DATA SECURITY AND COMPACTION TECHNIQUES

by

TIMOTHY FITZGERALD MURPHY
B.Sc. University of Liverpool, 1963

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the Department
of
MATHEMATICS

We accept this thesis as conforming
to the required standard

[redacted]
Dr. I.P. Sallaway

[redacted]
Dr. B.R. Johnson

[redacted]
Dr. F. Ruskey

[redacted]
Dr. W. Muir

[redacted]
Dr. L.P. Robertson

Supervisor: Dr. I.P. Sallaway

ABSTRACT

This thesis studies two common problems faced by computer analysts.

The first is data compaction; often this is required for economic reasons or to make best use of scarce external storage and associated interfaces.

The second problem is that of protecting data which has been obtained by unauthorized persons from exposure, by cyphering or encryption. This exposure can be accidental, such as a computer tape being lost, or a deliberate interception of a cyphered transmission.

The basic aim of this thesis is to examine the interrelation between compaction and security cyphering. While a connection appears probable little work has been done on this subject.

A number of data compaction techniques are evaluated and it is shown that data compaction is justified on modern computer systems. Theoretical limits to data compaction are given and, generally lower, practical limits are discussed.

The effect of data compaction on cypher security is demonstrated theoretically. Several modern computer cypher techniques are reviewed; data compaction is shown to be required to increase their security. Some weaknesses in modern computer security are reviewed and methods are given to increase this security. Methods of compacting computer data prior to encryption are given.

Computer Programs are supplied in the appendices for security encryption.

Examiners:

[redacted]

Dr. I.P. Sallaway

[redacted]

Dr. B.R. Johnson

[redacted]

Dr. F. Ruskey

[redacted]

Dr. W. Muir

[redacted]

Dr. L.P. Robertson

TABLE OF CONTENTS

LIST OF FIGURES

'The three basic requisites of a code; it should be easy to read, easy to write and difficult to detect'. FRANCIS BACON (1606)

SECTION I. INTRODUCTION.

The fields of Cryptology (that is the science of codes and cyphers) and of Data Processing overlap in many ways; but the practitioners of each science are largely ignorant of the existence of the other's subject. One area where there has been a merging of the two is in the large U.S. Government Agenices, such as the National Security Agency and the Central Intelligence Agency. The progress made there has been kept as secret as any other investigations by these agencies, however it is known that they make massive use of computers.

This thesis will attempt to correlate secrecy systems with data compaction systems giving particular attention to the practical applications. It will demonstrate that data compression and data security are interrelated. It will also be shown in Sections VI and IX that both can be maximized, all other things being equal, by the largest feasible code block.

It will be shown that, in general, the more compact the coding of a data source the more secure a cyphering of a message from that source will be, and that any redundancy in a message impairs secrecy.

The general theoretical background comes from the field of Information Theory covering Noiseless Coding which is based largely on C.E. Shannon's work since 1948. See also Pierce.

This Thesis is divided into sixteen sections.

Section II covers the trends in computer technology and cryptography over the past 30 years and will show why data compaction, despite the increase in available storage capacity, is required to a greater degree

than ever before. It will also show why the application of compression is more feasible than ever before.

Section III shows the economic advantages of data compaction to users of an average computer system. This applies to both disk and remote communication.

In Section IV some widespread computer data compaction techniques are reviewed and their effectiveness compared.

Section V covers the coding of noncomputer data; this is largely historical, but should give some useful insights.

Section VI will define the terms Entropy and Redundancy. These terms are central to this thesis.

Section VII covers the principles behind coding a message source and gives some examples of computer codings.

Section VIII discusses the entropy and redundancy of computer data.

Section IX gives the definition of cryptographic security — the unicity distance — and shows why security increases as redundancy is removed.

Section X defines codes and cyphers.

Section XI covers irreversible codes or one way codes.

Sections XII and XIII review some new, computer based, cypher techniques and show how their security will be imperilled if data redundancy is not minimized.

Section XIV reviews current methods of computer security and gives some ways in which these can be improved.

The penultimate section contains various ways of compressing and encrypting data allowing for the special nature of various types of computer data.

This is followed by the conclusion.

The majority of the analysis in this thesis is on the encryption and compression of alphabetic (text) data, not because this is overwhelmingly important, but simply because textual data is one of the few forms which lend themselves to a generalized practical solution. Here English, or occasionally Arabic, has been assumed for the language. The analysis of compression and coding of numeric data, computer files, or analogue data is very dependent on the structure. As shown by Rubin, coding can be effectively infinite in the worst case with analogue data; however some guide rules have been given for some of these cases.

Source code will be provided for programs to generate relatively secure and compact codes for data security and compaction, and for programs to compress and decompress such data.

Since it leads to no loss of generality, and fits the practical nature of this thesis, I have assumed binary coding for most of the theoretical background; that is, codes based upon the value of a bit being 0 or 1.

SECTION II. TRENDS IN TECHNOLOGY.

A. Security Demands

Recently there has been a growing concern among the general public about the need for improved computer data security. This has led to an upsurge of research and publications (almost all papers postdate 1970). In the United States legislatures, over 150 pieces of Privacy and Security Legislation were introduced in the first half of 1979 alone. There have been a number of important publications, among the most famous being those of Hellman (1976), Merkel (1978) and Rivest, Shamir and Adleman (1977). All have concentrated on the security aspect without particular concern for economy of resources.

The United Stated Federal Bureau of Standards has introduced a Standard Data Encrypting Algorithm (DES), but recently some doubt has been expressed on its security (Diffie and Hellman, 1977 and Campbell 1978). (See Section XII.) Despite such doubt the standard seems to be increasingly adopted, at least in part, because no really good alternative has been suggested and also because its use is mandated by law for many United States public bodies.

It appears that this, or some other, security encryption method is often not used even preparatory to remote data communication, and only very occasionally before writing to magnetic storage devices such as disk, diskette or tape. It was envisioned that the DES Standard Cryptographic Method could be implemented in a hardware CCD (charge coupled) device which would be fast enough to allow encryption without lessening the speed of a magnetic tape drive. But this has only just

happened as far as I am aware; the user demand has not been there previously.

B. Auxiliary Storage

Since 1956, when IBM[1] first announced the 305 RAMAC device, there has been an almost exponential increase in the size of online data bases which has been matched by the capacity on a single disk pack going from 10 million to 571 million characters installed today. The IBM 3380, with 1200 million bytes capacity, has just been announced for 1981 delivery. This information comes from a 1979 paper by A.S. Hoagland and manufacturer's advertisements. Data access times have decreased some 42 times during the same period (based on the ratio of the 3370 to the 305), and the number of packs to a data base has risen to over 30 in some of the larger sites. There are also a number of slower access mass storage devices which have a capacity in the order of hundreds of billions of bytes, for example the IBM 3850.[2]

Despite this increase in size it appears that the demand for disk capacity is constantly being overrun. This comes from the new applications now being considered feasible, from the general 'Parkinson' law and from the tendency to store data on disk rather than on magnetic tape so that random access techniques can be used. Another cause of growth in demand are the time sharing systems such as VM/370, Honeywell

---

[1] Throughout this thesis IBM designates the International Business Machine Corporation.

[2] The 3850 has a maximum capacity of 472 billion bytes.

TSS, and IBM TSO. Their use puts a premium on response time; this requires online disk data sets at present. Often with time sharing systems, tape is very difficult to use.

C. Central Processors

Over this same time period (1956-1980) Central Processing Unit (CPU) internal cycle times have decreased from 15 microseconds to under 50 nanoseconds, a factor of 300, and main memory sizes have risen from 4K to 8192K, a factor of 2000. Recently IBM's plans for a Josephson Junction (super cooled cyrogenic) Device Computer were outlined by W. Anacker (1979). The eventual expected internal cycle time of this computer would be of the order of 1 nanosecond with 256,000 bytes of 2 nanosecond access cache memory and 64 million bytes of 10 nanosecond random access main memory. The first model, which will execute about 70 million instructions per second (MIPS), was predicted to be out in 1984, with a 250 MIPS machine following.

An alternative technology which is receiving increasing attention is Gallanium Arsenide based circuits. Gallanium Arsenide has an electron migration speed 30 times faster than silicon and dissipates less heat. However, it lacks a resistive oxide and is very expensive in the purified state. Gallanium Arsenide operates at room temperature unlike Josephson Junctions.[1]

Another approach is Very Large Scale Integration (VLSI), using either Silicon or Gallanium Arsenide. VLSI will probably be used in military

---

[1] See Huatek (1980).

applications first.

D. Comparison

During the last few years it has been found that in many computer sites the disk drive data rate, and the disk controller and channel capacity, are the limiting factors in throughput, while the CPU itself is idle a relatively high proportion of the time. Even scientific calculations with little actual data to read or write may become I/O bound due to the paging necessitated by large arrays. This was certainly the case with the British Columbia Systems Corporation 3033 site in the last half of 1978.

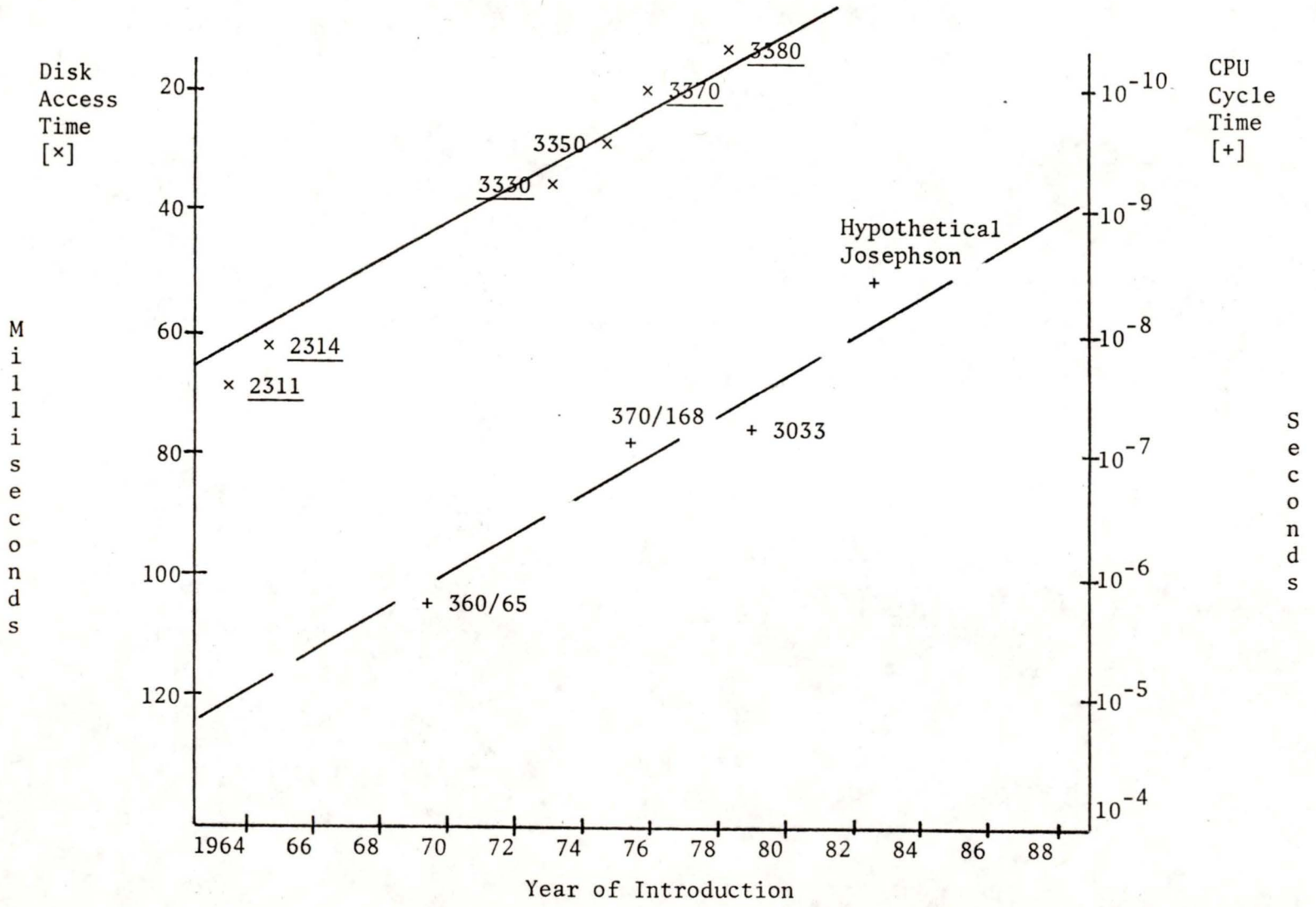
The new Josephson Junction device computers are predicted to be close to today's in cost and over 50 times more powerful, which will inevitably put still more load on the disk drives, controllers and channels. Personally, I feel that the Josephson Device computer will not be marketed before the end of the decade. IBM's profits were decreased by the introduction of the 4030 series, and a new generation in 1984 does not make financial sense.

Hoagland sees no radical new technology used in storage devices before the end of this century. He generally discounts bubble or laser type read/write memories.

Thus the next decade is almost certain to offer a considerable reduction in CPU speeds with a rather more modest reduction in magnetic storage access speeds.

As can be seen from Figure 1 the decrease in CPU time has been exponential while disk access time has decreased only linearly, which indicates the feasibility and desirability of compressing data where

Figure 1. Trends in Computer Speeds



possible. Naturally there are applications which are less suitable for compression due to the unpredictable nature of the data. Paging is an example. However, these are the applications which will be most likely making use of any new storage technology first. Also, as memories grow larger, paging should decrease in importance, or disappear entirely according to some authorities.

The number of instructions actually involved in access to a peripheral device is quite large and the steady increase in complexity of computer operating systems has meant that this number has increased almost as fast as computer speeds have increased. For example on an IBM 360/40 running under OS MFT an input/output (I/O) operation took about 1 millisecond or about 400 instructions. Under MVS, with address translation, an I/O operation can take several thousand instructions, which will take almost as long on a large 370 as on the older 360. Only by using large block sizes and compressed data is one able to cut the number of I/O operations involved and so decrease this overhead. With the shorter time for data transfer, throughput can be increased, particularly in a heavily loaded (Queue Building) system.

E. Remote Data Communication

The technological progress in the field of remote data communication has been far less productive than was the case with CPU performance, and costs have not decreased by the same magnitude. While digital network offerings, such as 'Dataroute' in Canada, and packet switching have cut the cost of low speed data transmission (that is up to 2000 baud), the cost of high speed connection has stayed constant in real terms.

Some form of data compaction has been standard for the last 20 years, but this has often been only a relatively ineffective compression, partly because of the small size and lack of sophistication of some of the remote terminals.

Because of the ease of interception, data security encryption now should be normal, at least before data communication, but this is very far from being actually so.

F. Summary

In summary, today, despite a widespread literature in coding theory, data compaction is uncommon and security encrypting is only slightly less so. This lack of data compaction inevitably leads to extra costs and performance problems due to the I/O bottlenecks which arise. The lack of security encryption is receiving legal exposure which will probably increasingly come before law courts (particularly in the U.S.) in the next few years. It seems fairly clear that these two processes could be combined in that both data compaction and data security could be addressed in one standard computer interface or sub-routine.

SECTION III. ECONOMIC JUSTIFICATION OF DATA COMPRESSION.

A. Magnetic Disk Devices

I have examined a number of computer files and found that the number of blanks (hexadecimal '40' on IBM computers) varies between 30% and 50%, often several blanks are grouped together. Normally the next most common character has an occurrence of less than 5%. Other files will have different characters of different occurrence frequency. For an example of the effect of data compression, one can consider the British Columbia Central Registers System.

The central registry system consists of two keyed (VSAM)[1] master files of approximately 1,150,000 (Serial-automobiles) and 460,000 (Alpha - other chattels) records, respectively. With the free space necessary for expansion as data is added, these files take 470 cylinders of IBM 3350 disk space; this is a little under 260 million characters. At 1978-79 British Columbia Systems Corporation rates, this disk space costs close to $6,000 a month.

It was found that 44% of the average Serial file record and 30% of the average Alpha record are blank; the next most common character has a 2% occurrence. Consider a compression technique which will decrease the disk space of these files by a nominal 33%. All other things being equal there will now be a charge of only $4,000 a month for disk space, though there will be an extra charge for the extra CPU time processing required.

---

[1] Virtual Storage Access Method. An indexed data management system for IBM 370 Computers.

For the central registry system about 0.15% of the file is updated daily, normally by the addition of new records, or more rarely deletion of records, and about 0.10% is subject to access for enquiry. The B.C.S.C. charge for this has been in the region of $280 per day or $6,914 per month. A reasonable assumption for a simple data compression/decompression routine is 10 instructions per character, this is about one second of extra CPU time per day on an IBM 3033. At B.C.S.C. rates (1978-1979) this would add $250 per month or about 12.5% of the saving in disk storage charge. More sophisticated routines could achieve a higher compression factor at little increase in processing charge.

It should be noted that under these circumstances above, the number of chargeable disk input/output transactions would decrease, due to a larger number of buffer 'hits'. This would provide some counteracting cost saving which would at least partially offset the extra CPU time cost. If 20% of the CPU cost of processing is ascribed to disk I/O[1] and the buffer hit ratio[2] can be increased by 10%, then CPU processing charges would be decreased by approximately $140 per month, which would be offset against the $250 extra for compression routines.

The critical factor is the effect of compression on the I/O buffer hit ratio. This in turn depends on the locality of reference which, for central registry, is relatively high on the more frequent updates but will likely be lower on enquiries. These calculations are assuming pure CPU charging rather than charging for channel and disk controller

---

[1] Probably low with VSAM.

[2] This measures the number of times disk I/O is unnecessary due to the required data already being in the computer memory.

utilization as well. When channel usage is taken into account, maybe half the cost would come from disk I/O activity.

In fact, compression will often more than offset the cost in most environments given a fair charging algorithm which charges for all facilities in proportion to their cost. At the Bell Northern Research Computer facility, D. Williams found that a cryptographic copy program which also compressed data was cheaper to run than a straight file copy under VM/370, due to the reduced number of blocks being written out. (See Williams, 1974.)

If the cost of computer services follows the trends which were mentioned in Section II, one would expect the cost of data compression to become negative even if the cost of disk space is ignored. Since channels and disk controllers are perceived to be the major system bottleneck, the charging algorithm will be adjusted to penalise their use and reward economies such as compression.

With the figures for central registry above, which are not atypical, it might be wondered why data compaction is not more widely adopted. The reasons seem to be the high cost of programming, the fact that some file redesign is often necessitated for the highest compression and the general fear of altering 'The Data'. Such work is not in the budget, and so is difficult to schedule. Only if compression routines were integrated into the system by some standard subroutine interface, thus becoming entirely transparent to programs, would data compaction find a widespread use in today's environment.

Some recent software, such as IMS/VS Release 1, allows the user to specify a compression/security user exit (subroutine) for particular

data fields which is entirely divorced from the actual IMS Programs. This is a very good step. Economy and the security aspects will encourage the use of similar data manipulation subsystems in the future (*viz.* the IBM programmed cryptographic packages - 5740-XY5), and this should ease the introduction of similar, or combined, compression routines.

J.P. Considine and J.J. Myers have pointed out that even if the user of a direct access device can get sufficient storage for his needs, this necessitates considerable 'housekeeping',[1] which is inherently non-productive work. For an example, under VM/370, users often have to spend a considerable amount of time compressing and decompressing their data files using the VM pack utility. If compression and security encryption could be done automatically by the system, this would free up user time for more productive work. (See also Williams 1974.)

B. Remote Data Communication

Contrary to the case with magnetic disk devices, the advantages of compression on data communication devices are widely recognized and implemented, because of the very high relative cost of data communication and the need to make maximum use of the communication lines. Unlike the cost of processing, the cost of data communication has been relatively static over the past 20 years. Also the availability of high speed communication lines has always been low, due both to cost

---

[1] Disk space maintainence.

and long delivery lead times. Since data often had to be translated into another code before transmission, there was not the same objection to the extra programming and data manipulation of compression.

The lowering of cost, due to new technology, which has been seen in computers and their peripherals, has had little impact on data communication up to now. New technologies for data communication certainly do exist (e.g., microwave, satellites, and, lately, glass fibre optics), but, with the possible exception of the glass fibre optics, all require such massive base investments that the savings to the end user are understandably modest. It is possible that by the end of the decade glass fibre based transmission will be sufficiently widespread to allow real cost savings.

C. General

In this section, it has been assumed that charges for computer resources bears some relationship to their cost. In an environment where problem program CPU time is the sole chargeback mechanism, or where there is no charging done at all, data compression is unlikely to be of economic benefit to a user.

As noted earlier computers are getting much more powerful, and in the future the use of more sophisticated compression routines can be foreseen. These will probably be combined with security.

In many cases, despite its advantages, data compaction is not considered until a situation is reached where hardware limitations make it mandatory. Often only when a particular application cannot be implemented, or continue processing, with the hardware in place, and when no

more can be obtained in a reasonable time will these matters be looked at. Similarly, security encryption is not considered until a major security violation is detected. Under these circumstances, the compression or security routine will probably be an inefficient patch, a far from optimum solution.

Compression and security of data should be planned at as early a stage as possible. The use of special purpose micro computers could becomes the best solution in the future. These should be commercially available soon.

SECTION IV. SOME COMPUTER COMPACTION TECHNIQUES.

Data compression on computers is generally undertaken for reasons of economy. The two major applications are data storage, and data communication. They have much in common, though security is a considerably greater problem in the second case. As will be shown in Section VIII the redundancy in computer data is fairly equally divided between the restricted probability of character occurrence and the large number of blanks grouped together.

The effect of data compaction, as a basis for security, is normally to decrease the redundancy of data. It also, particularly in the more sophisticated routines, increases the number of possible cypher keys as well. As will be seen later, only the most sophisticated of these codes are secure on their own, but many act as a good basis for other techniques. As shown in the sections on DES and public key cryptograms (Sections XII and XIII) data compression is often nearly mandatory for real security.

A. Code substitution

This technique makes use of the restricted character set used in many computer applications.

The most common form of compression is 'packing' numeric data or converting it to binary format. When converted to packed format, each character position after the first (to allow for the sign) contains two digits. Thus N packed digits will take (N+2)/2 positions.

Conversion to binary gives a potential for still higher compression, but this is not often done because of the difficulty of manipulating

words of other than 2 or 4 characters in most high level computer languages (e.g., COBOL or FORTRAN).

Another very common way of compressing data is to convert to another form of code by a simple substitution, particularly when the input code is in ebcdic with its cost of 8 bits and a sparse usage. Codes that are used include BCD (binary coded decimal), which is a 6 bit code and ASCII, which is 7 (after dropping the parity bit).

In both cases certain problems can arise. Firstly, many computer characters cannot be expressed by the shorter codes. For example, BCD cannot be used to code packed numeric data, or full Arabic text, though the latter is possible just with ASCII. On IBM 360/370 computers neither can be used to code compiler object code. All normal printable characters can be expressed in all three codes and ASCII can even code the lower case English alphabet. Secondly, the collating sequence (sorting order) is different in all three codes. This can lead to problems in sorting or with indexed look up; of course, an artificial code can be derived to solve that problem.

Despite the difficulties this technique is very attractive, particularly as many large computers have powerful instructions to convert codes and generally highly efficient implementations in high level languages such as COBOL or PLI. In the case of the Honeywell 66 level architecture there is an instruction which will translate one string of up to 2048 characters under control of a table and change the length of each character as it is moved.

This conversion technique can be combined with blank compression.

Another possible way of using short bit strings to code a large number of characters is translation of the input character set to correspondence or Murray (telegraph) code. This makes use of the low Markov transition probability from a numeric state to an alphabetic state and the converse. (See Section VIII.) This coding uses only 5 bits with one code reserved for numeric shift and one for alphabetic shift; all others doing double duty. The resulting 60 possible codes are sufficient to represent all upper case letters of the alphabet, numbers and common punctuation marks. These were the characters to be carried on the paper tape which was used to store data and then transmitted at a higher data rate. It is now used on low speed communication systems such as telex, and a similar code is used on the IBM 2741 terminal.

A more sophisticated code, with many states, could represent the entire ebcdic character set, and, as the Markov tables in Figure IV show, would be fairly compact. This could well be the best method in special applications.

B. Blank Compression

Blank compression is another very common data compaction technique which makes use of the very large number of conjoined blanks in computer data. Blank compression is often applied as a first step, before other data security or compression techniques.

The normal methodology is that two or more blanks are replaced in the data record by a special nontext character containing a count of the number of occurrences. (In versions based on the ebcdic character

set, use is made of the fact that no printable character starts below hex '40'.)

Even simpler is to delimit the end of a blank string by a special marker character if the actual number of spaces dropped is unimportant. This technique is very widely used in such machines as the IBM 3780, a hard wired 'remote job entry' terminal introduced in 1972. Another effective technique is to replace all trailing blanks in a data file with a nontext character and to use a compiler 'string' function to expand the data into a fixed format.

The effective compression will depend on the number of blanks in the data; thus it is particularly effective for the type of print files that are met on RJE type terminals. As an example, blank compression and file conversion to variable length reduced a sample file by 45% change using 800 byte blocks.

An even simpler compression technique is to convert the fixed length data file to variable length and to strip the blanks at the end of each data record. This requires less programming and is often surprisingly effective.

Some blank compression techniques depend on special computer hardware. On remote video type display devices with direct cursor addressing, such as the IBM 3270, only 5 or more blanks are compressed, as it takes four control characters to reposition the cursor; but this is often very effective in achieving compression on communication lines.

The fairly good results achieved by blank compression on computer data arise from the relatively enormous redundancy of most fixed form computer data; it will have little effect on other types of data such

as straight English text. However 20% blank compression is sometimes possible even with computer object text (report headings, etc.).

C. Trailing Blank Suppression

Another blank compression technique is used by database systems which inherently have knowledge of the types of data in the various fields and have to be able to handle these fields in isolation.

For example, consider ADABAS, an increasingly successful commercial database system produced by software ag. In this technique each field of the data files is compressed separately. A great advantage of such an integrated approach is that data compression is transparent to the user, which removes some of the political disadvantages of a programmed compression routine. Trailing blanks are removed from character fields and leading zeros are removed from numeric fields (these are also packed).

Software ag claims an average compression factor of 48% for ADABAS with minimal extra CPU time on accessing the file. In many cases, particularly with the 'monster' file design recommended by software ag, several consecutive fields could probably be all nulls or zero's. In this case a single such field is presented by a binary byte of '11000001', two such fields together by '11000010' and so on up to 64 null fields.[1]

The security of the file under browse inspection is little more than that of uncompressed data for either of these techniques, though it

---

[1] If the field is suppressed the field is also omitted from any indexes.

would seem that the ADABAS files would be more secure than simple blank compressed data particularly where numeric fields are concerned. In order to increase security, ADABAS allows the user to supply a hashing code of 8 numeric digits which is used to cypher the data giving at least a high order of browse security — this is basically an additive superencypherment.

D. Duplicate Character Compression

This is an extension of blank compression in that all strings of three or more identical characters are compressed. This requires a minimum of two characters for a string as against only one for blank compression. Since the space character is normally the only one which forms runs of any length, the compression factor can be less than occurs with simple blank compression unless a special set of characters are kept for blanks.

An example of this in common use is the 'packed' file of the conversational monitor system (CMS) under VM/370.

The user is allowed to specify the most common character in his file if it is not blank. This feature was found to cut data length by a further 5% on testing one sample of data.

In another test a sample (COBOL) program of 116,888 characters with a computed Bernoulli[1] entropy of 2.535 bits per character was reduced by the CMS pack command to 42,188 characters with a computed entropy of 5.631 bits per character.

---

[1] See Section VI.

This technique is also used in the RSCS component of VM/370, a multileaving RJE protocol.

The browse security of this method is only a little more than simple blank compression and probably less than that of an ADABAS type compression.

E. One and Seven Eights Code

This and the next method attack both forms of redundancy together; as can be imagined, they give rather better results.

Rather than collecting runs of characters other than blanks, which can give a rather low extra compression factor, the most commonly occurring characters are converted into a 6 bit code, and the remaining characters are represented by 2 six bit codes. The normal choice is to choose 26 alphabetic characters, 10 numeric digits and 12 special characters leaving 16 codes to make up the pairs.[1] On a typical file this gives close to 25% compression on top of that achieved by blank compression.

This method was used in the IBM Houston Automatic Spooling Program (HASP) for communication with intelligent RJE terminals, where it was called 1-7/8th code. It is also the data compaction technique used in 'LIBRARIAN', a source program maintenance package marketed by Applied Data Research.

This technique suffers from the disadvantage that the 'compressed' string can be longer than the original. 'LIBRARIAN', when it detects

---

[1] Some of these codes can represent special controls, as we only need 208 pairs out of the 256 possibilities.

this, will not compress the record, in which case much of the data security will be lost. Blank compression is still applied.

Some additional compression can be gained with COBOL programs, in that COBOL reserved words are replaced by special strings. (This is a step towards constructing a special alphabet.)

Contrary to precious examples the browse security is quite high. All the common, printable characters are replaced by 6 bit strings which can start on any even bit boundary. Because it will normally be combined with a different six bit character on each occurrence, frequency analysis will be an order of magnitude more difficult than is the case for a simple substitution code. Even a small degree of further hashing, say by a 'Boolean algebraic exclusive' or with a long random string, would make decryption analysis very time consuming and expensive.

The computational effort to compress the data is quite low, perhaps 10 instructions per character, and the cost is more than compensated for by the reduced cost of the CPU time necessary to do the I/O servicing.

F. Shrink-IMS & Shrink-2.

These are commercial packages produced by Informatics Inc. which uses Huffman code[1], optimally, after calculating the optimal coding set from a sample of the data. They claim a 50-70% compression ratio at a cost of about 1 CPU second per 100,000 characters on an IBM 370/158 — about 10 instructions per character. This is a reasonable estimate of what would be achieved with Huffman code on ebcdic, but, as will be shown, Huffman code on a single character basis does not approach maxi-

---

[1] See Section VII.

mum compression.

The exact nature of the algorithm used is a trade secret but, judging from the low number of instructions used on each character, and the modest compression ratio, it seems unlikely that the package uses Huffman code on anything other than single bytes.

G. Sparse Lists or Threaded Lists

All the above methods assume that the data is of the basic form of English, or other languages, and that the text is to be interpreted by human understanding. Sparse lists, or arrays, are more easily comprehended as tables of entries, most of which are null. In this case the data is already in packed form, (or floating point), so almost all values other than nulls are equally likely. Thus, a technique is used which codes the number of null fields in some form, either as a pointer to the next non-null field, or as a count of the number of omitted table entries; basically it is a form of blank compression tailored for scientific arrays. A rather similar method is the null compression described for ADABAS above. Another technique, developed by M. Visvalinham and used on census data, called 'indexing with coded deltas', records the distance to the next non-null value.

H. One Way Codes

In the case of one way codes, as described in Section XI it is unnecessary for the output string to be as long as the input string. If the coded result has enough bits to make the chance of an accidental match effectively zero, the output string is all that should be

retained for later comparison. Just 24 to 32 bits should give adequate security.

I. String Substitution

In many cases in computer files certain values are predominate; for instance in a file of names the word 'Smith' will be very common.[1] In a survey taken in the United States it was found that only 128 surnames make up 80% of the total. Similarly the name 'Vancouver' can occur very often in a file containing Canadian city names. Thus, there is a great advantage in replacing such fields with shorter codes. Even if the rest of the file is left uncompressed, there will be fairly efficient compression, as in these cases one character replaces about five.

One such case is the driver's license system of the province of B.C. A technique developed by L. Melder codes up to 220 common names and cities by a single byte. The technique of replacing a frequent set of values with a binary number, which effectively acts as an index, is a more normal and common procedure. For example, all ranks in the Canadian Army can be expressed by the binary numbers from 1 to about 30, which takes only 5 bits rather than having a field containing the word 'private', 'lance-corporal', etc. Normally this will take a full computer character, as any wasted bits have little effect on the final file size.

---

[1] One and a half million occurrences in the U.S. Social Security file!

J. Markov Techniques

A compression schema described by Synderman and Hunt is a memory based method, in that the altering of probabilities of the second character, after a particular first character is taken into account. This is the first order Markov probability though this term is not used in their article. The technique is to combine pairs of characters where the frequency of pairing warrants it; otherwise, the characters are left unpaired though translated to another code to aid the efficiency of the decompression technique. The strength of such a method is that each byte contains complete character(s), allowing simple implementation. Overall, this method was found to achieve a 35% compression ratio on upper case data with an overall conversion time of a little more than 12 instructions per character on IBM 360/370 computers. This is a higher packing factor than would be obtained by simple conversion to the BCD 6 bit code. The data security is fairly high, in that the data would not be recognizable to casual inspection.

SECTION V. NON COMPUTER APPLICATIONS

A. Morse Code

Invented in 1838 by Samuel Morse (1791-1872), the code is compact in that the most frequent characters in English text 'e', and 't' are coded with a dot and a dash respectively with longer strings for less frequent characters. Interestingly, Morse arrived at the frequency of letters in the English language by counting the letters in bins of printer's typescript.

B. Diplomatic Codes

These have a very long history; the most famous example also has the lowest security — that is the Caesar cypher mentioned by Julius Caesar in 'De Belli Gallica'.

By about 1400 almost all countries had adopted the nomenclator, a hybrid of code and cypher. A fair example might be additional MS9800 in the British Museum. This is a single sheet nomenclator given to Sir William Temple (1628-1699), the British ambassador at the Congress of Nijmegen (1675). It consists of a single sheet of six pre-printed columns containing about 700 commonly used words with space to write in extra words (normally names). These words were in rows of two to four words across by alphabet and the code numbers had been written in running down the columns starting at 672 giving in effect a mixed one part and two part code with codes for letters so that names and uncoded words could be spelt out. For example, able was 672, majesty was 806, parliament 1084, and Louvious M. was written in as 1070.

Interestingly, the clerk who wrote down the code completely destroyed the nomenclator security because he almost invariably numbered each group in ascending order as just one more than the preceding group; the ascending order was required by the form of the code, the unity gaps between groups were not. Judging from the preprinted sheets there were a number of similar codes. If this fault was habitual, as I suspect, once any one system was broken any further intercepted message, based on the same preprinted sheet, could be read simply by trying starting numbers around the lowest code appearing in the message until recognizable plaintext appeared. The security was unlikely to be helped by the then common practice of leaving much of the text uncoded allowing easy detection of the codes. Alternatively, some clerks laboriously coded 'I am your excellencies obediant servant...' etc. at the end of every dispatch.

Later examples of nomenclators were much larger and cut apparent redundancy by containing multiple codes for the more common groups such as 'e' or 'France'. Cypher clerks, traditionally underpaid, continued to use just the first variant.

Normally, these latter codes were true two part codes; despite this they usually offered little security to careful analysis. Probably some could be broken with a single intercept.

However some later codes were well designed and secure. It should be noted that 'Le Grand Chiffre' of Louis XIV was not broken until the middle of the nineteenth century, by which time the principles concerned were rather beyond worry. A code used by Sir Francis Walshingham, Elizabeth I's Ambassador to France and security secretary, is still

unbroken.

D. Telegraph Codes

These codes were invented mainly for economical transmission on telegraphs and so rose to their peak during the beginning of the 20th century. These codes are modified dictionaries published as two part codes. The codes typically consist of a set of messages which can be represented by a four, five, or ten letter group. Letters were used because they could normally be sent at a cheaper rate than numbers and they generated a more compact code. For example, the message 'DRWAE' could represent 'Captain and crew drunk', and 'DRUAG' could represent 'Captain and crew drunk and mutinous'. Effectively the messages were designed to cover the most common situations, and the code books had various flavours, in that some were mainly nautical, some commercial, etc.

Overall, it was found that one code group could replace five words — an 80% compression factor.

Experience showed that few messages were entirely error free. The five letter group was chosen so as to give high degree of protection from garbles. It was realized early on that if all code words differed from other code words by at least two letters most mistakes in transmission would be discovered and a book of 100,000 codes could still be constructed. Also similar messages differ in only a few letters, so even if it did not generate an invalid message, a transposition could give one with some chance of having a sense close to the one meant.

Economy, that is the compression character of the code books, was

the main impetus behind these codes and data security was a minor importance. At first telegraph companies tried to ban the use of the codes, as costing them revenue, and this led to the development of pronounceable codes. Later some telegraph regulations allowed only the use of regular published commercial codes. The apparent point of this was to prevent spies using the telegraph network and during World War II only two or three commercial book codes were allowed. It was always possible to produce a code book with the same groups taking different meanings or more simply to have a convention that, for example, the code sent being the 23rd code on page 43 should be read for the 32nd code on page 34. However, such a code would break Francis Bacon's third precept in that any enemy trying to decode the message would quickly realise that the messages were nonsense, and probably deduce such a simple substitution.

Kahn tells the story of smugglers during prohibition using even more complicated transpositions making use of several code books. It proved so complicated in fact that they did not code every word and so allowed their messages to be decrypted relatively easily by Elizabeth S. Friedman, one of the most renowned cryptanalysists of this century.

SECTION VI. ENTROPY AND REDUNDANCY.

A. Entropy

The entropy of data is of fundamental importance as it relates to both compression and security. It is used to calculate the redundancy in data giving a measure of the amount of compaction possible without loss of information. In turn, the redundancy is used to calculate a measure of the security of a code or cypher evolved from the data as will be shown in Section IX.

Definition: Any message has a self-information, I, defined by $I(x) = -\log_2 P(x)$; where x is the message and $P(x)$ is the *a priori* probability of this message. (Shannon (1948).)

Entropy is defined as the average value of self information. For a message source M the entropy is $S(M) = \sum_{x \in M} P(x)I(x)$.[1]

B. Coding

Typically the messages will be coded into binary by some encoding processes. (These processes will be discussed in detail in Section VII.) The coding will have an average length S_{MAX}.

Now $S(M) \leq S_{MAX}(M)$; (Fano, 1961). Thus S_{MAX} is the maximum entropy which can be encoded with a particular alphabet. It is a measure of the maximum amount of information encodable with this set of symbols.

---

[1] As described in Brillouin (1956) this definition of entropy is equivalent to that in classical thermodynamics.

C. Redundancy

A message encoding has a redundancy (D). This is a measure of the ratio of the information transmitted to the amount theoretically possible.

$$D = 1 - S(M)/S_{MAX}(M)$$

The redundancy of a data source should be calculated, at least approximately, before deciding on the amount of effort which should go into building algorithms to encrypt or compress the data. If the redundancy is already close to zero such algorithms, and efforts, are largely wasted.

D. Data Independence

The data issuing from a message source can have various degrees of independence considered character by character. This will be of considerable importance in calculating the entropy of computer data in Section VIII.

Consider a message M composed of a number of coded characters $C_1, C_2, C_3, \cdots, C_n$. That is:

$$M = C_1 || C_2 || C_3 || \cdots || C_n$$

It is realistic to consider that each character can have one of a distinct number of values: $V_1, V_2, \cdots, V_n$. That is:

$C_1 \cdots C_n$ belongs to the set $R(V_1, V_2, \cdots, V_n)$.

Now the separate values of C_1 to C_n can have two degrees of independence:

(a) Bernoulli Independence.

In this mode each value V_1 to V_n is equally likely to occur and the distribution is completely random. A classic example is the fall of a pair of dice where the numbers 1 to 6 are equally likely to occur.

Bernoulli dependence refers to an underlying probability distribution for the values V_1 to V_n. Thus some values will be more likely and some less likely to occur. An example is the value of the sum total derived from a pair of dice. Another example is the distribution of letters in English text.

(b) Markov Independence.

If the data has Markov independence the value of C_n is completely independent of any previous values of C_1 to C_{n-1}. That is, C_n has no correlation with $C_{n-1}, C_{n-2}, \cdots, C_1$. The fall of either one, or a pair, of dice is Markov independent.

Data is Markov dependent if the values of C_1, C_2, and/or C_{n-1} effect the value, or probability distribution of the value, of C_n. Examples are numbered balls being removed in a lottery draw or letters in English language text (e.g. i before e).

Markov data dependence can be first order (dependent only on the immediately preceding value, that is C_n on C_{n-1}), or second order or more.

The two types of independence, Bernoulli and Markov, are unconnected. Data can have either or both. English text has neither, that is, it is dependent on both modes.

E. The Effects of Data Dependence on Entropy

An example will clarify the effect of data dependence.

Consider the standard English alphabet. This is comprised of 27 symbols (26 letters plus space). Thus,

$$S_{MAX} = \log_2(27) = 4.758 \text{ Bits.}$$

Now if data was fully independent in both modes, this would also be the value of the entropy.

But English letters occurrence is known empirically to have neither form of occurrence. Considered purely in Bernoulli mode, the entropy of the English alphabet can be considered as

$$S = - \sum_{\text{space}}^{Z} P(x) \log_2 P(x) = 4.02 \text{ Bits,}$$

taking the probabilities in Figure II (called below the Bernoulli entropy). This gives a redundancy of

$$D = 1 - 4.02/4.758 = 0.1551.$$

If the Markov dependencies are added, the calculation of the Markov entropy is not possible from formulae. Shannon (1951) suggested a letter guessing technique which gives an empirical value of the entropy of about 1.4 bits. The resultant redundancy would be

$$D = 0.71.$$

SECTION VII. CODING OF DATA SOURCES.

A. Block Coding and Prefix Coding

The meaning of coding in this thesis is restricted to block coding in that each symbol of the source alphabet, X, is mapped into a fixed sequence of the code alphabet, C (Abramson, 1963). The selected characters of the code alphabet are concatenated together to make the output code message.

That is, if the selected sequence of the code alphabet are the symbols $C_1, C_2, C_3, \cdots, C_n$ (not all necessarily distinct) and the concatenated output code after adding C_n is S_n, then

$$S_1 = C_1 \ ;$$

and

$$S_{n+1} = S_n \cdot 2^z + C_{n+1}$$

where z can be either positive or negative and is normally an integer equal to the absolute binary length of C_{n+1}. Rissanen and Langdon (1979) have generalized this equation to cover arithmetic codes where $|z| < \text{len}(C_{n+1})$ and is not an integer.

The normal case can easily be implemented with concatenation and shifting as S_n and C_{n+1} have no bits in common in the output code message. That is true where no code is a prefix of another code. This is the prefix property.

Russanen and Langdon have demonstrated that arithmetic codes can be more compact than prefix codes but have not, so far, been demonstrated

to have simple practical implementations. Thus this thesis is largely restricted to codes having the prefix property.

B. Optimization

An optimum code can be defined as one in which the average coded message length, S_{MAX}, is as close as possible to the entropy, S.

Restriction of practical implementation generally prevents perfect coding. Many coding schemes allow only for an approach to the Bernoulli entropy in that only one symbol for the source alphabet is block coded at a time. Since the Markov entropy is often considerably less than the simple Bernoulli codes such codes cannot be optimum.

Extensions of symbols (grouping) will allow coding of both types of data dependence simultaneously.

Obviously little optimization is possible for codes of fixed (equal) length. For a source coding m symbols the minimum binary bit length, n, is the integer given by

$$\mathrm{Log}_2(m) \leq n < \mathrm{Log}_2(m) + 1.$$

Thus for 16 symbols $n = 4$, and for 17 symbols $n = 5$.

Since in almost all cases the frequencies of occurrence of the various symbols are going to be different, the most efficient coding of a source will use different lengths for different symbols. This will be either explicit, or implicit by the suppression of redundant information, i.e. null or blank suppression.

C. Shannon Noiseless Coding Theorem

For an arbitrary source alphabet, a, the entropy can be related to the minimum code length L. Shannon's noiseless coding theorem gives a limit to the code length, for this code.

$$\frac{S(a)}{\text{Log}_2(r)} \leq L < \frac{S(a)}{\text{Log}_2(r)} + 1$$

where r is the number of symbols in the code.

The source symbols can be combined to form extensions of the basic code. Then the limit of the optimum code length is given by

$$\frac{S(a)}{\text{Log}_2(r)} \leq \frac{L(n)}{n} < \frac{S(a)}{\text{Log}_2(r)} + \frac{1}{n}$$

where n is the *n*th extension of the code and L(n) is the average length of this code.

From this we can see that a code can be built with an average length arbitrarily close to the entropy as long as the code block also contains an arbitrarily large number of symbols.

Shannon's theorem can also be reversed in that if a coding exists with average code length L(n) then

$$\frac{S(a)}{\text{Log}_2(r)} \leq L(n).$$

That is, if a source can be coded by a binary coding of length L(n) then the entropy of the source is no greater than L(n). The value of L(n) can be called the implied entropy.

C.E. Shannon in his seminal work (1948), proved the existence of an optimal code for any alphabet and suggested a method, known as the

Binary Fission Method, for construction of such a code. Since this method is complicated and the codes produced by it may not, in fact, be optimum, it has been superseded by other methods.

D. The Shannon/Fano Method

This is a refinement of the Shannon Method. The codes are ordered in decreasing probability, then divided into groups of equal probability and each group, assuming a binary code, is given a one or zero as a prefix. This is continued for each group until all symbols have been coded. This code is optimum if and only if the equality

$$P(a_i) = 2^{-\ell_i}, \quad i = 1,2,3,\cdots,N$$

is true where ℓ_i is the length of the a_ith symbol and $P(a_i)$ is its probability.

Since the Shannon/Fano method only gives optimum codes with the above restriction and the Huffman code covered next is always optimum, the Shannon/Fano method would not be of use except for one advantage. Huffman codes can only be constructed when the number of symbols and their associated probabilities are known. In the Shannon/Fano method if the symbols with some portion of the total probability of occurrence are known codes can be constructed for these symbols without knowing the probabilities or even the number of the remaining symbols (with the restriction that no symbol in the uncoded set can have a higher probability than a coded one; in other words, the equality above must hold). This makes the construction of large code tables computationally feasible. Use could be made of this in building an English word code

table for a security or compression table.

E. Huffman Code

D.A. Huffman in his classic 1952 paper showed how to construct an exhaustive optimal code from any given alphabet. In this case he showed that the average length of the code letters will be less than or equal to any other method of constructing a code from the same alphabet. The method used is that the two letters of lowest probability of occurrence are used to form a prefix set with the combined probability of the two letters. This procedure is repeated until all letters have been coded.

It should be noted that the input source symbols need not be of the same length. In fact, some experimentation will show that codes of shorter average bit length can be constructed if this is not so and if the most common groups are set aside as a special code.

This result follows from the fact that the code is being built on Markov dependence data, at least partially, rather than Bernoulli.

F. Security of Huffman Code

It should be moderately easy to treat a simple Huffman code as a cryptogram and to break it by use of the prefix property, by trying a limited set of codes on a 'tree' and seeing if this continues to give only positions on the 'tree' (that is, the same set of codes). Since the shorter codes are by definition the ones with the higher probability of occurrence, frequency analysis has been done for us on the entire set of data rather than on the fragment which we have intercepted. With

English text we may even be able to read off the codes with the aid of a probability table similar to Figure II. Because of relabelling, more than one Huffman code can normally be constructed from any source, but this would present a very minor difficulty to the antagonist.

The security of Huffman code will be increased markedly, as will the compression factor, if special groups are coded separately, as this is effectively equivalent to having more than one symbol for the same source letter. Since the adversary will not know how many code groups there now are, decryption is more difficult. In the sample computer program supplied to create Huffman codes an extra group for multiple blanks can be added if this is found to be desirable. In order to increase security still further an optional superencypherment scheme has been supplied in the sample programs. This superencypherment is required for greater code security.

The run time of a Huffman coding and decoding routine will be more than the 10 instructions per character found with simpler algorithms. The actual run times of the programs supplied are probabilistic in that they decrease as the code generated tends towards the optimum; that is, the distribution of the characters in the input source matches that of the sample used to calculate the table. With the new generation of computers this should not be an important point.

G. Alphabetical Ordered Code

Gilbert and Moore in a 1959 paper showed how to extend this coding in a way which retains alphabetical (sorting) order. They call the

property of preserving alphabetical order the Strong Alphabetical Property.

The preservation of alphabetical order has often been shown to be a major weakness in code security, in that if codes N and N + 2 are known, then code N + 1 is often known as well. Despite this, the loss of security in preserving alphabetical orders is only moderate in many cases, and for efficient computing this could often be required. The choice could be in preserving alphabetical order or no security at all. Such would be the case when we are cyphering data record keys for an indexed file.

In alphabetical coding every prefix set must consist of all letters lying between two fixed letters of the alphabet. The average code length is extended by a minimal amount, and the codes generated are in the same mathematical increasing relationship as the input alphabet; a disadvantage is that the construction of a code is somewhat more difficult than in the straightforward Huffman code, and it could be broken even more easily. Despite this such a code could have considerable advantages in compression schemas for data indexes, and it is surprising that little use has been found for it; the slight weakness in security is unlikely to be the reason. (See also Knuth, vol. 3, Chapter 6, pp. 445-447.)

H. Larger Character Sets

One can see that codes composed of more than one base character will be more compact in principle and that this should give a gain in security.

As seen from Figure II, the English language probability of the letter 'Z' in standard text is 0.05%, while that of the trigram 'the' is 4.5% and the corresponding probability of 'ing' is 1.4% [DE02]; yet standard Huffman code will use 11 bits to express 'the' and 14 for 'ing' while using only 10 for 'Z'.

This inconsistency follows naturally from the form of English which has many rigid rules about spelling. For example all English words which contain the letter Q follow it by U so the U is redundant, as Shannon points out. Because J or V never end a word in English, the combination of the letter J or V followed by a space hardly every occurs; similarly these letters are almost never doubled.

The effect of these rules, and similar ones, is that there are only about 100,000 words of under 10 letters in the English language and these could be coded with 17 bits. However, if coded in Huffman code they will take on the order of 23 bits. Huffman code, as given in Figure II, can only take advantage of some 25% of the redundancy of the English language, that is the Bernoulli dependence. It would seem that coding by words rather than be letters would give a fairly substantial increase in compactness and in all probability a similar gain in security, even if a fixed length code word were used.

The full English dictionary of about 450,000 words could be represented with 19 bits, and the 850 words of basic English with only 10 bits. But is this the optimum approach, or is some other method better? It should be noted that on a sample of 100,000 English words, 26.668% were made up of just ten words, namely 'the', 'of', 'and', 'to', 'a',

FIGURE II.

EXAMPLES OF COMPUTER CODES

LTR	PROBABILITY	EBCDIC	ASCII	HUF CODE	ALPHABET CODE
SPACE	0.1850	01000	0100000	000	00
A	0.0642	11000001	1000001	0100	0100
B	0.0127	11000010	1000010	011111	010100
C	0.0218	11000011	1000011	1111	010101
D	0.0317	11000100	1000100	01011	01011
E	0.1031	11000101	1000101	101	0110
F	0.0208	11000110	1000110	001100	011100
G	0.0152	11000111	1000111	011101	011101
H	0.0467	11001000	1001000	1110	01111
I	0.0575	11001001	1001001	1000	1000
J	0.0008	11010001	1001010	0111001110	1001000
K	0.0049	11010010	1001011	01110010	1001001
L	0.0321	11010011	1001100	01010	100101
M	0.0198	11010100	1001101	001101	10011
N	0.0574	11010101	1001110	01001	1010
O	0.0632	11010110	1001111	0110	1011
P	0.0152	11010111	1010000	011110	110000
Q	0.0008	11011000	1010001	00111001101	110001
R	0.0484	11011001	1010010	1101	11001
S	0.0514	11100010	1010011	1100	1101
T	0.0796	11100011	1010100	0010	1110
U	0.0228	11100100	1010101	11110	111100
V	0.0083	11100101	1010110	0111000	111101
W	0.0175	11100110	1010111	001110	111110
X	0.0013	11100111	1011000	00111001100	1111110
Y	0.0164	11101000	1011001	001111	11111110
Z	0.0005	11101001	1011010	00111001111	11111111
COST	BITS	8	7	4.1195	4.1978

Note: This Figure is modified from Gilbert and Moore's article. The minimum fixed length word to code this costs 5 bits and would give a fairly efficient coding for this alphabet.

'in', 'that', 'is', and 'I'. (Geoffrey Deevy cited in the Encyclopedia Americana - 1977 ed.) Similar results will be found in Arabic; the Arabic word for 'the' is required before the noun and each adjective as well so it tends to outnumber all other words in a way not found in English.

This can be put in terms of entropy. As was seen in Section VI, the English language, regarded Bernoulli independently letter by letter, has an entropy of 4.02. If the symbol immediately preceding an unknown symbol is taken into account the entropy drops to 3.332 bits per symbol. When this is extended up to eight preceding letters the entropy is 2.4 bits per symbol. Carrying this further, by guessing experiments, Shannon (1951) arrived at an entropy of about 1 bit per character (0.6-1.3 range). Further work has refined this to 1.4 - 1.5 bits per character. (However, at this level the structure of the basic English word is not enough; guessing experiments are using lexical relationships as well — the entropy of any language is decreased by disallowed word combinations such as 'we is bottle' and so efficient coding would not need to represent this phrase in its lexion. See Smith and Wilson's Work on Linguistics.)

These types of language restrictions give us a target for efficient compression shemas, in that with straight alphabetic text we could get the same 1.4 bit per character instead of the 6 of standard BCD which is the smallest fixed length code possible if we allow for punctuation. If compression to more than the word level is not felt to be feasible with today's technology, a level of about 2 - 2.5 bits per character

could be achieved by codes which represent single words, or at least the most common ones.

SECTION VIII. ENTROPY OF COMPUTER DATA.

A. Coding

Due to the limitations and economics of computer circuit design, data representation (i.e., coding) is nearly always in basic fixed units or number of bits.

Digital computers generally have their main memory divided into two basic units: the word and the character or byte. The basic unit is the character (or byte) which is defined as the smallest directly addressable unit of data, and it varies in size from 6 to 8 bits. Some computer's architectures, such as Honeywell model 66's allow the size, for certain instructions, to be varied under program control as a program option. An eight bit byte allows the storing of packed data (two numeric characters to a byte), and all characters which can be printed on a standard computer printer. Eight being a direct power of two gives simpler internal circuit design as well. The adoption of the byte by the IBM 360 means that this is now close to an industry standard. Some other common internal codes are BCD at 6 bits/character and ASCII at 7 bits/character, though the latter is often held as an 8 bit character for ease of manipulation.

The next larger basic data unit is the computer word, which is normally the basic unit for binary or floating point arithmetic. The word is composed of up to 128 bits of data; again this can often be varied. Some large scientific computers, particularly array processors, can only address words, but this is not the normal case. Even then, macros are generally available to allow character addressing. However

word addressed computer instructions are often much faster than character addressed instructions which allow for the greater complexity of scientific calculations.

Thus, the naive answer to the entropy of an N bit character, or word, would be that it is N bits/character or word. However, this is actually rarely the case, though this is, of course, always the value of S_{MAX}. Consider what an entropy of 8 bits/character for an IBM/370 would mean. It would mean that any one of the 256 possible bytes is equally likely to occur and that the occurrence of each character is both Bernoulli and Markov independent of all the others. This is a fair description of a table of random (binary) numbers, but not of anything else. It certainly does not describe most computer data though it represents the compiled object code[1] of the IBM/370, but not the data portion. However, it must be clear that this is the only generalized answer, and that the lower values for entropy, which are calculated in the rest of this section, inevitably come from consideration of special common cases.

B. Frequency Analysis

When frequency analysis is run on computer data, the space character (X'40') is found to make up nearly 50% of the file in many cases, more in the case of source programs and reports. So consider a code of B'1' for space and B'ONNNNNNNN' where NNNNNNNN is the normal ebcdic character bit pattern. This unsophisticated coding would give unique decodability

---

[1] Executable machine language.

and would provide an average bit length of 5 bits per character. Hence by the reversal of Shannon's first theorem as was shown in Section VIII, this would be a maximum for the entropy of this data.

The preponderance of the space character, or null, follows from the programming practice of taking maximum length fields for variable data, such as names and padding the rest of the field with blanks. It is, thus, much more common than in non-computer data where the space character is rarely more than 10% of the total character. Use will be made later of the observation that the space character often occurs in runs, sometimes of considerable length, something which is uncommon for any other character.

Computer files can be divided into various types on a rather arbitrary basis; these are given in Figure III.

Figure III

TYPES OF COMPUTER FILES

A. Source programs and output listings.

B. Reports and textual data.

C. Object code and load modules, general software.

D. Scientific data.

E. Data files and data bases.

C. Markov States

When these are inspected visually the outstanding feature is that the data seems to be divided into a number of Markov states with rela-

tively low transition probabilities between them. (See Section VI.D.)

The states are the letters of the alphabet, the numbers, the other special printable characters and all others. Depending on the type of data there will be a preponderance of alphabetic data or of numeric data and only in type C above will the states be somewhat random. A number of Markov transition state tables were constructed as Figure IV. The characters '.' and ',' and the space character were treated as special cases, in that they could logically be members of any states. The transition probabilities have been found to be neither ergodic nor regular.

Having divided the ebcidic character set into groups, a code could be constructed for each group reserving 3 symbols to indicate the transition to another state. This code could be 4 bits long for numeric data (including + and space), 5 bits long for the alphabet and the special characters, and 8 bits long for the other characters. From the Markov transition probabilities, we can calculate the average code length, and from Shannon's theorem this is an upper bound on the entropy. It would be possible to use this method to calculate the upper bound on the entropy for the various types of data studied, but it was found Huffman code normally gave a lower result on test data. It should be noted that this is the principle behind the Murray (or correspondence) code used in such networks as Telex.

D. Huffman Code

With a program written to calculate Huffman code for any data source, it was a simple addition to the program to calculate the average

FIGURE IV.

FIRST ORDER MARKOV PROBABILITIES

LISTING TYPE: DOCUMENTATION

Markov State	Overall Ratio	Spaces ',' & '.'	Trans to Numeric	Trans to Alphabet	Trans to O. Print	Trans to Other
Numeric	6.107%	41.610%	81.628%	8.795%	9.521%	-
Alphabetic	81.634%	19.559%	0.966%	98.504%	1.128%	-
Other Printable	12.257%	18.346%	2.677%	9.573%	87.747%	-
All Other	-	-	-	-	-	-

LISTING TYPE: INPUT DATA

Markov State	Overall Ratio	Spaces ',' & '.'	Trans to Numeric	Trans to Alphabet	Trans to O. Print	Trans to Other
Numeric	45.824%	37.370%	84.197%	14.143%	1.658%	-
Alphabetic	49.264%	16.469%	8.320%	85.068%	6.610%	-
Other Printable	4.523%	1.081%	68.268%	20.540%	9.749%	-
All Other	-	-	-	-	-	-

coded bit length. This came out at about 2.2 bits per character for assembler source programs, and 2.8 bits per character for COBOL. Test data, which had been largely stripped of blanks had a result of 4.8 bits per character and a VM 'packed' file (see Section V) of 5.6 bits per character.

Such a coding gives a worthwhile improvement over such utilities as the VM pack program. However far more optimum codes could be constructed with a lower implied entropy, particularly on alphabetic data. The calculations and programming rapidly become very cumbersome, so some form of compromise should be looked for.

On inspection of sample source programs, the number of blanks occurring together are shown in Figure V. Similar, but not identical results would be obtained from other programs.[1]

Thus, one worthwhile extension to Huffman code, and one which adds little to programming complexity, and much to security and compaction, is to allocate an extra code for a group of blanks or to allow for this factor in some other way. Experimentation showed that, at least in the case of program source, the optimum size for grouped blanks would be in the range of 4 - 8 as is shown in the attached table of possible packing factors. The optimum value will depend on the number of comments, the length of labels and other factors.

These factors will vary extensively from source program to source program, but the number of blanks grouped will make little overall difference, inside the above range, to the effective compression of any par-

---

[1] The results are not averaged, but are simply to illustrate the tendency of the space character to occur in runs.

FIGURE V

GROUPED BLANKS

No. of Characters Together †	1	2	3	4	5	6
No. in Assembly Program	308	39	74	92	275	9
No. in COBOL Program	5245	319	250	102	25	72

No. of Characters Together	7	8	9	10	11	12
No. in Assembly Program	4	14	14	10	4	9/etc.
No. in COBOL Program	535	17	27	124	268	48/etc.

† Number of blanks between non-blank characters or end of record.

FIGURE VI

AVERAGE CHARACTER LENGTH FOR VARIOUS LENGTHS

OF GROUPED BLANKS

GROUPED BLANKS						
0	3	4	5	6	7	8
CALCULATED CHARACTER LENGTH (IMPLIED ENTROPY)						
COBOL PROGRAM						
2.535	2.188	2.142	2.095	2.077	2.050	2.083
ASSEMBLER 1						
2.072	1.801	1.751	1.789	1.789	1.787	1.762
ASSEMBLER 2						
2.365	2.151	2.130	2.071	2.070	2.050	2.129
TEST DATA						
4.738	4.738.	4.738	4.738	4.738	4.738	4.738

ticular program source.

If this grouping of blanks is done, one gets an implied entropy minimum of 2.076 bits per character working with all three programs combined and using a unique code for 6 space characters occurring together.[1]

Different data types require different techniques. For example, in the case of alphabetic (English) text the most common digrams and trigrams, such as 'th' and 'ing', should be coded as special Huffman code groups as well; it is simple to extend the supplied programs in this direction.

I found that a slightly greater improvement could be obtained with computer source programs by, rather than just grouping blanks, replacing all duplicate characters with a count field, and in this case the implied entropy was 2.032 bits per character for the same three source programs combined. This is a very small improvement over grouping blanks, but the number of Huffman codes which are used increases, which will increase security. This feature was implemented in the attached programs in Appendix B using the VM pack program to compress duplicate characters as an external program subroutine.

E. Summary

From the analysis above it will be seen that there is a considerable redundancy in much of computer data, more than is found in normal printed text, and that this can be ascribed to two roughly equal causes.

---

[1] The programs are of different lengths so this is not the same as the optimum average value.

Firstly, the large number of blanks or space characters; secondly, the preponderance of the few 'printable' characters in a normal computer data record.[1]

---

[1] Printable characters are those byte configurations which encode characters that are 'printed' by standard computer hardware. The space character is technically 'printable'. These characters are normally the most common.

SECTION IX. PRINCIPLES OF DATA SECURITY.

A. General

Coding or cyphering of a message involves transformation of an input message by a general system or algorithm involving a specific key. The basic requirement of classical message security is that the key should be kept secret. This is known as Kerchkhoff's law and states that the secrecy of a cryptographic system comes solely from the key and that the cypher algorithm itself is assumed to be known to the antagonist.

B. Bayes' Theorem

The secrecy of a message can be deduced from Bayes' theorem.

PE(M) = P(M).PM(E)/P(E), where

P(M) is the *a priori* probability of the message M.

PM(E) is the conditional probability of cryptogram E, that is, the sum of all probabilities which can produce E from message M.

P(E) is the probability of obtaining the E from any cause,

PE(M) is then the *a posteriori* probability of producing M if cryptogram E is intercepted.

For perfect secrecy PE(M) must equal P(M) for all E and M. That is PM(E) ≡ P(E).

C. Shannon's Theorem

In the 'Theory of secrecy systems' (1949) Shannon defined secrecy systems as a set of transformations of one space into a second space (the set of possible cryptograms). The encyphering process is a trans-

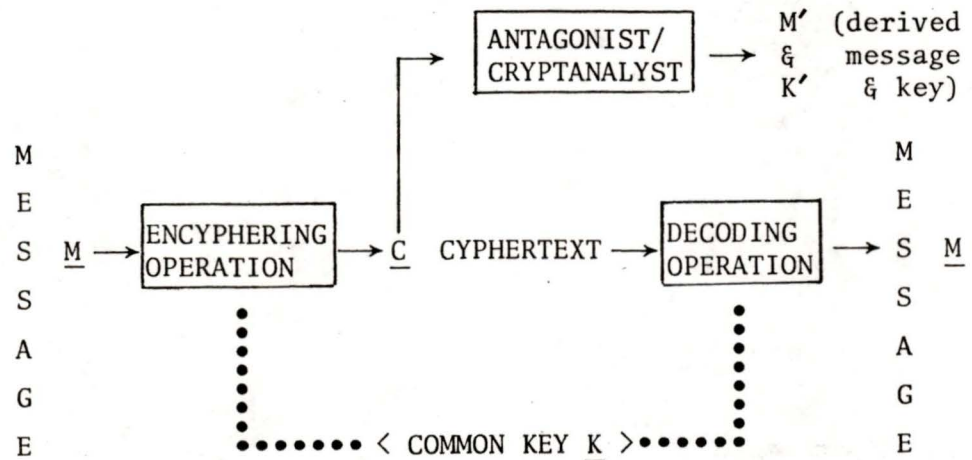
form f on data originating in a discrete source:

f(M,K) = C, where M is the message plaintext, K is the key, and C is the message cyphertext.

Also, $f'(C,K) = M$ is the decoding process. The coding and decoding process can be given in the figure following:

FIGURE VII

CODING PROCESSES



Normally the encyphering and decyphering keys are simple transforms of each other. Thus, they have to be distributed and guarded with extreme care. There are two variant cyphers where the keys are not simple transforms of each other. In public key cryptograms the calculation of the transformation algorithm of one key into another, while theoretically possible, will take on the order of millennia. In irreversible cyphers, which are discussed in Section XI, the decyphering

key does not exist.

The object of a cypher or code should be that the probability of an antagonist, decoding the message without the key should approach zero, i.e.

$P(f'(C,K') = M) \rightarrow 0$; where K' designates that the key is unknown to the cryptanalyst.

Alternatively, referring to Figure VII above,

$P(M' = M) \rightarrow 0$ and $P(K' = K) \rightarrow 0$.

D. Unicity Distance

In the 'Theory of Secrecy Systems' Shannon also showed that the difficulty in decrypting a language cryptogram, as opposed to a computer cryptogram, is related to the unicity distance 'U'. This is a heuristic measure of cypher security. The unicity distance is actually the number of characters which must be intercepted, on average, to allow a cryptanalyst, in theory, to be certain of a proposed solution's correctness. The unicity distance can be calculated by

$$U(A) = H(K)/D,$$

where H is the logarithm of $K!$, the number of possible keys, and D is the redundancy of the data.

For the 26 letters of the alphabet and the space character Shannon gives

$$H(K) = \text{Log}_{10}(27!) = 21.$$

In Section VI the value of D for the English substitution cypher was calculated as 0.7.

U(A) = 21/0.7 = 28. This predicts that a mono-alphabetic substitution cypher should be solvable if over 28 letters of cyphertext are available. This has been found to be in good agreement with experience. Friedman has stated that most English mono-alphabetic substitution cyphers of more than 25 letters can be solved (1973).

From this, one can see that the security of a code or cypher can be related to its effective key length and to the redundancy. Perfect security requires either infinite key length, which is the one time pad, or zero redundancy, which is perfect source coding. That is,

$$U \to \text{infinity as } K \to \text{infinity or } D \to 0.$$

Thus, if the coding is fully Markov and Bernoulli independent, the unicity distance of even a simple substitution cypher is infinite.

E. Data Compaction

An example should show the effect of data compaction.

The following table, Figure VIII, is an artificial sample of computer data with a compact code built for it. The table is limited to printable characters only. Taking the minimum fixed size to code this data, and assuming a simple substitution cypher,

$$S_{MAX}(a) = \text{Log}_2(64) = 6 \quad \text{and} \quad S(a) = 3.01.$$

Therefore,

$$D = 1 - 3.01/6 = 0.5.$$

FIGURE VIII

COMPACT CODE FOR PRINTABLE CHARACTER

CHARACTER	FREQUENCY	CODE	CHARACTER	FREQUENCY	CODE
0	55.5%	0	I	0.20	1111111011
1	6.7	100	T	0.15	11111111000
2	4.5	101	K	0.15	11111111001
8	3.5	11000	Y	0.13	11111111010
3	3.3	11001	X	0.12	11111111011
A	3.2	11010	G	0.10	111111111000
5	3.3	11011	J	0.10	111111111001
6	3.0	111000	O	0.06	111111111010
4	2.7	111001	Q	0.03	111111111011
9	2.2	111010	V	0.03	1111111111000
7	1.9	111011	W	0.01	1111111111001
F	1.5	1111000	.	< 0.01	1111111111010
B	1.2	1111001	-	"	1111111111011
BLANK	1.1	1111010	,	"	11111111111000
D	1.0	1111011	&	"	11111111111001
E	0.9	11111000	/	"	11111111111010
Z	0.7	11111001	+	"	11111111111011
P	0.6	11111010	<	"	111111111111000
N	0.5	11111011	)	"	111111111111001
U	0.4	111111000	(	"	111111111111010
C	0.4	111111001	%	"	111111111111011
H	0.4	111111010	=	"	1111111111111000
R	0.3	111111011	#	"	1111111111111001
M	0.3	1111111000	?	"	1111111111111010
L	0.3	1111111001	@	"	1111111111111011
S	0.25	1111111010			

AVERAGE CHARACTER LENGTH (allowing for frequency of occurrence) IS 3.01 BITS.

This coding represents 51 characters; so the number of possible keys are 51! giving a length of

$$H(K) = \text{Log}_2(51!) = 220 \text{ bits or } 38 \text{ characters.}$$

Therefore, the unicity distance is

$$U = 38/0.5 = 76 \text{ characters.}$$

Now since over half of this particular data sample is the zero character, one can assume that 1/2 of these zeroes are suppressed by an additional compression routine, and that the data is represented with the same Huffman code. Then

$$S'(a) = 5.1325, \quad D' = 0.144 \quad \text{and} \quad U' = 262 \text{ characters.}$$

But we have dropped 25% of the characters in the file so, in terms of the original data

$$U'' = U'/0.75 = 349 \text{ original characters.}$$

$$U''/U = 4.59.$$

Thus a very simple compression routine has had a dramatic effect on the security of encrypted data.

A considerable further increase in security is possible. From the high Markov probabilities of the numeric characters occurring together, the numeric data could be packed two to an 8 bit byte. The 8 bit byte is required because packed decimal data has 130 possible characters, allowing for the sign bit, and this gives us 171 possible codes,

$S_{MAX}(a) = 8, \quad S(a) \doteq 6, \quad H(K) = \text{Log}_2(171!),$

$D' = .75.$

The decrease in the file size is probably the same as in the last case, that, .75, so the true value of D (in terms of the old data) would be about 0.55.

In this case the unicity distance will be several orders of magnitude larger than in simple compression, again as a result of data compaction, and the coding of the source with even a simple encypherment is likely to be relatively secure.

In the case where packed decimal is not useful, such as where most of the data is non-numeric, a cypher based on two characters at a time would give H(K) = Log(2061!) at a modest cost in programming complexity and size. This would only be a problem in a microcomputer, where a Shannon/Fano code for the more common combinations might be the answer.

SECTION X. CODES AND CYPHERS.

A. Definitions

Codes and cyphers are terms which are often used incorrectly; a cypher is a system which works on short, fixed length blocks of text (typically just one or two letters). Codes transform natural blocks, such as words or phrases. Codes are linguistic, cyphers are non-linguistic.[1] Thus, substituting different letters of the alphabet for each other, as in Poe's 'The Gold Bug', is a cypher, and the commercial codes described in the section on coding examples below are true codes. Of course, many systems mix both processes. Coding is normally block coding, i.e. every coding of the same source will lead to the same coding with no variation.

In turn, cyphers are of two basic kinds: substitution and transposition. In substitution cyphers a letter, or group of letters, is changed into some other set of symbols. In transposition the text is shuffled through some predetermined permutation. Again, both kinds are often mixed.

For example, a phrase such as 'THE CAT SAT' could become '$TAXBR$XCR$' in substitution cypher and 'CH TESA ATT' in a transposition cypher.

B. Security

While the second is a little more secure, both examples above will be broken with sufficient textual intercept. Poe's 'The Gold Bug' gives

---

[1] This is the cryptographic definition. In information theory, both processes are coding — hence 'Huffman Codes'.

an excellent example of how these cyphers should be attacked. In fact, substitution cyphers are generally so easy to solve that they are popular newspaper puzzles under such names as "Daily Cryptoquote". Also, Hellman (1977) showed for an English language transposition cypher that the unicity distance is about 1.2 times greater than that for a substitution code, or approximately 32.

To break an intercepted code the cryptologist looks for an underlying pattern in the form of repeated groups of letters, or constant letter to letter distances. In substitution cyphers the cryptologist looks for repeated association of letter pairs and larger groups. The constant distances are of more importance in transposition or mixed cyphers. These patterns are effectively dependant on the first and subsequence order Markov probabilities. While the cypher key can normally be deduced at the same time this is not essential, and it is important to realize that messages have often been broken without the key being needed.

Additionally, the cryptanalyst tries to match cypher groups to particular portions of 'known text'. This may be a message which has already been obtained in clear by means of theft or a press release, or it may be the likelihood of groups occurring at the end of the message, meaning 'STOP'. The cryptanalyst can also guess likely words — 'Afghanistan' is a likely group today.

There is also a need to distinguish between global and local security. A code in which 10% of the message groups have had their meaning recovered may be potentially damaging simply because they are likely to be the most common groups.

C. Variations

In order to gain higher security, various approaches have been tried with varying success. David Kahn covers the vast majority in his classic book 'The Codebreakers'.

Perhaps the best is to apply two successive and different cyphers. This is known as superencypherment. A common example is, after applying a substitution cypher, or a code, to apply a transposition cypher which will destroy the 'pairing' effect which is of so much use to a cryptanalyst. The plaintext example above could generate the cyphertext 'BT$XACRXR$$' in this case.

Conversion of an alphabetic text into a group of numbers, either by a code book or a system such as a chequerboard table constructed from a key phrase, allows the superencypherment to be done by simple additive groups. This was proposed by P.E. Chase in 1859 and was the cryptographic standard for nearly 100 years until it was superseded by the Rotor machines.

Another, more historic, approach was to increase the number of substitution alphabets using the first for letters 1, N, 2N; the second for letters 2, N+1, 2N+1, etc. The Vingere is a famous example of this, though even much more complicated, and secure, examples exist. Shannon showed that the unicity distance becomes about 20N for N alphabets, that is, a vingere of 26 alphabets, which is the realistic limit, would have a calculated unicity distance of 520.[1]

---

[1] Practical examples turn out to be lower, through guessing the key phrase.

Since the basic tool of a cryptologist is the varying frequency of letters, the space character is often dropped and more than one code used for the most common letters E and T.

Repeated pairing of the same symbol is another basic tool used to break a code; a transposition, before or after the substitution, will make this less apparent. In other cypher systems, doubled letters either have an 'X' inserted between them or are not cyphered at all.

Still another approach to greater security is substitution by pairs of letters which will make the breaking of the cypher theoretically more difficult because the frequency analysis of letter pairs should require more intercept and be less certain. An example of this is the Playfair code which was widely used by the British during the First World War, a field cypher. The novelist, Dorothy L. Sayers, gives an example of breaking a fictional Playfair code in a detective story by guessing that it started with a date. This shows that the Germans probably found them equally easy, and William Friedman (1976) stated that 'their general solution is not difficult'.

Up to the advent of computers, cyphers could be of only moderate difficulty before the inevitable mistakes in coding and decoding meant that they were effectively unusable. Thus the Playfair and similar cypher were used, with at least some knowledge of their weaknesses, because they were about the limit of complication which could be handled in a cypher to be utilized by amateurs. Anything more complicated was found to result in an unacceptably high number of 'garbles' of undecodable or misunderstood messages. In fact, repeating of messages, requested

because of coding mistakes, seemed to be a major weakness at the beginning of the century because this often gave examples of the same plaintext coded in different keys.

D. One Time Pads

As stated in Section IX, in the Theory of Secrecy Systems, Shannon showed that perfect secrecy is possible, but requires a key which, while not infinite, is at least as long as the sum of all the messages which will be sent in this code system. This is the basis of the one time pad which has been heavily used by spies, since the end of the Second World War.

If properly implemented, this has been shown to be the only totally secure cypher method.[1] Each character, or word, of the plaintext is cyphered with a different random key which will never be used again. Shannon was able to show that such a message will remain totally secret, and can never be correctly decoded without the key. However the problem of security of the cypher key, of generating a large enough quantity of key text (the one time pad), of seeing that the key text is not reused and of ensuring that it is used to decode the correct text, make this technique almost unusable on computers, and in most other situations.

Elsewhere in this thesis, some ways in which a pseudo one time pad could be prepared have been given. These are similar in effect to the Vernam cypher (see below). It will be markedly less secure than a true one time pad, but could be adequate for a cheap and effective way of

---

[1] Shannon, 1949.

protecting sensitive data.

The Vernam Machines, invented in the '20's and rapidly refined, were an attempt at attaining the advantages of the one time pad without the problem of infinite key generation. With new possibilities such as single 'chip' transpositions for the DES cryptographic standard (Section XII), the complication of the cypher process can now be almost unlimited.

E. Problems

There are several points which should be learned from the history.

Firstly, that all codes are, in principle, breakable, and the best that can be done is to slow up the enemy until the damage can be minimized, or at least to make the breaking of a code more expensive than the gain to be expected from success.

Secondly, that the more data given an antagonist, the easier an antagonist will break the code. Therefore, compression of data is doubly useful as was demonstrated in Section IX. This is because the amount of coded text is less likely to be over the unicity distance, and also because compression of data, which removes redundancy from data, is going to increase the unicity distance of the final code. For a simple example, it has been found that a substitution code in which only the letter E has been deleted is much more difficult to decrypt, while at the same time any one message will be about 8% shorter. That is, both the redundancy and the amount of plaintext to be transmitted has been decreased.

The third point is that all block coding and cyphers can be regarded as examples of substitution cyphers with a large substitution

table or alphabet. This point seems obvious but may be often overlooked. The implications of this are that regardless of the algorithm used to actually code the message, the actual operation can always be replaced, at least in theory, by an operation consisting of taking one block of symbols and looking in a predefined table for a block which will always be used as substitute for this block. Decoding will be a similar operation. Thus, an algorithm, such as building a Playfair matrix, raising a number derived from the message to the *N*th power mod(N), or passing a block of plaintext to a DES chip, is only used because the construction of the table, which is inherently very large, is not practical or economical. The originator of the code loses sight of the fact that construction of the table, while uneconomical for him, may well be feasible and worthwhile for an antagonist.

E. Optimum Cyphers

As described above, the ease of breaking a cypher or code depends on the detection of underlying regularities in the text; that is, on the entropy of the code letters being different. If a cypher could be constructed such that every symbol could occur with almost equal probability, at least one of the main tools of a cryptanalyst would be useless. It was shown by Shannon that such a code has maximum entropy. Of course the other tools, such as the analysis of repeated strings, would still be available. Some permutation or hashing schema would be needed to ward off that methodology.

Shannon's Noiseless Coding Theorem shows that, as the groups of input source letters that are coded as a block get longer, the overall

code length per input source letter can approach a minimum. Concurrently, the number of code groups to be broken by an antagonist will increase. So the security of the code will grow as the average message length decreases. This can be considered another way.

Consider a discrete message source M. There is a set of all possible messages $M(S_1,S_2,S_3,\cdots,S_n)$ and a possible coding schema is to number each possible message.[1] i.e.,

$$S_1 = 1,\ S_2 = 2,\cdots,S_n = n.$$

This coding schema is indicated by Shannon's first theorem which shows that the longer the extensions to the code the closer to the entropy of the message the coding can be. If the messages were numbered in decreasing order of proability, and the code was the compact index number $1,2,3,\cdots$, it is clear that the code is optimum in that over the entire sequence of transmissions to be expected from the source, no other technique is going to produce a more compact set of codes. This is always the best coding for any message over the set of all possible messages. It should be noted that this alphabet is compact, at least in a part because of the lack of the prefix property; that is, the code is not instantaneous.

Now the information content of message $S(n)$ is $E = -\text{Log}(P_n)$ and the length is $L = \text{Log}(n)$. Since P_n is monotonically decreasing and n is strictly increasing $E \cdot L = -\text{Log}(P_n)\ \text{Log}(n)$ will be somewhat invariant. $E \cdot L$ will only be constant if P is proportional to $1/n$.

---

[1] See also Fano, pp. 6-8.

G. Zipf's Law

In 1949 Zipf analysed the frequency of occurrence of words in English language text. He showed that if the first N words are numbered in decreasing probability of occurrence, then the probability of word 'a' occurring is

$$P(a) = c/a, \quad \text{where } c \text{ is a constant such that}$$

$$\sum_{a=1}^{N} P(a) = \sum_{a=1}^{N} c/a = 1.$$

This formula is known as Zipf's Law.[1] This is the ratio required to get a minimum entropy for a numbered message source.

Actually the technique is quite old in specialized applications. A good example is the lanterns in the Boston old north church steeple: 'One if by land and two if by sea'.[2] If the code book is secure, and this is normally a major problem, then such a code is almost totally safe from decryption because the number of message repetitions, which is normally the main weakness in code security, is low. However, such an alphabet cannot be of general use. It is often impossible to predict all messages which could be sent from a source, and the problem of distributing code books is enormous.

If such a code is constructed, messages can be very long and will not have the instantaneous property in that the entire message will have

---

[1] Zipf's Law, or variations on it, occur in a number of different circumstances. See Knuth, Vol. 3, pp. 397-399.

[2] Paul Revere's Ride.

to be sent before it can be decoded. The effect of even a single bit transmission error will be particularly disastrous.

In any normal situation the number of message numbers will grow exponentially with the possible message length. Additionally, almost all the possible messages will not be transmitted; in the worst case for a number of messages N the maximum word length will be $Log_2(N)$. Thus, a practical limit has to be put on the transmission block size and the set of possible messages is now constructed from these blocks. For maximum compression the coded block size should be large enough to allow for the Markov transition probabilities between the internal component words, or at least a large proportion of it.

The nature of this block alphabet will be the main factor in the brevity and security of the messages coded from this alphabet. Also, the alphabet will now normally have to be coded to produce decodability as Lennon showed (1978), and this will add to the size of the codes though not more than $Log_2(Log_2N) + 3$ bits, for an alphabet of N symbols, as was shown by Rosenberg (1978).

H. Dictionary Codes

One possible code which is indicated by Zipf's law is to generate a code in which each word of English text is coded with a code length approximately proportional to its inverse probability of occurrence. Thus coding would be simply looking up each word in a dictionary.

If such a code was constructed, however, the table of words would have to be several million characters long, something that is only marginally feasible on today's machines. The programming complexity

would be high and such an approach should be possible only by the late 1980's. Programming techniques, such as hash codes, mean that it may not be necessary to actually store complete words.

The next sections consider the security of some codes, and show in what way compression can increase their security.

SECTION XI. IRREVERSIBLE CYPHERS

A. Password Encryption

There is a class of computer codes which, contrary to the normal concept of codes, does not have to be decoded. These are codes to be used as passwords and keys. It is very bad security for these passwords to be retained in their decrypted form. (This is the case with the IBM VSAM catalog or the VM/370 CP Directory passwords.)

Instead the password should be encrypted in some way, and only this should be retained, preferably as a binary string. Then if it is accidentally or maliciously obtained, it still cannot be used to breach security. Also, it will not stand out in a 'core' dump.

Passwords should be respecified directly by the user, and frequent changes should be mandated in a manner similar to that used in the IBM RACF package. What is important is that this can be done without involving a third person such as a security officer or system programmer. Since there is, and should be, no programmed way of finding a user's password, an override procedure should be available which will change any password to a one time password to allow a forgetful user to 'log on' and then to force the user to specify a new password at once.

There is no requirement that the password encryption should lead to a unique transform as long as the range of output codes is sufficiently large. Since only encrypted passwords are held, even if such an encrypted password is fraudulently obtained, security will not be breached as the password will not transform to itself. This is an obvious design parameter of the encryption process.

A possible algorithm would be, given an 8 character password, treat each 2 characters as a binary number N_1,N_2,N_3,N_4 and feed it into a polynomial such as:

$$Y = 3N_1^7 + 71.N_2^5 + 76.N_3^3 + 2.N_4^2 + 3456789, \text{ mod } 2^{32}.$$

For an example, the key 'arabian' would transform to the number -531,351,669, and 'camel' to -1,828,196,288.[1] A subroutine to implement this algorithm on an IBM 370 computer is attached as Appendix A.

B. Hash Codes

Hashing codes for superencypherment is another use for irreversable cyphers. The hash code should also be transformed, and should be used to make a long pseudo random string, for a Boolean algebraic exclusive or, from a shorter number of key phrases. In the case of the cyphering key being disclosed accidentally, security would be preserved to a degree if the algorithm was secure. (In normal cryptographic work the algorithm used is considered to be generally known.)

Normally such a key is used to cypher the message or record starting again at each record. This appears to make possible a known text attack by analyzing repeated strings in either the input or output which might be caused by a date, a customer name, a null or zero field or some other relatively invariant field. As Shannon showed, only if the key does not repeat over the entire message space, giving an effectively infinite key,

---

[1] On digital computers Mod 2^N arithmetic, where N is the word length, is much faster than any other base though perhaps slightly less secure.

do we have total security. While this is normally impossible, we can get an almost arbitrarily large non-repeating, though not truly random, key. We can generate a series of 'random' keys by using a good random number generator starting from a secret seed derived from the user's password in the way described above. By letting the pseudo random key continue to propagate through each successive record, we achieve a high degree of security, but only at the cost of being unable to read the file in any way except sequentially.

A circumvention is to use the same record index we used to read the record as either a secondary or primary cypher key. If the records are to be retrieved only by a single key path, and never sequentially, this method could lead to a very good security; however, these pre-conditions will prevent the use of almost all database packages where the use of multiple keys to access the data is generally essential. In that case we have to repeat the cyphering on each record with the same key, and so an antagonist's frequency analysis on each position of the record should be successful.

A superencypherment or a transposition cypher will be helpful, but the cypher can still be broken. It appears that we are stuck with somewhat flawed methods, and the object of study is to achieve as high a degree of security as is possible with a limited cost in other requirements.

A mechanism to generate a superencypherment key has been supplied, according to the method outlined above, in the compression and decompression routines supplied in Appendix B. This uses a simple random

number generator, of the linear congruential type[1] using a user supplied number as a seed; any key that cycles in too short a period is rejected. In the program supplied this feature is optional, but it could easily be made compulsory.

---

[1] See Knuth, Vol. 2, pp. 305-339.

SECTION XII. THE DES STANDARD CODE.

A. Description

This code[1] was proposed by the United States National Bureau of Standards to provide a secure, and relatively simple method of encrypting data both for communication, and on archival storage devices. It is also sometimes known as the NBS Code. The method is simple to implement in a hardware device or 'chip'. The method used is a permutation of the data, taken in 64 bit blocks by a subsection of a bit key. While 64 bits were the limit of technology at the time the standard was initially proposed (1976), it is clear that this was too short, in that insufficient allowance was made for the progress of technology, and that 128 or 256 bits should have been used, at least as an option.

B. Security Problems

Concentrating on this weakness, Diffie and Hellman (1977) suggested a method for the regular cryptanalysis of these codes, by building a computer to try all possible keys in parallel in which solutions on a per code basis would be produced at a relatively low cost, and in a matter of hours. This is aided by the fact that only part of the key is used at each permutation. IBM has gone so far as to state that they will not build a 'Diffie' machine, which tends to confirm its feasibility.

There have been suggestions that this weakness is, at least in part, deliberate, so as to allow such organizations as NSA at least a chance

---

[1] Technically speaking this is a cypher — but it is normally called a code in the literature.

of breaking transmissions using the DES codes. Also see Campbell.

A different approach to breaking the codes might be successful. As shown above, all codes can be treated as substitution cyphers. Given the massive amounts of data which would be intercepted on a stolen tape, the 64 bit block would be liable to frequency analysis, and so could be solved as an 8 character substitution cypher.

Given enough intercepts with aid of a high speed computer, a solution should be possible. The structured nature of computer data and the large number of blanks will aid any solution. With uncompressed data, and assuming normal English text with an entropy of 3 bits per 8 bit ebcdic character, the unicity distance can be calculated as about $\text{Log}_{10}(6561!) \doteq 200{,}000$ characters, which is not large enough to preclude a partial solution. In his paper, Diffie pointed out the disadvantages of using ASCII for coding messages to be encrypted by DES as the parity bit will aid any attempt to break the DES code.

If a tape could be copied on a regular basis, a reasonable assumption, there is an even easier approach. Assume that we are interested in a firm's file of customers as an example. Simply add a spurious customer, or become one ourselves, and find the new or changed records within the file. This will give an antagonist several probable code meanings, and as orders are added, meanings could be deduced.

Even without devices such as those suggested by Diffie, a competitor could deduce a large and possibly fatal amount of information.

C. Counter Measures

The obvious counter measures are: firstly, to apply the maximum feasible security measures, secondly, to pre-compress the data with Huffman code or equivalent.

Frequent changes of code are not always effective for an obvious reason. If it had been deduced that code group 'X' means, say, 'J. Smith', and this is found to be replaced on the next generation of data by group 'Y' in the same position, it is fairly clear what group 'Y' means. All that has been achieved is to build a false sense of security in the authorized users at the cost of some minor inconvenience. However code changes are effective where the plaintext is different for each generation. This is closer to the classical cryptographic problem.

SECTION XIII. PUBLIC KEY CRYPTOGRAMS.

A. Description

Public key cryptograms, also called trapdoor codes, recently have been developed to deal with two major problems of cryptography: message authentication, and code distribution. The method has been described in a number of articles particularly by Hellman (1979). The term trapdoor code comes from the analogy with a trapdoor which is easy to fall through, but difficult to get out the same way. The term public key cryptograms describes a particular class of transform on the data such that the transform can be made public, but the converse transform, while uniquely determined, can still be held secret. In many of these the algorithm is such that, for authentication purposes, the converse transform can be applied before the public one and produce the original plaintext.

Two methods of achieving this have been produced: the first by Rivest *et al.* (1977) uses the factoring into primes of very large numbers, the second produced by Diffie and Hellman (1976b) uses two related matrices to convert a simply solvable knapsack problem (that is, the problem of which of a set of N numbers was added to form a particular sum) into a computational insolvable one, and *vice versa*. The first of these matrices, only, is made public. Both these cryptosystems are NP complete problems in that the principle of solving the code, by either factoring the product of primes, or finding a solution to a knapsack, takes exponential time with current algorithms.[1] It has been proved that

---

[1] See Knuth, Vol. 2.

the full breaking of the code would be an equivalent problem. Figures of two billion years have been quoted. Since history is full of unbreakable codes which have fallen, the cynical might suspect that this could be an overestimate. Consider the Japanese code purple which was felt, by the Japanese, to be utterly unbreakable.[1]

B. Security Problems

As shown in Section X, in all cases but the one time pad, coding consists of substituting strings of data for other strings in a repeatable manner, effectively as a block substitution cypher. Now these strings can be attacked by the normal tools of cryptography, such as frequency analysis. In public key cryptograms these strings are, hopefully, very long, but this is only a partial protection.

In the case of trapdoor codes, one can describe a method which will, in principle, give partial solutions to these codes in a rather shorter time. Since this is very similar to the method suggested by Diffie for attack on the DES standard code, it is surprising that it has been overlooked. The trapdoor code has one feature that makes it particularly vulnerable and that is the ability of an antagonist to generate coded messages at will with known text.[2]

This makes this type of code particularly open to attack. Firstly the fact that the interceptor of a message can guess the probable texts

---

[1] This was a wired rotor type cypher machine developed in 1937. See Kahn, pp. 21-27.

[2] This is normally extremely difficult if not impossible with all other codes.

in some cases, and can then code these probable plaintexts with the public code, and compare the resulting cyphertext. This alone could have disastrous consequences, though paraphrasing of messages, could certainly give a measure of protection.[1]

The second problem of public codes is more basic. Both methods so far proposed translate the message to a numeric string. The method suggested in the published paper is a basic substitution code of space=00, A=01, B=02 etc. in which 'cab' would become 030102. This seems needlessly inefficient; P.E. Chase was suggesting far more sophisticated methods 120 years ago. If the letters were coded as before but each number was added and then multiplied by 27, rather than 100 as suggested, then 'cab' could be coded as 2234, and the cryptogram could contain four times as much data with no loss of information. Then the redundancy would be reduced by 70%.

With single messages this gives no advantage, as the trapdoor cypher methods produce text of constant length which are slightly shorter on average than the input message string. However, there is inevitably a maximum length of string after which the remainder of the message goes into subsequent strings.[2] With the past full of impregnable cyphers broken, this seems to give the antagonist an unnecessary advantage.

The rules of the English language will limit the number of possible

---

[1] Other easy protection methods are deliberate misspelling of words or translation into another language and then back.

[2] In the RAS algorithm of Rivest *et al.* the number generated from the string must be less than the product of the two primes.

messages to a subset of the maximum possible which would be 27^N where N is the maximum number of characters that can be coded in a block. From the studies on the entropy of the English language quoted above there will probably be only 1/7 of this number. If a device were built to try all the possible messages under a public key and to construct a code book of these messages, the cost would be substantially less than the device proposed by Diffie to crack the DES codes. With perhaps 1000 'chips', each capable of producing a coded text in 1/1000th of a second, a code 'book' of 3,600,000,000 possible solutions could be built up in an hour. This would not break all messages or parts of messages, but it would produce partial solutions in rather less than the 2 billion years envisioned.[1] Thus the local code security will be breached, though the code will be likely to remain globally secure.

C. Countermeasures

Obviously, the more the data is compressed the greater the possible number of blocks and the less such an attack will produce a worthwhile effort. Another partial protection from this cryptographic attack is quite easy, and that is to add to every string of data one or two non-sense words, such as 'ccchft'. This is similar to the nulls which are often used in other, noncomputer, coding techniques.

Thus with maximum compression of data, paraphrasing and very long strings the attack outlined should fail, or at least be an order of

---

[1] During World War II solutions to cyphers such as Purple were never complete and some groups could only be guessed.

magnitude more difficult and expensive.

Since these codes are designed for the public who will probably not perceive the need for the extra security methods, this weakness will probably not be corrected.

SECTION XIV. COMPUTER SECURITY.

Typically, computer security, at least up to the mid 60's, consisted of keeping everyone out of the computer room — an idea which still has proponents. While this is partially adequate for a batch site, once a computer terminal is attached to the computer by a remote line, the problem of security transcends physical security. If the terminal is attached to the system through TSO, CMS, or some other interactive programming package, there is a still greater problem of data access and security. The data stored at a computer site, and to a lesser degree in off site backup storage, needs protection in three basic ways. Protection in each case must be different.

A. Browse Protection

This covers users looking at sensitive data such as payroll or customer name files, but not trying to alter this information. In many cases confidential data have been widely distributed in many companies with senior management being totally ignorant of this security breach.[1] Since in many cases simple curiosity is the cause, even a simple hashing code or password would make access difficult enough to discourage the majority of violators. However, access to very sensitive data must be logged and passwords must be separated from the access programs where possible.

---

[1] I have known, personally, of a programmer who kept a print out of the salary file in his desk, so as to be able to look up salaries of people he met.

Browse protection must protect both against the accidental exposure and the deliberate search for information. Security can be breached by a small program being loaded into the same memory as a large one and then a program bug, or a deliberate program fault, causing a core dump.[1] This will give a picture of the last program data in this memory area, possibly fully decrypted (or maybe just the last password). Certain, but not all, operating systems have protection against this.

B. Update Protection

This covers users altering files, again such as payroll or debtor lists, often with criminal intent. Protection in this case is nominally easier, as general ledger balances should show misuse, and passwords are more common and acceptable for write access. However the criminal often turns out to be a present, or past, authorized user who knows how to circumvent these controls. In fact the start of their felonious attempts is often the discovery of some weakness in security, and they are generally using an otherwise legitimate transaction.

Newspapers seem to be full of the stories of people unfortunate or incompetent enough to be caught. For an example, they have been caught by getting married with the name change causing an error, or, in one recent case, the bank wished to give an award to the first and last customer account — the criminal had a ficticious name of 'ZZEIBF'. The consensus is that computer criminals are common, but are very rarely caught and are still more rarely prosecuted.

---

[1] A print out of the content of memory, normally in character and binary form.

The people involved do not think of themselves as really criminal in the way a bank robber is; to steal from a computer gives insulation.

C. Trojan Horse Protection

A Trojan horse is a computer program which, while fulfilling a legitimate purpose, also has illigitimate, quite different, purpose buried in the program code. This is smuggled into the computer system and the special instructions are 'switched on' by an apparently harmless set of circumstances.

Examples include sending copies of orders to an unauthorized terminal or debiting other accounts to credit a special Swiss bank account. Despite popular mythology the only common examples seem to be date testing routines, left by disgruntled employees, which destroy system or master files after a certain date is passed.

The best protection against this is good operation procedures. Of course, never let fired personnel have any further access to the computer.

D. Other Problems

The misuse of computer time, for games and hobbies, is a gray area rather like use of an office XEROX. A total restriction is probably unenforceable and has, in at least one case, led to a criminal, who was in charge of a programming department, blackmailing minor offenders to ignore his major crime. It is better to license a strictly limited amount of 'self development' than to make 'criminals' of otherwise honest staff.

Data encryption will certainly add to security, but in itself it

is not enough. It must be combined with a password system which logs all access to secure data and notifies the appropriate personnel of any access attempts with incorrect passwords. H. Gladney gives a good review of this area.

SECTION XV. COMBINING COMPRESSION AND SECURITY.

A. Analysis

In order to combine compression and security the procedures below could be followed on any computer file. The object is first to remove as much redundancy as possible from the data. Then further security measures could be applied according to the sensitivity of the data; this latter must be a matter of judgement. The steps suggested are:

Print a portion of the file. Look for standard fields; these can be of various types.

B. Conversion

Numeric data. Convert to binary using only the number of bits required to hold the data and its sign. That is, if the number is 3 signed digits, 11 bits will contain it.

Tabular data. See if a table can be constructed for a field which contains a known range of data such as car models, city or country of birth. Again use only the required number of bits to index to the table entry. For example, if there are 100 possible entries, 7 bits ($2^7 = 128$) will hold the index. Surnames are possible as tables if some method is devised for handling the occasional rarer name. (Knuth, vol. 3, pp. 486-492 gives some example. See also Melder.)

All binary fields should be grouped together, as it will probably be necessary to round binary fields up to a byte boundary.

C. Blank Compression

Alphabetic data. This will have trailing blanks (leading in Arabic), and a considerable space saving can be achieved by suppressing these and giving a byte count at the beginning. Another method of blank suppression is to use the COBOL or PLI 'string' function to break out compressed fields, which requires that a special character not in the text is used as a delimiter.

D. Recoding

If the alphabetic characters are pure (that is, contain only a small subset of the full character set of the computer), convert to another code and pack these into a string. Five bits will code upper case letters with some punctuation and six will add lower case or numbers.

A better generalized solution is to generate Huffman code from your data and use this to code the data. Appendix B contains a program which will generate a Huffman code at the byte level for any data on an IBM computer. This program, and the programs to code and decode the data, also given in Appendix B, could be generalized for other machines if required.

An alphabetic ordered code should be considered if part of the data is used for a key. In order to give fast and efficient access, key fields, particularly if they are a low percentage of the data, should not be compressed, but numeric fields should still be packed.[1] If access

---

[1] Watch the sign, +1 can become X'1C' or X'1F', -1 becomes X'1D'.

is changed from index by key to truly random, by randomizing the key to give a block address, the objection to compressing the key disappears. This can require a redesign of the application and a careful consideration of overflow chains. It is sometimes advantageous to build a separate key field derived from the base field, as this could speed access while still allowing full compression.

A still higher compaction on alphabetic data can be achieved if the Huffman code is build on groups of 2 or 3 characters. The program in Appendix B could be modified for this if required, though it would require a larger amount of computer memory to run. If a multi-character Huffman code is used there will be little extra space saved by blank compression for the count field will probably be coded by as many bits as several blanks. The program supplied can treat multiple blanks as a single code group which maximizes compression with certain types of data. By extending this to the groups shown to be common with other types of data, a secure and compact code could be generated with a moderate program size and run time.

With textual data another technique is to code words and build a Huffman code on words, or at least the five thousand most common. This should end up with a code that contains about two bits per input character.

If successive records of a sequential file contain many common fields a special count field could indicate this with a substantial space saving. Also, if several fields are empty or null this can sometimes be indicated in the first of the empty fields. It should be noted that as the space character is often represented as a single bit with

Huffman code, replacing a number of blanks with a count is sometimes less efficient than coding each blank separately.

For example, if 7 bytes of blanks could be replaced with the value HEX'70' seven blanks will take 7 bits, while HEX'70' will probably take in the range of 10 - 18 bits. Only blanks at the end of each record should be dropped in such cases.

E. Variable Length Records

Pack the data into variable length records. With standard software this adds 4 characters to the record length but a sophisticated user might consider using only 2 bytes. Security would be increased since this would make the file unreadable by normal manufacturer's software.

F. Superencypherment

Now apply a security superencypherment. The routines supplied in Appendix B allows the user to supply an 8 digit number, which is used as a seed of a random number generator, to generate successive strings of data which are used to superencypher the compacted data. Other methods could be used, such as the DES standard code or one of the public key cryptograms depending on the required degree of security. This must be a matter of judgement.

SECTION XVI. CONCLUSIONS.

It is clear that the regularity of computer data can reduce security. Data compaction will destroy much of the regularity, and thus enhance security.

During the next decade the increase in capacity of the newest generation of computers is likely to be limited by lack of fast secondary storage devices. The gains in raw CPU power will not be matched by the modest gains in magnetic storage capacity and speed. Under these circumstances there should be an increasing tendency to use some of the otherwise unharnessed CPU cycles to compress data. This will reduce costs and make the best use of the limited secondary storage.

This compression of data will have the secondary affect of increasing security by making the data less readable by unauthorized users. The increase in security gained from removing redundancy follows naturally from Shannon's security theorem in the 'Theory of Secrecy Systems'. Despite this, Hellman (1977) states that this result is 'unexpected' (p. 291). Apart from this, the covariation of security, and compression, is almost unknown.

This thesis has shown that this result is predicted by theory. Many practical demonstrations showing compression and security as useful or necessary companions have been given. It has also been shown that compression, except under non-realistic charging algorithms, will give a cost saving. It is suggested that security and compression routines should be supplied as standard interfaces to computer operating systems, which could remove much of the load of security and compression

of data from the actual end user. The user would then be restricted to specifying what security he wants for his data, and passwords, and security codes.

Recently, several new methods of cyphering data have been produced for use on computers; while they are, in principle, much more secure than historic cyphers, this thesis has shown that misuse will imperil their security and that compression of data before security encryption will improve the security of these new codes and will often decrease cost.

A section on irreversible cyphers gives some applications of cyphers which have no inverse transform; they are particularly useful when storing data for validation of passwords.

As Appendixes, a number of computer programs have been supplied. These are written in IBM 360/370 assembler. Appendix A is a single program which will convert a string of 8 characters into a 31 bit signed binary number as an example of irreversible cypher. Appendix B consists of three programs to generate the optimum Huffman code for a data sample and to code and decode data; this data should have close to the same frequency distribution.

The decompression routine in the last program should be converted to a simple 'tree' method once a particular data probability distribution is fixed upon. The method given allows fast alteration for different data samples but will be relatively inefficient.

SECTION XVII. DEFINITIONS.

Antagonist. A term common in cryptography describing the unauthorized individual who is attempting to read the plaintext of coded messages.

Baud. A unit to measure the speed of transmission. This is named after Baudot, a pioneer of telegraph transmission studies. It was originally twice the number of Morse code dots transmitted per second, but now is usually defined simply as bits per second.

Block Code. This is a code in which the same set of symbols from the message source is always encoded in the same way. This is not generally desirable for maximum security.

Channel. This is the communication path between the CPU and the various peripheral devices. This is a complicated interface processor with, normally, separate access to the main computer memory.

Cyphertext. Ths fully coded message which is ready for transmission or storage is called cyphertext.

Controller. To cut cost, the logical control of a number of computer peripherals is concentrated in the controller, which will be the interface to one or more of the channels.

CPU. (Central Processing Unit) This is the main arithmetic and logic component of a computer. This abbreviation often describes the time spent by a computer activity in this type of function.

Cryptogram. A coded message to be decoded without the key being available (i.e. an unauthorized decoding is implied) is a cryptogram.

Cryptography. This is the science of codes and cyphers.

Cryptanalyst. This is a person (other than the authorized recipient) who is trying to decode a message. (See antagonist)

Cycle or **Key Cycle.** When a cypher is used in which a different part of the key is used with each code block it is the distance between reuse of the same sub-key. Thus, it is 1 for a simple substitution cypher, 26 for a vignere, and infinite for a one time pad. It is 1 for the DES code and public key cryptographs.

DES. The United States National Standard Bureau data encrypting standard algorithm is abbreviated to DES.

Entropy. The entropy S of a source of symbols representing events is the average number of binary codes required to identify the events, or the average length of code required to code the events. Thus a random, ebcdic source has a source entropy of 8 and an ASCII source one of 7, while BCD has an entropy of 6.

Instantaneous Code. This is a code which can be decoded immediately upon receipt without waiting for any more data from the source. (See prefix codes and the Kraft inequality.)

Intercept. The portion of coded message(s) which an antagonist has intercepted or received is called the intercept. The amount required before a good cryptologist should be able to break the code is a measure of the security of the message and is known as the Unicity distance.

I/O. This common computer abbreviation refers to input and/or output.

Key. The cyphering and decyphering process consists of applying some particular processes on the data with a key which can be periodically changed. Kerchkoffs' Law states that secrecy refers to the key and not

the secrecy in the particular algorithm.

Kraft inequality [KR01]. Given an alphabet message source $a = a_1, a_2, a_3, \cdots, a_m$ with assigned code word lengths of $\ell_1, \ell_2, \ell_3, \cdots, \ell_m$, assume a code alphabet of r numbers. A necessary and sufficient condition for the existence of an instantaneous code with the word lengths above is that the $\sum_{i=1}^{m} r^{-\ell i} \leq 1$. This is the Kraft inequality.

Plaintext. The original, uncoded message which should be recovered on full decoding of the cyphertext is called plaintext.

Prefix code. This code can always be uniquely decoded, as no code is a prefix of any other code. Such a code is also by definition instantaneous, so code 0, 01 and 001 is not a prefix code because the word 0 is a prefix to 01 and 001. Morse code is an example of a common code which is not a prefix code.

Redundancy. This is the measure of the information actually contained in a message, as against the amount theoretically possible to code with this source alphabet. For a source entropy $S(M)$ and a maximum possibly entropy for the same source $S_{MAX}(M)$, the redundancy D is

$$D = 1 - S(M)/S_{MAX}(M)$$

$$0 \leftarrow D \leftarrow 1.$$

Superencypherment. This is the successive application of separate encyphering processes upon the plaintext. Sometimes, but not always, the resultant cyphertext is more secure.

Unicity distance. This measures the quantity of cyphertext needed to give an unambiguous translation into plaintext, without the key being available. See also intercept. In general for a cypher of K

symbols with redundancy, D, the unicity distance U(A) is

$$U(A) = \text{Log}_2(K!)/D.$$

Vernam Cypher. This device encryphared telex in which two short sets of random codes, prepared on paper tape, with different cycles were logically merged together to produce a key with an effective repetition cycle of about one million characters.

Vingere Cyphere. This is a substitution cypher where each successive letter is encoded according to the formula:

On the first letter, A=C, B=D, C=E ... Z=B;

On the next letter, A=D, B=E, C=F ... Z=C

and so on. There can be up to 26 cyphers before a return to the beginning of the cypher. If the cycle is one, this is a Caesar Cypher.

SECTION XVIII. SELECTED BIBLIOGRAPHY.

Abramson, Norman. 'Information and Coding Theory'. New York: McGraw Hill, pp. 12-89, 1963.

Anacker, Wilhem. 'JJ-Based Processor with 1 NSEC Cycle Planned by IBM'. Speech quoted by Computer World. October 1, 1979.

Adabas Introduction (ADA-410-000). March, 1979. pp. 14-15, Software ag.

Ash, Robert. 'Information Theory'. Chapter 2, pp. 21-24. New York: John Wiley (Interscience) 1965.

Auerbach Data Processing Manual. 'Distributed Embezzlement'. 1-05-16, 1978.

_____. 'Security Risk Assessment'. 1-03-01, 1976.

_____. 'Internal Security Problems'. 1-03-01, 1973.

_____. 'The Audit of Security'. 2-06-07, 1976.

_____. 'Designing Proper Controls for Terminals'. 2-06-09, 1977.

Bacon, Francis (1561-1625). 'The Advancement of Learning'. London, 1606.

Bayes, Thomas (1702-1761). Paper in Philosophical Transactions, 1763.

Bently, Jon Louis. 'An Introduction to Algorithm Design'. Computer, February 1979, pp. 66-78.

Blom, Rolf J. 'Bounds on Key Equivocation for Simple Substitution Cyphers'. IEEE IT-25, pp. 8-18, January 1979.[1]

British Museum Library. Additional MS 9800 FF134-5.

Botto, Mario. 'Data Protection Methods'. Canadian Datasystems, pp. 47-48, April 1978.

Bolour, Azad. 'Optimal Properties of Multiple Key Hashing Functions'. J. of the ACM, pp. 196-210, April 1979.

Brassard, Gilles. 'A Note on the Complexity of Cryptography'. IEEE, IT-25, pp. 232-233, March 1979.

---

[1] IEEE always refers to the IEEE transaction on information theory.

Brillouin, Leon. 'Science and Information Theory'. Chapter 9, pp. 114-116. New York: Academic Press, 1962.

Campbell, Duncan. 'The Danger Signals Flash'. Computing Europe, October 1978.

Chase, P.E. 'Mathematical Monthly'. (Article) 1859.

Cocke, J. & Raviv, J. 'Data Compaction and Security System'. IBM Technical Disclosure Bulletin, vol. 14, no. 8, January 1972.

Considine, J.P. & Myers, J.J. 'MVS Archival Storage and Recovery Programs'. IBM Systems Journal, vol. 16, no. 4, pp. 378-397, 1977.

'Comterm 140-1 User's Guide'. Comterm Ltd. publication, 990-1400-00, September 17, 1979.

'Data Encryption Standard'. U.S. Federal Information Processing Standard, Publication 46, National Bureau of Standards, Dept. of Commerce, Washington, D.C., January 1977.

Diffie, W. & Hellman, M. 'New Directions in Cryptography'. IEEE, IT-23, pp. 644-654, November 1976.

_____. 'Multiuser Cryptographic Techniques'. National Computer Conference, 1976.

_____. 'Privacy and Authentication'. IEEE invited paper for May 1979 issue (draft).

_____. 'Exhaustive Cryptanalysis of the NBS Data Encryptation Standard'. Computer, pp. 74-84, June 1977.

Ehrsam, W., Matyas, S., Meyer, C. & Tuchamn. 'A Cryptographic Key Management Scheme for Implementing the Data Encryption Standard'. IBM Systems Journal, vol. 17, no. 2, 1978.

Ellis, Clarence A. 'Analysis of Some Abstract Measures of Protection in a Computer System'. International J. of Computing Science, vol. 7, no. 3, 1978.

Fano, R. 'Transmission of Information', MIT Press and John Wiley, 1961.

Friedman, W.F. 'Cryptology' in Encyclopedia Britanica, p. 848, 1973.

_____. 'Elements of Cryptanalysis'. Aegean Park Press, 1976.

Fussenger, Frank & Gabow, Harold N. 'A Counting Approach to Lower Bounds for Selection Problems'. J. of the ACM, pp. 227-238, April 1979.

Gallager, Robert G. 'Information Theory and Reliable Communication'. p. 65. New York: John Wiley, 1968.

Gilbert, E.N. & Moore, E.F. 'Variable Length Binary Encoding'. Bell Systems Technical Journal, July 1959.

Gladney, H.M. 'Administrative Control of Computer Service'. IBM Systems Journal, vol. 17, no. 2, pp. 151-178, 1978.

Gravina, C.M. 'National Westminister Bank Mass Storage'. IBM Systems Journal, vol. 17, no. 4, pp. 344-358, 1978.

Guiasu, Silvi. 'Information Theory with Application'. Chapter 6, pp. 126-131. New York: McGraw-Hill, 1977.

Hanaratty, M.E. Speech at National Computer Converence quoted in 'Computerworld', June 25, 1979.

Hallman, Grant. 'How to Squeeze Data into Smaller Spaces'. Canadian Data Systems, pp. 27-99, September 1978.

Hellman, Martin E. 'An Extention of the Shannon Approach to Cryptography'. IEEE, IT-23, no. 3, pp. 289-293, May 1977.

_____. 'The Mathematics of Public Key Cryptography'. Scientific American, pp. 130-139, August 1979.

Hoagland, A.S. 'Storage Technology: Capabilities and Limitations'. Computer, pp. 12-18, May 1979.

Hoffman, Lance. 'Modern Methods for Computer Security and Privacy'. pp. 65-67, 1977.

Huatek, Eugene R. 'Higher Density Technologies'. Computer Design, vol. 19, no. 2, pp. 156-159, February 1980.

Huffman, D.A. 'A Method for the Construction of Minimum-Redundancy Codes'. Proc. IRE, vol. 40, pp. 1098-1101, 1952.

IBM VM Facility/370 CMS Commands and Reference Manual, GC28-1818-1, pp. 50-51, 1977.

OS/VS2 MVS Resource Access Control Facility (RACF) General Information Manual. GC28-0722, 1977.

IMS/VS Version 1 System/Application Design Guide. 5740-XX2. REL 1.5, pp. 4.128, 1978.

Ingles, Franklin M. 'Information and Coding Theory', Chapter 5, pp. 108-134. Intext.

Kahn, David. 'The Code Breakers'. Macmillan, 1967.

Katona, Byula O.H. & Nemetz, Tibor O.H. 'Huffman Codes and Self Information'. IEEE, IT-22, 1976.

Kerchkhoffs, A. 'La Cryptographie Militaire'. Paris, 1883.

Knuth, Donald E. 'The Art of Computer Programming - Vol. 1 - Fundamental Algorithms'. Chapter 2, 1969.

_____. 'The Art of Computer Programming - Vol. 2 - Seminumerical Algorithms'. Chapter 3, pp. 9-17, 1969.

_____. 'The Art of Computer Programming - Vol. 3 - Sorting and Searching'. Chapter 6, 1969.

Kraft, L.G. 'A Device for Quantizing, Grouping and Coding Amplitude Modulated Pulses'. M.S. Thesis, Electrical Engineering Dept. M.I.T. 1949.

Leung-Yan-Cheonbor, Sik K. and Cover, Thomas M. 'Some Equivalences between Shannon Entropy and Kolmogorov Complexity'. IEEE, IT-24, no. 3, pp. 331-338, 1978.

Lennon, R.E. 'Cryptographic Architecture for Information Security'. IBM Systems Journal, vol. 17, no. 2, pp. 138-150, 1978.

Lewis, T.G. and Smith, M.Z. 'Applying Data Structures'. Houghton Mifflin Co., Boston, 1976.

Matyas, S.M. and Meyer, C.H. 'Generation, Distribution, and Installation of Cryptographic Keys'. IBM Systems Journal, vol. 17, no. 2, 1978.

McIlroy, M.D. 'The Number of 1's in Binary Integer Bounds and Extremal Properties'. SIAM J. of Comp., vol. 3, no. 4, December 1974.

Merkle, R.C. 'Secure Communications over Insecure Channels'. Communications of the ACM, pp. 294-299, April 1978.

Melder, Lloyd. 'A data compaction technique'. Personnel Communication - BCSC, 1978.

Pierce, J.R. 'The Early Days of Information Theory'. IEEE, IT-19, no. 1, pp. 3-8, 1973.

Poe, Edgar Allen (1809-1849). 'The Gold Bug'. Philadelphia Dollar Magazine, 1843.

Rissanen, J. and Langdon, G.G. Jr. 'Arithmetic Coding'. IBM Journal of Research & Development, vol. 23, no. 2, pp. 149-162, March 1979.

Rivest, R., Sharmir, A. & Adleman, A. 'A Method for Obtaining Digital Signatures and Public Key Cryptograms'. MIT Technical Memo LCS/TM82, April 1977.

Rivest, Ronald. 'The Impact of Technology on Cryptography'. ICC, 1978.

Rosenberg, Arnold L. 'Data Encodings, and Their Cost'. ACTA Information, vol. 9, pp. 273-292, 1978.

Rubin, Izhak. 'Data Compression for Communications Networks: The Delay - Distortion Function'. IEEE, IT-22, no. 6, pp. 655-665.

Sayers, Dorothy L. 'Have His Carcass'. Collancz., Chapter 28, 1932.

Shannon, C.E. 'Mathematical Theory of Communication'. Bell System Technical Journal, vol. 27, 1948.

_____. 'Communication Theory of Secrecy Systems'. Bell Technical Systems Journal, vol. 28, no. 4, pp. 656-715, October 1949.

_____. 'Prediction and Entropy of Printed English'. Bell Technical Systems Journal, vol. 30, pp. 50-64, January 1951.

Smith, Neil and Wilson, Deirdre. 'Modern Linguistics'. Penguin Books, pp. 52-58, 1979.

Snyderman, M. and Hunt, B. 'Myriad Virtues'. Datamation, pp. 36-40, December 1970.

Svanks, Maija I. 'Optimizing the Storage of Alphanumeric Data'. Canadian Datasystems, pp. 38-40, May 1975.

Tritton, A.S. 'Teach Yourself Arabic'. Holder & Stoughton, pp. 13-14, 1977.

Upton, Molly. 'Memory Seen Dominant in 2000'. Computerworld, pp. 1-5, September 24, 1979. Quoting J.B. Keyes.

Visvalinham, M. 'Indexing with Coded Deltas — A Data Compaction Technique'. Software Practise and Experience, vol. 6, pp. 403.

Williams, Dewi L. 'Keeping Computer Files Confidential'. Canadian Datasystems, pp. 40-48, August 1974.

Young, J.F. 'Information Theory'. Butterworth (London) 1977.

Ziv, Jacob and Lempel, Abraham. 'A Universal Algorithm for Sequential Data Compression'. IEEE, IT-23, no. 3, May 1977.

Zipf, G.D. 'Human Behaviour and the Principle of Least Effort'. Addison-Wesley (1949).

APPENDIX A: APPENDA ASSEMBLE

```
TITLE 'CREATE IRREVERSABLE CODE FROM 8 BYTE PASSWORD.  T.F. MURPHY'
*********************************************************************
*                                                                   *
*    THIS PROGRAM TAKES THE 8 BYTE PASSWORD PASSED (ADDRESS IN REG 1)*
*    AND CONVERTS IT TO A BINARY NUMBER BY THE ALGORITHM BELOW.     *
*    THE NUMBERS CAN EASILY BE CHANGED.                             *
*                                                                   *
*                                                                   *
*       Y = 3.N1 ** 7 +71.N2 ** 5 + 76.N3 ** 3 + 2.N4 ** 2 +3456789*
*                                                                   *
*(MOD 2**32).                                                       *
*                                                                   *
*    EVEN WITH THESE NUMBERS KNOWN THERE IS NO REVERSE TRANSFORM    *
*    EXCEPT TRYING ALL POSSIBLE KEY WORDS.                          *
*                                                                   *
*********************************************************************
        SPACE
        REGEQU
        SPACE
CSECT   CSECT
        USING *,15
SAVE    SAVE  (14,12)       OMMITT ,* FOR SECURITY
        LH    R2,0(R1)      GET THE HALF WORDS
        LH    R3,2(R1)      GET THE HALF WORDS
        LH    R4,4(R1)      GET THE HALF WORDS
        LH    R5,6(R1)      GET THE HALF WORDS
        SPACE
        L     R1,=F'3456789'          INITIALIZE OUTREG
        LR    R9,R2                   TO WORK
        MR    R8,R2                   MULTIPLY TO **2
        MR    R8,R9                   MULTIPLY TO **4
        MR    R8,R2                   MULTIPLY TO **5
        MR    R8,R2                   MULTIPLY TO **6
        MR    R8,R2                   MULTIPLY TO **7
        M     R8,=F'3'                MULTIPLY BY 3
        AR    R1,R9                   TO WORK
        SPACE
        LR    R9,R3                   TO WORK
        MR    R8,R3                   MULTIPLY TO **2
        MR    R8,R9                   MULTIPLY TO **4
        MR    R8,R3                   MULTIPLY TO **5
        M     R8,=F'71'               MULTIPLY BY  71
        AR    R1,R9                   TO WORK
        SPACE
        LR    R9,R4                   TO WORK
        MR    R8,R4                   MULTIPLY TO **2
        MR    R8,R4                   MULTIPLY TO **4
        MR    R8,R3                   MULTIPLY TO **3
        M     R8,=F'76'               MULTIPLY TO  76
        AR    R1,R9                   TO WORK
```

```
SPACE
LR     R9,R5                    TO WORK
MR     R8,R5                    MULTIPLY TO **2
M      R8,=F'2'                 MULTIPLY BY 2
AR     R1,R9                    TO WORK
SPACE
RETURN  (14,12),RC=0
SPACE
LTORG
SPACE
END
```

APPENDIX B: HPACK ASSEMBLE

```
TITLE 'PACK DATA WITH HUFFMAN CODES. 3RD DEC,1979. 14 MOMARAM 1400.'
*********************************************************************
*                                                                   *
*  THIS PROGRAM CONVERTS A FILE UNDER CMS TO HUFFMAN CODE.  THE INPUT*
*  FILE MUST BE PACKED BY THE VM PACK COMMAND.  (IT IS ASSUMED THAT  *
*  IT CONTAINS NO X'FF' CHARACTERS AS THIS OTHERWISE MARKS THE END   *
*  OF THE PACKED STRING).  THE BITS ARE ACCUMULATED AND ARE PLACED   *
*  IN THE OUTPUT BUFFER ONE BYTE AT A TIME AS THIS SIMPLIFIES THE    *
*  CODE AND GIVES GOOD RUN TIME.  THE HUFFMAN CODE TABLE MUST BE IN  *
*  COLLATING SEQUENCE.                                              *
*                                                                   *
*********************************************************************
         SPACE 3
         PRINT  OFF
         MACRO
&NAME    BEGIN &A
         USING    ,15          GET TEMP ADDRESSABILIITY
&NAME    SAVE  (14,12),,
         CNOP  0,4             ALIGN ON WORD BOUNDARY
         BAL   14, +76         BRANCH ROUND SAVE AREA
         USING  ,13            GET ADDRESSABLIIT
         DC    F'0'            SET HIGH ORDER WORD TO 0 FOR PL1
         DS    17F
         DROP  15
         ST    13,4(14)        SAVE REGS 13 AND
         ST    14,8(13)        BACK CHAIN)
         LR    13,14           GET SAVE AREA ADDRESS
         AIF   (T'&A NE '0').B
         MEXIT
.B       ANOP
&NAME    B     A23K&SYSNDX     BRANCH ROUND RETURN
         SPACE 1
&A       L     13,4(13)        GET OLD SAVE AREA
         L     14,12(13)       GET REG 14
         LM    0,12,20(13)     GET REST OF REGS
         BR    14
         SPACE 1
A23K&SYSNDX    DS    0H          START OF MAIN LINE CODE
         MEND
         PRINT  ON,NOGEN
         SPACE 3
         MACRO
&NAME    MAC   &CODE,&LEN,&MASK
&NAME    DC    AL1(&LEN),&MASK
         MEND
         SPACE
          REGEQU
          SPACE 2
```

APPENDIX B: HPACK ASSEMBLE

```
*         ENTRY POINT
PACK      CSECT
          BEGIN  RET              ENTRY POINT
          SPACE 2
          BAL    R10,PARMA        GO GET AORMS
          SPACE 1
          OC     SEED,SEED        ANY SEED
          BZ     FSOPEN           NO - GO STRAIGHT TO OPEN FILES
          SPACE
          L      R8,SEED          GET SEED FOR COMPARE
          LR     R6,R8            SAVE
          LA     R7,256           TRY 256 TIMES
          SPACE
CLOOP     BAL    R9,CYCLE        GENERATE RANDOM NUMBERS
          C      R8,SEED          HAVE WE CYCLED
          BE     FAILC
          BL     CLOOP1
          L      R8,SEED
          B      CLOOP1
          SPACE
FAILC     WRTERM 'SEED CYCLES - NOT USEABLE.'
          B      RET
CLOOP1    BCT    R7,CLOOP
          ST     R6,SEED             RESTORE ORIGINAL SEED.
          SPACE
FSOPEN    FSOPEN  FSCB=INDASD
          FSSTATE FSCB=OUTDASD
          LTR     R15,R15            IS IT THERE
          BNZ     OPENOUT
          SPACE   1
          FSERASE FSCB=OUTDASD
          SPACE   1
OPENOUT   FSOPEN  FSCB=OUTDASD
          SPACE 2
          BAL     R10,NEWBLOCK
          SPACE
PARLOOP   BAL     R10,FSREAD
          SR      R0,R0            CLEAR POINTERS
          SR      R1,R1
          SPACE
*         R12 = CURRENT BUFFER IN, R11 CURRENT BUFFER OUT.
PAR       AP      CHARS,=P'1'    ADD TO PACKED WORD COUNT
          SR      R2,R2          CLEAR
          IC      R2,0(R12)      GET CHAR
          C       R2,=F'255'     END
          BNE     PARF
          SPACE
```

APPENDIX B: HPACK ASSEMBLE

```
          LA      R9,1            SHOW LAST
          SPACE
PARF      LA      R12,1(R12)      GET NEXT
          SLL     R2,2            MULT BY TABLE LENGTH = 4
          LA      R2,TABLE(R2)    GET
          SPACE
PARC      L       R3,0(R2)        GET CHAR
          L       R4,=F'16777215'     'X'00FFFFFF'
* RO CURRENT BITS IN REG 1; R4 NEW CHAR, R3 NO BITS IN R3.
          NR      R4,R3           NOW CONTAINS BIT PATERN
          SRL     R3,24           NOW CONTAINS NUMBER OF BITS
          SPACE
          LA      R2,32           MAX
          AR      R0,R3           NEW NO OF CHARS
          SR      R2,R0           OFFSET
          STC     R2,SHIFT+3      SHIFT R4
SHIFT     SLL     R4,0
          AR      R1,R4           ADD THIS CHAR
          SPACE
PARCO     C       R0,=F'8'        SIZE
          BL      PARBCT
          SPACE
          STCM    R1,8,0(R11)
          LA      R11,1(R11)
          S       R0,=F'8'        CUT BY 8 BITS
          SLL     R1,8            SHIFT OUT THIS BYTE
          SPACE
          LA      R15,BLOCK1+800
          CR      R15,R11
          BH      PARCO
          SPACE
          STM     R0,R1,SAVE10    SAVE CURENT VALUE OF 1 AND 0
          BAL     R10,FSWRITE
          LM      R0,R1,SAVE10    GET  CURENT VALUE OF 1 AND 0
          B       PARBCT
          SPACE
PARBCT    BCT     R9,PAR          GET NEXT
          C       R0,=F'0'
          BE      PARLOOP
          STCM    R1,8,0(R11)
          LA      R11,1(R11)
          B       PARLOOP
          SPACE
FSWRITE   FSWRITE FSCB=OUTDASD,ERROR=RETURN
          SPACE
          BAL     R9,CYCLE
          SPACE
```

APPENDIX B: HPACK ASSEMBLE

```
NEWBLOCK  LA       R11,BLOCK1
          BR       R10
          SPACE
*
FSREAD    FSREAD   FSCB=INDASD,ERROR=READERR
          LA       R12,BLOCK      START OF DATA
          SPACE    1
          LR       R9,R0     GET LENGTH CODE
          SPACE
          BR       R10
          SPACE
READERR   C        R15,=F'12'     TEST
          BE       EOF
          SPACE 1
          B        RETURN
          SPACE 2
          SPACE    2
EOF       BAL      R10,FSWRITE         PURGE BUFFERR
          FSCLOSE  FSCB=INDASD
          FSCLOSE  FSCB=OUTDASD
          SPACE
RETURN    CVB      R15,CHARS
          SPACE
          B        RET              THIS IS THE END
          SPACE
          SPACE    2
PARMA     CLI      0(R1),X'FF'    NOTHING
          BE       ERROR10
          CLI      8(R1),X'FF'    NOPARM
          BE       ERROR10
          CLI      16(R1),X'FF'   NOTHING
          BE       ERROR10
          SPACE
          MVC      INDASD+8(16),8(R1)      MOVE FILE NAME
          MVC      OUTDASD+8(8),8(R1)     OUTNAME
          SPACE
          CLI      24(R1),X'FF'    NOTHING
          BER      R10             NO SEED
          SPACE
          OC       DUB,24(R1)      MAKE ALL NUMERIC
          PACK     DUB,DUB         PACK IT
          CVB      R1,DUB          BINARY
          ST       R1,SEED         STORE SEED
          SPACE
          BR       R10
          SPACE
```

APPENDIX B: HPACK ASSEMBLE

```
ERROR10   WRTERM 'PARM INFO MUST BE ''FILENAME FILETYPE (SEED)'''
          B      RET
          SPACE
CYCLE     L      R15,SEED
          M      R14,=F'12345677'      MULTIPLY SEED
          D      R14,=XL4'7FFFFFFF'    MOD 2**31 - 1.
          ST     R14,SEED
          BR     R9
          SPACE
CYCLE2    ICM    R15,15,SEED           GET SEED WITH TEST
          BZR    R9
          LA     R7,200                GET SIZE
          LA     R1,BLOCK
          SPACE
GGG       M      R14,=F'12345677'          MULTIPLY SEED
          D      R14,=XL4'7FFFFFFF'        MOD 2**31 - 1.
          ST     R14,SEED
          XC     0(4,R1),SEED
          LA     R1,4(R1)
          BCT    R7,GGG
          BR     R9
          TITLE   'LITERALS AND CONSTANTS.'
          LTORG
          SPACE  2
INDASD    FSCB   'A          A        A1     ',BUFFER=BLOCK,RECFM=F,  XXXX
               BSIZE=800
          SPACE
OUTDASD   FSCB   'A          PACKED   A1     ',RECFM=F,BUFFER=BLOCK1, XXXX
               BSIZE=800
          SPACE  2
CUB       DC     0D'0',CLB'00000000'
CHARS     DC     0D'0',PL8'0'
SAVE10    DC     D'0'          SAVE REG 0 AND 1
SEED      DC     F'0'          RANDOM NUMBER SEED
          SPACE
BLOCK     DS     CL800
BLANK     DC     CL256' '
BLOCK1    DC     800XL1'00'
          SPACE  3
*         NOW FOLLOWS THE HUFFMAN CODE.
TABLE     MAC    000,012,BL03'010101000101'                    19.
          MAC    001,007,BL03'0110101'                        687.
          MAC    002,008,BL03'10001110'                       381.
          MAC    003,008,BL03'11110110'                       498.
          MAC    004,008,BL03'11110000'                       488.
          MAC    005,008,BL03'11101011'                       471.
          MAC    006,008,BL03'01010111'                       322.
          MAC    007,010,BL03'1000111111'                      99.
```

APPENDIX B: HPACK ASSEMBLE

```
MAC     008,007,BL03'0011100'                524.
MAC     009,007,BL03'1111110'               1029.
MAC     010,007,BL03'0101100'                641.
MAC     011,010,BL03'1110101001'             117.
MAC     012,010,BL03'1011010011'             105.
MAC     013,009,BL03'011010010'              172.
MAC     014,008,BL03'01010010'               290.
MAC     015,010,BL03'0101010010'              76.
MAC     016,009,BL03'010100011'              145.
MAC     017,010,BL03'0110100010'              85.
MAC     018,011,BL03'10100011100'             50.
MAC     019,011,BL03'10000000111'             46.
MAC     020,011,BL03'10000011001'             47.
MAC     021,011,BL03'01010101011'             39.
MAC     022,011,BL03'10000010110'             46.
MAC     023,010,BL03'0101010000'              74.
MAC     024,010,BL03'0101000001'              72.
MAC     025,010,BL03'1011011010'             108.
MAC     026,011,BL03'10100011010'             49.
MAC     027,010,BL03'0101000100'              72.
MAC     028,009,BL03'111100111'              247.
MAC     029,011,BL03'11101010001'            158.
MAC     030,010,BL03'0011101000'              62.
MAC     031,010,BL03'0100001011'              71.
MAC     032,009,BL03'010000100'              140.
MAC     033,011,BL03'11101010100'             58.
MAC     034,011,BL03'10110110111'             56.
MAC     035,011,BL03'10110100100'             52.
MAC     036,010,BL03'1110100110'             114.
MAC     037,011,BL03'11101010000'             57.
MAC     038.010,BL03'1000000000'              87.
MAC     039,010,BL03'0011101010'              66.
MAC     040,011,BL03'11101000000'             56.
MAC     041,011,BL03'01101000110'             43.
MAC     042,011,BL03'11101010110'             60.
MAC     043,010,BL03'0101010011'              76.
MAC     044,010,BL03'1010001111'             104.
MAC     045,012,BL03'100000110000'            23.
MAC     046,011,BL03'00111011110'             34.
MAC     047,011,BL03'10100011101'             51.
MAC     048,011,BL03'10000000010'             44.
MAC     049,011,BL03'01101000111'             43.
MAC     050,011,BL03'10000000110'             45.
MAC     051,011,BL03'00111011100'             33.
MAC     052,012,BL03'111010000010'            28.
MAC     053,010,BL03'0100001110'              71.
MAC     054,012,BL03'100000110101'            24.
```

APPENDIX B: HPACK ASSEMBLE

```
MAC     055,011,BL03'01010100011'                          38.
MAC     056,011,BL03'11101010101'                          59.
MAC     057,012,BL03'101101001010'                         26.
MAC     058,012,BL03'010101101101'                         20.
MAC     059,012,BL03'101101101101'                         27.
MAC     060,010,BL03'0011101011'                           66.
MAC     061,011,BL03'00111011111'                          34.
MAC     062,012,BL03'111010101110'                         29.
MAC     063,012,BL03'101101001011'                         26.
MAC     064,003,BL03'000'                                 8147.
MAC     065,010,BL03'1000001010'                           92.
MAC     066,013,BL03'0101011011100'                        10.
MAC     067,016,BL03'0101010001001000'                      1.
MAC     068,012,BL03'100000110001'                         23.
MAC     069,016,BL03'0101010001001001'                      1.
MAC     070,015,BL03'001110111011000'                       2.
MAC     071,020,BL03'01010110111111100100'                  0.
MAC     072,016,BL03'0101010001001010'                      1.
MAC     073,020,BL03'01010110111111100101'                  0.
MAC     074,020,BL03'01010110111111100110'                  0.
MAC     075,006,BL03'110010'                              1730.
MAC     076,013,BL03'1000000001110'                        11.
MAC     077,006,BL03'001000'                              1033.
MAC     078,008,BL03'10000001'                             360.
MAC     079,009,BL03'001110110'                            134.
MAC     080,010,BL03'0100001111'                            71.
MAC     081,020,BL03'01010110111111100111'                  0.
MAC     082,020,BL03'01010110111111101000'                  0.
MAC     083,020,BL03'01010110111111101001'                  0.
MAC     084,020,BL03'01010110111111101010'                  0.
MAC     085,020,BL03'01010110111111101011'                  0.
MAC     086,020,BL03'01010110111111101100'                  0.
MAC     087,020,BL03'01010110111111101101'                  0.
MAC     088,020,BL03'01010110111111101110'                  0.
MAC     089,020,BL03'01010110111111101111'                  0.
MAC     090,020,BL03'01010110111111110000'                  0.
MAC     091,014,BL03'00111011101010'                        4.
MAC     092,008,BL03'11110010'                             489.
MAC     093,006,BL03'001001'                              1036.
MAC     094,010,BL03'1000000010'                           91.
MAC     095,020,BL03'01010110111111110001'                  0.
MAC     096,006,BL03'100001'                              1471.
MAC     097,009,BL03'010110101'                            166.
MAC     098,020,BL03'01010110111111110010'                  0.
MAC     099,020,BL03'01010110111111110011'                  0.
MAC     100,020,BL03'01010110111111110100'                  0.
MAC     101,020,BL03'01010110111111110101'                  0.
```

APPENDIX B: HPACK ASSEMBLE

```
MAC     102,020,BL03'01010110111111110110'          0.
MAC     103,020,BL03'01010110111111110111'          0.
MAC     104,020,BL03'01010110111111111000'          0.
MAC     105,020,BL03'01010110111111111001'          0.
MAC     106,020,BL03'01010110111111111010'          0.
MAC     107,007,BL03'1111010'                     982.
MAC     108,020,BL03'01010110111111111011'          0.
MAC     109,016,BL03'0101010001001011'              1.
MAC     110,012,BL03'011010000110'                 21.
MAC     111,012,BL03'100000110100'                 23.
MAC     112,020,BL03'01010110111111111100'          0.
MAC     113,020,BL03'01010110111111111101'          0.
MAC     114,020,BL03'01010110111111111110'          0.
MAC     115,020,BL03'01010110111111111111'          0.
MAC     116,020,BL03'10100011011101100000'          0.
MAC     117,020,BL03'10100011011101100001'          0.
MAC     118,020,BL03'10100011011101100010'          0.
MAC     119,020,BL03'10100011011101100011'          0.
MAC     120,020,BL03'10100011011101100100'          0.
MAC     121,020,BL03'10100011011101100101'          0.
MAC     122,009,BL03'010110100'                   158.
MAC     123,016,BL03'0101011011111100'              1.
MAC     124,020,BL03'10100011011101100110'          0.
MAC     125,006,BL03'010111'                     1308.
MAC     126,008,BL03'01011011'                    327.
MAC     127,012,BL03'010101010100'                 19.
MAC     128,007,BL03'1000110'                     769.
MAC     129,007,BL03'1011000'                     833.
MAC     130,007,BL03'0100000'                     551.
MAC     131,009,BL03'101101110'                   224.
MAC     132,008,BL03'11110001'                    488.
MAC     133,009,BL03'010100001'                   144.
MAC     134,009,BL03'100000111'                   189.
MAC     135,008,BL03'10110101'                    426.
MAC     136,009,BL03'100011110'                   194.
MAC     137,009,BL03'111010001'                   226.
MAC     138,009,BL03'101101100'                   214.
MAC     139,009,BL03'100000100'                   183.
MAC     140,009,BL03'010101100'                   157.
MAC     141,009,BL03'010101011'                   155.
MAC     142,009,BL03'011010011'                   174.
MAC     143,009,BL03'010000110'                   141.
MAC     144,010,BL03'1110100111'                  115.
MAC     145,010,BL03'1011010000'                  104.
MAC     146,010,BL03'1011010001'                  104.
MAC     147,010,BL03'1010001100'                   99.
MAC     148,010,BL03'0101000000'                   71.
```

APPENDIX B: HPACK ASSEMBLE

```
MAC     149,010,BL03'0011101001'                    65.
MAC     150,011,BL03'01010110100'                   39.
MAC     151,010,BL03'1000111110'                    96.
MAC     152,011,BL03'10000011011'                   48.
MAC     153,010,BL03'0101000101'                    72.
MAC     154,011,BL03'01010101000'                   38.
MAC     155,011,BL03'01010101001'                   38.
MAC     156,011,BL03'10000010111'                   46.
MAC     157,011,BL03'01101000010'                   41.
MAC     158,011,BL03'01000010100'                   34.
MAC     159,012,BL03'010101010101'                  19.
MAC     160,012,BL03'101101101100'                  26.
MAC     161,013,BL03'1110101011111'                 16.
MAC     162,011,BL03'01010110101'                   39.
MAC     163,009,BL03'111010010'                    228.
MAC     164,012,BL03'100000000110'                  22.
MAC     165,012,BL03'010101101100'                  19.
MAC     166,012,BL03'011010000111'                  21.
MAC     167,013,BL03'0101011011101'                 10.
MAC     168,013,BL03'0011101110111'                  9.
MAC     169,013,BL03'1010001101111'                 13.
MAC     170,013,BL03'1000000001111'                 11.
MAC     171,013,BL03'1010001101100'                 12.
MAC     172,013,BL03'0101010001000'                  9.
MAC     173,015,BL03'101000110110110'                3.
MAC     174,014,BL03'00111011101011'                 4.
MAC     175,014,BL03'11101000001110'                 7.
MAC     176,014,BL03'10100011011010'                 6.
MAC     177,015,BL03'001110111011001'                2.
MAC     178,014,BL03'01010100010011'                 5.
MAC     179,015,BL03'101000110110111'                3.
MAC     180,015,BL03'101000110111000'                3.
MAC     181,014,BL03'01010110111100'                 5.
MAC     182,014,BL03'01010110111101'                 5.
MAC     183,015,BL03'101000110111001'                3.
MAC     184,014,BL03'11101010111101'                 8.
MAC     185,014,BL03'11101000001111'                 7.
MAC     186,015,BL03'101000110111010'                3.
MAC     187,013,BL03'1110100000110'                 14.
MAC     188,014,BL03'01010110111110'                 5.
MAC     189,013,BL03'0011101110100'                  8.
MAC     190,014,BL03'11101010111100'                 7.
MAC     191,015,BL03'001110111011010'                2.
MAC     192,015,BL03'001110111011011'                2.
MAC     193,005,BL03'10111'                       3400.
MAC     194,007,BL03'1110111'                      954.
MAC     195,005,BL03'11011'                       3652.
```

APPENDIX B: HPACK ASSEMBLE

```
MAC     196,005,BL03'00101'                                  2104.
MAC     197,004,BL03'0111'                                   5503.
MAC     198,006,BL03'110011'                                 1732.
MAC     199,007,BL03'0110111'                                 714.
MAC     200,007,BL03'1010000'                                 786.
MAC     201,005,BL03'10011'                                  3169.
MAC     202,020,BL03'10100011011101100111'                      0.
MAC     203,020,BL03'10100011011101101000'                      0.
MAC     204,020,BL03'10100011011101101001'                      0.
MAC     205,020,BL03'10100011011101101010'                      0.
MAC     206,020,BL03'10100011011101101011'                      0.
MAC     207,020,BL03'10100011011101101100'                      0.
MAC     208,020,BL03'10100011011101101101'                      0.
MAC     209,011,BL03'01000010101'                              35.
MAC     210,008,BL03'10100010'                                400.
MAC     211,005,BL03'10101'                                  3370.
MAC     212,006,BL03'111110'                                 1983.
MAC     213,005,BL03'01001'                                  2321.
MAC     214,005,BL03'11100'                                  3728.
MAC     215,005,BL03'00110'                                  2158.
MAC     216,010,BL03'1110100001'                              113.
MAC     217,005,BL03'11010'                                  3579.
MAC     218,020,BL03'10100011011101101110'                      0.
MAC     219,020,BL03'10100011011101101111'                      0.
MAC     220,020,BL03'10100011011101110000'                      0.
MAC     221,020,BL03'10100011011101110001'                      0.
MAC     222,020,BL03'10100011011101110010'                      0.
MAC     223,020,BL03'10100011011101110011'                      0.
MAC     224,020,BL03'10100011011101110100'                      0.
MAC     225,020,BL03'10100011011101110101'                      0.
MAC     226,005,BL03'10010'                                  3135.
MAC     227,005,BL03'11000'                                  3448.
MAC     228,006,BL03'100010'                                 1475.
MAC     229,006,BL03'001111'                                 1110.
MAC     230,008,BL03'11110111'                                498.
MAC     231,007,BL03'0110110'                                 687.
MAC     232,008,BL03'11101101'                                474.
MAC     233,010,BL03'0110100000'                               83.
MAC     234,020,BL03'10100011011101110110'                      0.
MAC     235,020,BL03'10100011011101110111'                      0.
MAC     236,020,BL03'10100011011101111000'                      0.
MAC     237,020,BL03'10100011011101111001'                      0.
MAC     238,020,BL03'10100011011101111010'                      0.
MAC     239,020,BL03'10100011011101111011'                      0.
MAC     240,005,BL03'01100'                                  2637.
MAC     241,006,BL03'101001'                                 1617.
MAC     242,007,BL03'1011001'                                 835.
```

APPENDIX B: HPACK ASSEMBLE

```
MAC     243,007,BL03'1111111'                          1032.
MAC     244,007,BL03'0100011'                           575.
MAC     245,008,BL03'11101100'                          471.
MAC     246,008,BL03'01010011'                          294.
MAC     247,009,BL03'101101111'                         224.
MAC     248,009,BL03'111100110'                         243.
MAC     249,007,BL03'0100010'                           569.
MAC     250,020,BL03'10100011011101111100'                0.
MAC     251,020,BL03'10100011011101111101'                0.
MAC     252,020,BL03'10100011011101111110'                0.
MAC     253,020,BL03'10100011011101111111'                0.
MAC     254,019,BL03'0101011011111110000'                 0.
MAC     255,016,BL03'0101011011111101'                    1.
 SPACE
 END
```

APPENDIX B: HUFF ASSEMBLE

```
         TITLE 'QSAM ANAL. T F MURPHY'
         MACRO
&NAME    BEGIN &A
         USING    ,15          GET TEMP ADDRESSABLIITY
&NAME    SAVE  (14,12),,*
         CNOP  0,4             ALIGN ON WORD BOUNDARY
         BAL   14,*+76         BRANCH ROUND SAVE AREA
         USING *,13            GET ADDRESSABLIIT
         DC    F'0'            SET HIGH ORDER WORD TO 0 FOR PLI
         DS    17F
         DROP  15
         ST    13,4(14)        SAVE REGS 13 AND
         ST    14,8(13)        BACK CHAIN)
         LR    13,14           GET SAVE AREA ADDRESS
         AIF   (T'&A NE '0').B
         MEXIT
.B       ANOP
&NAME    B     A23K&SYSNDX     BRANCH ROUND RETURN
         SPACE 1
&A       L     13,4(13)        GET OLD SAVE AREA
         L     14,12(13)       GET REG 14
         LM    0,12,20(13)     GET REST OF REGS
         BR    14
         SPACE 1
A23K&SYSNDX    DS    0H          START OF MAIN LINE CODE
         MEND
         SPACE 3
         REGEQU
         SPACE
LL       EQU   36
CHR      EQU   4
HOT      EQU   5
CST      EQU   28
CNT      EQU   32
         SPACE
         GBLA  &SPN,&SPR     NUMBER OF SPACES TO LOOK FOR
&SPN     SETA  6             SET TO 6 - CALCUALTED AS BEST.
&SPR     SETA  &SPN-1        SET TO 1 LESS FOR MULTIPLY
         SPACE
C        CSECT
         BEGIN RET
     OPEN (DCB,(INPUT))
         SPACE 1
LP4      GET   DOB
         LR    R2,R1          GET RECORD ADDRESS
         AP    TOT1,=P'1'     ADD TO TOTAL
         SPACE 1
         CP    TOT1,=P'10000' TEST NUMBER
```

APPENDIX B: HUFF ASSEMBLE

```
         BE    ENDR1
         SPACE 1
         LH    R3,DCB+82          GET LRECL
         SR    R4,R4              CLEARRECL
         SPACE
         TM    DCB+38,X'40'       IS IT VARIABLE LENGTH
         BO    LOOP
         SPACE
         LA    R2,4(R2)           SCIP OVER LENGTH CODE
         S     R3,=F'4'
         BNP   LP4                SAFETY
         SPACE
LOOP     C     R3,=A(&SPN)        TEST IF ENOUGH ROOM LEFT
         BL    LOOP1
         CLC   0(&SPN,R2),=CL&SPN'     '
         BNE   LOOP1
         SPACE
         L     R7,=A(TABLAST)          GET MULTIPLE BLANK TABLE
         AP    CNT(4,R7),=P'10'         ADD 10
         AP    TOT2,=P'10'     ADD TO CHARACTER COUNT
         A     R2,=A(&SPN)
         S     R3,=A(&SPN)
         BZ    LP4
         B     LOOP
         SPACE
LOOP1    IC    R4,0(R2)     GET CHAR
         LR    R5,R4
         MH    R5,=A12(LL)       MULT BY TABLE LENGTH
         LA    R7,TABPOINT(R5)   GET ADDRESS
         AP    CNT(4,R7),=P'10' ADD TO COUNT  GO BY 10
         SPACE
         LA    R2,1(R2)        TO GIVE EXTRA WEIGHT
         AP    TOT2,=P'10'     ADD TO CHARACTER COUNT
         BCT   R3,LOOP
         SPACE
         B     LP4
         SPACE 1
ENDR1    CLOSE DCB
         EJECT
* THIS CODE CALCULATES HUFFMAN CODES.
*        FIRST PLACE IN THE CHAR CODE.
         LA    R4,257
         L     R5,=A(TABPOINT)
         SR    R7,R7
         SPACE
TABSORT  STCM R7,15,1(R5)          PLACE CHAR AND CLEAR ADDRESS
         LA    R7,1(R7)
         SPACE
```

APPENDIX B: HUFF ASSEMBLE

```
         ZAP   CST(4,R5),CNT(4,R5)       GET ARTICIAL COUNT
         SRP   CST(4,R5),63,0            COUNT=(COUNT-1)/10
         SPACE
         LA    R5,LL(R5)                 AND GET NEXT TABLE ENTRY
         BCT   R4,TABSORT
         SPACE
         L      R7,=A(TABLAST)     FIRST WORD PF THIS WILL BE 1
         XC    0(4,R7),0(R7)      CLEAR THIS 1
         SPACE
*        NOW BUILD THE HUFFMAN CODE
TABHUFF  BAL   R10,FINDLOW]         REG 2
         BAL   R10,FINDLOW2         REG 3
         SPACE
         SR    R6,R6                WORK REGS
         LR    R9,R2                 "   "
         SPACE
RF1      IC    R6,HCT(R9)          OFFSETT
         MVC   WORK(20),6(R9)       SAVE BEFOR SHIFT
         EX    R6,MOVEO             SHIFT OVER
         MVI   6(R9),C'0'           FIRST
         LA    R6,1(R6)             INCREMENT
         STC   R6,HCT(R9)           PUT BACK
         SPACE
         ICM   R9,15,0(R9)          IS IT THE END
         BP    TF1
         SPACE
         LR    R9,R3
         SPACE
TF3      IC    R6,HCT(R9)          OFFSETT
         MVC   WORK(20),6(R9)       SAVE BEFOR SHIFT
         EX    R6,MOVEO             SHIFT OVER
         MVI   6(R9),C'1'           SECOND
         LA    R6,1(R6)             INCREMENT
         STC   R6,HCT(R9)          PUT BACK
         SPACE
         LR    R1,R9                SAVE
         ICM   R9,15,0(R9)          IS IT THE END
         BP    TF3
         SPACE
         ST    R2,0(R1)             LST ZERO POINTER
         SPACE
         AP    CNT(4,R3),CNT(4,R2)     ADD THEM TOGETHER
         ZAP   CNT(4,R2),=P'9999998' FLAG HIGH
         SPACE
         CP    CNT(4,R3),TOT2        HAVE WE ADDED THEM ALL
         BL    TABHUFF
*        CODES ARE NOW IN BUILT
         SPACE
```

APPENDIX B: HUFF ASSEMBLE

```
* THIS CODE PRINTS THE HUFFMAN CODES
        OPEN    (OUT,(OUTPUT))          PRINT HEADER
          PUT   OUT,BLANK
          SPACE 1
          PUT   OUT,HEAD
          PUT   OUT,BLANK
          SPACE
          L     R5,=A(TABPOINT)         GET START OF TABLE
          LA     R4,257
          SPACE 1
          SR    R9,R9                   FOR TOTAL NO OF BITS OUT
          SR    R6,R6                       WORK (IC USE)
          SPACE
LOOPX     MVC   WORK,BLANK                  CLEAR PRINT AREA
          MVC   WORK+9(3),=CL3'MAC'         MACRO
          SPACE
          IC    R6,CHR(R5)                  GET DECIMAL VERSION OF
          CVD   R6,DUB                          EBDIDIC
          OI    DUB+7,X'OF'                           CHARACTER
          UNPK  WORK+18(3),DUB+6(2)                 IN PRINT AREA
          SPACE
          IC    R6,HCT(R5)              GET NUMBER OF BITS
          CVD   R6,DUB                  PRINT NUMBER
          OI    DUB+7,X'OF'                       OF BITS
          UNPK  WORK+22(3),DUB+6(2)                       IN
          MVI   WORK+21,C','                                CODE
          SPACE
          MVC   WORK+25(6),=CL6',BL03'''    LITERAL
          LA    R8,WORK+31(R6)               WILL BE END
          MVI   0(R8),C''''                 END
          SPACE
          ZAP   DUB,CST(4,R5)           GET CHAR COUNT
          CVB   R15,DUB                 MULT BY NO OF BITS
          MR    R14,R6
          AR    R9,R15                  ADD TO TOTAL
          SPACE
          BCTR  R6,R0
          EX    R6,MOVE                 MOVE OVER BIT PATTERN
          SPACE
          MVC    WORK+57(9),=XL9'40202020202021204B'
          ED     WORK+57(8),CST(R5)
          SPACE
          PUT   OUT,WORK                PUNCH
          LA    R5,LL(R5)
          BCT   R4,LOOPX                GET NEXT
          SPACE  2
```

APPENDIX B: HUFF ASSEMBLE

```
*  NOW PRINT ENTROPY AND TOTAL CHARACTERS.
         L     R15,=F'1000'          NOW MULT BY 1000
         MR    R14,R9
         SP    TOT2,=P'257'          CUT BE PRIME FACTOR.
         SRP   TOT2,63,0             DIVIDE BY 10 FOR TRUE NO
         SPACE
         L     R7,=A(TABLAST)             ADD NUMBER OF GROUPED BLANKS
         ZAP   DUB,CST(4,R7)         MOVE TO WORK AREA
         MP    DUB,=P'&SPR'               WE HAVE ALREADY ADDED THIS ONCE
         AP    TOT2,DUB
         SPACE
         CVB   R9,TOT2                    TOTAL NO OF CHARS
         DR    R14,R9                     TOTAL NO OF BITS * 1000
         CVD   R15,DUB                    DEC ANSWER
         SPACE
         MVC   WORK,BLANK
         SPACE
         MVC   WORK+4(10),=CL10'ENTROPY:'          EDIT
         MVC   WORK+20(7),=XL7'4021204B202020'        IT
         ED    WORK+20(7),DUB+5                         TO
         MVC   WORK+44(12),=CL12'CHARACTERS:'             PUNCH
         MVC   WORK+57(9),=XL9'40202020202021204B'              LINE
         ED    WORK+57(8),TOT2+4
         SPACE
         PUT   OUT,WORK              PUNCH
         SPACE
         PUT   OUT,BLANK
         SPACE 2
         CLOSE OUT
         B     RET
         SPACE  2
FINDLOW1 L     R5,=A(TABPOINT)
         LA    R5,0(R5)
         LA    R2,TABMAX
         LA    R4,257
         SPACE
FINDLOWA CP    CNT(4,R5),CNT(4,R2)
         BNL   FINDLOWB
         SPACE
         LR    R2,R5
         SPACE
FINDLOWB LA    R5,LL(R5)
         BCT   R4,FINDLOWA
         BR    R10
         SPACE
FINDLOW2 L     R5,=A(TABPOINT)
         LA    R5,0(R5)
         LA    R3,TABMAX
```

APPENDIX B: HUFF ASSEMBLE

```
         LA    R4,257
         SPACE
FINDLOWC CR    R2,R5
         BE    FINDLOWD
         CP    CNT(4,R5),CNT(4,R3)
         BNL   FINDLOWD
         SPACE
         LR    R3,R5
         SPACE
FINDLOWD LA    R5,LL(R5)
         BCT   R4,FINDLOWC
         BR    R10
         SPACE
MOVE     MVC   WORK+31(0),6(R5)      MOVE BIT MASK TO OUT AREA
MOVEO    MVC   7(0,R9),WORK          MOVE OVER ALL BITS
         EJECT
         LTORG
OUT      DCB   DSORG=PS,MACRF=(PM),LRECL=80,BLKSIZE=80,        XXXXXXXXX
               DDNAME=SYSPRINT,RECFM=F
BLANK    DC    CL080' '
WORK     DC    CL080' '
HEAD     DC    CL080'1     HUFFMAN CODE TABLE.'
DUB      DC    D'0'
TOT1     DC    PL8'0'   END OF RUN.'
TOT2     DC    PL8'257' NUMBER OF CHARACTERS  + 257
         SPACE 1
DCB      DCB   DDNAME=MASTER,MACRF=(GL),EODAD=ENDR1,DSORG=PS
         SPACE 1
TABMAX   DC    CL32'PAD',PL4'9999999'   MAX
* MUST BE LAST
* FORMAT IS A(POINT-TO-NEXT),X'CHAR',ALI(HUFF-LENGTH),CL22'HUFF-CODE'
*        COUNT OF CHARS.
         CNCP  0,8
TABPOINT DC    2304A(28)          WILL BE INITIALISED TO X'1C'
TABLAST  DC    9A(28)      (FOR MULTIPLE SPACES)
         SPACE 1
         END
```

APPENDIX B: UNPCK ASSEMBLE

```
  TITLE 'UNPACK DATA FROM HUFFMAN CODE. 3 DEC,1979. 14 MOMARAM 1400.'
*********************************************************************
*                                                                   *
*     THIS PROGRAM CONVERTS A FILE UNDER CMS FROM HUFFMAN CODE. THE *
*     INPUT UST HAVE BEEN CONVERTED FROM A PACKED FILE INTO HUFFMAN *
*     CODE USING THE SAME TRANSLATE TABLE. THE TABLE MUST BE SORTED *
*     INTO ASCENDING CODE LENGTH ORDER AS THE LOGIC IS THE POSSIBLE *
*     HUFFMAN CODES IN THE INPUT STREAM ARE LOADED INTO A REGISTER  *
*     WITH BITS STRINGS OF 3 THEN 4 THEN 5 ... BEING TESTED IN ORDER*
*     THIS GIVES A PROBALISTIC CHANCE OF THE CORRECT STING BEING FOUND*
*     IN A SHORT TIME.  THE CODE HERE MAKES USE OF THE FACT THAT THE*
*     ONLY X'FF' MAKES THE END OF PACKED DATA (THE GARBAGE AT THE   *
*     END IS NOT REPRODUCED); AND THE FACT THAT THE FIRST STRING HAS*
*     THREE BITS. THIS MUST BE MODIFIED IF THE ASSUMPTIONS ARE FALSE.*
*                                                                   *
*********************************************************************
          SPACE 3
          PRINT  OFF
          MACRO
&NAME     BEGIN &A
          USING   *,15          GET TEMP ADDRESSABLIITY
&NAME     SAVE  (14,12),,*
          CNOP  0,4             ALIGN ON WORD BOUNDARY
          BAL   14,*+76         BRANCH ROUND SAVE AREA
          USING *,13            GET ADDRESSABLIIT
          DC    F'0'            SET HIGH ORDER WORD TO 0 FOR PL1
          DS    17F
          DROP  15
          ST    13,4(14)        SAVE REGS 13 AND
          ST    14,8(13)        BACK CHAIN)
          LR    13,14           GET SAVE AREA ADDRESS
          AIF   (T'&A NE '0').B
          MEXIT
.B        ANOP
&NAME     B     A23k&SYSNDX     BRANCH ROUND RETURN
          SPACE 1
&A        L     13,4(13)        GET OLD SAVE AREA
          L     14,12(13)       GET REG 14
          LM    0,12,20(13)     GET REST OF REGS
          BR    14
          SPACE 1
A23K&SYSNDX     DS    0H          START OF MAIN LINE CODE
          MEND
          PRINT ON,NOGEN
          SPACE 3
          MACRO
&NAME     MAC   &CODE,&LEN,&MASK
&NAME     DC    AL1(&CODE,&LEN),&MASK
```

APPENDIX B: UNPCK ASSEMBLE

```
          MEND
          SPACE
           REGEQU
          SPACE 2
*         ENTRY  POINT
PACK      CSECT
          BEGIN  RET            ENTRY POINT
          SPACE 2
          BAL    R10,PARMA      GO GET PARMS
          SPACE 1
          OC     SEED,SEED      ANY SEED
          BZ     FSOPEN         NO - GO STRAIGHT TO OPEN FILES
          SPACE
*  IF NOT USING THE SECURITY HASHING REMOVE ALL CODE FOR EFFICIENCY
          L      R8,SEED        GET SEED FOR COMPARE
          LR     R6,R8          SAVE
          LA     R7,256         TRY 256 TIMES
          SPACE
CLOOP     BAL    R9,CYCLE      GENERATE RANDOM NUMBERS
          C      R8,SEED        HAVE WE CYCLED
          BE     FAILC
          BL     CLOOP1
          L      R8,SEED
          B      CLOOP1
          SPACE
FAILC     WRTERM 'SEED CYCLES - NOT USEABLE.'
          B      RET
CLOOP1    BCT    R7,CLOOP
          ST     R6,SEED           RESTORE ORIGINAL SEED.
          SPACE
FSOPEN    FSOPEN  FSCB=INDASD
          FSSTATE FSCB=OUTDASD
          LTR     R15,R15           IS IT THERE
          BNZ     OPENOUT
          SPACE   1
          FSERASE FSCB=OUTDASD
          SPACE   1
OPENOUT   FSOPEN  FSCB=OUTDASD
          SPACE 2
          LA      R11,BLOCK1
          SPACE
          LA      R12,1          FOR VALUE 1 FOR 80NS ARITH
          SR      R2,R2        NUMBER IN REG 4
          SR      R4,R4        CLEAR THIS WORD
          SPACE
          BAL     R10,FSREAD
          SPACE
```

APPENDIX B: UNPCK ASSEMBLE

```
*   REG 9 IS OUTBLOCK LEFT, R2 = NUMBER IN REG 4, R8 NUMBER IN REG 5
*         R7 = CURRENT BUFFER IN, R11 CURRENT BUFFER OUT.
*   REG 12 IS 1.
SERCH     LA      R3,TABLE       TABLE
          SPACE
          C       R8,=F'4'             DO WE HAVE ENOUGH CHARS
          BL      PAR                  WILL RUN OUT SOON
          SLDL    R4,3                 START RIGHT
          LA      R2,3
          SR      R8,R2                SET COUNTS RIGHT
          B       PAR2                 SAVES SOME TIME
          SPACE
PAR       CLM     R2,1,1(R3)     DO WE HAVE ENOUGH CHARACTER2
          BE      PAR2           YES WE CAN TEST CHARACTERS
          SPACE
          SLDL    R4,1           SHIFT BY 1
          AR      R2,R12         ADD 1
          SR      R8,R12         SUBTRACT 1
          BP      PAR            TRY AGAIN
          SPACE
          BAL     R10,GET4       GET NEXT CHARACTER
          B       PAR
          SPACE
PAR2      CLM     R4,7,2(R3)     TEST BIT PATERN
          BE      PARFOUND
          SPACE
          LA      R3,5(R3)      GET NEXT TABLE ELEMENT
          B       PAR
          SPACE
PARFOUND  MVC     0(1,R11),0(R3)    MOVE CHARACTER
          AP      CHARS,=P'1'       BYTE COUNT
          SPACE
* THIS ASSUMES THAT THE UNPACKED DATA DOES NOT CONTAIN X'FF'
          CLI     0(R3),X'FF'       ASSUMED TO MARK EOF ONLY
          BE      EOF
          SPACE
          AR      R11,R12             NEW NO OF CHARS
          SR      R2,R2               CLEART
          SR      R4,R4
          LA      R10,BLOCK1+800    ARE WE AT END
          CR      R11,R10
          BNE     SERCH               NEW CHAR
          SPACE
          LA      R10,SERCH
FSWRITE   FSWRITE FSCB=OUTDASD,ERROR=RETURN
          SPACE
          LA      R11,BLOCK1
```

APPENDIX B: UNPCK ASSEMBLE

```
          BR       R10
          SPACE
*
FSREAD    FSREAD   FSCB=INDASD,ERROR=READERR
          SPACE    1
          BAL      R9,CYCLE2
          SPACE
          LR       R9,R0     GET LENGTH CODE
          SPACE
          LA       R7,BLOCK   START OF IN BUFFER
          SPACE
GET4      S        R9,=F'4'        CUT BY 4
          BNP      FSREAD
          SPACE
          L        R5,0(R7)     GET WORD
          SPACE
          LA       R7,4(R7)
          LA       R8,32        AND IN REG 5
          BR       R10          END
          SPACE
          SPACE
READERR   DS       0H
          C        R15,=F'12'   TEST
          BE       EOF
          SPACE 1
          B        RETURN
          SPACE 2
          SPACE  2
EOF       BAL     R10,FSWRITE        PURGE BUFFER
          FSCLOSE FSCB=INDASD
          FSCLOSE FSCB=OUTDASD
          SPACE
RETURN    CVB      R15,CHARS
          SPACE
          B      RET                THIS IS THE END
          SPACE 2
PARMA     CLI    0(R1),X'FF'      NOTHING
          BE     ERROR10
          CLI    8(R1),X'FF'      NOPARM
          BE     ERROR10
          CLI    16(R1),X'FF'     NOTHING
          BE     ERROR10
          SPACE
          MVC    INDASD+8(16),8(R1)      MOVE FILE NAME
          MVC    OUTDASD+8(8),8(R1)     OUTNAME
          SPACE
          CLI    24(R1),X'FF'    NOTHING
          BER    R10             NO SEED
          SPACE
          OC     DUB,24(R1)      MAKE ALL NUMERIC
```

APPENDIX B: UNPCK ASSEMBLE

```
          PACK  DUB,DUB          PACK IT
          CVB   R1,DUB           BINARY
          ST    R1,SEED          STORE SEED
          SPACE
          BR    R10
          SPACE
ERROR10   WRTERM 'PARM INFO MUST BE ''FILENAME FILETYPE (SEED)'''
          B     RET
          SPACE
          SPACE
CYCLE     M     R14,=F'12345677'         MULTIPLY SEED
          D     R14,=XL4'7FFFFFFF'       MOD 2**31 - 1.
          ST    R14,SEED
          BR    R9
CYCLE2    ICM   R15,15,SEED       GET SEED WITH TEST
          BZR   R9
          LA    R7,200            GET SIZE
          LA    R1,BLOCK
          SPACE
GGG       M     R14,=F'12345677'         MULTIPLY SEED
          D     R14,=XL4'7FFFFFFF'       MOD 2**31 - 1.
          ST    R14,SEED
          XC    0(4,R1),SEED
          LA    R1,4(R1)
          BCT   R7,GGG
          BR    R9
          TITLE  'LITERALS AND CONSTANTS.'
          LTORG
          SPACE  2
INDASD    FSCB   'A        A        A1     ',BUFFER=BLOCK,RECFM=F,  XXXX
               BSIZE=800
          SPACE
OUTDASD   FSCB   'A        P2       A1     ',RECFM=F,BUFFER=BLOCK1, XXXX
               BSIZE=800
          SPACE  2
DUB       DC     0D'0',CL8'00000000'
CHARS     DC    0D'0',PL8'0'
SEED      DC    F'0'          RANDOM NUMBER SEED
          SPACE
BLOCK     DS    CL800
BLOCK1    DC    800XL1'00'
          DC    F'0'   PAD
          SPACE 3
*         NOW FOLLOWS THE HUFFMAN CODE.
*         FOR UNPACK THIS TABLE MUST BE IN ASCENDING ORDER OF CODE LEN
TABLE    MAC      064,003,BL03'000'                              8147.
         MAC      197,004,BL03'0111'                             5503.
         MAC      193,005,BL03'10111'                            3400.
```

APPENDIX B: UNPCK ASSEMBLE

```
MAC     195,005,BL03'11011'                3652.
MAC     196,005,BL03'00101'                2104.
MAC     201,005,BL03'10011'                3169.
MAC     211,005,BL03'10101'                3370.
MAC     213,005,BL03'01001'                2321.
MAC     214,005,BL03'11100'                3728.
MAC     215,005,BL03'00110'                2158.
MAC     217,005,BL03'11010'                3579.
MAC     226,005,BL03'10010'                3135.
MAC     227,005,BL03'11000'                3448.
MAC     240,005,BL03'01100'                2637.
MAC     075,006,BL03'110010'               1730.
MAC     077,006,BL03'001000'               1033.
MAC     093,006,BL03'001001'               1036.
MAC     096,006,BL03'100001'               1471.
MAC     125,006,BL03'010111'               1308.
MAC     198,006,BL03'110011'               1732.
MAC     212,006,BL03'111110'               1983.
MAC     228,006,BL03'100010'               1475.
MAC     229,006,BL03'001111'               1110.
MAC     241,006,BL03'101001'               1617.
MAC     001,007,BL03'0110101'               687.
MAC     008,007,BL03'0011100'               524.
MAC     009,007,BL03'1111110'              1029.
MAC     010,007,BL03'0101100'               641.
MAC     107,007,BL03'1111010'               982.
MAC     128,007,BL03'1000110'               769.
MAC     129,007,BL03'1011000'               833.
MAC     130,007,BL03'0100000'               551.
MAC     194,007,BL03'1110111'               954.
MAC     199,007,BL03'0110111'               714.
MAC     200,007,BL03'1010000'               786.
MAC     231,007,BL03'0110110'               687.
MAC     242,007,BL03'1011001'               835.
MAC     243,007,BL03'1111111'              1032.
MAC     244,007,BL03'0100011'               575.
MAC     249,007,BL03'0100010'               569.
MAC     002,008,BL03'10001110'              381.
MAC     003,008,BL03'11110110'              498.
MAC     004,008,BL03'11110000'              488.
MAC     005,008,BL03'11101011'              471.
MAC     006,008,BL03'01010111'              322.
MAC     014,008,BL03'01010010'              290.
MAC     078,008,BL03'10000001'              360.
MAC     092,008,BL03'11110010'              489.
MAC     126,008,BL03'01011011'              327.
MAC     132,008,BL03'11110001'              488.
MAC     135,008,BL03'10110101'              426.
```

APPENDIX B: UNPCK ASSEMBLE

```
MAC     210,008,BL03'10100010'                  400.
MAC     230,008,BL03'11110111'                  498.
MAC     232,008,BL03'11101101'                  474.
MAC     245,008,BL03'11101100'                  471.
MAC     246,008,BL03'01010011'                  294.
MAC     013,009,BL03'011010010'                 172.
MAC     016,009,BL03'010100011'                 145.
MAC     028,009,BL03'111100111'                 247.
MAC     032,009,BL03'010000100'                 140.
MAC     079,009,BL03'001110110'                 134.
MAC     097,009,BL03'010110101'                 166.
MAC     122,009,BL03'010110100'                 158.
MAC     131,009,BL03'101101110'                 224.
MAC     133,009,BL03'010100001'                 144.
MAC     134,009,BL03'100000111'                 189.
MAC     136,009,BL03'100011110'                 194.
MAC     137,009,BL03'111010001'                 226.
MAC     138,009,BL03'101101100'                 214.
MAC     139,009,BL03'100000100'                 183.
MAC     140,009,BL03'010101100'                 157.
MAC     141,009,BL03'010101011'                 155.
MAC     142,009,BL03'011010011'                 174.
MAC     143,009,BL03'010000110'                 141.
MAC     163,009,BL03'111010010'                 228.
MAC     247,009,BL03'101101111'                 224.
MAC     248,009,BL03'111100110'                 243.
MAC     007,010,BL03'1000111111'                 99.
MAC     011,010,BL03'1110101001'                117.
MAC     012,010,BL03'1011010011'                105.
MAC     015,010,BL03'0101010010'                 76.
MAC     017,010,BL03'0110100010'                 85.
MAC     023,010,BL03'0101010000'                 74.
MAC     024,010,BL03'0101000001'                 72.
MAC     025,010,BL03'1011011010'                108.
MAC     027,010,BL03'0101000100'                 72.
MAC     030,010,BL03'0011101000'                 62.
MAC     031,010,BL03'0100001011'                 71.
MAC     036,010,BL03'1110100110'                114.
MAC     038,010,BL03'1000000000'                 87.
MAC     039,010,BL03'0011101010'                 66.
MAC     043,010,BL03'0101010011'                 76.
MAC     044,010,BL03'1010001111'                104.
MAC     053,010,BL03'0100001110'                 71.
MAC     060,010,BL03'0011101011'                 66.
MAC     065,010,BL03'1000001010'                 92.
MAC     080,010,BL03'0100001111'                 71.
MAC     094,010,BL03'1000000010'                 91.
```

APPENDIX B: UNPCK ASSEMBLE

```
MAC     144,010,BL03'1110100111'                115.
MAC     145,010,BL03'1011010000'                104.
MAC     146,010,BL03'1011010001'                104.
MAC     147,010,BL03'1010001100'                 99.
MAC     148,010,BL03'0101000000'                 71.
MAC     149,010,BL03'0011101001'                 65.
MAC     151,010,BL03'1000111110'                 96.
MAC     153,010,BL03'0101000101'                 72.
MAC     216,010,BL03'1110100001'                113.
MAC     233,010,BL03'0110100000'                 83.
MAC     018,011,BL03'10100011100'                50.
MAC     019,011,BL03'10000000111'                46.
MAC     020,011,BL03'10000011001'                47.
MAC     021,011,BL03'01010101011'                39.
MAC     022,011,BL03'10000010110'                46.
MAC     026,011,BL03'10100011010'                49.
MAC     029,011,BL03'11101010001'                58.
MAC     033,011,BL03'11101010100'                58.
MAC     034,011,BL03'10110110111'                56.
MAC     035,011,BL03'10110100100'                52.
MAC     037,011,BL03'11101010000'                57.
MAC     040,011,BL03'11101000000'                56.
MAC     041,011,BL03'01101000110'                43.
MAC     042,011,BL03'11101010110'                60.
MAC     046,011,BL03'00111011110'                34.
MAC     047,011,BL03'10100011101'                51.
MAC     048,011,BL03'10000000010'                44.
MAC     049,011,BL03'01101000111'                43.
MAC     050,011,BL03'10000000110'                45.
MAC     051,011,BL03'00111011100'                33.
MAC     055,011,BL03'01010100011'                38.
MAC     056,011,BL03'11101010101'                59.
MAC     061,011,BL03'00111011111'                34.
MAC     150,011,BL03'01010110100'                39.
MAC     152,011,BL03'10000011011'                48.
MAC     154,011,BL03'01010101000'                38.
MAC     155,011,BL03'01010101001'                38.
MAC     156,011,BL03'10000010111'                46.
MAC     157,011,BL03'01101000010'                41.
MAC     158,011,BL03'01000010100'                34.
MAC     162,011,BL03'01010110101'                39.
MAC     209,011,BL03'01000010101'                35.
MAC     000,012,BL03'010101000101'               19.
MAC     045,012,BL03'100000110000'               23.
MAC     052,012,BL03'111010000010'               28.
MAC     054,012,BL03'100000110101'               24.
MAC     057,012,BL03'101101001010'               26.
```

APPENDIX B: UNPCK ASSEMBLE

```
MAC     058,012,BL03'010101101101'                    20.
MAC     059,012,BL03'101101101101'                    27.
MAC     062,012,BL03'111010101110'                    29.
MAC     063,012,BL03'101101001011'                    26.
MAC     068,012,BL03'100000110001'                    23.
MAC     110,012,BL03'011010000110'                    21.
MAC     111,012,BL03'100000110100'                    23.
MAC     127,012,BL03'010101010100'                    19.
MAC     159,012,BL03'010101010101'                    19.
MAC     000,012,BL03'010101000101'                    19.
MAC     160,012,BL03'101101101100'                    26.
MAC     164,012,BL03'100000000110'                    22.
MAC     165,012,BL03'010101101100'                    19.
MAC     166,012,BL03'011010000111'                    21.
MAC     066,013,BL03'0101011011100'                   10.
MAC     076,013,BL03'1000000001110'                   11.
MAC     161,013,BL03'1110101011111'                   16.
MAC     167,013,BL03'0101011011101'                   10.
MAC     168,013,BL03'0011101110111'                    9.
MAC     169,013,BL03'1010001101111'                   13.
MAC     170,013,BL03'1000000001111'                   11.
MAC     171,013,BL03'1010001101100'                   12.
MAC     172,013,BL03'0101010001000'                    9.
MAC     187,013,BL03'1110100000110'                   14.
MAC     189,013,BL03'0011101110100'                    8.
MAC     091,014,BL03'00111011101010'                   4.
MAC     174,014,BL03'00111011101011'                   4.
MAC     175,014,BL03'11101000001110'                   7.
MAC     176,014,BL03'10100011011010'                   6.
MAC     178,014,BL03'01010100010011'                   5.
MAC     181,014,BL03'01010110111100'                   5.
MAC     182,014,BL03'01010110111101'                   5.
MAC     184,014,BL03'11101010111101'                   8.
MAC     185,014,BL03'11101000001111'                   7.
MAC     188,014,BL03'01010110111110'                   5.
MAC     190,014,BL03'11101010111100'                   7.
MAC     070,015,BL03'001110111011000'                  2.
MAC     173,015,BL03'101000110110110'                  3.
MAC     177,015,BL03'001110111011001'                  2.
MAC     179,015,BL03'101000110110111'                  3.
MAC     180,015,BL03'101000110111000'                  3.
MAC     183,015,BL03'101000110111001'                  3.
MAC     186,015,BL03'101000110111010'                  3.
MAC     191,015,BL03'001110111011010'                  2.
MAC     192,015,BL03'001110111011011'                  2.
MAC     067,016,BL03'0101010001001000'                 1.
MAC     069,016,BL03'0101010001001001'                 1.
```

APPENDIX B: UNPCK ASSEMBLE

```
MAC     072,016,BL03'0101010001001010'              1.
MAC     109,016,BL03'0101010001001011'              1.
MAC     123,016,BL03'0101011011111100'              1.
MAC     255,016,BL03'0101011011111101'              1.
MAC     254,019,BL03'01010110111111110000'          0.
MAC     071,020,BL03'01010110111111100100'          0.
MAC     073,020,BL03'01010110111111100101'          0.
MAC     074,020,BL03'01010110111111100110'          0.
MAC     081,020,BL03'01010110111111100111'          0.
MAC     082,020,BL03'01010110111111101000'          0.
MAC     083,020,BL03'01010110111111101001'          0.
MAC     084,020,BL03'01010110111111101010'          0.
MAC     085,020,BL03'01010110111111101011'          0.
MAC     086,020,BL03'01010110111111101100'          0.
MAC     087,020,BL03'01010110111111101101'          0.
MAC     088,020,BL03'01010110111111101110'          0.
MAC     089,020,BL03'01010110111111101111'          0.
MAC     090,020,BL03'01010110111111110000'          0.
MAC     095,020,BL03'01010110111111110001'          0.
MAC     098,020,BL03'01010110111111110010'          0.
MAC     099,020,BL03'01010110111111110011'          0.
MAC     100,020,BL03'01010110111111110100'          0.
MAC     101,020,BL03'01010110111111110101'          0.
MAC     102,020,BL03'01010110111111110110'          0.
MAC     103,020,BL03'01010110111111110111'          0.
MAC     104,020,BL03'01010110111111111000'          0.
MAC     105,020,BL03'01010110111111111001'          0.
MAC     106,020,BL03'01010110111111111010'          0.
MAC     108,020,BL03'01010110111111111011'          0.
MAC     112,020,BL03'01010110111111111100'          0.
MAC     113,020,BL03'01010110111111111101'          0.
MAC     114,020,BL03'01010110111111111110'          0.
MAC     115,020,BL03'01010110111111111111'          0.
MAC     116,020,BL03'10100011011101100000'          0.
MAC     117,020,BL03'10100011011101100001'          0.
MAC     118,020,BL03'10100011011101100010'          0.
MAC     119,020,BL03'10100011011101100011'          0.
MAC     120,020,BL03'10100011011101100100'          0.
MAC     121,020,BL03'10100011011101100101'          0.
MAC     124,020,BL03'10100011011101100110'          0.
MAC     202,020,BL03'10100011011101100111'          0.
MAC     203,020,BL03'10100011011101101000'          0.
MAC     204,020,BL03'10100011011101101001'          0.
MAC     205,020,BL03'10100011011101101010'          0.
MAC     206,020,BL03'10100011011101101011'          0.
MAC     207,020,BL03'10100011011101101100'          0.
```

APPENDIX B: UNPCK ASSEMBLE

```
MAC     208,020,BL03'10100011011101101101'          0.
MAC     218,020,BL03'10100011011101101110'          0.
MAC     219,020,BL03'10100011011101101111'          0.
MAC     220,020,BL03'10100011011101110000'          0.
MAC     221,020,BL03'10100011011101110001'          0.
MAC     222,020,BL03'10100011011101110010'          0.
MAC     223,020,BL03'10100011011101110011'          0.
MAC     224,020,BL03'10100011011101110100'          0.
MAC     225,020,BL03'10100011011101110101'          0.
MAC     234,020,BL03'10100011011101110110'          0.
MAC     235,020,BL03'10100011011101110111'          0.
MAC     236,020,BL03'10100011011101111000'          0.
MAC     237,020,BL03'10100011011101111001'          0.
MAC     238,020,BL03'10100011011101111010'          0.
MAC     239,020,BL03'10100011011101111011'          0.
MAC     250,020,BL03'10100011011101111100'          0.
MAC     251,020,BL03'10100011011101111101'          0.
MAC     252,020,BL03'10100011011101111110'          0.
MAC     253,020,BL03'10100011011101111111'          0.
 SPACE
 END
```

VITA

Surname: MURPHY Given Names: TIMOTHY FITZGERALD

Place of Birth: LONDON, U.K. Date of Birth: August 15, 1942

Educational Institutions Attended, with Dates of Entering and Leaving:

LIVERPOOL UNIVERSITY, LIVERPOOL, U.K.	1960	to	1963
UNIVERSITY OF VICTORIA, B.C,	1977	to	1980
	____	to	____

Degrees, Diplomas, Etc., Awarded, with Dates and Names of Institutions:

B.Sc.	1963	Liverpool University, Liverpool, U.K.
	____	
	____	

Honors and Awards:

Publications:

PARTIAL COPYRIGHT LICENSE

I hereby grant the right to lend my thesis or dissertation (the title of which is shown below) to users of the University of Victoria Library, and to make *single copies only* for such users or in response to a request from the library of any other university, or similar institution, on its behalf or for one of its users. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by me or a member of the University designated by me. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Title of Thesis

DATA SECURITY AND COMPACTION TECHNIQUES

Author [redacted]

Signature

TIMOTHY FITZGERALD MURPHY

Name

18/03/1981.

Date