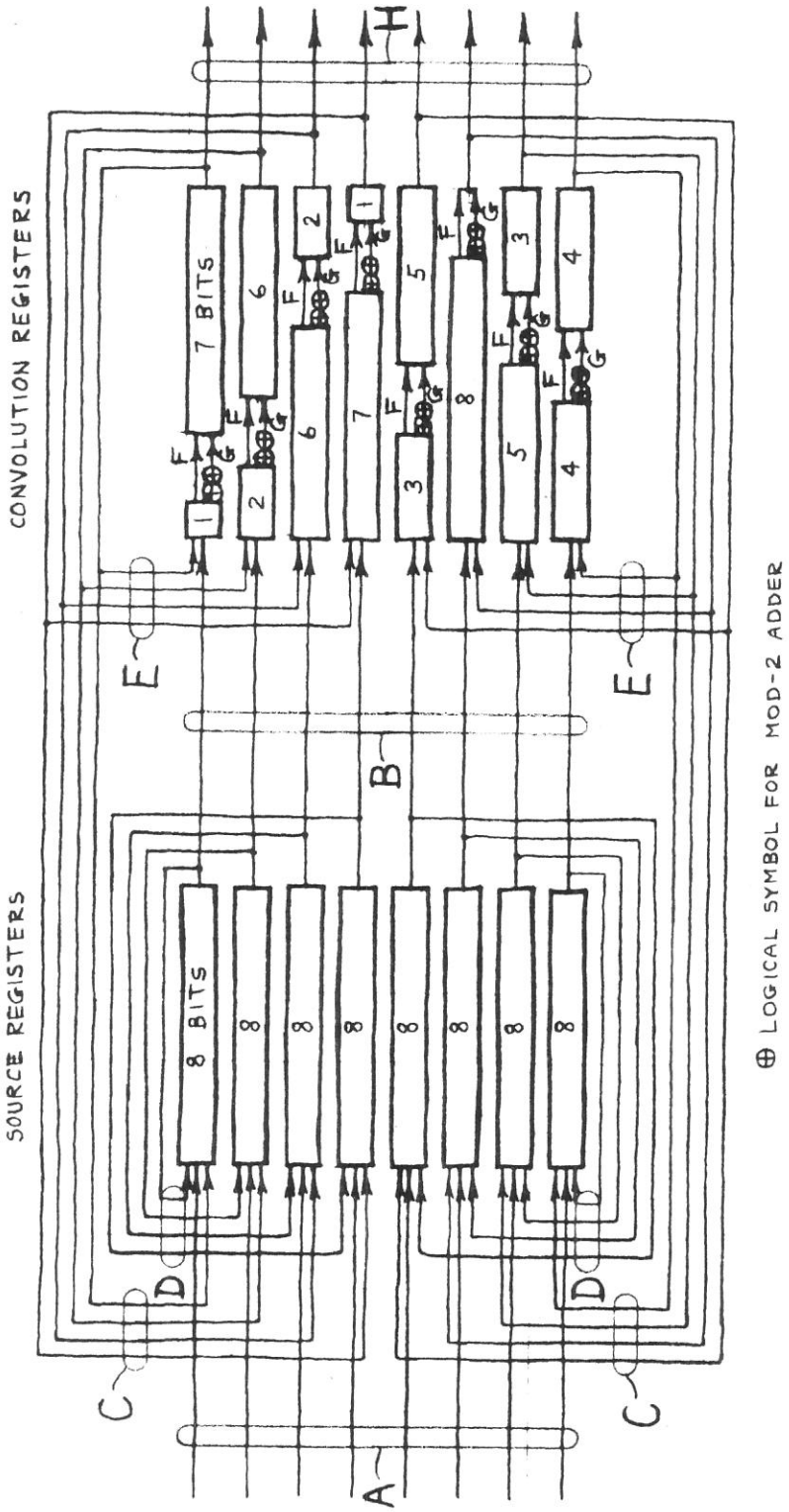


FIG.2. LUCIFER DATA FLOW— REGISTER CIRCULATION

A message is loaded into the registers by applying each byte in turn to signal lines A, with only data-transfer lines B and F enabled, and causing all the data registers to shift one position for each byte. After 16 such, eight bytes will have been stored in the convolution registers and eight in the source registers. The eight bits of any byte are arrayed vertically on the diagram, with each bit contained within a different register.

During the ciphering operation, there are two essentially different patterns that the data-flow paths assume. While confusion-interruption-diffusion is taking place, only lines D, E, and G are enabled, and each shift register recirculates upon itself for eight shifts. While the interchange is taking place, only lines B, C, and F are enabled, and after eight shifts the respective contents of the source registers and convolution registers will have been exchanged.

Output of processed data is on lines H, with only lines B, C, and F enabled; after 16 shifts, all 16 bytes will have been made available at H, and all bytes will have been retained in the registers.

Figure 3 shows the structures involved in performing the confusion, interruption and diffusion functions. The recirculation paths of the data registers shown in Fig. 2 are not repeated here. The key registers, to be dealt with in more detail in Section 6 together with the key ROS, are recirculating shift registers. The choice of key source (ROS or shift registers) is under manual control.

[redacted]

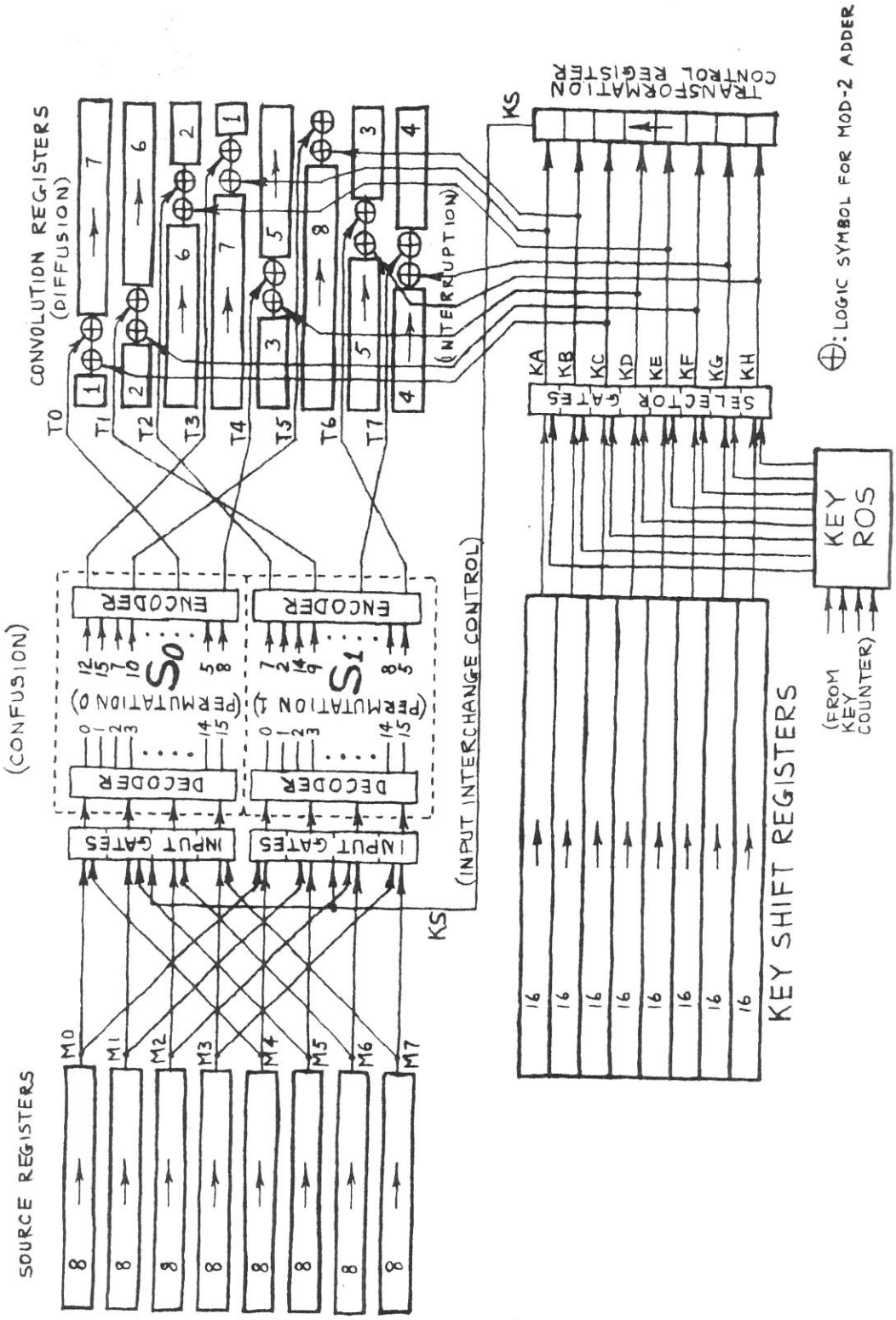


FIG.3. LUCIFER DATA FLOW— CONFUSION-INTERRUPTION-DIFFUSION

The remainder of this section shows how the functions for the cipher system previously described are performed in Lucifer hardware. Hereafter, "confusion-interruption-diffusion" will sometimes be referred to as "c-i-d".

Confusion (Refer to Figure 3) A byte copied from the source registers, M0...M7, is transformed into a byte T0...T7 under control of a key bit KS in the following manner. According as KS is a 1 or a 0, the source bits M0, M1, M2, M3 are respectively interchanged with bits M4, M5, M6, M7 or not. The resulting two sets of four bits are applied as inputs to two nonlinear transformation networks S0 and S1 within the dotted-line enclosures in the figure. Each of S0 and S1 consists of a four-bit decoder, which produces a signal on one of 16 lines according to the bit pattern of the inputs; these 16 lines are permuted before being applied as inputs to a four-bit encoder, which produces a unique bit pattern on its four output lines according to which of its 16 inputs carries a signal. The permutations of these 16 lines in each of S0 and S1 are different, and are chosen so that S0 and S1 produce two different nonlinear transformations on four bits. The eight outputs of S0 and S1 are permuted as indicated in the figure and the results are the bits T0...T7.

The two transformations S0 and S1 are given in Table 1, and in Table 2 is the complete set of byte transformations of M to T for all 256 possibilities. The transformation is designated as T(0) when the control bit is a 0 and T(1) when it is a 1. For convenience, the eight-bit bytes have been converted to decimal.

TABLE 1. FOUR-BIT TRANSFORMATIONS

ARG	*S0*	*S1*
0000	1100	0111
0001	1111	0010
0010	0111	1110
0011	1010	1001
0100	1110	0011
0101	1101	1011
0110	1011	0000
0111	0000	0100
1000	0010	1100
1001	0110	1101
1010	0011	0001
1011	0001	1010
1100	1001	0110
1101	0100	1111
1110	0101	1000
1111	1000	0101

TABLE 2. BYTE TRANSFORMATIONS

M	T(0)	T(1)	M	T(0)	T(1)	M	T(0)	T(1)	M	T(0)	T(1)
0	87	87	32	207	117	64	215	23	96	219	20
1	21	223	33	141	253	65	149	159	97	153	156
2	117	207	34	237	237	66	245	143	98	249	140
3	54	211	35	174	241	67	182	147	99	186	144
4	23	215	36	143	245	68	151	151	100	155	148
5	55	95	37	175	125	69	183	31	101	187	28
6	20	219	38	140	249	70	148	155	102	152	152
7	84	67	39	204	97	71	212	3	103	216	0
8	116	195	40	236	225	72	244	131	104	248	128
9	118	199	41	238	229	73	246	135	105	250	132
10	22	203	42	142	233	74	150	139	106	154	136
11	53	75	43	173	105	75	181	11	107	185	8
12	85	91	44	205	121	76	213	27	108	217	24
13	119	71	45	239	101	77	247	7	109	251	4
14	52	79	46	172	109	78	180	15	110	184	12
15	86	83	47	206	113	79	214	19	111	218	16
16	223	21	48	211	54	80	95	55	112	67	84
17	157	157	49	145	190	81	29	191	113	1	220
18	253	141	50	241	174	82	125	175	114	97	204
19	190	145	51	178	178	83	62	179	115	34	208
20	159	149	52	147	182	84	31	183	116	3	212
21	191	29	53	179	62	85	63	63	117	35	92
22	156	153	54	144	186	86	28	187	118	0	216
23	220	1	55	208	34	87	92	35	119	64	64
24	252	129	56	240	162	88	124	163	120	96	192
25	254	133	57	242	166	89	126	167	121	98	196
26	158	137	58	146	170	90	30	171	122	2	200
27	189	9	59	177	42	91	61	43	123	33	72
28	221	25	60	209	58	92	93	59	124	65	88
29	255	5	61	243	38	93	127	39	125	99	68
30	188	13	62	176	46	94	60	47	126	32	76
31	222	17	63	210	50	95	94	51	127	66	80

TABLE 2 (CONTINUED)

M	T(0)	T(1)	M	T(0)	T(1)	M	T(0)	T(1)	M	T(0)	T(1)
128	195	116	160	203	22	192	91	85	224	79	52
129	129	252	161	137	158	193	25	221	225	13	188
130	225	236	162	233	142	194	121	205	226	109	172
131	162	240	163	170	146	195	58	209	227	46	176
132	131	244	164	139	150	196	27	213	228	15	180
133	163	124	165	171	30	197	59	93	229	47	60
134	128	248	166	136	154	198	24	217	230	12	184
135	192	96	167	200	2	199	88	65	231	76	32
136	224	224	168	232	130	200	120	193	232	108	160
137	226	228	169	234	134	201	122	197	233	110	164
138	130	232	170	138	138	202	26	201	234	14	168
139	161	104	171	169	10	203	57	73	235	45	40
140	193	120	172	201	26	204	89	89	236	77	56
141	227	100	173	235	6	205	123	69	237	111	36
142	160	108	174	168	14	206	56	77	238	44	44
143	194	112	175	202	18	207	90	81	239	78	48
144	199	118	176	75	53	208	71	119	240	83	86
145	133	254	177	9	189	209	5	255	241	17	222
146	229	238	178	105	173	210	101	239	242	113	206
147	166	242	179	42	177	211	38	243	243	50	210
148	135	246	180	11	181	212	7	247	244	19	214
149	167	126	181	43	61	213	39	127	245	51	94
150	132	250	182	8	185	214	4	251	246	16	218
151	196	98	183	72	33	215	68	99	247	80	66
152	228	226	184	104	161	216	100	227	248	112	194
153	230	230	185	106	165	217	102	231	249	114	198
154	134	234	186	10	169	218	6	235	250	18	202
155	165	106	187	41	41	219	37	107	251	49	74
156	197	122	188	73	57	220	69	123	252	81	90
157	231	102	189	107	37	221	103	103	253	115	70
158	164	110	190	40	45	222	36	111	254	48	78
159	198	114	191	74	49	223	70	115	255	82	82

It is clear that this method of doing keyed byte transformations for confusion is not the most general one that could be conceived; to implement the most general method would be a prodigal use of hardware for a dubious benefit. It is believed that this method is a satisfactory compromise between hardware and utility as it produces one of two nonlinear transformations on M which are different for all arguments except for 16 anticipated instances of symmetry.

Since Lucifer has a serial-parallel organization, in each c-i-d cycle the eight bytes from the source registers are transformed sequentially by the single network just described, under the control of eight key bits utilized sequentially. At the beginning of a c-i-d cycle, eight key bits are stored in the transformation control register (see the figure); this register, shifting concomitantly with the data registers, furnishes these bits one at a time for controlling the transformation of each succeeding byte from the source registers.

Key interruption: Eight key bits, KA...KH, are brought simultaneously to the mod-2 adders in the convolution registers, where they are to be added in along with the confused byte, T0...T7. (The fact that these key bits are not added to the T-bits first is immaterial in this particular arrangement.) The key registers also shift concomitantly with the data registers, so that for each succeeding confused byte that is produced there is furnished a succeeding set of eight key bits; in all, 64 key bits are delivered in this manner in each c-i-d cycle.

Diffusion: The construction of the convolution registers in Lucifer is special; in each there is inserted a pair of mod-2 adders between two bit-storage cells. See Figure 3. When the data-flow paths containing these adders are enabled, the act of convolving a T bit and a key bit with one of the registers consists of a shift; after each shift, the storage cell to the right of an adder pair will contain the mod-2 sum of three bits: the bit that had been stored in the cell to the left of the adders, a key bit, and a T bit.

The permutation function, which is a vital part of the diffusion process, is accomplished simply by the placement of the pairs of mod-2 adders within the registers so that there is an adder pair at each of the eight byte positions (the eight bits of any byte are arrayed vertically in the figure). By this simple means, every single confused byte and key-interruption byte will potentially affect one bit of all eight bytes in the convolution registers after the shift, and after the eight shifts of the source registers and convolution registers required to perform each single c-i-d cycle, all 64 bits in the convolution registers will have been potentially affected after each cycle.

It should be obvious that the permutation performed in this manner is systematic and not "random". Randomness per se is not required; what is required is only that each "confused" byte and key-interruption byte be scattered and dispersed in some way throughout the convolution registers.

The alternation of confusion-interruption-diffusion and interchanges described in the previous section is achieved by

appropriately enabling and disabling the data transfer lines already described, with continuous shifting of the data registers. This is described in detail in the next section.

5. THE LUCIFER ALGORITHM (Refer to Figs. 3 and 4)

It is assumed that the message group (for an encipher operation) or the cipher group (for a decipher operation) has already been loaded into the data registers, and that the cipher key has already been stored in the key shift registers or else a key ROS has been plugged in. It is further assumed that the cipher key is in the correct position order for beginning an operation--that is, if this is an encipher operation, the key must be in "normal" shift position or, if this is a decipher operation, the key must have been moved half-way around. This matter, as well as the details of key-bit accessing, is explained in the following section. The use of an eight-bit control counter to manage the sequencing is implicit, and there is a four-bit key counter to furnish the addresses to access the key when a ROS is used.

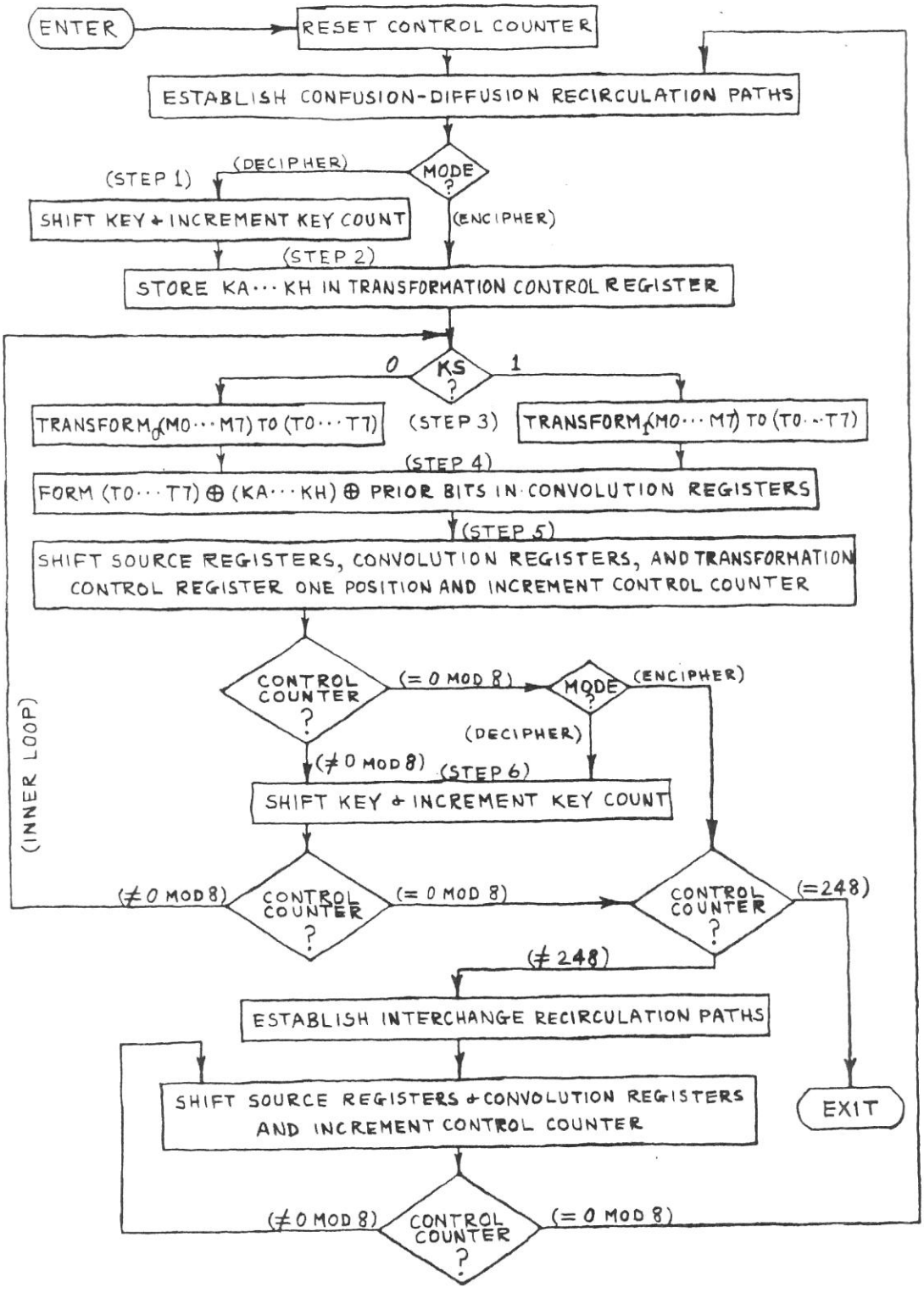


FIG. 4. LUCIFER CIPHER ALGORITHM

The Confusion-Interruption-Diffusion Cycle for One Round
(recirculation on lines D, E, G)

Step 1. (Executed only once each cycle) If this is a decipher operation, the key registers are shifted one position and the key counter is incremented by one.

Step 2. (Executed only once each cycle) The outputs of the key selector gates KA...KH are stored in the transformation control shift register (serializer) whose single output is KS.

Step 3. The outputs of the source registers M0...M7 are transformed into bits T0...T7 according to the value of KS (see Table 2).

Step 4. Eight mod-2 sums are formed of eight triplets of bits: the convolution-register bits at the left of the adders, the corresponding bits T0...T7, and the corresponding key bits KA...KH (see Fig. 3).

Step 5. The source registers, the convolution registers, and the transformation control register are all shifted one position. (After the shift, there are new M bits and a new KS bit; the mod-2 sums are stored in the register cells at the right of the adders.)

Step 6. The key registers are shifted one position and the key counter is incremented by one. (There is now a new set of key bits KA...KH). N.B. This step is omitted for an encipher operation if it has already been executed seven times in this current c-i-d cycle.

The inner loop, consisting of Steps 3, 4, 5, and 6, is repeated for a total of eight times, with the important exception noted in Step 6 for encipher.* This completes the cycle for one round.

The Interchange Cycle (recirculation on Lines B, C, F)-- The contents of the source registers and convolution registers are interchanged by the performing of eight shifts.

The c-i-d cycle and the interchange cycle repeat in alternation for a total of 16 c-i-d cycles and 15 interchange cycles. After the 16th c-i-d cycle, the enciphering or deciphering operation is complete. The time required is for 248 shifts.

Figure 4 is a flow chart for the Lucifer algorithm with indication of the manner of sequencing the various steps.

6. KEY-BYTE ACCESSING ORDER

An important aspect of the Lucifer design is the order of accessing the bits of the key. This is arranged so that, whether the operation is for encipher or decipher, the key shifting and counting are always in the forward direction. Moreover, since the key registers are all 16 bits long and since there are just 16 iterations of the c-i-d cycle (with a fixed number of key shifts and counts in each), at the conclusion of any operation on one 16-byte group the key is bound to be in the correct position order

---

* Note that for encipher, the key shift-and-count occurs seven times in each c-i-d cycle, while for decipher it occurs nine times.

for beginning the same operation on the next 16-byte group. The only long key shifts and counts that ever occur are for an encipher/decipher change, corresponding to a transmit/receive change at the terminal.

In Fig. 5a are shown the key registers with indication of the method of loading in which the registers are serially connected, and of interest is the facility for forming the mod-2 sum of two or more keys by repeated loading. After the serial loading, the registers are serial-parallel connected and the key is accessed eight bits at a time (a key byte). The numerals atop Fig. 5a refer not to the bit positions but are labels for the key bytes themselves. As the key counter advances for each shift, the label of the byte being accessed is indicated by the value of the count. This counter also furnishes the four-bit addresses for a 16-byte ROS so that the byte-accessing schedule is the same whether the key is from the shift registers or the ROS.

At the time when there is a transmit/receive change at the terminal, it is necessary for Lucifer to do an encipher/decipher state change. It will be seen shortly (Fig. 5b) that, while every encipher operation begins with the use of key byte 0, every decipher operation begins with the use of key byte 9. Therefore, when Lucifer is in its quiescent phase, a key shift-and-count of eight will occur for state changes, conditioned by one of the following:

1. If there is to be an <u>encipher</u> operation and if the key count is <u>not 0</u>, the key registers are shifted and the counter is incremented until the count is 0.

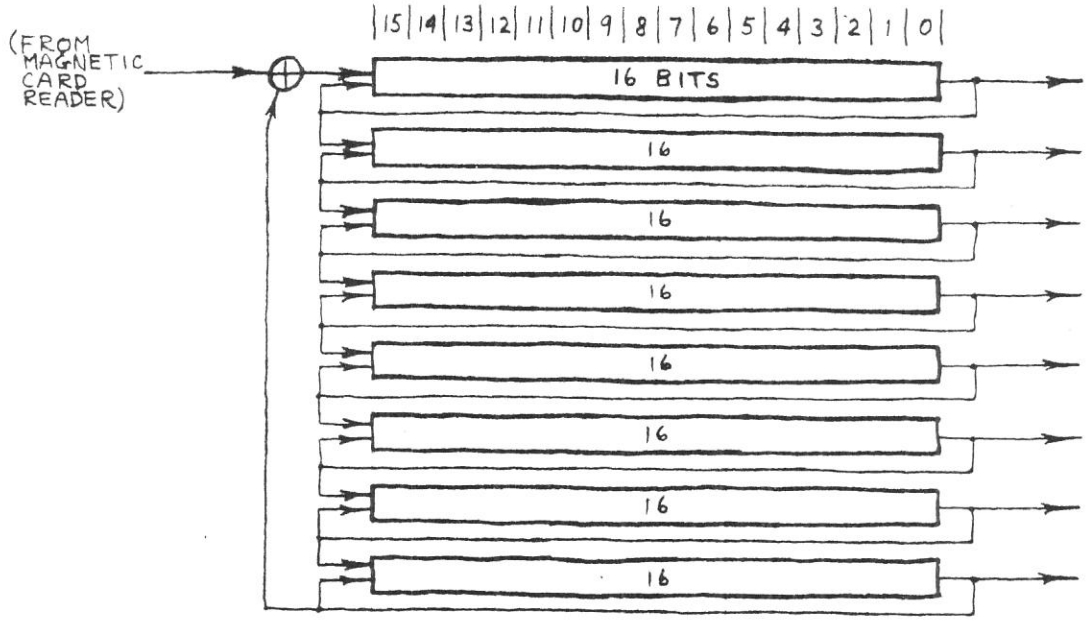


FIG. 5A. KEY SHIFT REGISTERS

ENCIPHER →									
ROUND 1: ↓	0	1	2	3	4	5	6	7
2:	7	8	9	10	11	12	13	14
3:	14	15	0	1	2	3	4	5
4:	5	6	7	8	9	10	11	12
5:	12	13	14	15	0	1	2	3
6:	3	4	5	6	7	8	9	10
7:	10	11	12	13	14	15	0	1
8:	1	2	3	4	5	6	7	8
9:	8	9	10	11	12	13	14	15
10:	15	0	1	2	3	4	5	6
11:	6	7	8	9	10	11	12	13
12:	13	14	15	0	1	2	3	4
13:	4	5	6	7	8	9	10	11
14:	11	12	13	14	15	0	1	2
15:	2	3	4	5	6	7	8	9
16: ↑	9	10	11	12	13	14	15	0
DECIPHER →								

FIG. 5B. KEY-BYTE ACCESSING SCHEDULE

2. If there is to be a decipher operation and if the key count is not 8, the key registers are shifted and the counter is incremented until the count is 8. (Recall that for decipher, Step 1 in confusion-interruption-diffusion is a single key shift-and-count so that key byte 9 is actually the first one used.)

In Fig. 5b is shown the schedule for accessing the key bytes during the series of 16 rounds of c-i-d and interchange cycles, one row for each round. The entries are the key-byte labels. For any particular c-i-d cycle, the corresponding row in the schedule has this interpretation: the key byte whose label is at the extreme left within the enclosure is the one stored in the transformation control register whence each bit in succession controls a transformation; all eight bytes in succession, from left to right, are accessed and are added into the convolution registers for key interruption by the successive eight executions of Steps 4 and 5 of the algorithm.

For the encipher operation, the schedule is read starting at the top from left to right and down, as in the ordinary reading scan (byte 0 taken first). For the decipher operation, the rounds are performed in the reverse order and the schedule is read starting at the bottom from left to right and up (byte 9 taken first).

IBM Research

THE DESIGN OF LUCIFER,
A CRYPTOGRAPHIC DEVICE
FOR DATA COMMUNICATIONS

J. L. Smith

April 15, 1971

RC 3326

Yorktown Heights, New York

San Jose, California

Zurich, Switzerland

As was noted in the description of the algorithm for the c-i-d cycle, seven key shifts and counts occur in each cycle for encipher, but nine for decipher. This is reflected in the schedule, as each entry there is the mod-16 sum of 7 and the entry above, or the sum of 9 and the one below.

As the rounds performed in a ciphering operation are distinguished by the key bytes that are accessed for each round, it is clear that, according to the schedule of key-byte accessing, the order of performing the c-i-d cycles for deciphering is the reverse of that for enciphering. But it is also clear that within each cycle the eight executions of the inner loop in the algorithm are done in the same order. The important concept here is that each execution of the inner loop is really independent of all the others and with proper selections of the key bits they could be done in any arbitrary order. For example, with reverse shifting in all the registers and with suitable data-flow modifications in the convolution registers, the initial c-i-d cycle for deciphering would be performed thus: store key byte 9 in the transformation control register and use those bits in the reverse order; for the key interruption access the key bites in the order 0 15 14 13 12 11 10 9. This would produce the exact same result as is produced by the process as stated using forward shifts, but it requires a complicated shifting regime and complicated switching and is clearly impractical.

The desiderata in a key-byte accessing schedule for Lucifer for one complete cipher operation (16 rounds) include:

(1) Each key bit is to be used once for controlling a byte transformation for confusion.

(2) Each key bit is to be used eight times for interruption, as early as possible the first time and regularly thereafter, each time in a different place in the developing cryptogram.

(3) Distinctions in the algorithm between Encipher and Decipher are to be simple and easy of implementation.

(4) Reverse (or long) register shifts are to be avoided ("random" key-byte selection is useless).

Schedules other than the one in Fig. 5b can readily be devised that provide some of the above, but none other has been found that provides all to the same degree.

7. KEY LOADING

As mentioned previously, there are two methods of providing the cipher key for Lucifer. One is by a pluggable 16-byte read-only-store mounted on a thin printed-circuit card easily carried on one's person. When in use, this key would simply be plugged into a receptable on the operator's panel with the key selector on ROS. There is no internal storage required for the ROS key because the key bytes are repetitively accessed directly from the ROS as needed. The other method is by magnetic recording on a wallet-sized card which would be read by a card reader such as the 2960, with storage of the bit pattern in Lucifer's key shift registers.

If the cipher key to be used is stored on a magnetic card, loading of the key via the 2960 card reader consists of the

following steps. (See Plate 3).

(1) Move the key selector switch to Mag Card.

(2) Push the POR button to erase any previous bit pattern that might exist in the key shift registers and to enable key loading.

(3) Push the Load Key button. The adjacent indicator lamp turns on.

(4) Insert the key card in the 2960 carrier, move the carrier all the way to the left and release. The card is read on the return stroke when it is moved past the reading head by the card carrier which is spring-loaded and governor-controlled. The outputs of the 2960 that are used are a data bit line and a synchronizing signal which controls the shifting and counting for the key shift registers as the key bits are stored. After the 128th bit has been read, the indicator lamp goes out, and the loading is finished.

(5) In order to enable Lucifer for operating, push the Start button. However, if it is desired, before pushing the Start button, another card can be loaded by repeating Steps 3 and 4. This causes the mod-2 sum of the previous key-register contents and the key currently being read to be stored in the shift registers, and as many keys as desired may be so entered. By this means, the actual cipher key in use need not be stored on any single magnetic card, and there is additional protection against unwarranted disclosure of the key through loss or theft or other malfeasance.

Special means are required for the original storing of key-bit patterns for both ROS keys and magnetic-card keys. The ROS keys could be endowed with "personality" after manufacture, as by a technique of junction break-down or "fuse-blowing" by controlled current pulses. The development of such devices is well advanced in the Components Division of IBM under W. Shutler, and advertisements of byte-wide programmable ROS modules by other manufacturers have appeared. For experimental purposes, the ROS keys actually used are standard off-the-shelf TTL code converter modules connected in different ways; this fact in no way affects the validity of the experiment. Parenthetically, no difficulty has been experienced with electrical contacts between card and receptable, which are dual-redundant. For greater reliability, these contacts should perhaps be quadruple-redundant.

If magnetic cards are used, the keys have to be recorded by a machine such as the 2954 in order that the cards be compatible with the 2960.

8. MESSAGE INTEGRITY

In an environment characterized by the existence of data banks comprising sensitive confidential files whose accessing and updating are managed by a data processor responsive to commands from a collection of remote terminals, there are attractive opportunities for sophisticated mischiefmakers who would profit from their mischief. It would be difficult to foresee all the ways that tampering might be done. Provided that the cipher keys belonging to legitimate

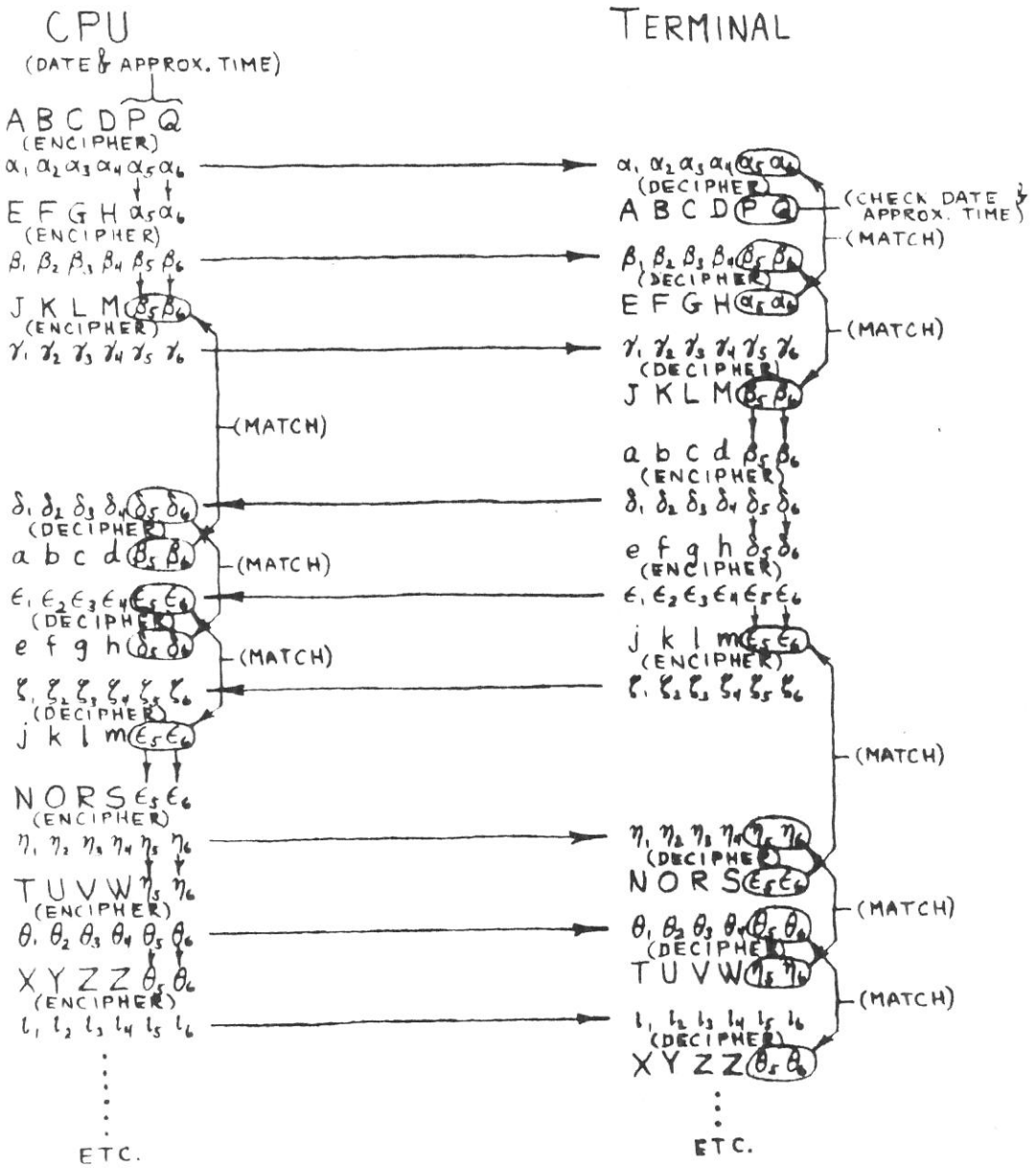
users (and the copies required by the data processor) are sufficiently well guarded, it is believed that the cryptographic system embodied in Lucifer can well enough prevent messages from being read by interlopers and spurious messages of impostors from being intelligible. The system cannot protect legitimate messages from being destroyed by interference with the transmitted ciphertext because even one wrong bit in a cipher group makes the entire deciphered group chaotic. That being the case, the system offers almost infallible indication that interference is occurring, over and above the excellent error-detection feature of the EBCDIC 2770 (two cyclic redundancy check bytes for every transmission block).

Beyond this, the system provides a way of ensuring message integrity, or message coherence. This is a technique specially adapted for use with Lucifer which can practically guarantee that each character group received is in true context with the preceding one. Moreover, messages having identical information content will be enciphered differently each time.

These benefits are provided by a method of character-group chaining, in which successive groups are linked together by a number of bytes from one cipher group included as part of the succeeding cleartext and consequently diffused throughout the next cipher group. Although continuous chaining can take place throughout an interchange of messages, and although progressive superenciphering occurs, chaos caused by an error does not propagate beyond the group in which the error occurs. A brief description of the technique follows.

Each group of 16 characters to be enciphered for transmission consists of two fields: an information field and a verification field. (The length of the verification field is variable from zero to a maximum of eight characters.) For message transmission, enough information characters are taken to fill the information field. The verification field comprises the characters from the corresponding field of the previously transmitted cipher group or, for the first group of a message, from the last deciphered verification field. By this means, there can be an unbroken chain of continuity during an exchange of messages, however long, between CPU and terminal which can be continuously monitored by identity checks on the verification fields. Figure 6 illustrates the technique where, for ease of exposition but without loss of generality, message groups are shown as being only six characters of which the two right-hand ones are for verification.

Initialization is done in a special way at the CPU by use of the (unique) date and time in the verification field; this ensures that even though the information field is stereotyped, as it naturally would be from a CPU immediately after sign-on from a terminal, the corresponding cryptogram for the first group will be unique. As the chaining for successive groups utilizes corresponding fields of previous ciphertext, which for practical purposes consist of random bits, there is a very high probability that cryptograms during an exchange of messages will never be the same twice, no matter what the information content.



NOTES: MESSAGES FROM CPU ARE UPPER-CASE ROMAN.
MESSAGES FROM TERMINAL ARE LOWER-CASE ROMAN.
CRYPTOGRAMS ARE LOWER-CASE GREEK.

FIG. 6. VERIFICATION BY MESSAGE CHAINING
(COMPLETE VERIFICATION AT BOTH ENDS)

In Figure 6, indication is given of the fields which must match at either end of the communications line if complete verification checking is to be done at both ends. This requires storage facilities for only two verification fields at any one time and could, of course, have been incorporated into Lucifer together with automatic comparing circuits. But since verification is more important at the CPU than at the terminal, and in order to save hardware, automatic checking is omitted from Lucifer. Both the CPU and Lucifer perform character-group chaining and the CPU would conduct a complete verification check.

In order for the 2770 operator to be able to perform certain visual checks on the verification fields, various options in supplying these fields at the CPU may be taken. One scheme, illustrated in Fig. 7, provides that the first verification field of any message received at the terminal will be identical to the last verification field of the preceding message it received.

The successful application of this method of character-group chaining does not depend on a sequence of preagreed passwords, nor upon continual comparisons between time-varying passwords as from clocked counters which need close synchronism, nor is any additional storage required to do this form of redundancy chaining. The expense is in the loss of throughput; the ratio of ciphertext length to message length is the ratio of 16 to 16 minus the number of verification bytes.

Each group of message-plus-verification characters is enciphered into 16 ciphertext bytes and is transmitted forthwith.

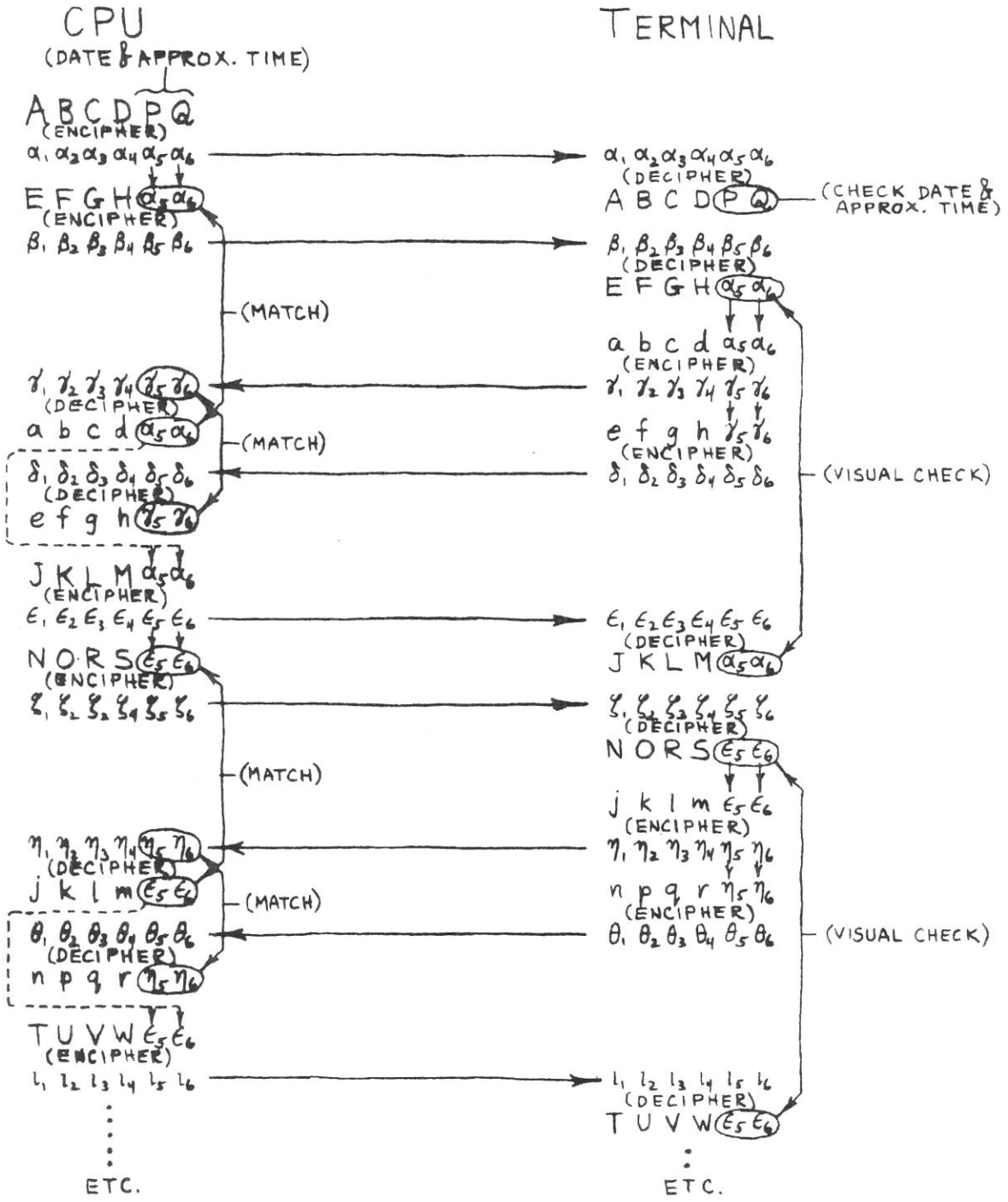


FIG. 7. VERIFICATION BY MESSAGE CHAINING
(COMPLETE VERIFICATION AT CPU ONLY)

Insofar as the deciphering is concerned, each group is independent of the others and can be deciphered forthwith as it is received. For the same reason, errors do not propagate beyond the character group in which they occur; a failure in the verification check will propagate to the next group only if the error had occurred in the verification field itself.

Lucifer performs message chaining through the simple act of saving all its "left-overs". When a cleartext message is to be loaded into Lucifer for enciphering, only the information field is loaded; the verification field consists of the left-overs from either the last group of a message just received, or the last group of an enciphered message just transmitted, however it may be. The length of the verification field is determined by a manual-switch setting which, of course, must agree with the field-length parameter in force at the CPU.

When Lucifer is deciphering incoming messages, the terminal operator may opt to have the verification fields suppressed; this in no way affects character-group chaining in Lucifer, nor verification checking at the CPU.

There is an additional feature by which the CPU can prime Lucifer with any arbitrary verification field to initiate a new independent chain without prematurely aborting the old one, which would foreclose the matching of certain fields at the terminal. This is accomplished by the simple procedure of transmitting a verification field only (no information field) from the CPU as a

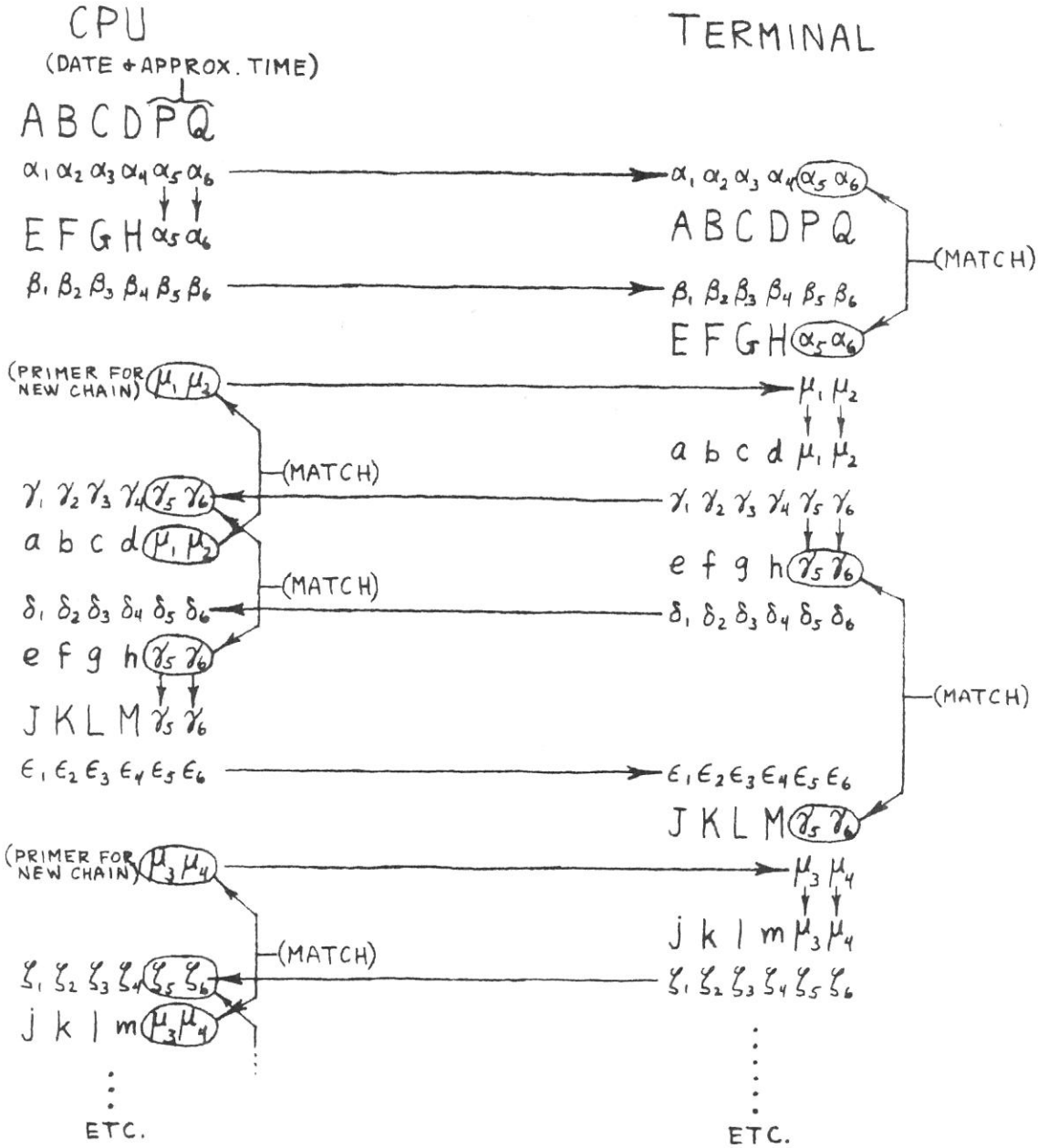


FIGURE 8. INITIATION OF INDEPENDENT VERIFICATION CHAINS

trailer to a normal ciphered message. This trailer is an incomplete group, and Lucifer will make no attempt to decipher it and will not deliver it up for display on an output device, but instead will save it for use as the verification field for the first group of the next ciphered message from the terminal. Figure 8 illustrates this procedure in which the CPU (on two occasions) initiates a new chain of verification fields.

Decisions as to the course of action to be followed at the CPU in case of failure of any verification check in a message would depend on circumstances. If warranted, any such message even if otherwise meaningful would be ignored by the CPU. The CPU could reinitialize the verification chain and ask for a retransmission, and as a last resort could disconnect the communications line. This would prevent tampering with sensitive files by sophisticated impostors who, for example, might be "playing back" recorded versions of previous legitimate transactions.

9. THE LUCIFER-2770 INTERFACE

This and the following two sections are intended to provide some insight into the operational characteristics of the 2770-Lucifer combination. For complete comprehension, a slight knowledge of the principles of operation of the 2772 Control Unit[3] and of Binary Synchronous Communications[4] discipline will be helpful to the reader.

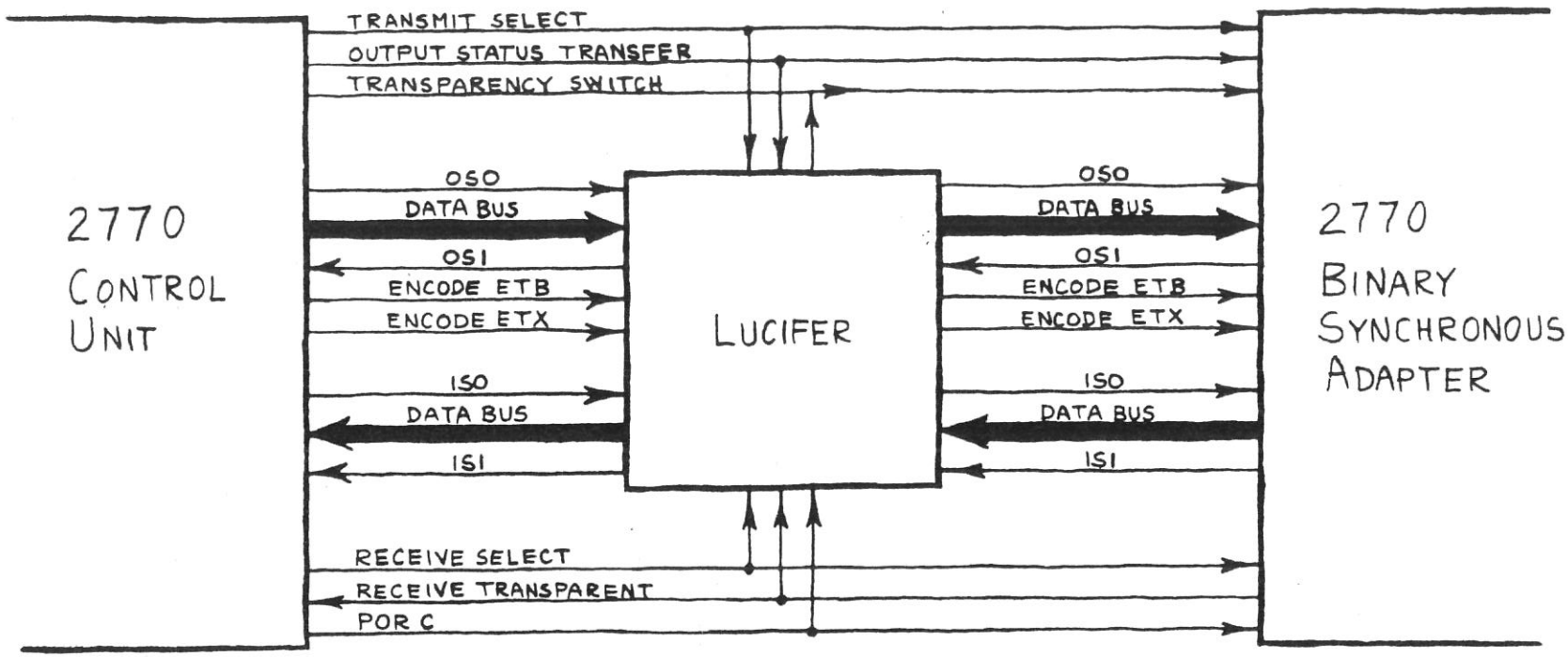
In these three sections, the term "block" used in connection with data transmission refers to the contents of one of the 2770 buffers (up to 128 or 256 bytes).

Lucifer is installed at the interface between the 2770 buffers in the Control Unit (referred to simply as CU) and the Binary Synchronous Adapter (BSA) which is a semiautonomous part of the 2770 performing line-control functions and managing the transmitting and receiving. This arrangement provides cryptographic protection from the interface to the BSA outwards onto the communications line, and the same protection would apply for whatever I/O device may be selected. An alternative would be to install Lucifer at the interface to a particular I/O device; although this would broaden the protected zone to include the CU itself and all the other I/O devices, the former option is preferable because of the greater flexibility it affords.

Figure 9 shows a portion of the CU/BSA interface into which Lucifer is fitted. This fitting involves the breaking of 22 lines and tapping onto six others. No physical wires are actually cut; four of the short ribbon cables connecting logic boards B1 and B2 on the A gate are removed and a pair of longer cables is substituted for each. These longer cables are also plugged into an auxiliary board mounted on the A gate, where the required breaks and connections are done, from which three other cables plug into Lucifer.*

---

* A better arrangement is to prepare a special cable "harness" which is plugged in place of the four short cables, having the appropriate wires tapped or physically cut, with the Lucifer cables attached to and integral with it. With this harness, Lucifer could be attached to any 2770.



NOTE: ONLY THOSE INTERFACE LINES THAT CONCERN LUCIFER ARE SHOWN.

FIG. 9. THE LUCIFER/2770 INTERFACE

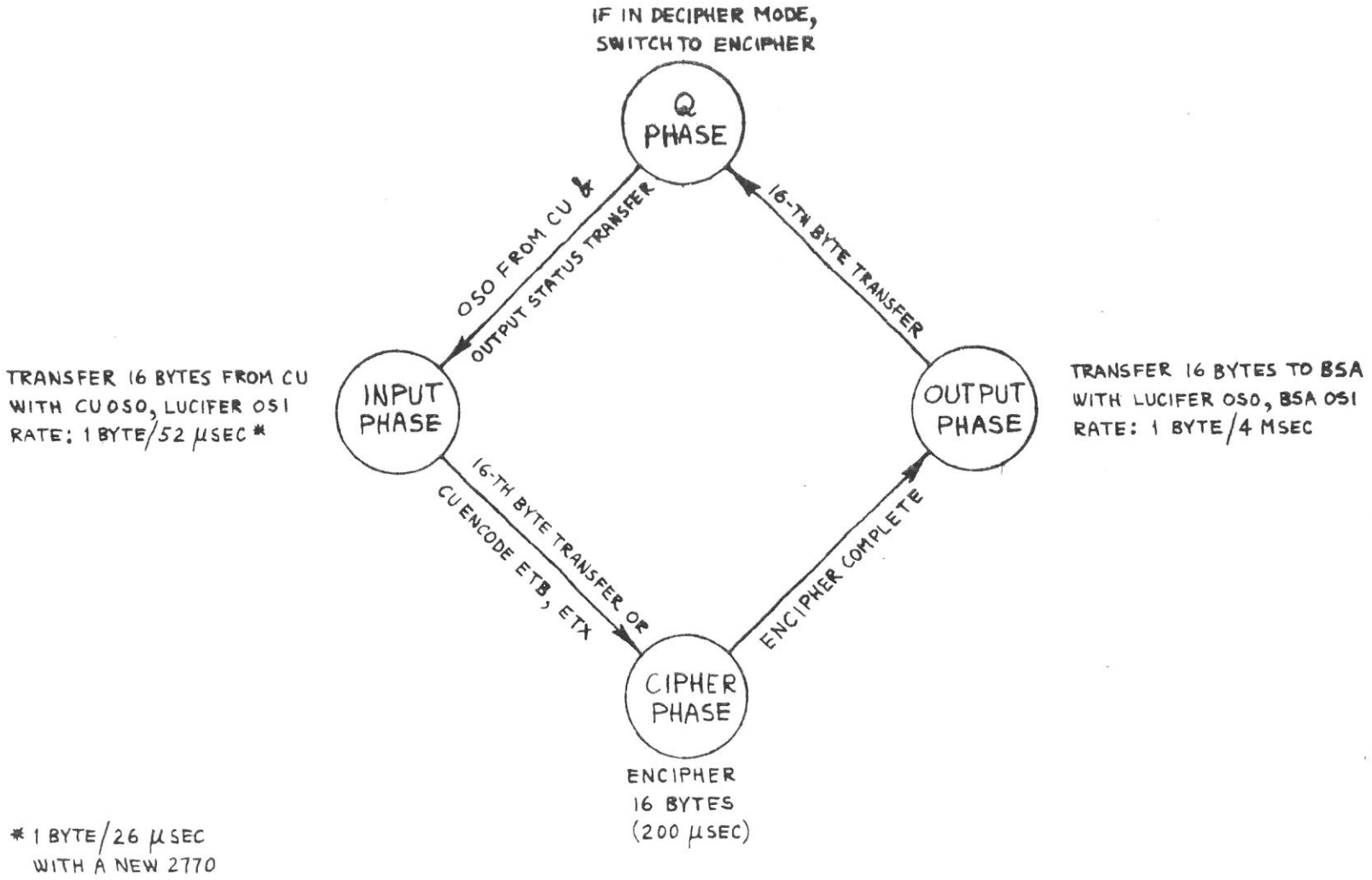


FIG. 10. LUCIFER OPERATION FOR TRANSMIT CIPHER

When Lucifer is resting in the Quiescent (Q) Phase and when the Lucifer Mode switch is set to Normal, all of these broken lines are effectively "spliced" together through active circuits and the 2770 operates in its normal manner.

It was seen fit to make only one slight modification in the 2770, which is optional by jumper placement. By removing the jumper, the Output Block Transparent signal is prevented from going to the matrix printer interface circuits so that, by this option, the printer will respond functionally to all format-control characters it receives, whether the output block is transparent or not. Hence, messages containing format-control characters received either under cipher protection (transparent) or in the clear (nontransparent) will be printed in the same way. The only disadvantage is that an automatic new line is not furnished at the end of each transparent block.

10. OPERATION OF LUCIFER FOR TRANSMIT CIPHER (see Fig. 10)

The Lucifer Mode switch must be set to the Send Cipher position. This switch setting conditions Lucifer to operate for transmitting, and it also overrides the setting of the Transparent Switch on the 2770 Operator's Console causing that switch to be effectively on so that the BSA will transmit in transparent mode.

If Lucifer is resting in Q Phase and the Receive Select signal from the CU is absent, Lucifer will go into the encipher state if it is not already in encipher. When Transmit Select, Output Status

Transfer, and the first CU Output Service Out (OSO) appear, Lucifer goes to Input Phase and is operational. In anthropomorphic terms, Lucifer faces the CU and pretends to be the BSA, responding to each OSO signal from the CU by storing a character from the data bus and furnishing its own Output Service In (OSI) signal to the CU. Lucifer takes the characters as fast as the CU can provide them--one character every 52 microseconds (26 for a new 2770)--until 16-X characters have been transferred. X is the number of bytes reserved for the verification field--from zero to eight--and is determined by a rotary switch setting on Lucifer. (This presupposes that the remote receiver is capable of correctly interpreting the information field and verification field.) If fewer than 16-X characters are available from the CU, after the last one the CU will furnish either Encode ETB or Encode ETX; when either of these is detected, Lucifer will quickly fill out this group with null characters to make a total of 16-X. (Lucifer prevents the BSA from getting Encode ETB or Encode ETX at this time.) The full group of 16 characters that will be enciphered comprises these 16-X characters plus X bytes of whatever had been left in the Lucifer storage registers after their last usage.

Lucifer now goes to Cipher Phase, generates 16 bytes of the cryptogram in about 200 microseconds, and goes immediately to Output Phase.

In Output Phase, Lucifer faces the BSA and pretends to be the CU, putting a cryptogram byte on the data bus, giving its own OSO

signal and responding to OSI from the BSA with another byte, etc. The BSA accepts the data at a rate determined by the data set clock (in this case one byte every four milliseconds). After the 16th byte has been transferred to the BSA, Lucifer goes immediately to Q Phase.

If at this time the CU has not yet provided either of the signals Encode ETB or Encode ETX, there is more of the message to be transmitted and Lucifer goes immediately to Input Phase to repeat the cycle.

However, if Encode ETB or Encode ETX has been provided, Lucifer is halted in Q Phase and only now passes that signal along to the BSA with OSO from the CU. The end-of-record sequence occurs normally.

If the 2770 is programmed to do a Monitor Print after each transmitted block, Lucifer is not responsive to the CU OSO signals provided for the printer because Transmit Select is absent, nor is it responsive to OSO when the subsequent Encode ETX is given because Output Status Transfer is absent.

11. OPERATION OF LUCIFER FOR RECEIVE CIPHER (see Fig. 11)

The Lucifer Mode switch must be set to a position other than Normal. Lucifer will then be switched in or out of operation depending on whether the message block is sent from the remote transmitter in transparent mode or not; transparent blocks may be intermixed with non-transparent blocks and Lucifer will operate only on the transparent ones, passing the others along unaltered.

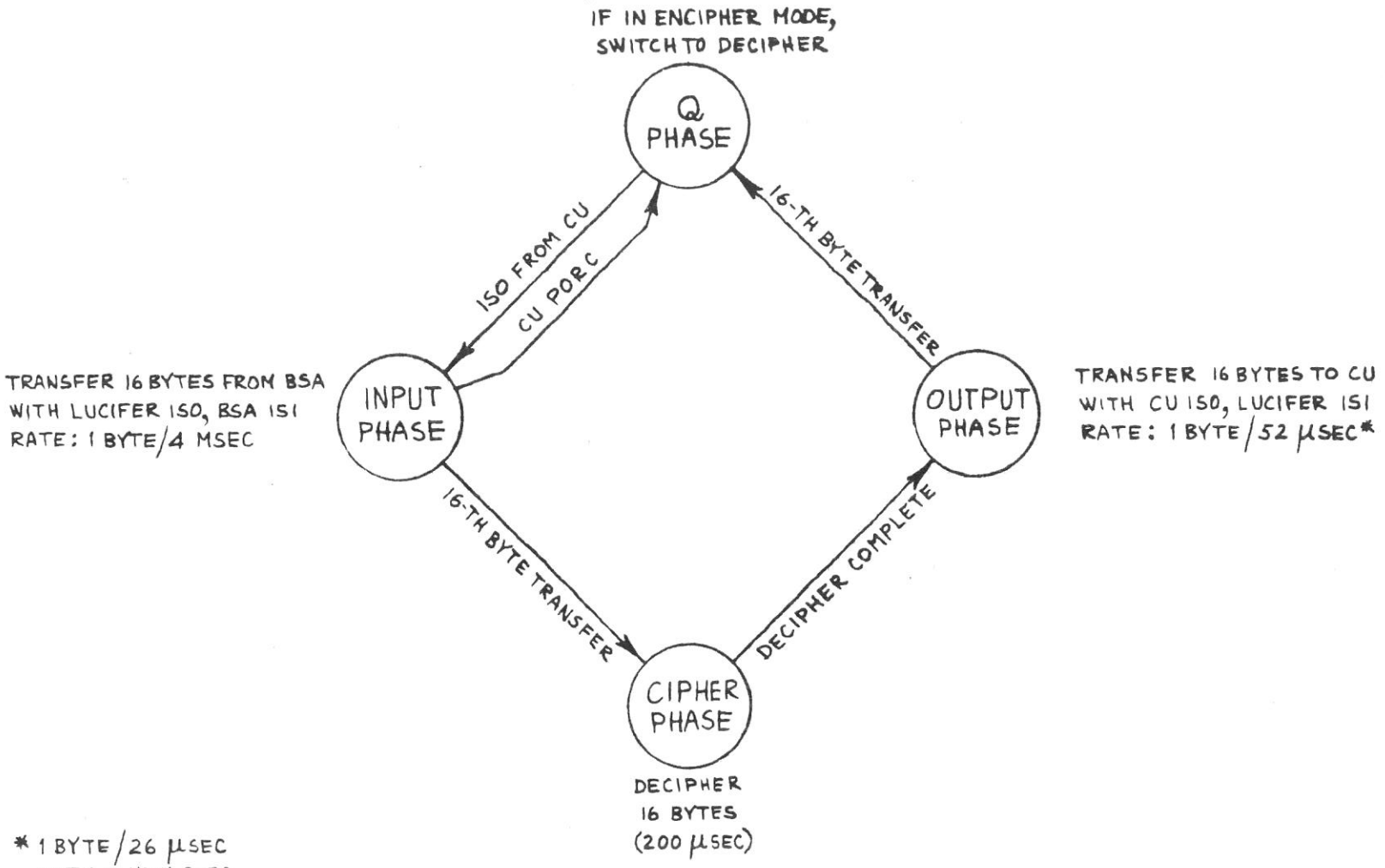


FIG. 11. LUCIFER OPERATION FOR RECEIVE CIPHER

If Lucifer is resting in Q Phase and the Receive Select signal from the CU is present, Lucifer will go into the decipher state if it is not already in decipher. If the BSA receives an initial DLE STX, it will produce the Receive Transparent signal, and when that signal and the first Input Service Out (ISO) from the CU appear, Lucifer goes to Input Phase and is operational.

In Input Phase, Lucifer faces the BSA and pretends to be the CU, giving its own ISO, and waiting for the BSA to respond with a byte and Input Service In (ISI). Sixteen bytes of the incoming cryptogram are collected in this manner, at a rate determined by the transmission speed (in this case one byte every four milliseconds).

Lucifer goes to Cipher Phase, deciphers the 16 bytes in about 200 microseconds, and goes immediately to Output Phase.

Lucifer now faces the CU, pretends to be the BSA, and responds to each CU ISO by transferring a character of the deciphered message along with its own ISI. The process continues at a rate of one character every 52 (or 26) microseconds until 16 characters have been transferred or, if the Verification Field Suppress switch is set to Suppress, until 16-X characters have been transferred. (This presupposes that the remote transmitter has the equivalent capability of providing an X-byte verification field in the cleartext groups before enciphering takes place.) If the Suppress option is taken, Lucifer skips over the last X characters of each group without transferring them to the CU; if the Print option is taken, all 16 characters are transferred.

Lucifer goes to Q Phase and thence to Input Phase because the CU, expecting more of the message, gives another ISO. If the BSA receives more message, the process repeats; if the BSA receives either DLE ETB or DLE ETX, the Receive Transparent signal will be removed and Lucifer will be temporarily inoperative in Input Phase. The BSA and the CU now interact normally, and a Normal Job Ending eventuates. When the CU produces its normal Power On Reset (POR C), Lucifer is forced back to Q Phase.

It is usually the case that the remote transmitter furnishes ciphertext in integral multiples of 16 bytes. If it does not, then the last (curtailed) group of bytes will not be deciphered but instead will be saved in Lucifer's storage registers for use in the verification field during the next Transmit Cipher.

12. OFF-LINE TEST

A necessary adjunct for the initial debugging of Lucifer and for trouble-shooting is a means for running a thorough test off line from the 2770. There is no such thing as being able to judge whether or not Lucifer is operating "almost" correctly, or to locate faults, by examining only the final results of a ciphering operation; one mishap anywhere produces chaos everywhere. It is necessary to provide a means of running Lucifer in "slow motion" so that individual microsteps can be checked.

The basic timing of Lucifer is set by a free-running oscillator, whose frequency is adjusted to be about 5 MHz, driving a two-flip-flop ring from which is derived a cycle of T-pulses at a repetition

rate of about 1.25 MHz. "Slow motion" is achieved by the appropriate gating of the oscillator pulses to produce the desired cycle of T-pulses, under manual control.

Off-line testing is performed manually by means of switches, push buttons, and indicator lamps on the special built-in "CE" panel (see Plate 4).

Features provided for performing the off-line test are:

(1) A toggle switch to effectively "stop" the clock, and a clock-advance push button to give single T-pulses, single cycles of T-pulses, or (in Cipher Phase) groups of eight cycles of T-pulses, according to the setting of a rotary selector switch. A clock indicator lamp shows whether the clock is running or stopped--bright for stopped, dim for running.

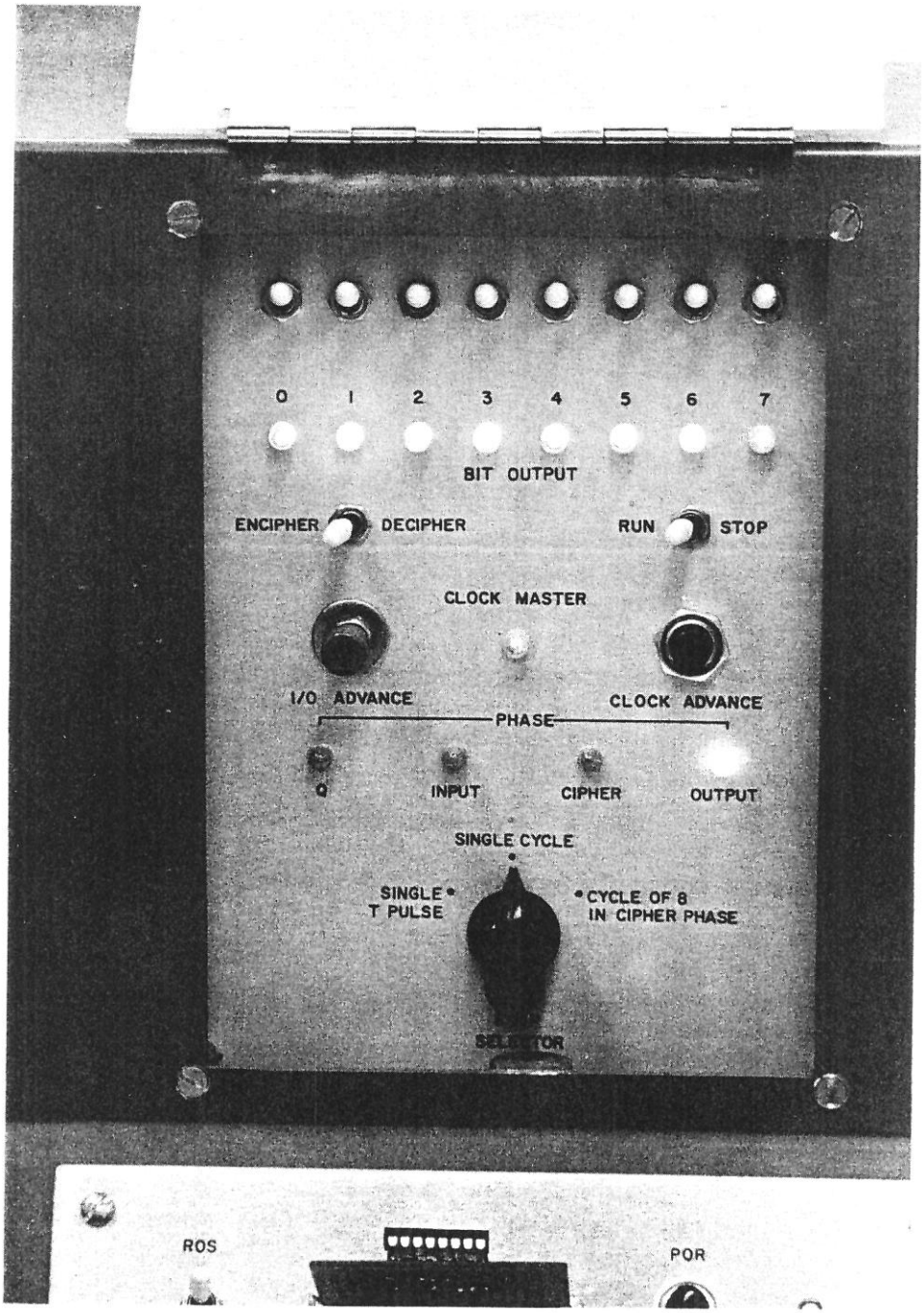
(2) A set of eight toggle switches which stores a byte of data to simulate input from the CU data bus.

(3) A set of eight indicator lamps, each individually pluggable to any wire-wrap pin in Lucifer, but normally plugged to the outputs of the convolution registers (lines H on Fig. 2).

(4) A toggle switch to determine Encipher or Decipher state for the off-line test.

(5) Four indicator lamps for the phases: Q, Input, Cipher, and Output.

(6) An I/O Advance push button which, for each excursion, will produce a single simulated service signal according to the phase and state, as in the accompanying table.


0 1 2 3 4 5 6 7
BIT OUTPUT
ENCIPHER DECIPHER
RUN STOP
CLOCK MASTER
I/O ADVANCE
CLOCK ADVANCE
PHASE
Q
INPUT
CIPHER
OUTPUT
SINGLE CYCLE
SINGLE T PULSE
CYCLE OF 8 IN CIPHER PHASE
ROS
POR

Service Signals Simulated by I/O Advance

	Input Phase	Output Phase
Encipher	CU OSO	BSA OSI
Decipher	BSA ISI	CU ISO

A software adjunct for testing is an APL function* (see the appendix) which is an exact analog of Lucifer's behavior as exhibited by the indicator lamps plugged to the storage registers. When the clock is advanced in slow motion, a byte-by-byte comparison can be made between the lamps and the APL displays. This permits Lucifer's activities to be checked in as fine detail as one wishes.

For Encipher in Input Phase, a byte of data from the input switches is stored in the data registers with each simulated CU OSO furnished by I/O advance; after sixteen such, Cipher Phase begins which can be executed as rapidly or as slowly as desired for comparing with the APL displays. In Output Phase the final cryptogram can be observed byte-by-byte with each simulated BSA OSI. After the sixteenth, Lucifer returns to Q Phase.

For Decipher in Input Phase, the cryptogram recirculates upon itself through the output and input data buses, one byte for each simulated BSA ISI. After the sixteenth, Cipher Phase begins

---

* This is distinct from the software cryptographic function used by the data processor with which the 2770 communicates.

and deciphering is done in slow motion as desired. In Output Phase, the original data from the switches can be observed on the indicator lamps, one byte for each simulated CU ISO.

By this means, all of Lucifer's data flow logic and control and all the circuits for interfacing the 2770 can be tested, with the exception of a few circuits at the interface for voltage-level limiting and signal inverting.

When Lucifer is attached to the 2770, the encipher operation can be checked out similarly by putting the 2770 in Line Simulation Test. Lucifer's clock is "stopped" and run in slow motion as desired and the same comparisons can be made with the APL displays as before. The I/O advance push button is not used for this test, nor is the set of eight input switches.

13. HARDWARE SUMMARY

Lucifer is implemented in standard TTL/SSI (small-scale integration) and TTL/MSI (medium-scale integration) modules. Altogether, 178 TTL modules are used, mounted on four 60-module Augat wire-wrap boards, plus a few resistors and diodes.

The accompanying table shows a breakdown of the logic elements by main functional groups. These figures do not relate directly to TTL module counts but can be used to project the implementation in other technologies. Not included are the electrical interface circuits for voltage-level limiting.

A complete set of logic diagrams for Lucifer is available from the author on request.

	DATA FLOW	CONTROL	KEY STORAGE *	INPUT INTERFACE	OUTPUT INTERFACE	OFF-LINE TEST	TOTAL
FLIP-FLOP		7	4	5	5	2	23
MONOSTABLE MULTIVIBRATOR		3				1	4
4-BIT COUNTER		3					3
4-BIT DECODER	2						2
EXCLUSIVE OR GATE	16	3	1				20
NAND GATE (INCL. INVERTER)	16	71	20	31	36	10	184
2×2 AOI GATE	32		8	10	16		66
4-BIT TAPPED SHIFT REGISTER	16						16
8-BIT SHIFT REGISTER	8		16				24
8-BIT SERIALIZER	1						1

* NOT NEEDED FOR ROS KEY

SUMMARY OF LUCIFER COMPONENTS

14. ACKNOWLEDGMENTS

Many persons have participated in the development of Lucifer and in its successful incorporation in the 2770 system. Their contributions are gratefully acknowledged.

In re the cipher system:

Acknowledgment is made to Horst Feistel for the basic cryptography notions embodied in Lucifer, and appreciation is expressed for many fruitful discussions.

In re message verification:

This is an outgrowth of the work of Feistel, and acknowledgment is also due W. A. Notz for his contribution to the development of the particular method.

In re the physical realization:

1. Special acknowledgment is made to J. Vanginderen, Central Scientific Services, who planned all the TTL circuit layouts and wiring, who was in charge of the major portion of the construction, and who greatly assisted in the initial debugging.
2. Acknowledgment is due V. DiLonardo for the detailed design of the Lucifer cabinet and overseeing its fabrication, for the design of the printed-circuit cards for ROS keys, and for ably assisting in the initial debugging.

In re the 2770:

1. Special thanks are due W. F. Emmons, SDD Raleigh, for providing on loan a preproduction model of the 2770.

2. Most grateful acknowledgment is made to James Martin, SDD Raleigh, for remote diagnoses during the period of initial debugging of the 2770, and also to David Baron, SDD Raleigh (formerly Research), for assisting in the debugging.

3. Thanks are due Robert Davis, Jon Fox and L. E. Smith, all of SDD Raleigh, for helpful discussions and advice leading to the successful integration of Lucifer in the 2770.

In re inputting the cipher key:

Special thanks are due Jack Gonzales and Tom Anderson, Custom Systems, San Jose, for providing on loan a 2960 magnetic card reader, and also for prerecorded key cards.

The author expresses gratitude to M. G. Smith, W. A. Notz, J. P. Cedarholm, and P. E. Green for their encouragement and stimulation throughout the course of this work.

REFERENCES

1. H. Feistel, "Cryptographic Coding for Data-bank Privacy," RC 2827, March 18, 1970.

2. C. E. Shannon, "Communication Theory of Secrecy Systems," BSTJ, Vol. 28, No. 4, October 1949, pp. 708-713.

3. 2772 Field Engineering--Theory of Operation, Form SY27-0081.

4. Binary Synchronous Communications, SRL Form GA27-3004-1.

APPENDIX

For illustrative purposes, some APL displays are given which show the divergence characteristics of the Lucifer algorithm when there is an initial minuscule difference in either the message or the key. Imagine that there are two identical Lucifer units operating in synchronism with identical cipher keys and which are ciphering two message groups that are identical except for one bit. Exhibit 1 shows as x's the corresponding bits that are <u>different</u> in the two units at the end of each round. (For convenience, the bits are displayed in pairs of bracketed horizontal rows, the upper row of each pair being the bits in the source registers and the lower one the bits in the convolution registers.) The 0th pair of rows shows the single bit difference in the original messages, the 1st pair of rows the differences after the first round, etc. The 16th row pair shows the differences in the final cryptograms. Note that the differences develop early and rapidly diffuse throughout the entire groups in apparently random fashion.

Exhibit 2 shows the differences for a similar situation in which the message groups are identical but with one bit different in the two cipher keys.

Exhibits 3 and 4 are horizontal plots of the <u>cardinal number</u> of corresponding bits that are different in the two Lucifer units after each round, averaged over 50 trials, beginning with a single arbitrarily chosen message-bit difference for each of the 50 trials

in the first case and with single key-bit differences in the second. Note that, in either case, the number of bits that are different increases very rapidly and hovers around 64 after about the eighth round, which is the expected number of differences between random sets of 128 bits.

Exhibit 5 is a display of the APL function LUCIFER which can be used for comparing with the action of Lucifer in performing the off-line test. SB0 and SB1 are four-bit transformations equivalent to the contents of Table 1. P is the vector corresponding to the permutation to obtain bits T0...T7 of Fig. 3, and Q is the vector specifying the locations of the mod-2 adders in the convolution registers (the diffusion permutation). The index origin is 0.

In the LUCIFER function, the arguments K and M are the key and message respectively, and may be specified globally to correspond to the actual key in use and the message to be ciphered. For initial debugging, it is convenient (from the standpoint of hardware) to use a key of all 1's and a message of all 1's. For this case one would simply issue the command

```
Z ← 1 LUCIFER 1 ,
```

and a complete encipher-decipher could be commanded by

```
Z ← 1 LUCIFER 1 LUCIFER 1,
```

specifying Encipher for the first execution and Decipher for the second.

```
      BITCHANGE
CHANGE MESSAGE OR KEY? TYPE M OR K
M
MESS[0;5:1] CHANGED TO 1
INDIVIDUAL BIT DIFFERENCES
```

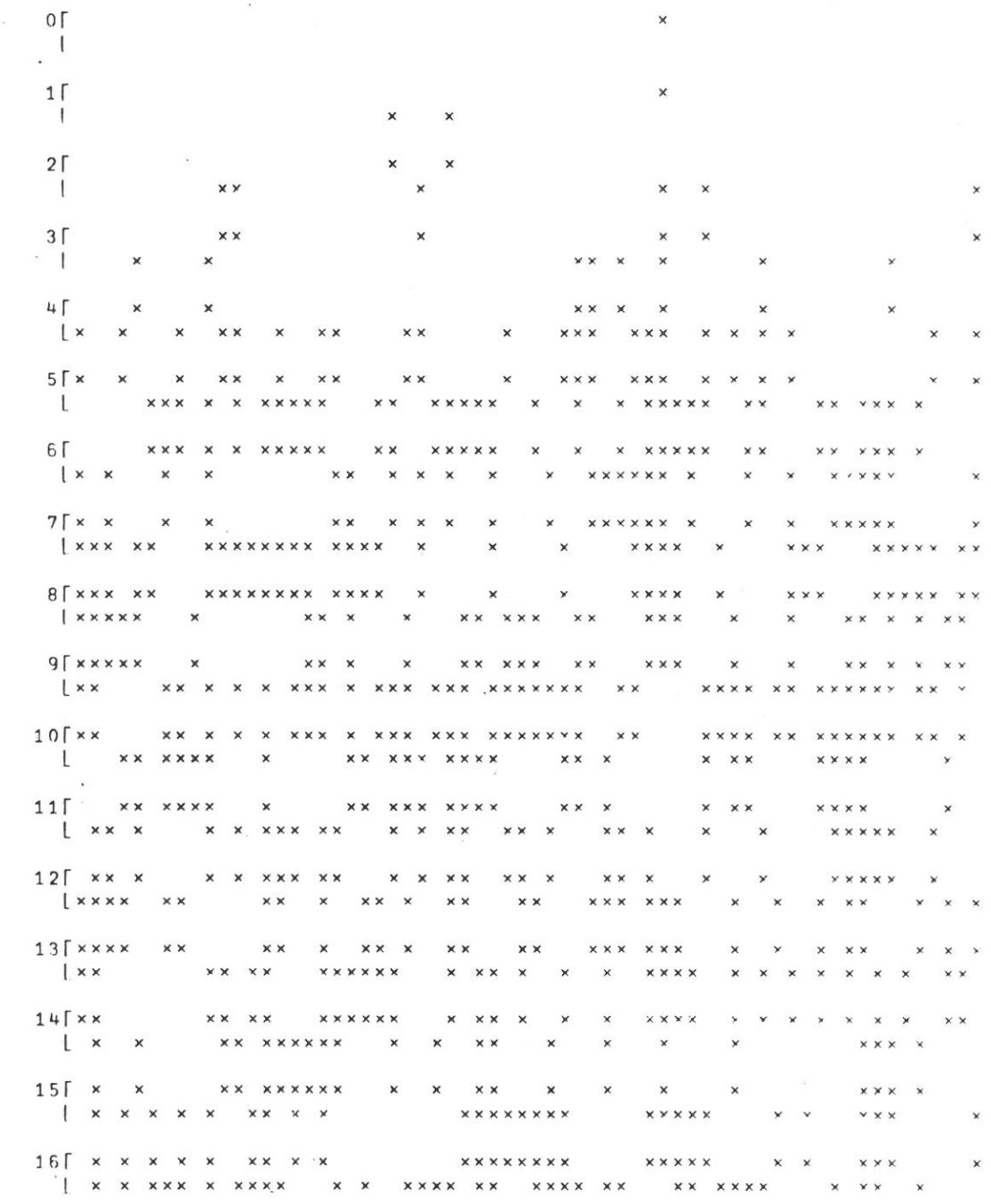


EXHIBIT 1

```
      BITCHANGE
CHANGE MESSAGE OR KEY? TYPE M OR K
K
KEY[3;1] CHANGED TO 0
INDIVIDUAL BIT DIFFERENCES
```

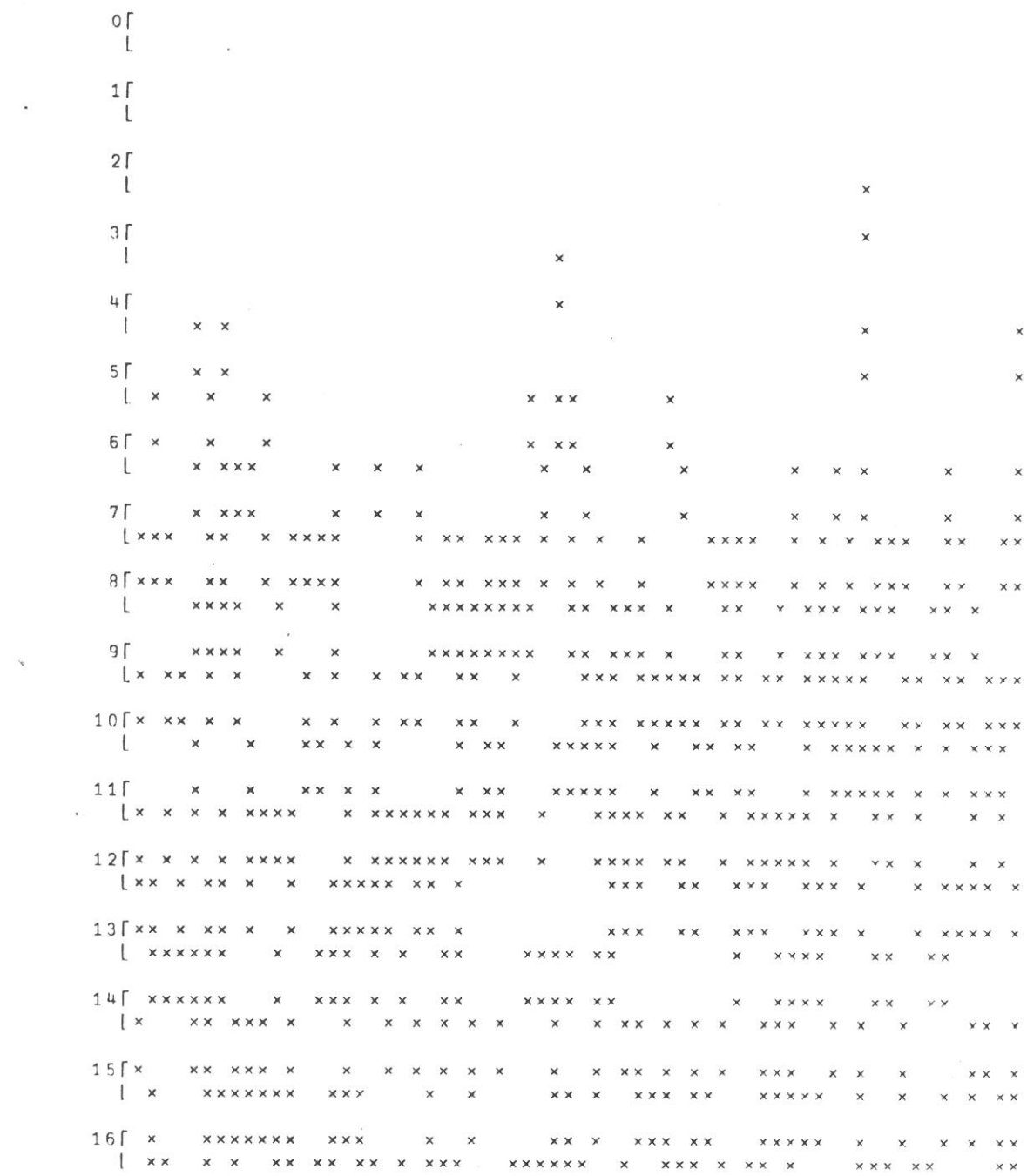


EXHIBIT 2

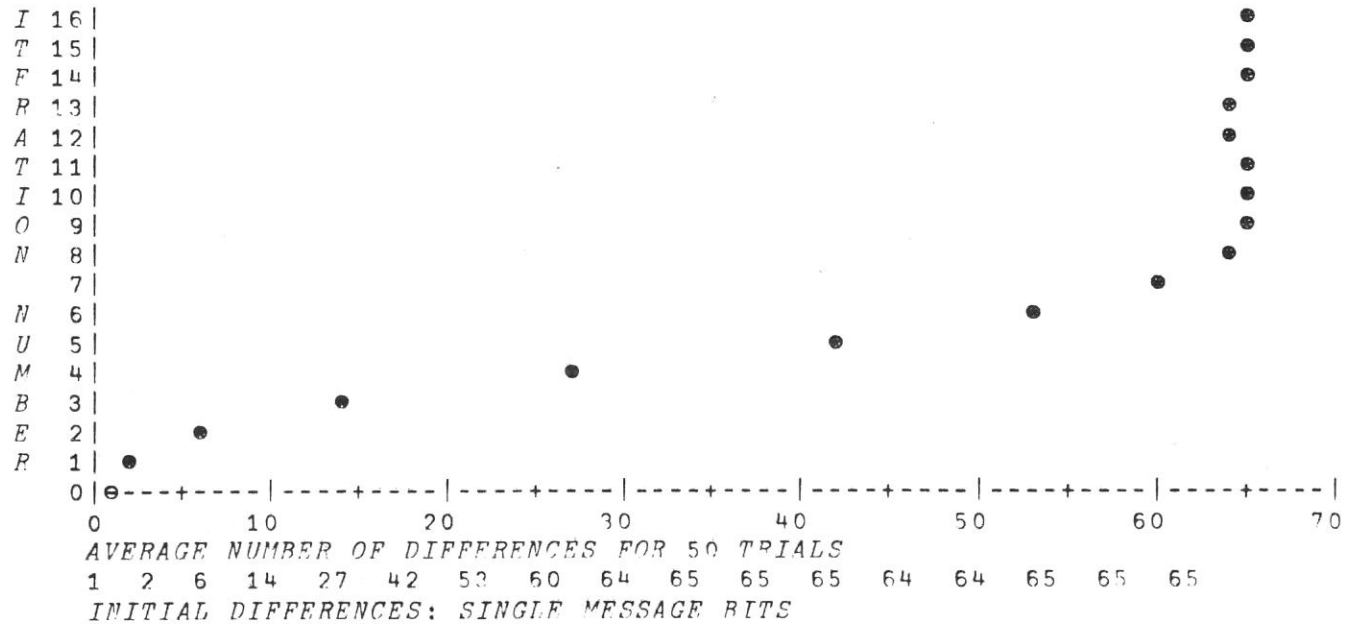

ITERATION NUMBER
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
0
10
20
30
40
50
60
70
AVERAGE NUMBER OF DIFFERENCES FOR 50 TRIALS
1 2 6 14 27 42 53 60 64 65 65 65 64 64 65 65 65
INITIAL DIFFERENCES: SINGLE MESSAGE BITS

EXHIBIT 3

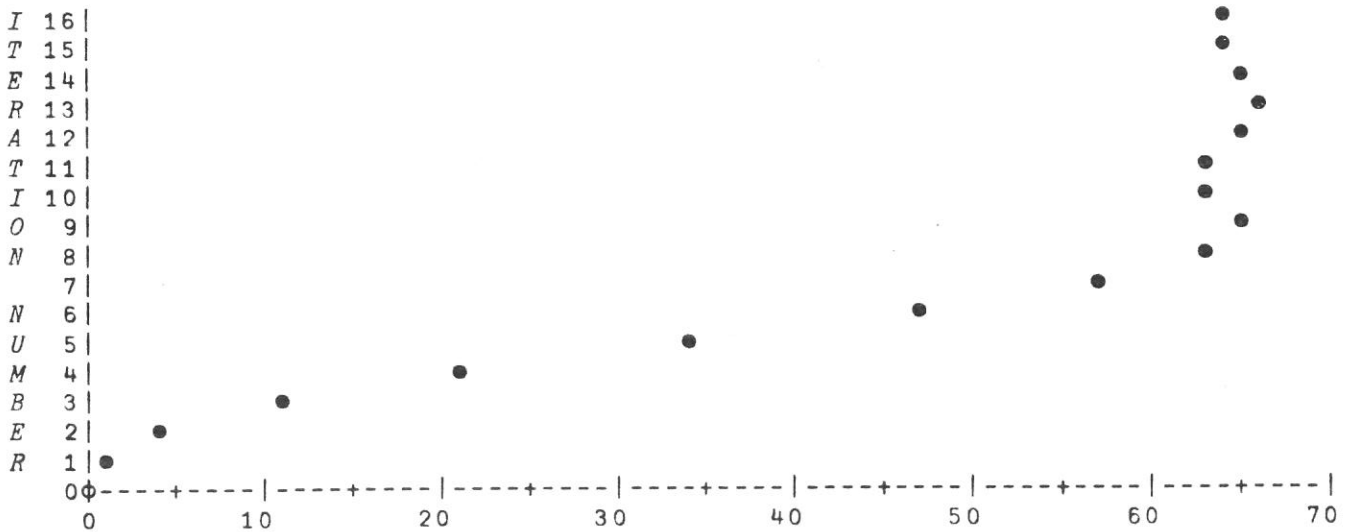

ITERATION NUMBER
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
0 10 20 30 40 50 60 70
AVERAGE NUMBER OF DIFFERENCES FOR 50 TRIALS
0 1 4 11 21 34 47 57 63 65 63 63 65 66 65 64 64
INITIAL DIFFERENCES: SINGLE KEY BITS

EXHIBIT 4

```
      ∇LUCIFER[□]∇
     ∇ Z←K LUCIFER M;D;J;TP;SM;PR
[1]   L5:→(2=D←1⍴'ED'⍳⍞)/L5,(J←1),⍴□←'ENCIPHER OR DECIPHER? TYPE  E  OR  D'
[2]   L6:→(2=PR←1⍴'NY'⍳⍞)/L6,⍴□←'PRINT INTERMEDIATE RESULTS? TYPE  YES  OR  NO'
[3]    0/⍴K←(8×D)⊖K;'KEY:';'01'⎕K← 16 8 ⍴K; 2 1 ⍴' ';'INPUT:';'01'⎕M← 2 8 8 ⍴M
[4]   L1:TP←(K←D⊖K)[0;]⊖(⍉ 2 4 8 ⍴⍉M[1;;])+.× 8 4 2 1
[5]    SM←(-C)⊖((K[⍳8;]≠⍉SB0[;TP[0;]],[0] SB1[;TP[1;]]),[0] 8 8 ⍴0)[;P]
[6]    →(PR=0)/L3,(⍴M[0;;]←M[0;;]≠SM[⍳8;]),⍴K←(7+D)⊖K
[7]    2 1 ⍴' ';'ROUND ';J;': OUTPUT DURING C-I-D';'01'⎕M[0;;]
[8]   L3:→((16=J),PR=0)/L2,L4,0/⍴M[0;;]←M[0;;]≠SM[8+⍳8;]
[9]    2 1 ⍴' ';'ROUND ';J;': OUTPUT DURING INTERCHANGE';'01'⎕M[0;;]
[10] L4:→L1,(J←J+1),⍴M←⊖M
[11] L2: 2 1 ⍴' ';'OUTPUT:';'01'⎕Z←M
     ∇

      SB0
 1 1 0 1 1 1 1 0 0 0 0 0 1 0 0 1
 1 1 1 0 1 1 0 0 0 1 0 0 0 1 1 0
 0 1 1 1 1 0 1 0 1 1 1 0 0 0 0 0
 0 1 1 0 0 1 1 0 0 0 1 1 1 0 1 0

      SB1
 0 0 1 1 0 1 0 0 1 1 0 1 0 1 1 0
 1 0 1 0 0 0 0 1 1 1 0 0 1 1 0 1
 1 1 1 0 1 1 0 0 0 0 0 1 1 1 0 0
 1 0 0 1 1 1 0 0 0 1 1 0 0 1 0 1

      P
2 5 4 0 3 1 7 6

      C
7 6 2 1 5 0 3 4
```

EXHIBIT 5