

Copyright © 1975, by the author(s).  
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

**COSTS OF DATA ENCIPHERMENT ON COMPUTERS OF VARYING POWER**

by

**James Jatzczynski**

**Memorandum No. ERL-M493**

**January 1975**

**ELECTRONICS RESEARCH LABORATORY**

**College of Engineering  
University of California, Berkeley  
94720**

### Abstract

Encipherment appears to be an attractive means of protecting sensitive data in computer systems, but the costs of encipherment are not yet well known. A series of timing experiments was conducted which measured performance of enciphering tasks given each of two application models, four additive encryption methods, and three different computers. The applications simulated were file encryption and communication traffic encryption; the encryption schemes were a baseline null transformation and a short key, long key, and pseudo-random key method; the computers used were the CDC 6400, PDP-11/45, and PDP-10/50.

Encipherment rates attained by the assembly language programs ranged from 22,700 by/s for random key communication encipherment on the PDP-11/45 to 1,380,370 by/s for null file encipherment on the CDC 6400. Estimated monetary costs ranged from .672¢ to 11.06¢ to encipher 1000 card images and from 6.60¢ to 34.91¢ to support a 300 baud terminal for one hour.

# Costs of Data Encipherment<sup>†</sup>

## 1. Summary

The purposes of this study are to 1) explicitly define two distinct data processing applications in which encipherment is likely to be used, 2) determine attainable enciphering rates in each application using various additive encipherment methods, 3) determine the relative performance of various computers in execution of enciphering tasks, and 4) obtain an estimate of the monetary cost of encipherment.

First, models of the two applications are defined. Then encipherment rates are determined by actually running enciphering program segments on the computers under consideration and are verified by using instruction timing data from manufacturers' manuals. Cost estimates are based on computer center accounting rates.

Measured enciphering rates ranged from 22,700 by/s<sup>1</sup> to 1,380,370 by/s (a factor of 60) depending on the application model, enciphering method, and computer considered. For a single model and computer, rates between the simplest and most complex enciphering methods differed by up to a factor of nine. Expected costs of encryption were surprisingly low.

The significant range of performance between various enciphering algorithms and various processors indicated a need for careful consideration of equipment selection for enciphering applications. A particularly interesting point brought out by the use of two application models was the desirability of choosing a processor to fit the size of its intended task in order that efficiency be maximized.

---

<sup>†</sup>Partially supported by National Science Foundation Grant GJ 36475.

<sup>1</sup>by/s = byte/second where a byte contains 8 bits.

## 2. Introduction

### 2.1 Background

With the increased interest in security and privacy aspects of computer systems has come the recognition of enciphering methods as an effective means of protecting data. It has been recognized that cryptographic methods are applicable to both file storage and intercomputer transmission of data (Pet, Ska, Bar, Fei, Hof, Fri). Encipherment may be effected by either hardware or software; however, only software encipherment will be considered here.

The goal of any enciphering method (or privacy transformation) is to disguise data in such a way that they cannot be understood by unauthorized persons. The datum under consideration at any time is called a message. Its undisguised form is called cleartext and its disguised form is called ciphertext. The transformation from cleartext to ciphertext must be performed in such a way that the reverse transformation can be performed at a later time, and furthermore that it can be performed only by authorized persons (Kah, Hof). Thus an enciphering scheme normally consists of 1) an algorithm which gives the general method to be used to transform the cleartext into ciphertext and 2) a key which is a parameter of the algorithm and is somehow combined with the cleartext by the algorithm to generate the ciphertext; in essence, the key selects a particular instance of the general algorithm. The algorithm is designed to be invertible so that anyone who knows the inverse algorithm and the key used in a particular instance can recover the cleartext message from the ciphertext. Therefore, secrecy of the key is a necessary but not sufficient condition for data security; Baran has suggested that secrecy regarding the algorithm is neither necessary nor desirable (Bar). Thus the encipherment process can be symbolized as follows (where T is the enciphering algorithm or transformation, M is the cleartext message, K is the key, and C

is the ciphertext) (Hof):

$$T(M,K) \rightarrow C \quad (1) .$$

Deciphering can be viewed as

$$T^{-1}(C) = T^{-1}(T(M,K)) \rightarrow M \quad (2) .$$

The classical methods of encipherment include addition of cleartext and key, substitution of one alphabet for another, and transposition of characters of cleartext (Hof, Fri, Kah). Enciphering methods with non-linear properties have also been considered (Fei). Because additive ciphers use the exclusive or operation to effect addition they are extremely well suited for software implementation, particularly from the standpoints of ease of programming and speed of execution. Therefore only additive ciphers will be considered here.

An additive enciphering method works schematically as follows:

$$\text{CLEARTEXT} + \text{KEY} \rightarrow \text{CIPHERTEXT} \quad (3)$$

where + symbolizes the exclusive or operation. Because of the additivity, deciphering is algorithmically equivalent to enciphering, namely:

$$\text{CIPHERTEXT} + \text{KEY} \rightarrow \text{CLEARTEXT} \quad (4) .$$

The text may be any length and it may naturally be divided into segments or blocks such as characters or words. The key may be as long as the message or shorter in which case it is used repeatedly in a cyclic manner until the entire text is encrypted/decrypted. The natural portion of the text which becomes available at a given time or which is most convenient to consider will be called a segment.

Variation among additive methods arises from schemes of key generation. A constant key which has length one segment is most convenient, but offers little data protection. Longer constant keys (10, 100, 1000 or more segments) offer more protection at some added expense in algorithm complexity and key storage. Infinite key methods using pseudo random number generators to produce

an ever changing key provide potentially excellent security but have the expense of generating the "random" numbers (Bar, Car, Kah).

The relative efficiency and absolute cost of various methods of encipherment are of practical interest. Some of the data available to date are the following:

1) Carrol and McClelland (Car) report an encipherment rate of 37,000 by/s for a pseudo random key method running on a PDP-10/50.

2) Friedman and Hoffman (Fri) investigated several techniques on the CDC 6400. They were able to move data (null "encipherment") at 1,575,600 by/s, effect constant one-segment-key encryption nearly as fast, long key encryption (125-segment key) at 909,075 by/s, and infinite pseudo random key encryption at 374,100 by/s. The 60-bit word was their segment.

3) Tests reported in (Hof) for the 360/67 showed a data transfer rate (as in 2) of 700,560 by/s, one-segment-key rate of 514,800 by/s, long key rate of 352,080 by/s, and infinite random key rate of 243,000 by/s.

Additional figures are given in (Fri).

## 2.2 Scope and Limitations

Three additive enciphering methods (constant key, long key, and infinite random key) were studied here. Three different computers (CDC 6400, PDP-11, and PDP-10) were used. Two different data processing applications were considered.

Certain encipherment methods were not studied because they closely resemble other methods (e.g. long key vs. double key; see (Fri)) or because they were deemed unsuitable for software implementation (e.g. Lucifer which handled 96,970 by/s in a hardware implementation but only 2910 by/s on the Honeywell 6180 (Ben)). (However, see Appendix I)

Thus we consider only a very small subset of the three-dimensional space of application vs. enciphering method vs. processor. In addition, our cost

figures are very crude and should be judged as possibly in error by a factor of 2 or 3.

### 3. Discussion

#### 3.1 Application Models

Two models of application areas in which encipherment is likely to be used are considered here. Application Model I concerns encryption of files within a computer system (batch, timesharing, management information); Model II concerns encryption of communication traffic between a computer system and a character-oriented terminal typically used in timesharing applications. These applications represent two extremes of data availability: In application I a large mass of data is immediately available for processing; in application II data are available only on a character by character basis, and furthermore, character arrivals are separated by relatively long periods of time. These models imply simple but significant differences in the programs necessary to handle encryption.

Application Model I. Encryption of a file within a system might be accomplished by a command such as

```
ENCRYPT input-file output-file key
```

where the parameters are self-explanatory. The input file is immediately available in total, and buffering techniques for input/output allow the encryption routine to use the full power of the machine to read, encrypt, and write data very rapidly. More specifically, once the machine's fast registers have been loaded with parameters required by the encryption routine, it proceeds to completion with essentially no additional overhead. The processing sequence is illustrated in Figure 1. In this model, the segment size is one machine word.

Application Model II. In this situation single character encryption occurs at intervals as characters are received at and transmitted by the system. We



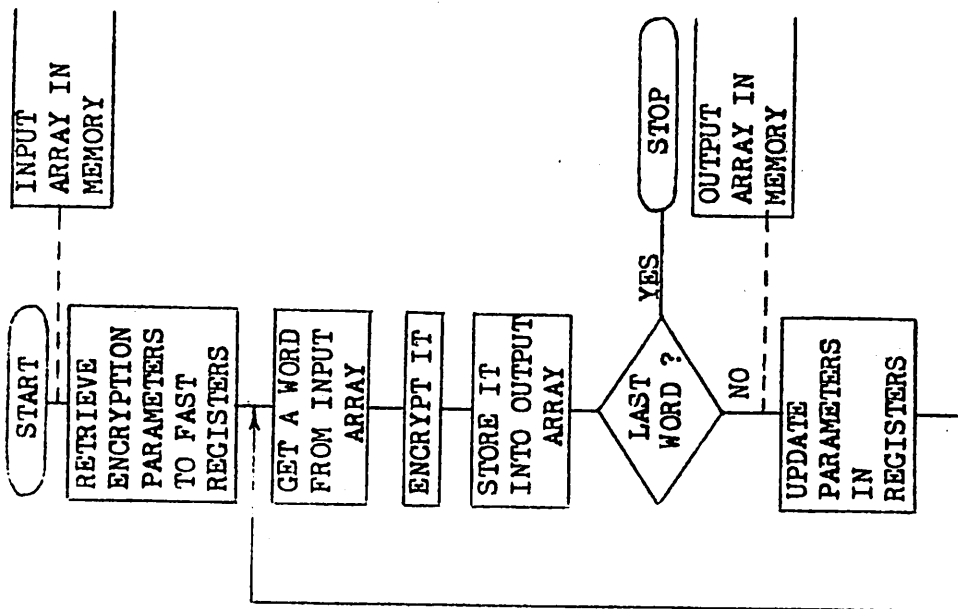


Figure 1. Application Model I Processing Sequence

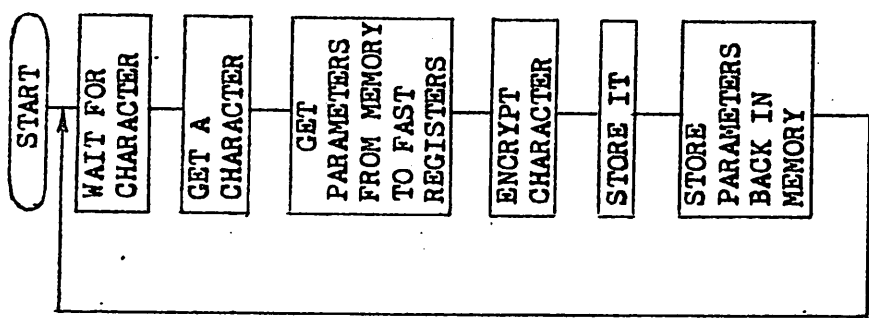


Figure 2. Application Model II Processing Sequence

assume highly interactive applications where any character may be significant, and thus each character must be deciphered (remember the algorithmic equivalence of enciphering and deciphering) as it is transmitted or received. (Note that enciphering of entire messages for transmission could occur as in Model I.) Here, in addition to enciphering time, significant overhead is incurred because 1) enciphering routine parameters must be retrieved from memory for each character (since characters are widely separated in time) and 2) a (say) 60-bit machine operating on an 8-bit character is using only a fraction of its processing power. The processing sequence is illustrated in Figure 2. (note that in the figure "encrypt" is synonymous with "decrypt."). In this model, the segment size is one byte.

### 3.2 Encipherment Methods

A "null encipherment" and three additive enciphering methods were considered in this study. The null encipherment consisted of simply moving data from one place to another in memory. The additive methods were those using a short one-segment key, a long 128-segment key, and an "infinite" pseudo-random key. Let  $M_i$ ,  $E_i$ , and  $K_i$  be the  $i$ -th segment of the message, the enciphered message, and the key respectively. Then the enciphering methods can be symbolized as follows:

Method#

$$1. \text{ Null} \quad E_i = M_i \quad (5)$$

$$2. \text{ Short} \quad E_i = M_i + K \quad (6)$$

$$3. \text{ Long} \quad E_i = M_i + K_i \text{ mod } 128 \quad (7)$$

$$4. \text{ Random} \quad E_i = M_i + K_i \quad (8).$$

In (6),  $K$  is a constant one segment long. In (7),  $K$  is a table of 128 key segments. In (8), successive words of  $K$  are produced by a linear congruential pseudo-random number generator. As before, "+" symbolizes the exclusive or operation.

### 3.3 Computers

Three computers were used in these experiments, namely the (Control Data

Corporation) CDC 6400 running under the scope operating system (CDC), the (Digital Equipment Corporation) DEC PDP-11/45 running under UNIX (Unx), and the DEC PDP-10 (KA10 processor) running under the Timesharing Monitor (P10). The main machine characteristics are:

	word size	memory cycle	clock
CDC 6400	7.5 by	.5 $\mu$ s	high frequency
PDP-11/45	2.0 by	.85 $\mu$ s	line
PDP-10	4.5 by	1.65 $\mu$ s	high frequency

The memory "cycle" for the CDC 6400 is quoted as .5 $\mu$ s because extensive memory interleaving makes the .5 $\mu$ s access time rather than the 1.0 $\mu$ s cycle time significant for the considerations here. Clock refers to the clock source used to obtain timing information. The high frequency clocks provide timing accuracy of better than 1 ms and the line clock provides accuracy of 16.6 ms. Thus much longer computer runs had to be made on the PDP-11 to obtain reasonable timing accuracy. Note that each of these machines can execute a full word exclusive or via one instruction. These machines were chosen because one represents the class of high-power processors (CDC 6400), one the class of medium-scale processors (PDP-10), and one the class of minicomputers (PDP-11).

### 3.4 Experiments

Twenty-four separate timing experiments were performed. For each of the two application models, four enciphering method programs were timed, and each of these eight combinations was run on all three computers.

Computer Programs. For each computer eight programs were written. Two programs performed the null transformation, two the short key encipherment, two the long key encipherment, and two the infinite random key encipherment. In each case, one program simulated application Model I and the other simulated

application Model II. Figures 1 and 2 mirror the differences in programming between application models very closely. In Model I programs, whole words were enciphered at a time and enciphering parameters were kept in registers. However in Model II programs, only character-at-a-time encipherment was performed, and furthermore parameters were moved back and forth between memory and registers for each character in order to reflect the fact that other processes would run and thus usurp the registers between character times. In addition, of course, a loop control test had to be added to Model II's programs.

All programs were written in assembly language for the particular machine. In addition to the enciphering task each program contained code needed to obtain CPU time information from the particular operating system.

Computer Runs. Each program was run 50 times on each computer in order to obtain an average. However, there was very little deviation from run to run. For the CDC 6400 and PDP-10 the programs were allowed to loop 20,000 times on each run (i.e. to encrypt 20,000 segments be they words in Model I or bytes in Model II). (See Fri for a statistical analysis of CDC 6400 results). For the PDP-11 with its less accurate timer, the programs were allowed to loop 524,288 times. Initially, variation of the number of loops showed that the additional time introduced by the timing code was insignificant.

For each run, the enciphering rate in by/s and the enciphering time per byte in s/by was computed. Results of all runs were combined to give an aggregate enciphering rate and time for each model/method/computer combination. Only the aggregate rates are presented here.

### 3.5 Timing Computations

In addition to measured times, theoretical execution times based on manufacturer's timing information were computed in order to increase confidence in the measured times. All computed times were within 5% of the measured times.

An example follows.

The segment of code which realizes the inner loop of Model II, enciphering method 2 on the CDC 6400 consists of 7 instructions which fit into 3 words. According to manufacturers' figures, the sequence should take 7.0 $\mu$ s. That is, one character can be encrypted in 7.0 $\mu$ s. The measured figure was 7.26 $\mu$ s for an error of 3.7%. In most cases, errors could be accounted for by memory interference not considered in the calculations.

### 3.6 Cost Computations

A cost figure was computed for each model/method/computer combination. For Model I, the cost of encrypting 1000 cards (= 80,000 characters) was computed. For Model II, the cost of supporting one 300-baud terminal transmitting at full speed for one hour (= 108,000 bytes) was computed. Costs were based entirely on CPU time charges estimated as follows (Cos):

CDC 6400	\$7.00 per CPU minute
PDP-11/45	\$4.00 per CPU minute
PDP-10	\$5.50 per CPU minute.

This is somewhat unjustified because memory usage may be significant in some of the applications (especially for file buffers in Model I). However, these figures should give reasonable estimates of marginal costs of adding encryption procedures to existing system routines.

## 4. Results

The objectives of this study were to determine time and money costs of each enciphering model/method/CPU combination.

### 4.1 Timing Results

Table 1 gives the enciphering rate in bytes per second and the CPU time per byte in  $\mu$ s. For example, the CDC 6400 operating under Model I constraints is

Method	Model I			Model II		
	CDC 6400	PDP 11	PDP 10	CDC 6400	PDP 11	PDP 10
Null	1,380,370 .72	308,000 3.25	259,740 3.85	180,070 5.55	109,000 9.17	87,730 11.40
Short	1,332,540 .75	215,000 4.65	217,860 4.59	137,740 7.26	64,300 15.55	47,250 21.16
Long	897,850 1.11	95,700 10.45	162,600 6.15	81,280 12.30	32,000 31.25	33,670 29.70
Random	159,980 6.25	48,200 20.75	140,650 7.11	38,250 26.14	22,700 44.05	28,360 35.26

Table 1. Measured Raw Enciphering Rates (by/s) and Enciphering Time per Byte ( $\mu$ s)

Method	Model I			Model II		
	CDC 6400	PDP 11	PDP 10	CDC 6400	PDP 11	PDP 10
Null	1.00	1.00	1.00	1.00	1.00	1.00
Short	1.04	1.44	1.19	1.31	1.70	1.86
Long	1.54	3.23	1.60	2.22	3.41	2.61
Random	8.68	6.41	2.85	4.71	4.80	3.09

Table 2. Enciphering Time Coefficients for Processor/Model Combinations

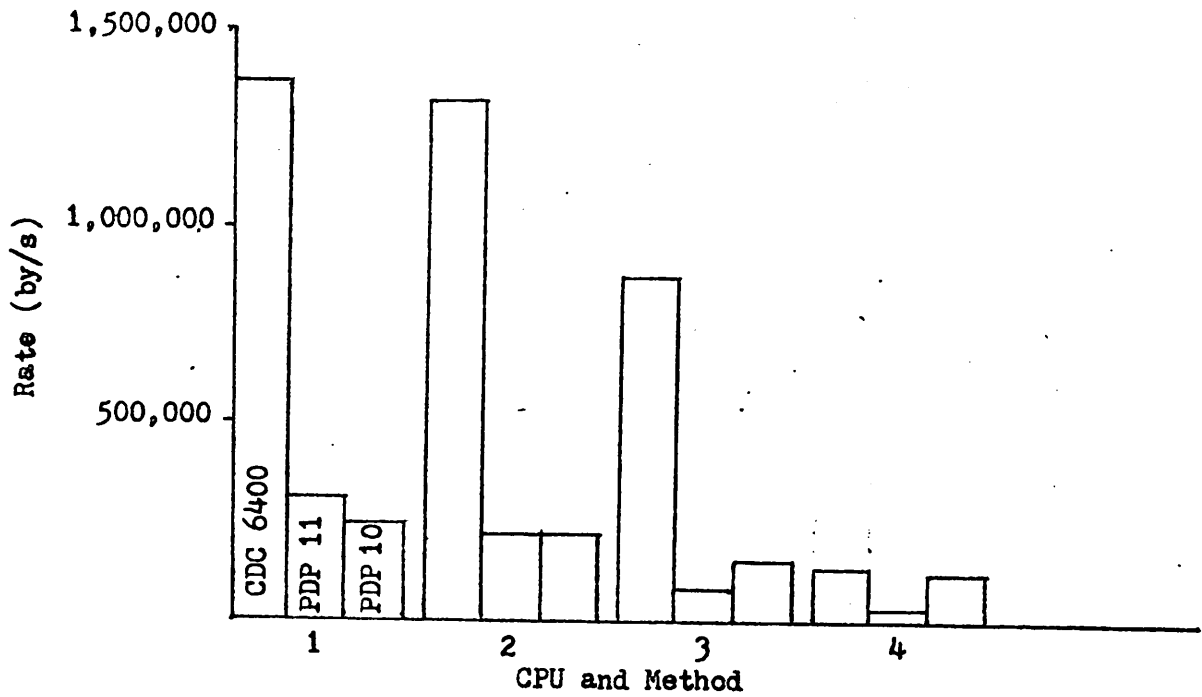


Figure 3. Model I Raw Enciphering Rates

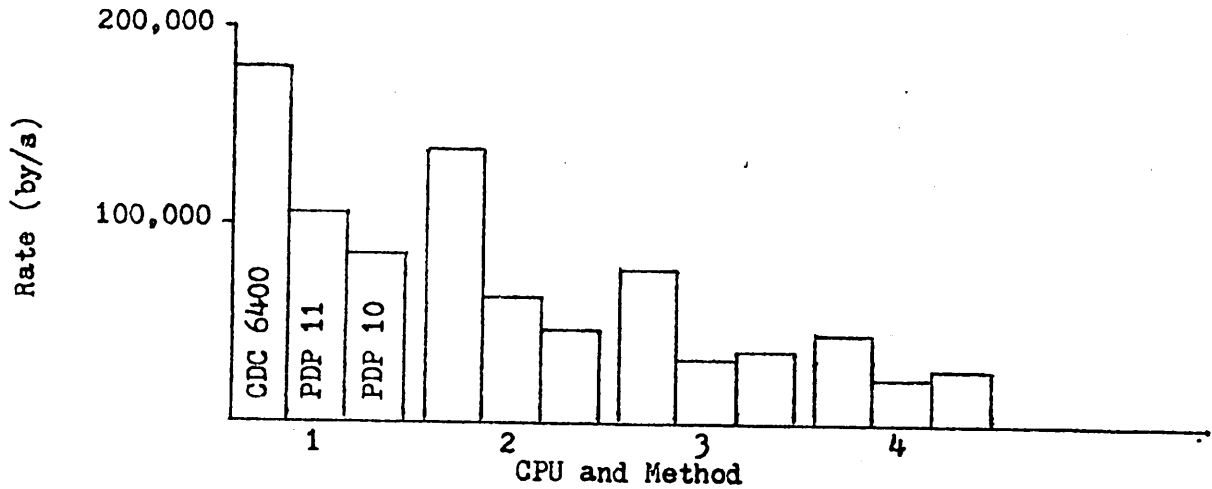


Figure 4. Model II Raw Enciphering Rates

Method	Model I			Model II		
	CDC 6400	PDP 11	PDP 10	CDC 6400	PDP 11	PDP 10
Null	92,020	131,000	95,240	90,040	92,700	144,750
Short	88,840	91,400	79,882	68,870	54,700	77,960
Long	59,860	40,700	59,620	40,640	27,200	55,560
Random	10,665	20,500	51,572	19,125	19,300	46,790

Memory Cycle ( $\mu$ s)	.5	.85	1.65	.5	.85	1.65
Word Size (by)	7.5	2.0	4.5	1.0	1.0	1.0

Table 3. Normalized Enciphering Rates (by/s)

Method	Model I Cost to encrypt 1000 cards (cents)			Model II Cost to encrypt 108,000 bytes (cents)		
	CDC 6400	PDP 11	PDP 10	CDC 6400	PDP 11	PDP 10
Null	.672	1.73	2.82	6.99	6.60	11.29
Short	.700	2.48	3.37	9.15	11.20	20.95
Long	1.04	5.57	4.51	15.50	22.50	29.40
Random	5.83	11.06	5.21	32.94	31.72	34.91

Table 4. Costs of Encryption



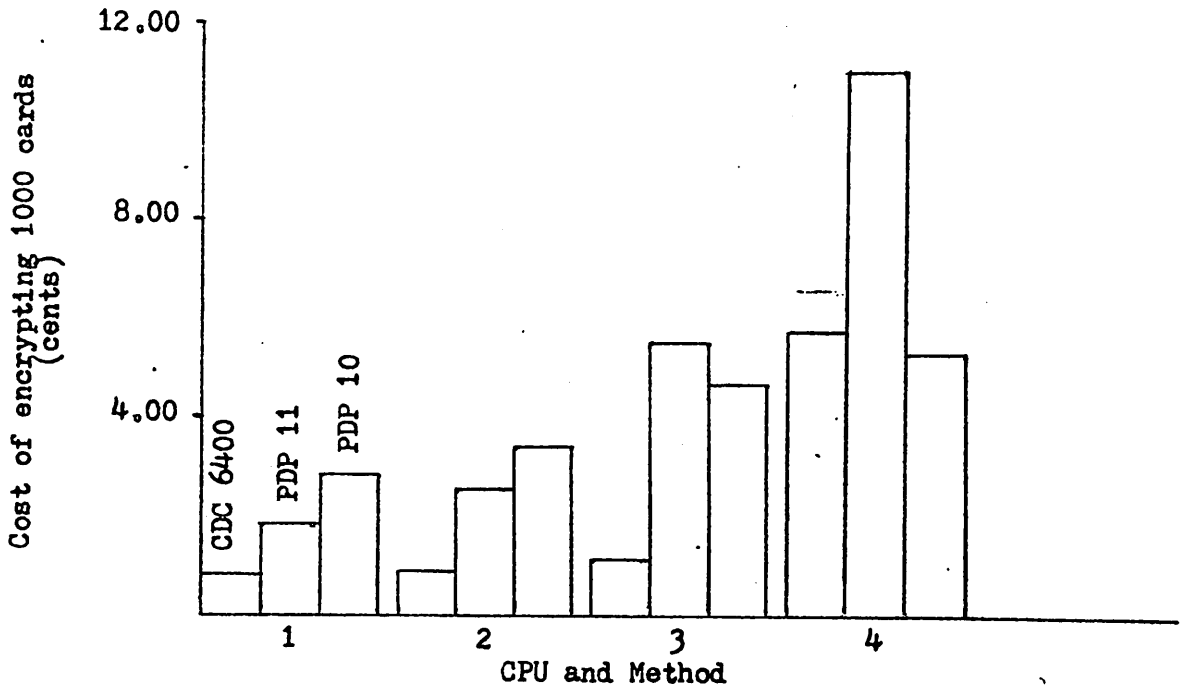


Figure 5. Model I Encryption Costs

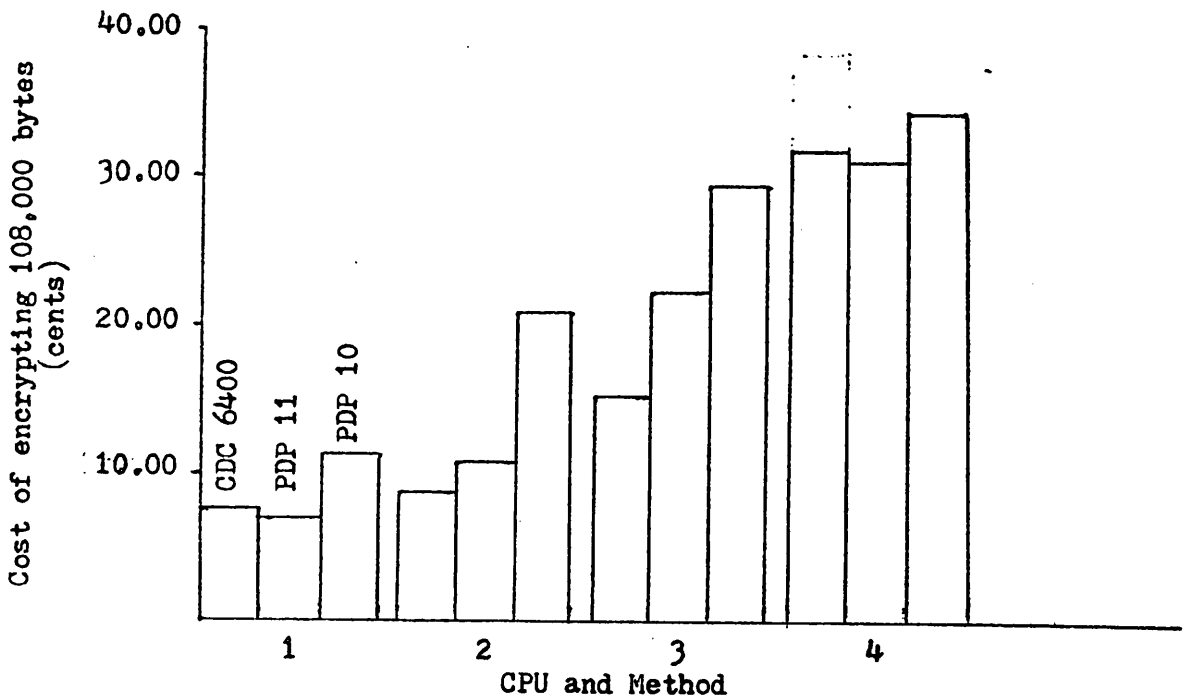


Figure 6. Model II Encryption Costs

capable of enciphering 897,850 by/s using a long key method; this is equivalent to an enciphering time of  $1.11\mu\text{s}$  per byte. Figures 3 and 4 show this information graphically.

Table 2 gives the enciphering time coefficients (Fri) for each model/CPU combination taken across all methods. The enciphering time coefficient is the ratio of the time taken to encipher a segment to the time needed to perform the null transformation on that segment. Thus, for the example of the last paragraph, the enciphering time coefficient is  $1.11/.72 = 1.54$ . That is, it takes 1.54 times as much CPU time to encipher a segment using the long key method as it takes to simply move that segment.

Table 4 gives the estimated costs of encryption based on the time figures in Table 1 and the accounting figures mentioned previously. Figures 5 and 6 depict these figures graphically. Note that the amounts of data considered under Models I and II differ. This was done in order to anchor the cost results in the real world-related to each application model.

#### 4.2 Gross Analysis

Since the null encryption method effects no transformation of the input it was fastest in all cases. The short key method requires the addition of an exclusive or operation between input segment and key and thus takes somewhat longer. The long key method requires a table lookup, exclusive or, and maintenance of a pointer into the key table, and thus takes even longer. Finally the random key method requires calling a random number generator for computation of the next pseudo random number by a linear congruential method and thus requires a multiplication and an addition to generate the key, and then an exclusive or. In Tables 1 to 4 the trend in execution time with increasing complexity is apparent.

Since in Model I a word is enciphered at a time and in Model II only a byte

is enciphered at a time, one would expect a ratio of about the CPU's word length in bytes between comparable boxes in I and II of Table 1. For example, for the CDC 6400 the ratio of the Model I to Model II null transformation is  $1,380,370/180,070 = 7.67$  and the word length is 7.5 by. This, however is only a rough rule since other factors influence this ratio, most notably the differences in program structure for Model I vs. Model II.

Obviously, costs increase with the complexity of the enciphering algorithm. One would expect, however, that for a given model and enciphering method, the cost of encipherment would be roughly comparable over CPU's (given well known economic laws). For Model I we have cost differences up to a factor of  $5\frac{1}{2}$ , and for Model II the differences go up to a factor of 2. These differences are due to at least two factors:

- 1) Charges for CPU time may be based on factors other than the power of the CPU and
- 2) Inaccuracies in CPU time charge estimates.

Nothing more will be said about costs since only preliminary estimates were intended.

#### 4.3 Detailed Analysis

It is the intent of this section to account for the differences in enciphering times between processors based on CPU characteristics. The contributions of both hardware and instruction set characteristics will be considered.

One comment should be made before beginning, however. The random key method for Model I on the CDC 6400 requires two calls of the random number generator. This is because the generator produces only a 48-bit random number, and the word is 60 bits long. This anomaly does not arise on the other two machines.

Hardware Contributions. Two main hardware factors effect enciphering rates,

namely word size and memory cycle time. With a larger word size, more data can be manipulated at once (this, of course, is applicable in Model I but not Model II). Similarly with a faster memory cycle, data can be manipulated more rapidly and instructions can be fetched more rapidly.

Let  $R$  be the enciphering rate determined in a box of Table 1,  $C$  the memory cycle time in  $\mu s$ , and  $W$  the word size in bytes. Then we define  $N$  the normalized enciphering rate by

$$N = R(C/W) \quad (9).$$

Table 3 presents the normalized enciphering rates. Whereas in Table 1 we have rate differences across processors of up to  $9\frac{1}{2}$ , in Table 3, these differences are reduced to at most a factor of 3 (except for the pathological case mentioned in the second paragraph of this section); the factor here is typically 1.5 to 2.0. We believe that this normalization accounts handily for the primary hardware factors.

Instruction Set Contributions. Extremely detailed analysis is possible here on several fronts. To shorten the presentation, one microscopic example will be considered followed by broadbrush treatment of additional factors.

Table 2 will be the primary vehicle.

Consider the enciphering time coefficients for Model I, short key enciphering. The CDC 6400 exhibits a near unity coefficient while those for the PDP-11 and PDP-10 are considerably larger. This stems from the peculiar CDC 6400 architecture which allows loading into registers 1 to 5 only and storing from 6 and 7 only. For the null transformation we load say register 1, move its contents to register 6, and store from register 6. For the short key method, we replace the move with an exclusive or with a previously loaded key. A programming peculiarity which required the inclusion of a NO OP instruction introduced the small additional time. On the PDP-11 the null move is accomplished

by a single memory to memory move instruction. However, three instructions are needed to effect the short key method, namely move to register, exclusive or, store. Thus the rather large time increase is seen. In the PDP-10 only a single instruction is added to a 4 instruction loop, namely exclusive or to register; thus the increase in time is smaller.

With reference to the enciphering time coefficients in Table 2, two general points of comparison between processors' efficiencies can be made:

1) PDP-11 coefficients in general grow the most rapidly. This occurs because the instruction set architecture is such that as additional algorithmic complexity is added, additional or multi-word instructions must be used, whereas with the other processors simple modifications within instructions often suffice.

2) PDP-10 coefficients seem to grow the most slowly, perhaps due to the fact that the PDP-10 has a relatively small spread in instruction execution times (about 4 to 1) when compared to, say, the CDC 6400 (11 to 1) on the instructions added to the enciphering programs as their complexity increased.

Only these two points will be mentioned here, but any aspect of inter-processor comparison will yield to detailed analysis of instruction set architecture and timing.

#### 4.4 Accuracy

The accuracy of timing of CPU operations is of concern in estimating the validity of these results. It is believed that the figures here represent good working estimates so only brief consideration of their accuracy will be made. This belief is based on 1) close agreement between theoretical and actual timing results and 2) use of various methods to eliminate timing errors.

Friedman and Hoffman (Fri) performed a careful statistical analysis of their results on the CDC 6400 and quote a 95% confidence level for their timing figures. It is believed that their analysis of timing accuracy is

applicable here since tests were performed at the same installation. In any case, observed accuracy in the present study was typically better than 1 part in 10,000.

The PDP-10 clock is inherently more accurate than the CDC 6400 system clock because its handling routine is interrupt driven. Variance over test runs was practically nonexistent. Accuracy here was typically 1 part in 25,000.

The PDP-11 line frequency clock was quite imprecise in that over any interval it is guaranteed to be accurate only within 16.7ms. Timing accuracy here ranged from 1 part in 200 to 1 part in 1400. Thus, only three significant figures are quoted in PDP-11 results (in many cases, this should be reduced to two).

## 5. Conclusions

The time taken to encipher streams of data with two application models and three additive encryption methods was measured on three distinct computers. For comparison purposes, the time taken to effect a null transformation was also measured. Enciphering rates varied by a factor of 60 over all applications, methods, and computers. Rates decreased with increasing complexity of the enciphering method. Monetary costs were also estimated, and these varied by as much as a factor of 16 (6) over all methods and CPU's in Model I (Model II); overall, these costs were quite low. It was found that much of the difference in rates between CPU's could be accounted for by word size and memory cycle time. Further differences are accounted for by appeal to differences in detail of instruction set architecture and instruction timing. Results indicate that a CPU should be chosen carefully to fit the particular enciphering application considered while providing maximal efficiency.

## 6. Recommendations

Several specific recommendations are implied by this study:

- 1) Since only a small sampling of enciphering methods was considered, an expanded study should be made which would include other methods such as Lucifer.
- 2) More realistic figures could be obtained by e.g. adding encryption code directly to a COPY utility or the communication interface procedure of a given system.
- 3) The choice of a processor to perform encryption should be based on the particular application at hand; as seen in Model II figures, raw computing power does not imply significant cost effectiveness. Benchmark data stream simulations should be set up and tested on several computers.
- 4) Standards for coding encipherment simulation and benchmark programs are needed in order to insure comparability between implementations. It is suggested that application models be formalized and program schema be agreed upon.

## Appendix I

### Preliminary Investigation of Lucifer

Lucifer was deemed inappropriate for inclusion in this study for the following reasons:

- 1) Its applicability to Model II is questionable because it deals with blocks of 16 bytes instead of single bytes.
- 2) Its complexity is an order of magnitude above that of the methods considered here and furthermore the algorithm is not in the spirit of the other algorithms considered (i.e. successively more complex methods of key generation). Thus, in a sense, they are incomparable; Lucifer would have been a somewhat ad hoc addition to the experimental design.
- 3) The cost effectiveness of software implementation of Lucifer is questionable.

However I have performed a preliminary analysis of Lucifer's performance and cost, and the results are given here. First, existing results are considered.

Benedict wrote an assembly language implementation of Lucifer for the Honeywell 6180 (Ben). He achieved an enciphering rate of only 2910 by/s for a byte enciphering time of  $344\mu\text{s}$ . Since the Multics (block) communication interface program required only  $100\mu\text{s}$  to process a character, he concluded that serious degradation in performance would occur if Lucifer were used on most terminal communication. Generalizing this result adds weight to the inapplicability of a software Lucifer system to general terminal communication. Estimating the CPU time charge for the 6180 at \$5.00/CPU minute, encryption of 1000 cards would cost about \$2.30 and support of a 300 baud line for 1 hour would cost \$3.10. These figures are significantly higher than those for the methods considered in the text. However, tradeoffs of degree of protection vs. cost must be considered.

A preliminary program has been written for the CDC 6400 and manufacturer's



timing figures indicate an attainable enciphering rate of 8000 by/s or  $125\mu\text{s}/\text{by}$ . This gives costs at least 20 times greater than those for the random key method used here. Encryption of 1000 cards would cost \$1.16 and support of the terminal as above would cost \$1.60. These figures are given credence by the estimate that the CDC 6400 is about two to three times as fast as the Honeywell 6180.

The above figures should be compared with those determined for a hardware Lucifer system. A rate of 96,970 by/s or  $10.31\mu\text{s}/\text{by}$  was achieved. No estimates of cost effectiveness were available.

## Appendix II

### Comparison of Results with those of Friedman & Hoffman

The following table compares the results given in (Fri) with results determined here for the CDC 6400 Model I programs.

	(Fri)	Present study
Null	4.76	5.40
Short	4.78	5.63
Long	8.25	8.32
Random	20.05	46.88

Time to Encrypt 7.5 byte word ( $\mu$ s)

Differences in the first three algorithms can be accounted for by single instruction differences in the loop which processes a word.

The large differences in the random key algorithm is due to the calling rather than in-line invocation of the random number generator. Only descriptions (no listings) of random number generators could be obtained for the PDP-10 and PDP-11. It had been decided that existing generators should be used because their correctness could be reasonably assumed. The common denominator implied by these two constraints was that the system routines be called rather than modified and placed in-line. Functional similarity of routines was assured by close examination of their specifications.

It remains to account for the timing differences explicitly. Timing for subroutine call and return for the CDC 6400 is as follows:

call subroutine	2.1 $\mu$ s
jump to return instruction	1.3 $\mu$ s
jump back to calling routine	1.3 $\mu$ s
TOTAL	4.7 $\mu$ s

The random number routine must check its argument to determine which of several functions it is to perform. This requires:

load parameter to X register	1.2 $\mu$ s
jump if non-zero (fails)	.5 $\mu$ s
non-overlapped instruction fetch	.2 $\mu$ s
penalty for jump in slot two of instruction word	.1 $\mu$ s
TOTAL	2.0 $\mu$ s
GRAND TOTAL	6.7 $\mu$ s

All the above must be done twice for a total of 13.4 $\mu$ s. However, we still have to account for 13.43 $\mu$ s/word or 6.72 $\mu$ s per call of the generator. This time comes from the required loading and storing of registers to avoid conflicts with the generator. However a few microseconds are left unaccounted for.

It is suggested that the standard program schema mentioned in the recommendations (p. 15) use in-line invocation of the generator.

## 7. References

- (Hof) Hoffman, L. J. Class Notes for Computer Science 244. University of California, Berkeley. Fall 1974.

These notes were used in a graduate level course in computer security engineering. They include consideration of hardware and software safeguards, operating systems, physical and administrative security, and models of secure systems. They include broad coverage of encryption.

- (Fri) Friedman, T. D. and Hoffman, L. J. Execution Time Requirements for Encipherment Programs. Communications of the ACM. Vol. 17, No. 8 (August 1974) pp. 445-449.

This paper reports a study similar to the work done here. A null enciphering method and three additive methods were timed on the CDC 6400. The enciphering time coefficient was introduced.

- (Pet) Peterson, H. E. and Turn, R. System Implications of Information Privacy. Proc. AFIPS 1967 SJCC. Vol. 30, AFIPS Press, Montvale, N. J., pp. 291-300.

The paper lists many threats to information privacy and categorizes countermeasures into the areas of access management, processing restrictions, threat monitoring, and privacy transformations. It provides a good cross tabulation of threats and countermeasures.

- (Ska) Skatrud, R. O. A Consideration of the Application of Cryptographic Techniques to Data Processing. Proc. AFIPS 1969 FJCC, Vol. 35, AFIPS Press, Montvale, N. J., pp. 111-117.

In this paper, two cryptographic techniques are described. The first is a substitution system using an additive method with two key memories and the second is a route transposition system. Consideration is given to the security of the methods.

- (Bar) Baran, P. On Distributed Communications: IX. Security, Secrecy, and Tamper-Free Considerations. The Rand Corporation, Memorandum RM-3765-PR, August 1964.

This paper states Baran's principle regarding secrecy about secrecy. It presents some basics of cryptography, describes a real world application to a distributed message communication system, and provides a critical examination of this system.

- (Fei) Feistel, H. Cryptography and Computer Privacy. Scientific American, Vol. 228, No. 5 (May 1973), pp. 15-23.

The paper describes recent work done at IBM on nonlinear block ciphers (the Lucifer system). It provides a lucid description of the problems involved in producing a ciphering method which is highly resistant to cryptanalysis.

- (Kah) Kahn, D. The Codebreakers. (Macmillan, 1967)

This book provides an extensive and fascinating account of the techniques and history of cryptography from ancient times to the end of World War II.

- (Car) Carrol, J. M. and McLelland, P. M. Fast "Infinite-Key" privacy Transformations for Resource-Sharing Systems. AFIPS 1970 FJCC, Vol. 37, AFIPS Press, Montvale, N. J. pp. 223-230.

This paper describes privacy threats and countermeasures. It discusses a two-stage random number generator and an implementation of the associated privacy transformation. It also gives an analysis of the goodness of the generator and gives timing figures for an encryption using the generator on a PDP-10/50.

- (Ben) Benedict, G. G. An Enciphering Module for Multics. MAC Technical Memorandum 50. Massachusetts Institute of Technology, Computer Systems Research Division, Project MAC.

This memo describes both a PL/I and an assembly language implementation of the Lucifer system on the Multics System. Complete program listings are provided and results of timing experiments are summarized.

- (CDC) Control Data 6400/6500/6600 Computer Systems Reference Manual. Pub. No. 60100000, Control Data Corporation.(1969).

- (P10) PDP10 Reference Handbook. Digital Equipment Corporation (1970).

- (P11) PDP-11/45 Processor Handbook. Digital Equipment Corporation (1973).

These three manuals contain all the information needed to program the particular computer in machine language. In addition, each gives complete instruction timing information.

- (Unx) UNIX Handbook. University of California, Berkeley. Department of Electrical Engineering and Computer Sciences.

This manual contains reprints of several manuals written at Bell Laboratories which describe the operation of the UNIX timesharing system for the PDP-11/45.

(Cos) Cost estimates for CPU time.

These are my own estimates of CPU time charges. The CDC 6400 figure is that actually charged for a particular grade of service at the University of California, Berkeley Computer Center. The PDP-10 figure was obtained in a private communication at the Lawrence Livermore Laboratory. The PDP-11 figure has been widely circulated in the literature of small timesharing companies.