Enhancing Secrecy by Data Compression : Theoretical and Practical Aspects

Colin Boyd
Communications Research Group
Electrical Engineering Laboratories
University of Manchester
Manchester M13 9PL, UK
(Email: BOYD@uk.ac.man.ee.v1)

Abstract

It was recognised by Shannon that data compression increases the strength of secrecy systems when applied prior to encryption. Compression techniques have advanced considerably in recent years. This paper considers the extent to which these techniques can increase security. Estimates are obtained for how far practical compression schemes can increase unicity distance of symmetric ciphers. It is noted that there are other good reasons for using data compression prior to encryption. Comparison is made with homophonic coding and it is suggested that data compression is more worthwhile for practical sources such as natural language.

1 Introduction

Shannon's theory of secrecy systems [11] uses key equivocation as an index to measure the security of a cryptosystem. The key equivocation is the conditional entropy of the key given the ciphertext, and can be expressed as a function of the number of plaintext characters encrypted. Shannon showed that the rate at which this equivocation falls is approximately linear with slope equal to the redundancy of the source language.

The unicity distance U can be defined as the least number of symbols such that the key equivocation, after U symbols are encrypted, is expected to be zero. Shannon also showed that U is approximately inversely proportional to the source redundancy in bits per symbol, D, for all reasonable cryptosystems

[11]. If $H(K)$ is the initial key entropy then U is approximated by

$$U = \frac{H(K)}{D}$$

An alternative definition of U is the average number of plaintext symbols required so that the redundancy (in bits) exceeds the key length. If the length of the encrypted message is less than the unicity distance then it is not possible to decide the key or plaintext uniquely from knowledge of the ciphertext alone. It is quite possible for ciphers to have an infinite U, and Shannon called such ciphers *ideal*.

In Shannons other landmark paper [12] he showed that it is possible to reduce the redundancy of sources by suitable coding. He realised that this source coding, or *data compression*, if implemented prior to encryption, would strengthen secrecy systems by reducing the rate at which the key equivocation function decreases or, equivalently, increasing U. If all source redundancy can be removed before encryption then any reasonable cipher will be ideal. The use of data compression has become widespread in recent years due to the availability of easily programmed algorithms. The benefits of data compression are obvious in terms of savings in storage and communications costs.

This principle of using compression prior to encryption is now widely known and has been repeated often, for example in modern texts such as [14]. However, the literature has not been helpful in providing quantitative measures for how far security can be extended when using practical techniques. In this paper the practical, as well as the theoretical, aspects of applying compression prior to encryption are considered, with particular reference to recent advances in data compression techniques. As a result a quantitative measure for how far the best compression schemes can be expected to increase unicity distance is obtained.

The paper is divided into three main sections, the first two of which are concerned solely with the question of how far unicity distance can be extended by practical compression. Firstly modern coding techniques are examined to see how close they can get to achieving zero redundancy. In order to do this, new bounds on the overhead of arithmetic coding are obtained which show that unicity distance can be made very large if the only overheads are due to coding inaccuracies. The next section considers the complementary problem of modelling real sources and how this affects practical compression. This allows quantitative statements to be made on how far modern compression techniques can be expected to extend unicity distance. We conclude that in the practice, for English text, only a modest increase of unicity distance is possible, by a factor of between 3 and 6. The final main section considers other aspects of using compression prior to encryption, in particular the effects on known and chosen plaintext attacks, and the possible effects on processing speed.

When considering the effect of compression on unicity distance, comparison is made with the alternative source coding technique of homophonic coding [6]. Although homophonic coding appears best in a purely theoretical approach, we suggest that it is probably not worthwhile for practical encryption of complex sources such as natural language, in comparison with compression methods.

1.1 Huffman, Ziv-Lempel and arithmetic coding

In a theoretical sense the compression coding problem was solved in 1952 when Huffman published his coding algorithm [5]. He showed that it could not be improved upon in the sense that there is no prefix free code with a shorter average word length. Despite this, the most successful modern methods of practical compression do not use Huffman coding. The reason is partly that modern methods do not encode in a symbolwise manner as Huffman coding does, but instead deal with source symbols many at a time. In addition, and perhaps more importantly, Huffman coding is not well suited to exploit the adaptive models used in the most successful methods.

The most common modern methods are arithmetic coding and dictionary coding [1]. The set of dictionary coding methods, which are variations on the method of Ziv and Lempel [18], have proved very popular recently largely due to their speed and ease of implementation, as well as their very good compression. Arithmetic coding, although less widely used, is a very simple and effective idea. It has the property that it separates the coding and modelling functions very clearly which not only makes it suitable for virtually any modelling technique, but also makes it amenable to analysis. With sophisticated modelling techniques arithmetic coding is frequently the most effective choice in terms of achievable compression.

1.2 Coding and Modelling

One of the most important recent influences on data compression has been the realisation that the whole compression process can be divided into two essentially distinct parts [10]; the modelling process, which provides a probability distribution for the source alphabet, and the coding process which uses the stream of source symbols tc be compressed, together with the probability distributions from the model, to produce an encoding. In an information theoretic model of communication various kinds of source model are usually considered. For example, symbols may be assumed to be independently emitted, or more generally a Markov model may be assumed. A practical approach to compression, and to cryptography, needs to take into account that a real source may not conform to any tractable model, or if it does then the exact probability distribution may be unknown.

The next two sections reflect the coding and modelling viewpoint. First we ignore the modelling problem by making an assumption that an exact model for the source is known. Following that we turn to the practical situation by including the modelling problem. For dictionary methods the division between modelling and coding is rather difficult to define. For Huffman and arithmetic coding the division is very clean, so we concentrate on these.

2 Coding

By Shannon's Noiseless Coding Theorem [12] it is known that for a given source it is possible to compress the output arbitrarily close to its entropy, but no further without loss of information. We are only concerned here with compression that can be completely reversed since this is necessary in practice for text.

For any message m, which is a sequence of l source symbols, we may define its information content by $I(m) = -\log p_m$ if m occurs with probability p_m. The overhead for a particular message m is then:

$$coding\ overhead = \frac{encoded\ length - I(m)}{l}$$

We assume that all encodings are binary so that the coding overhead is measured in bits per symbol (*bps*). A conventional way of measuring the effectiveness of a source code is to find the *expected* coding overhead. It will suit our purposes to look simply at bounds on the coding overhead for any message. This makes calculations simpler and also conforms to the convention in cryptography of assuming worst case conditions.

2.1 Limitations of Huffman Coding

The basic algorithm for Huffman coding [5] assumes that the model of the source is fixed and that symbols are emitted independently. Schemes for adaptive Huffman coding have also been developed but they are not as flexible as arithmetic coding in accepting different models. The limitation of Huffman coding is that an exact number of bits must be used for the encoding of each symbol. To illustrate worst case conditions note that when a binary source of just two symbols is encoded, each symbol must occupy one bit. If one symbol is far more likely than the other then the information content of that bit will be very small. As a result, the best general bound for the coding overhead is one bit per symbol. It is well known that there is zero coding overhead if all symbols have probabilities equal to an integral power of 1/2.

For a source with 'random' symbol probabilities we might expect an average overhead of 1/2 bit per symbol.

A common way of reducing the overhead is to extend the source alphabet by using blocks of two or more symbols. In this way the overhead is spread over the block, so that for an alphabet of two-symbol blocks the maximum overhead is 1/2 bit per symbol. This idea can be exploited as far as is desired in theory by taking larger and larger blocks, but the practical drawback is that the storage required increases exponentially in the block size. For ASCII text a block length of 3 or 4 is about as far as is reasonable, leading to a maximum overhead of 0.25 bits per symbol.

2.2 Limitations of arithmetic coding

Arithmetic coding encodes a source string as a number in the unit interval $[0,1)$. Unlike Huffman Coding, there is no fixed codeword for each symbol, but how each symbol is encoded depends on the rest of the string. A complete description can be found in [7] or [17]. The idea is that each symbol in the message is assigned a distinct subinterval of the unit interval of length equal to its probability. This is the *coding interval* for that symbol. As encoding proceeds a nesting of subintervals is defined. Each successive subinterval is defined by reducing the previous subinterval in proportion of the current symbol's probability. The final interval, when all symbols have been encoded, has length equal to the product of all the symbol probabilities and can be transmitted by sending any member of it. It can be easily shown that in any interval of length L there is a number that can be represented in a number of bits that is the least integer greater than $-\log L$. This is equal to the entropy of the message encoded so in theory virtually perfect compression is achieved.

The limitations of arithmetic coding are a little harder to evaluate than those of Huffman coding, since they depend on parameters defined by the implementation. In order to make reasonable estimates we use the parameters reported for a published implementation [17]. Alternative concrete implementations are considered in [2] and [7]. The overhead can be divided into three parts.

- Overhead due to finite arithmetic. Because only finite arithmetic is used to calculate the current interval at each stage, a truncation must be ma-le, resulting in a smaller interval than would be obtained with infinite precision arithmetic. Using arithmetic to 32 bits of precision, this has a very small effect on the compression obtained, since the interval has to be increased to twice its correct size before one extra bit is used.

  In [17] the overhead due to finite arithmetic is stated to be around 10^{-4}

bits per symbol, from empirical calculations. A concrete bound may also be obtained which indicates that this is, if anything, an overestimate of the effect.

- Overhead due to end effects. In practice it is not necessary to encode all symbols of a message before obtaining some output bits. However, after the final symbol is encoded it is necessary to send some additional bits to disambiguate the final symbol. This results in an overhead of not more than 2 bits per message (see [17, p. 535]).

- Overhead due to halting problem. The decoder needs to have some means of knowing when to stop decoding since the transmitted string represents any interval it is contained in. A naive way of achieving this is to send the message length prior to the message, and this clearly results in an overhead of log l bits for a message of length l. In practice this method may not be very satisfactory and so in [17] an end-of-message symbol (eom) is included in the model, which is appended to each message as the last symbol. The effect of this is that extra bits are used to send this symbol, and also that the coding interval of every other symbol is slightly shortened to accommodate the coding interval for eom. The exact overhead depends on the way that the end-of-message symbol is modelled. In [17] both a fixed and an adaptive model are implemented.

  For the the fixed model each source character is given a weight, the sum of all weights being 8000. In addition the eom symbol also has weight 1 and so has probability 1/8001, and each other symbol has its probability reduced by the factor 8000/8001. The resulting overhead for one message is

  $$-\log(8000/8001)bps + \log 8001 = 0.00018bps + 13.$$

  For the adaptive model the eom symbol is given a fixed weight of 1 while all other characters start off with weight 1 but can increase, subject to a maximum total weight of 2^{14}. Thus if the eom symbol originally has probability 1/257 then it has probability $1/(257+l)$ at the end of a message of length l. Thus the overhead for a message of length l is $\log(257+l)$ for coding the eom symbol itself plus

  $$-\log(256/257)(257/258)...(256+l)/(257+l) = \log 257 + l - 8.$$

  In other words the total overhead due to eom is $2\log(257+l) - 8$.

  Because of the maximum allowed total weight, the weights are all halved periodically and so the above formula is no longer valid. Instead, for long messages the eom symbol has a frequency in the adaptive model of between 1 in 2^{13} and 1 in 2^{14}. This results in a minimum probability of eom of 2^{-14}, while the reduction to other symbols is not more than $1 - 2^{-13}$. For long messages then, the eom results in an overhead of not more than $14 + 0.00018bps$.

In conclusion, adding the three factors together we arrive at a fair estimate for the overhead in an implementable arithmetic coding scheme of

$$0.00028 \textit{ bits per symbol} + 16 \textit{ bits per message}$$

The bounds derived above were verified by performing some simple experiments using the implementation of arithmetic coding in 'C' given in [17]. Using simple fixed models it was found that the overhead was almost exactly as predicted by the estimates.

2.3 Homophonic coding

Although arithmetic coding has very small redundancy it is possible to do better. Homophonic coding actually achieves zero redundancy even though it may result in data expansion. A useful way of viewing this surprising effect is to consider that homophonic coding works by 'adding randomness' to the source. Since an ideal cipher requires only zero redundancy, and not compression, this method is just as useful from a cryptographic viewpoint. The recent methods of Günther [4], and of Jendal, Kuhn and Massey [6] are guaranteed to have small data expansion. The links between homophonic coding and data compression are strong - using the model of [6] homophonic coding can be viewed as a homophonic mapping followed by perfect compression. Homophonic coding allocates a number of homophones (infinite in general) to each source symbol. The coding space of each symbol is divided up in such a way that each of its homophones has a part and so that the homophones can be perfectly compressed.

In comparison with compression methods, homophonic coding has two disadvantages. Firstly the implementation is more expensive - a random source of bits is required to choose the homophones and more resources are required to store or calculate the codewords. Secondly there is considerable loss in data compression. It was shown [6] that the expected coding overhead for homophonic coding is not more than 2 bits per symbol, and that this can be further reduced for memoryless sources by blocking as explained for Huffman coding. However, even in this simple case the overhead will be considerably worse than for arithmetic coding.

2.4 The Effects of Compression Coding on Unicity Distance

For Huffman coding the coding overhead will depend on the extent to which the source alphabet may be extended. If we assume an overhead of 0.125

bits per symbol, which may be typical for a source with alphabet extended to sets of four symbols, then the unicity distance for a cryptosystem will become eight times the key length in bits. For example a cryptosystem with 56 bit key length should achieve around a 450 character unicity distance.

Consider next the situation for arithmetic coding. If we assume again a cipher with a key length of 56 bits then the unicity distance U is defined by

$$16 + 0.00028U = 56$$

so that U is over 142 000 characters. This is quite a large size of message (about four times the size of this paper) and represents a vast increase compared with using ASCII, and around 300 times better than with Huffman coding.

Finally for homophonic coding we have zero redundancy and so an infinite unicity distance is achieved. Thus in a theoretical sense homophonic coding allows achievement of ideal ciphers.

3 Practical Compression Schemes

We are only concerned here with 'one-pass' compression schemes - those that use models that are either chosen beforehand and fixed, or which adapt according to the statistics of previous symbols encoded. The alternatives are two pass schemes which examine the statistics of the message first and then transmit a model along with the compressed message. There are a number of reasons that this restriction is made. From the viewpoint of increasing unicity distance, transferring a model ensures that there is a considerable amount of redundancy present. From the point of view of achieving good compression it has been shown [1] that two pass schemes are in practice no better then one-pass. Finally, one-pass schemes are far more convenient in a communications environment.

Practical compression schemes often use adaptive models which change the statistics of the source model according to those symbols encoded so far. Arithmetic coding is particularly suited to adaptive modelling since a new probability distribution can be used after every symbol is encoded. As long as the procedure for updating the model is well defined both encoder and decoder maintain the same model at each stage. A comprehensive survey of such modelling techniques is presented by Bell, Cleary and Witten [1].

3.1 The effects of the best compression methods on unicity distance

It is very difficult to make a definitive statement on what is the best data compression method currently known. For one thing techniques are constantly improving and there is much active research. But in any case we need to define what is meant by 'best'. Perhaps the first concern in the present context is how much compression is achieved, but even this matter will normally depend on the type of data being compressed. Just as important in a communications environment is the question of the cost of compression, particularly in terms of time but also in memory required. As might be expected, there is in general a trade-off between compression achieved and resources required. We will try to give a brief summary of the current situation concentrating on how well "ordinary" English text can be compressed, since this is the case for which the data exists to compare the effect on unicity distance.

A comparison of ten different compression schemes is presented in by Bell, Cleary and Witten [1]. This comparison ranges over a variety of samples including technical and non-technical English prose encoded in ASCII, binary object code, programming source code in ASCII and graphics images. Over the English text the best compression achieved (using the PPMC method with arithmetic coding) was 2.25 bits per character for technical prose, and 2.48 bit per character for fiction prose. The unicity distance achieved on encrypting such compressed text will depend crucially on the true figure of the information content per English character for the source used. There have been various methods devised for estimating the entropy of ordinary English, many using human experiments. (See [15] for a survey, and [3] for a recent method of computer analysis.) The figures obtained typically vary between 1 and 1.3 bits per symbol and depend on authors and styles of writing. However there are no studies which suggest that the entropy of English is above 1.3 bits per symbol for messages of any reasonable length and we may safely take this value as an upper bound. Thus even for best compression schemes the redundancy per symbol is at least 1 bit per character. This compares with 3.7 bits per character of redundancy for the basic 26 letter alphabet, or 6 bits per character for English text encoded as 7-bit ASCII.

Coding	Bits per char	Redundancy per char
26 letter alphabet	4.7	3.4
7 bit ASCII	7	5.7
Best compression schemes	2.3	1

Table 1: Redundancies for English Text

Thus, since unicity distance is inversely proportional to redundancy, the effect of the best compression schemes on unicity distance is to increase it between 3 and 6 times, depending on the coding used and on the true value of the information content. The best methods use arithmetic coding for which the coding overhead is a small fraction of one bit per symbol and thus contributes a small part of the total redundancy. The only way to significantly improve unicity distance for natural languages beyond these levels is to use better modelling techniques. These are the subject of much current research.

Estimates for the redundancy of real sources other than natural languages do not seem to be so readily available. Clearly many examples, such as programming source code, can be expected to have a less rich structure than natural language, and it may well be that redundancy per character can be made much lower than 1 bit per character for these sources. In [1] Lisp and Pascal source code is reported to be compressed to 1.9 and 1.84 bits per character respectively. It is possible to conceive of sources for which extremely accurate models are available and where practically infinite unicity distance may be obtained. At present this does not appear to be achievable for natural languages.

Although unicity distance may be improved by a factor of 3 to 6 it will still be far too small in general to allow keys to be changed before the unicity distance is achieved, which would guarantee some unconditional security. For example, the 56 bit key length of DES would result in a unicity distance of no more than 56 characters of plaintext. Even so, the removal of so much structure of the plaintext gives good confidence that breaking the cryptosystem using a ciphertext-only attack will be made considerably harder than without compression. It is well understood that a small unicity distance does not mean that a cryptosystem is weak - thus with ASCII encoded English, DES has a unicity distance of about 8 characters, but nobody knows a better way to break DES than with a brute force key search.

Shannon defined the *work factor* associated with a cryptosystem [11, p. 702] to be the amount of work required to break it from a given amount of ciphertext. The idea is that with more plaintext redundancy being used, the effort needed to break the cipher would be reduced. Shannon suggests that there is a strong relationship between work factor and unicity distance. If the work factor were directly proportional to the unicity distance then using compression would increase this also by a factor of 3 to 6. In practice, however, the work factor may be asymptotic to some value, and since there is frequently practically limitless ciphertext available, this should perhaps not be taken as very comforting.

3.2 Is homophonic coding worthwhile?

It was shown in section 2 that practical coding methods can compress with a coding overhead of around 0.0003 bits per symbol for long messages. However, as we have just seen, the best compression methods (including those which achieve this sort of coding efficiency) can only compress natural language with remaining redundancy at least 1 bit per character. In other words nearly all the redundancy remaining is due to approximations in the modelling and not to problems of coding. Even with Huffman coding the majority of the overhead is due to modelling.

The reason for the modelling overhead is that the incorrect probabilities are used for each message. Thus a particular message m of length l may have an actual probability p_m of ocurring, but the model assumes that it has a probability of p'_m. Thus if we write

$$Modelling\ Overhead = \frac{\log(p_m/p'_m)}{l}$$

then the total overhead for compression is

$$\frac{Encoded\ length - I(m)}{l} = Coding\ Overhead + Modelling\ Overhead$$

For homophonic coding the coding overhead is zero but the modelling problem, and thus the modelling overhead, is the same as for compression coding. Thus, for natural language, using homophonic coding would achieve little advantage in loss of redundancy while resulting in considerably higher implementational costs and loss of much compression. Thus from the viewpoint of reducing redundancy there seems little point in using homophonic coding. The reason that homophonic coding does not achieve optimal compression, even though it does achieve optimal coding, is that randomness is introduced. We cannot say that this addition of randomness will not make cryptanalysis harder, but equally other methods of adding randomness may be just as effective.

4 Other Factors

Up till now we have been considering the benefits of compression only in terms of increasing the unicity distance of a cipher - in other words we have been aiming at an approximation to an ideal cipher of Shannon. In this section other factors are considered, including the benefits of compression against other forms of attack, and the implications for speed of processing.

4.1 Using adaptive modelling to increase security

Consider a known plaintext attack against a cryptosystem to which compression is applied beforehand. On the surface it may seem that compression provides no extra defence against such an attack. For an attacker who obtains known ciphertext and the corresponding plaintext which was compressed before encryption, may compress the plaintext himself to obtain plaintext/ciphertext pairs for the basic cryptosystem. It is here that the use of adaptive modelling can provide additional security.

The use of adaptive modelling for cryptographic purposes has previously been considered by Witten and Cleary [16]. They proposed the use of arithmetic coding with adaptive modelling as a cryptosystem itself, using the initial state of the source model as the key. They argue that the model will become so complex and dependent on the whole of the plaintext that analysis for a cryptanalyst is impractical. However they do not provide any concrete evidence that the idea is secure.

By using instead the compressed output as the input to a cryptosystem in which confidence has been established, we can be fairly sure of having made cryptanalysis considerably harder. However, the benefits of the above idea can also be claimed. Instead of using a key to initialise an adaptive model, a random initialiser can be used as a prefix for each message. This will indeed make the model unpredictable and have a complicated affect on its future states (note that such models usually expand in size over time). In addition such a randomizer would make a known or chosen plaintext attack more difficult to mount if the initialiser were not itself known or chosen, much in the manner that initialisation values for block ciphers can. This initialiser need not be sent in advance as long as its length is agreed beforehand.

It is perhaps worth extending this analogy with block ciphers. A common mode of operation of block ciphers is to chain the ciphertext blocks together so that the ciphertext of each block depends on all the preceding blocks. This property is also true when adaptive modelling is used but now on a character by character basis. This property ensures that recurring pieces of plaintext are encrypted differently. It is also useful in maintaining the integrity of the message, in that parts of the encrypted text cannot be deleted or transposed without completely altering the subsequent decrypted text.

4.2 Speed

When considering the practical implication of using data compression prior to encryption it is important to examine the speed of compression. In a communications environment it may not be tolerable for a large delay to be introduced. Not surprisingly there is again a trade-off here, this time between speed of compression and compression achieved. Nevertheless even the best compression techniques are not likely to have too problematic an effect. In [1] the speed of the PPMC method mentioned above is reported as 2000 characters per second on a Vax. Schemes which achieve less compression go faster - a version of Ziv-Lempel coding is reported in [1] to run at 6000 characters per second for encoding and nearly twice as fast decoding.

The effects of speed when applying compression prior to encryption will depend also on the speed of the underlying encryption. There will of course be less material to encrypt if compression is used. If we assume also that pipelining of algorithms is possible then we are looking for compression algorithms which run at similar speeds to the encryption algorithm in order that there is similar performance. Implementations of DES in software run typically between 20 and 100 kbit/s [13]. (More recent products claim speeds of around 200 kbit/s.) If 8-bit ASCII is used for encrypting English this translates to between 2500 and 12 000 characters per second. Clearly it is difficult to draw precise conclusions from these figures on different machines, but it appears that compression may result in a small slow down in running speed, perhaps around a half in some cases.

Hardware encryption with, say, DES, can go much faster than this, up to 45 Mbit/s [13]. Compression in software cannot match these speeds. Hardware designs exist [9] which promise adaptive data compression at 100 Mbit/s. If this can be achieved then the use of data compression prior to encryption will be able to increase the speed of encrypting a message, in addition to its other benefits. For slower encryption algorithms, such as RSA, compression prior to encryption may also result in an increase of speed.

4.3 Error propagation

The one clear disadvantage in using data compression is the effect it has upon error propagation. Whilst Huffman coding with a fixed model has quite good re-synchronisation properties [8], coding with adaptive models appears not to have any such properties. Once a single bit is lost or changed the whole model thereafter is out of synchronisation and will have catastrophic effects. Whether this is a major problem will depend on the channel being used and if essentially error free transmission can be achieved. A silver lining to this cloud is the benefit of such a property in terms of authentication of the message as discussed in 4.1.

5 Conclusion

In this paper we have made the following points.

1. Modern coding techniques can reduce the coding overhead to less than 0.0003 bits per symbol plus 16 bits per message in practice.

2. The major overhead is due to limitations in modelling. This results in a current best achievement of around 2.3 bits per symbol for ordinary English text.

3. Practical compression of English text results in increase of unicity distance of between 3 and 6 times.

4. Homophonic coding suffers from the same modelling limitations and thus has little benefit over compression coding in reducing redundancy.

5. Using compression coding helps reduce resources used, and may somctimes even improve the speed of encryption.

6. Using a random initialiser with an adaptive model appears to make practical cryptanalysis very hard.

In summary we have seen that ideal cryptosystems cannot be achieved either by using compression or homophonic coding. However modern compression, such as arithmetic coding, used prior to encryption, results in both theoretical and practical strengthening of the cryptosystem with little or no loss of performance while also saving on storage and/or transmission costs. The only negative consideration is whether errors are likely. Further research on practical implementations of combined compression and encryption would be useful.

6 Acknowledgement

I am very grateful to Ian Witten for his interest in this paper and for making suggestions for important improvements.

References

[1] T.Bell, I.H.Witten and J.G.Cleary, *Modeling for Text Compression*, ACM Computing Surveys, 21,4, December 1989.

[2] G.V.Cormack and R.N.S.Horspool, *Data Compression using Dynamic Markov Modelling*, The Computer Journal, 30, 6, 1987.

[3] P.Grassberger, *Estimating the Information Content of Symbol Sequences and Efficient Codes*, IEEE Transactions on Information Theory, IT-35, 3, pp 669-675, 1989.

[4] C.G.Günther, *A Universal Algorithm for Homophonic Coding*, Proceedings of Eurocrypt 88, Springer-Verlag, 1988.

[5] D.A.Huffman, *A method for the Construction of Minimum-Redundancy Codes*, Proceedings of the IERE, 40,9,1952.

[6] H.N.Jendal, Y.J.B.Kuhn and J.L.Massey, *An Information-Theoretic Treatment of Homophonic Substitution*, Proceedings of Eurocrypt 89, Springer-Verlag, 1990.

[7] G.G.Langdon, *An Introduction to Arithmetic Coding*, IBM Journal of Research and Development, 28,2, 1984.

[8] D.A.Lelewer and D.S.Hirschberg, *Data Compression*, ACM Computing Surveys, 13,3,1987.

[9] R.Phillips and S.Jones, *A 100 MBit/s Adaptive Compressor Chip*, Abstracts of Second Bangor Symposium on Communications, May 1990.

[10] J.Rissanen and G.G.Langdon, *Universal Modelling and Coding*, IEEE Transactions on Information Theory, IT-27, 1, pp 12-23, 1981.

[11] C.E.Shannon, *Communication Theory of Secrecy Systems*, Bell Systems Technical Journal, 656-715, 1949.

[12] C.E.Shannon and W.Weaver, *The Mathematical Theory of Communication*, University of Illinois Press, 1949.

[13] M.Smid and D.Branstad, *The Data Encryption Standard: Past and Future*, Proceedings of the IEEE, 76,5,1988.

[14] H.C.A.van Tilborg, *An Introduction to Cryptology*, Kluwer Academic Publishers, 1988.

[15] D.Welsh, *Codes and Cryptography*, Clarendon Press, Oxford, 1988.

[16] I.H.Witten and J.G.Cleary, *On the Privacy Afforded by Adaptive Text Compression*, Computers and Security, 7, 1988, pp397-408.

[17] I.H.Witten, R.Neal and J.G.Cleary, *Arithmetic Coding for Data Compression*, Communications of the ACM, 30,6,1987.

[18] J.Ziv and A.Lempel, *A universal algorithm for sequential data compression*, IEEE Transactions on Information Theory, IT-23,3,pp 337-343, 1977.