A Fast Homophonic Coding Algorithm Based on Arithmetic Coding

W T Penzhorn

Department of Electrical and Electronic Engineering
University of Pretoria, 0002 Pretoria, South Africa
e-mail: walter.penzhorn@ee.up.ac.za

Abstract. We present a practical algorithm for the homophonic coding of a message source, as required for cryptographic purposes. The purpose of homophonic coding is to transform the output of a non-uniformly distributed message source into a sequence of uniformly distributed symbols. This is achieved by randomly mapping each source symbol into one of a set of homophones. The selected homophones are then encoded by means of arithmetic coding, after which they can be encrypted with a suitable cryptographic algorithm. The advantage of homophonic coding above source coding is that source coding merely protects against a ciphertext-only attack, whereas homophonic coding provides additional protection against known-plaintext and chosen-plaintext attacks. This paper introduces a fast algorithm for homophonic coding based on arithmetic coding, termed the *shift-and-add* algorithm, which makes use of the fact that the set of homophones are chosen according to a dyadic probability distribution. This leads to a particularly simple, efficient implementation, requiring no multiplications but only shifts and additions. The usefulness of the algorithm is demonstrated by the homophonic coding of an ASCII textfile. The simulation results show that homophonic coding increases the entropy by less than 2 bits per symbol, and also provides source encoding (data compression).

1 Introduction

The history of cryptology shows that most secret-key cipher systems were broken by exploiting the fact that plaintext characters are not uniformly distributed. The technique of *homophonic substitution*, or *homophonic coding*, (sometimes also called *multiple substitution*) is a well-known method for converting a given plaintext sequence into a (more) random sequence. Such a coding scheme was, for example, used by the Duke of Mantua in his correspondence with Simeone de Crema and was also used in the well-known Beale ciphers [6].

In *simple substitution* each plaintext source symbol is replaced with a corresponding codeword by means of a *one-to-one* mapping. *Homophonic coding* is similar, except that the mapping is *one-to-many*, as each source symbol is

mapped into a *set of codewords* referred to as *homophones*. The term *homophonic* means to *"sound the same"* which indicates that different codewords refer to the same source symbol. Source symbols which occur frequently are mapped into more than one codeword, so as to flatten the frequency distribution of the resulting codewords. An important attribute of homophonic coding is that in the one-to-many mapping, a codeword is picked at *random* from the set of homophones to represent the given source symbol, thereby introducing external randomness into a given message.

2 Illustration of Homophonic Coding Based on Huffman Source Coding

To illustrate the main idea behind homophonic coding, consider the example in Fig. 1, due to Massey [9] and Günther [4]. The latter introduced *variable-length homophonic coding*, which was subsequently put into a more general framework in [5]. In this example, the homophonic coder consists of the following three elements: a memoryless binary message source (BMS), a homophonic channel and a binary Huffman source encoder. The message source U produces symbols from the two-letter alphabet $U = \{u_1, u_2\}$, with probability $P(u_1) = P(U_i = u_1) = 1/4$ and $P(u_2) = P(U_i = u_2) = 3/4$.

The (memoryless) homophonic coding channel substitutes the source symbols U_i for the homophones taken from the alphabet $V = \{v_1, v_2, v_3\}$. It is characterized by the set of transition probabilities $P(V = v_j|U = u_i)$ such that for each j there is exactly one i such that $P(V = v_j|U = u_i) \neq 0$. Those v_j for which $P(V = v_j|U = u_i) \neq 0$ are the *homophones* for u_i. For the example in Fig. 1 we have $P(v_1|u_1) = 1$, $P(v_2|u_2) = 2/3$ and $P(v_3|u_2) = 1/3$ from which the probability of occurrence of the homophones readily follows as $P(v_1) = 1/4$, $P(v_2) = 1/2$ and $P(v_3) = 1/4$. Finally, the selected homophones are encoded as binary codewords in the alphabet $X = \{x_1, x_2\}$ by means of Huffman source coding.

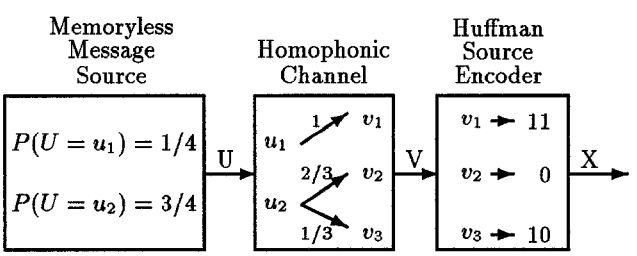

Fig. 1. Variable-length homophonic coder with Huffman source coding.

The operation of the homophonic coder is as follows: When the message source produces the symbol u_1, the homophonic channel outputs the codeword v_1, and

the resulting binary codeword produced by the Huffman coder is $x_1 = 11$. Note that this is a deterministic mapping, since u_1 is always represented as v_1. However, source symbol u_2 is represented as either v_2 or v_3, selected randomly: v_2 is chosen with probability 2/3, and v_3 with probability 1/3, as indicated in Fig. 1. Note that the resulting binary codewords produced by the Huffman coder are of unequal length ($l_2 = 1$ and $l_1 = l_3 = 2$ respectively). It is easily verified that the resulting bit string output by the Huffman source coder is uniformly distributed, and that the average codeword length $E[w]$ is 3/2. Fig. 1 also introduces the general form of homophonic coding, or multiple substitution, which consists of a *message source*, a *reversible memoryless mapping* and a *source encoder*. Conceptually, the required mapping is performed by the *homophonic channel*, whose output is then "compressed" by a suitable source encoder. The homophonic channel and source encoder will be jointly referred to as the *homophonic coder*. Furthermore, in Fig. 1 it is implicitly assumed that an external *randomiser* provides the appropriate random numbers for the selection of the homophones.

In spite of the elegance and simplicity of this scheme, its main drawback is the fact that it is not easily adaptable to changing source statistics, as was also noted in [10]. For this reason we propose the use of arithmetic source coding in homophonic coding.

3 Information-theoretic Analysis of Homophonic Coding

In this section we briefly review some important information theoretical results from [5] and [10]. In Fig. 2 the block diagram of a general homophonic coding scheme is shown. The *message source* U produces a sequence of random variables $U_1, U_2, U_3 \ldots$. The U_i are taken from the L-ary alphabet $U = \{u_1, u_2, \ldots, u_L\}$, $2 \leq L < \infty$, according to the probability distribution $P(u_1), P(u_2), \ldots, P(u_L)$. For simplicity we shall assume that the source is *memoryless* and *stationary*, so that the coding problem reduces to the encoding of the single variable U_i.
The homophonic coder, which consists of a memoryless homophonic channel and a source encoder, maps the output of the message source into the sequence $X_1, X_2, X_3 \ldots$. The randomizer provides the random numbers needed for the selection of the homophones. In [5] an elegant method is introduced for the generation of the required "awkward" random numbers by means of a random binary symmetric source (BSS). For the purpose of this analysis we shall regard the output of the homophonic coder as the *plaintext* sequence to be encrypted.

Let X^n and Y^n denote the *finite* plaintext sequence $\{X_1, X_2, \ldots, X_n\}$ and ciphertext sequence $\{Y_1, Y_2, \ldots, Y_n\}$, respectively. As is customary, and as Fig. 2 suggests, we assume that the secret key Z is statistically independent from the plaintext sequence X^n for all n. A cipher system is called *non-expanding* if the plaintext digits and ciphertext digits take values in the same D-ary alphabet and there is an increasing infinite sequence of positive integers $n_1, n_2, n_3, \ldots$ such that, when Z is known, X^n and Y^n uniquely determine one another for all $n \in S = \{n_1, n_2, n_3, \ldots\}$. We shall also call a sequence of L-ary random

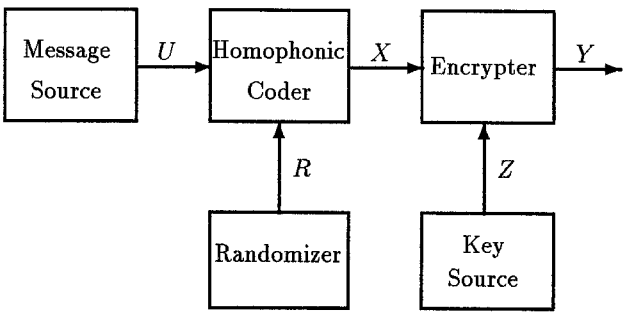

Fig. 2. Schematic diagram of a general homophonic coding scheme.

variables *completely random* if each of its digits is statistically independent of the preceding digits and is equally likely to take on any of the L possible values. Shannon [12] defined the *key equivocation*, or *conditional entropy*, of the secret key Z, given the first n digits of ciphertext as

$$H(Z|Y_1, Y_2, \ldots, Y_n) = H(Z|Y^n) \ .$$

Since $H(Z|Y^n)$ can only decrease as n increases, Shannon called a cipher system *ideal* if the key equivocation approaches a non-zero value as n tends toward infinity, and *strongly ideal* if $H(Z|Y^n)$ is constant, i.e.

$$H(Z|Y^n) = H(Z) \quad \forall n$$

which is equivalent to the statement that the ciphertext sequence is statistically independent of the secret key. The following important results were proved in [5, 10] and are quoted here without proof:

Proposition 1. *If the plaintext sequence encrypted by means of a non-expanding secret-key cipher is completely random, then the ciphertext sequence is also completely random, and is also statistically independent of the secret key.*

Corollary 2. *If the plaintext sequence encrypted by a non-expanding secret-key cipher is completely random, then the cipher is strongly ideal (regardless of the probability distribution for the secret key). Moreover, the conditional entropy of the plaintext sequence, given the ciphertext sequence, satisfies*

$$H(X^n|Y^n) \approx H(Z)$$

for all n sufficiently large.

Therefore, in a ciphertext-only attack, the cryptanalyst can do no better than guessing at random from among as many possibilities as there are possible values of the secret key Z. This illustrates the fact that virtually any secret key cipher can be used as the cipher in a strongly ideal cipher system, *provided that the plaintext source emits a completely random sequence.* The goal of homophonic substitution is precisely to convert a non-uniformly distributed source into a completely random one.

Definition 3. A Homophonic coding scheme is called *perfect* if the resulting encoded binary sequence $X_1, X_2, X_3, \ldots$ is completely random.

Proposition 4. *For the homophonic coder shown in Fig. 1 the expression*

$$H(U) \leq H(V) \leq E[W] = \sum_i P(v_i) l_i$$

holds with equality on the left if and only if the homophonic channel is deterministic, and with equality on the right if and only if the homophonic coder is perfect. Moreover, there exists a binary arithmetic source coding of V such that the scheme is **perfect** *if only if $P(v_i)$ is dyadic, i.e. if and only if $P(V = v_i) = 2^{-l_i}$ holds for all values v_i of V where l_i is the length of the binary codeword assigned to v_i.*

For a proof, see [5].

Definition 5. A homophonic coding scheme is called *optimum* if it is perfect and minimizes the expected length $E[W]$ of the binary codeword assigned to the homophonic channel output V, when the input is the message source output U.

Definition 6. The associated probability distribution $P(v_1), P(v_2), \ldots, P(v_L)$ of a message source V is called *dyadic* if $-\log_2 P(v_i)$ is an integer for all values of i.

Proposition 7. *A homophonic channel is optimum if and only if, for every $u \in U$ its homophones equal (in some order) the terms in the unique dyadic decomposition*

$$P(U = u_i) = \sum_{j \geq 1} P(v_i)^{(j)}$$

where the dyadic decomposition is either a finite or an infinite sum.

For a proof, see [5].

Proposition 4 shows that the task of designing an optimum homophonic coder requires a homophonic channel that minimizes the entropy V. According to Proposition 7, this is equivalent to the requirement that the homophones are to be associated according to the *dyadic* decomposition of the source probabilities $P(u_i)$.

As an example, consider again the homophonic channel shown in Fig. 1. The source symbols occur with probabilities $P(U = u_1) = 1/4$ and $P(U = u_2) = 3/4$. Since $P(U = u_1) = 1/4 = 2^{-2}$ is already in dyadic form, no further decomposition of u_1 is required, and v_1 may be assigned directly to it, i.e. $P(v_1) = 1/4$. For the assignment of homophones to u_2, several possibilities exist. One possibility would be the following decomposition: $P(u_2) = 3/4 = 1/4 + 1/4 + 1/4$, with the associated three homophones each occurring with probability 1/4. According to Proposition 4 this decomposition is not optimal, since it is not dyadic. Another possibility is $P(U = u_2) = 3/4 = 1/2 + 1/8 + 1/16 + 1/32 + \ldots$, which is also not optimal.

Finally, $P(u_2) = 1/2 + 1/4$ gives the desired optimal (dyadic) decomposition. The homophones v_2 and v_3 are assigned to u_2 with probability 1/2 and 1/4 respectively to form a dyadic homophonic channel. Hence, this is an example of a *perfect* homophonic channel. Jendahl et al [5] have also derived the following useful upper bound on $H(V)$ for an optimum homophonic coder.

Proposition 8. *For an optimum binary homophonic coder*

$$H(U) \leq H(V) = E[W] < H(U) + 2$$

In the remainder of this article we will show how arithmetic coding may be applied in optimum homophonic coding.

4 Arithmetic Coding

Arithmetic source coding was first introduced by Rissanen and Langdon [7, 8, 11], and may viewed as a generalisation of Shannon-Fano-Elias coding [3]. In spite of the fact that arithmetic coding is more complex than Huffman coding, it offers some important advantages:

1. As a source coding scheme, arithmetic coding is able to approach the entropy of the source arbitrarily close (for details see [3, 7, 11]).
2. A clear separation exists between modelling of source statistics and encoding of source symbols, which has distinct practical advantages.
3. The algorithm is easily adaptable to varying source statistics.
4. It is not necessary to arrange the symbol probabilities in any particular order, as is required for Huffman coding.

The main idea behind arithmetic source coding is best illustrated by means of an example. We will find it beneficial to take a slight detour, and to consider Huffman source coding first. Thereafter, arithmetic will readily follow as a generalization of Huffman coding.

Table 1. Sample source statistics.

Symbol	Huffman Codewords	$P(v_i)$ (in binary)	Cumulative probability $\sum P(v_i)$	Associated subinterval
v_1	11	.01	.00	$[.0 - .01)$
v_2	0	.1	.01	$[.01 - .11)$
v_3	10	.01	.11	$[.11 - 1)$

4.1 Review of Huffman Source Coding

Consider a source V, producing symbols from the alphabet $V = \{v_1, v_2, v_3\}$ with associated probabilities 1/4, 1/2 and 1/4 respectively, as shown in Table 1. Note that the probabilities are expressed as binary fractions.

The Huffman codewords shown in column two are *prefix-free*, i.e. no codeword is the prefix of another. Suppose the following string of source symbols is to be encoded: v_2, v_2, v_3, v_1. The encoding process is illustrated below by means of Table 2.

Table 2. Illustration of Huffman encoding.

Symbol no.	Symbol	Codewords					
1	v_2	0					
2	v_2		0				
3	v_3			1	0		
4	v_1					1	1
Codeword		0	0	1	0	1	1

The resulting codeword is 001011, obtained by *non-overlapping concatenation* of the individual codewords for each symbol in the sequence. Decoding amounts to a comparison process, starting with the first bit of the received code string. Note that this code is an example of a First-In-First-Out code (FIFO). This is a highly desirable attribute, which implies that the first received symbol can be decoded before the last symbol has been received.

4.2 Review of Arithmetic Source Coding

Next, consider the encoding of the same sequence by means of arithmetic coding. In arithmetic coding, the codewords representing the source symbols can be viewed as *code points* on the half-open unit interval $[0, 1)$, which divide the unit

interval into non-overlapping subintervals. Each codeword (code point) is equal to the *cumulative sum* of the probabilities of the preceding symbols, $\sum P(v_i)$, as shown in column four of Table 1. Note that the probabilities are expressed as *binary fractions*. Each symbol is identified with the *lower* point of the corresponding subinterval, and customarily this is referred to as the *code point* C. For example, the code point for symbol v_1 is .0 and its associated subinterval is $[.0-.01)$, and for symbol v_3 the code point is .11 and the corresponding subinterval is $[.11-1)$. The notation "[.11, 1)" means that .11 is included in the interval, as well as all fractions equal to or greater than .11 but less than 1. Symbol v_2 is assigned 1/2 of the unit interval, and symbols v_1 and v_3 each 1/4, as illustrated in Fig. 3. The size of the interval above each code point corresponds to the probability of that symbol, and is referred to as the *interval width* A. It is important to note that the symbols in Table 1 appear with their probabilities $P(v_i)$ arranged in *arbitrary* order.

In arithmetic coding, encoding essentially amounts to repeated division of subintervals. The encoding of the given symbol string v_2, v_2, v_3, v_1 is done as follows: For the encoding of the first symbol, v_2, the interval [.01, .11), is selected. To encode the second symbol, v_2, the subinterval [.01, .11) is divided into the same proportions as the original unit interval. The subinterval assigned to the second symbol, v_2, is then [.001, .101). For the third symbol this subinterval is again divided, and the corresponding subinterval for v_3 is [.1001, .1010). The encoding process is graphically illustrated in Fig. 3. This repeated subdivision is continued until all symbols in the sequence have been encoded. Decoding amounts to magnitude comparison, essentially following the inverse of this procedure.

This description of arithmetic coding leads tot he conclusion that two recursions are needed, one for the *code point* C, and one for the *interval width* A. This leads to the following two encoding steps.

Step 1: New code point
The first recursion determines the new code point as the sum of the current code point C, and the product of the width A of the current interval and the cumulative probability $\sum P(v_i)$ of the symbol v_i:

$$C_k = C_{k-1} + A_{k-1} \times \sum P(v_i) \quad ; k = 1, 2, 3 \cdots \qquad \text{with } C_0 = 0 \text{ and } A_0 = 1$$

Step 2: New interval width
The second recursion determines the width A of the new interval, which is the product of the probabilities of the data symbols encoded so far. Thus, the new interval width for the symbol v_i is:

$$A_k = A_{k-1} \times P(v_i) \quad ; k = 1, 2, 3 \cdots \qquad \text{with } A_0 = 1$$

In this way the next code point and its interval width are recursively calculated from the current code point C and interval width A. This double recursion is central to arithmetic coding, and facilitates FIFO-encoding, which is a highly desirable feature for any practical source coding scheme, as noted in the previous section.

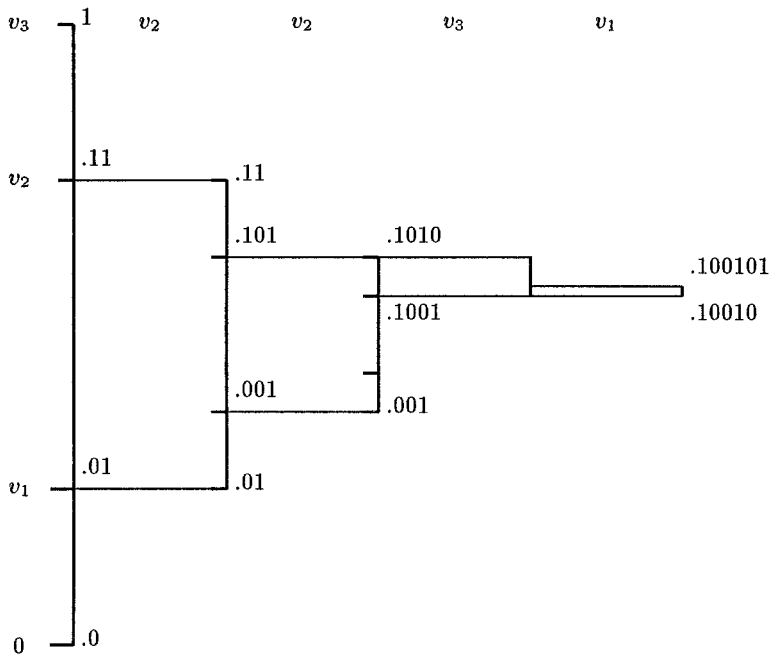

Fig. 3. Illustration of arithmetic coding.

In the arithmetic coding literature the expression $A_{k-1} \times \sum P(v_i)$, which is added to the previous code point C_{k-1}, is often referred to as the *augend* [7]. In Table 3 it is illustrated that the encoding process amounts to the addition of properly scaled cumulative probabilities $\sum P(v_i)$ (augends) to the code string.

Table 3. Arithmetic coding as sums of augends.

Symbol no.	Symbol	Augends				
1	v_2	.0	1			
2	v_2		0	1		
3	v_3			1	1	
4	v_1					0
Codeword		.1	0	0	1	0

The bit string output by the encoder is .10010. Note that 5 bits are needed to represent the given source string, compared to the 6 bits required for Huffman

coding. Comparison with the Huffman coding shown in Table 2 shows that arithmetic coding determines the code string by *overlapped addition* of the individual codewords, instead of *non-overlapped* concatenation. This is the primary reason why arithmetic coding is able to approach the source's entropy arbitrarily close.

The encoding of the third source symbol exemplifies a small problem, called the *carry-over problem.* After encoding the first two symbols, we find that when encoding the third symbol, v_3, all the bits in the codeword are being changed due to the carry-bit, which has a ripple-effect on the already calculated codeword bits. This problem is easily alleviated by means of *bit-stuffing* [8], whereby an additional 0 is introduced into the codeword bit string if a certain number of consecutive 1s have occurred.

4.3 Arithmetic Coding: Decoder Operation

The receiver obtains the code string .10010, which tells the decoder what the encoder did. In essence, the decoder recursively "undoes" the encoder's recursion. This is done in three steps:

Step 1: Decoder comparison
The decoder compares the magnitude of the received binary code string with the cumulative probabilities $\sum P(v_i)$ in Table 1. This shows that the magnitude of the binary string .10010 is equal to, or greater than .01 but less than .11. Hence the first received symbol is decoded as v_2, since the received code string value lies in the sub-interval $[.01, .11)$. Had the first symbol been, for example, v_1, then the code string magnitude would have been less than .01. Once the symbol is decoded it is possible to use the same recursion as the encoder to calculate the interval width:

$$A_k = A_{k-1} \times P(v_i) \quad ; k = 1, 2, 3 \cdots \qquad \text{with } A_0 = 1 \ .$$

For the first decoded symbol this gives: $A_1 = 1 \times .1 = .1$

Step 2: Decoder readjust
Next, the decoder subtracts from the received code string the value of the cumulative probability $\sum P(v_i)$ which corresponds to the decoded symbol v_i. For the first decoded symbol the value $\sum P(v_2) = .01$ is subtracted: $.10010 - .01 = .01010$.

Step 3: Decoder scaling
During encoding the new code point C_k is determined by multiplying $\sum P(v_i)$ with the current interval width A_k. The effect of this multiplication can be "undone" by division with the interval width, or alternatively, by multiplication with the inverse of the interval width. This gives: $.01010 \times 2 = .1010$.

4.4 Discussion

A salient feature of arithmetic coding is the fact that *modelling* and *coding* are separable by a clear-cut line. This separation enables the encoder to adapt to changing source statistics "on the fly". Furthermore, the encoder can be combined with any suitable source modelling algorithm, which has distinct practical advantages when encountering widely varying source statistics.

It should be noted that there does not exist just one *single* arithmetic coding algorithm. Rather, several classes of arithmetic coding can be identified [1, 7, 8, 11]. Although the underlying idea of arithmetic coding is simple, in practice various ad hoc tricks are needed to cater for some awkward practical problems. The specific version of arithmetic coding proposed in this paper, which we shall refer to as the *shift-and-add algorithm*, is much simpler than any of the other cited implementations.

This is a direct consequence of the fact that the symbol probabilities were chosen to be dyadic, i.e. negative powers of two. As shown in Table 3, encoding amounts to the adding of correctly shifted augends. At first glance the requirement that symbol probabilities are constrained to negative powers of two may appear severely restrictive. However, in the next section it will become clear that this requirement can be utilized to great advantage in the case of homophonic coding.

5 A Homophonic Coding Algorithm Based on Arithmetic Coding

We are now in a position to introduce the algorithm for homophonic coding based on arithmetic coding to homophonic coding. In the sequel each of these steps will be discussed briefly.

Step 1: Source modelling
The first step is to estimate the statistics of the source, which requires source modelling. It is beyond the scope of this paper to discuss this important topic, but the interested reader is referred to the book by Bell et al [1], and the articles by Langdon and Rissanen [7][11]. At the end of this section we present some simulation results of the homophonic coding of an ASCII text file. Although no sophisticated source modelling was employed, the simulations still yielded good, useful results.

Step 2: Design of the homophonic channel
The next step is to design the homophonic channel. As mentioned earlier, this requires a dyadic decomposition of the source symbol probabilities. This step is best illustrated by means of an example. Consider a message source which generates symbols from the 5-letter alphabet $U = \{u_1, u_2, u_3, u_4, u_5\}$. Suppose the following sequence of 15 symbols is to be encoded:

$$u_1\ u_1\ u_2\ u_3\ u_3\ u_5\ u_2\ u_1\ u_4\ u_4\ u_5\ u_1\ u_2\ u_1\ u_4\ .$$

Since the purpose is to illustrate the design of the homophonic channel, it suffices to obtain a crude estimate of symbol probabilities by counting the frequency of occurrence of each symbol. The resulting estimated source probabilities are shown in Table 4, expressed as a decimal fraction in column 2, and as a binary fraction in column 3.

Table 4. Example of designing a homophonic channel.

Source Symbols				Homophones		
Symbol	$P(u_i)$ (decimal)	$P(u_i)$ (binary)	$P(u_i)$ (truncated)	Symbol	$P(v_{ij})$ (binary)	$\sum P(v_{ij'})$ (binary)
u_1	.3333	.01010101	.0101	v_{11}	.01	.0000
				v_{12}	.0001	.0100
u_2	.2	.00110011	.0011	v_{21}	.001	.0101
				v_{22}	.0001	.0111
u_3	.1333	.00100010	.0010	v_{31}	.001	.1000
u_4	.2	.00110011	.0011	v_{41}	.001	.1010
				v_{42}	.0001	.1100
u_5	.1333	.00100010	.0010	v_{51}	.001	.1101

One important aspect of the shift-and-add arithmetic coding algorithm is the need to represent source probabilities with finite precision arithmetic, which depends on the register-size of the arithmetic processor being used. For purposes of illustration the precision has been limited to 4 bits, as shown in column 4 of Table 4. The dyadic decomposition of the truncated source probabilities readily follows, and each symbol is mapped to a finite number of homophones. This is illustrated below for u_1:

$$\begin{aligned} P(u_1) &= 0.0101\ldots \qquad \text{(truncated)} \\ &= 1/4 + 1/16 + \epsilon \\ &= P(v_{11}) + P(v_{12}) + \epsilon \end{aligned}$$

As a result of the truncation of the probabilities to a finite precision there will always occur a small error ϵ in the dyadic approximation. However, the magnitude of this error depends on the choice of register-size can be made arbitrarily small.

The right-most column of Table 4 contains the accumulated symbol probabilities $\sum P(v_{ij})$, which represent the codewords for the homophones, as previously discussed. Once the $P(v_{ij})$ and $\sum P(v_{ij})$ have been calculated, the design of the homophonic channel is completed.

In Fig. 4 the allocation of homophones to the source symbols is illustrated. Note that the source symbols do not fill the entire interval $[0, 1)$, but that a small error occurs as a result of the truncation of symbol probabilities to a resolution of 4 bits.

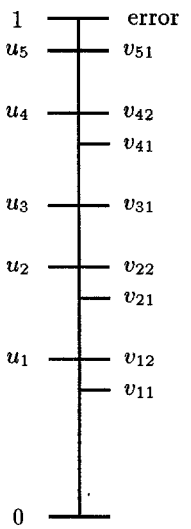

Fig. 4. Illustration of the allocation of homophones.

Step 3: Random selection of homophones
In the third step of the homophonic coding algorithm, each source symbol to be encoded is mapped into one of its associated homophones, chosen at random. The homophones are selected by means of an external randomiser. This introduces additional randomness into the message, which accounts for the increase in entropy of maximally 2 bits/symbol (see Proposition 8). In [5] a novel method is introduced for the generation of the required "awkward" random numbers by means of a binary symmetric source (BSS).

It is interesting to note that the sequence of random binary digits produced by the BSS could be used to transmit useful information. This provides a low-speed subliminal channel, which could be used very effectively in a practical cryptosystem.

Step 4: Arithmetic coding of the homophones
The final step is to encode each randomly selected homophone by means of the shift-and-add arithmetic coding algorithm introduced in the previous section. The output of the arithmetic coder is a string of binary digits. The homophonic

coder thus maps a given (non-uniformly distributed) sequence of source symbols into a uniformly-distributed bit string. It should be pointed out that any implementation of arithmetic coding could be used in this step, once the probabilities $P(v_{ij})$ are known. However, experiments have shown that the shift-add-add algorithm introduced here is about 25% faster than the algorithm given in [1].
In summary, the homophonic coding algorithm consists of the following steps:

1. Modelling: Estimate source statistics.
2. Design of homophonic channel: Decompose symbol probabilities dyadically to form the homophones.
3. Homophonic coding: Map each symbol to be encoded into one of its associated homophones, chosen at random.
4. Arithmetic coding: Encode each selected homophone with the shift-and-add arithmetic coding algorithm into a sequence of bits.

To demonstrate the operation of the homophonic coding algorithm, a file consisting of about 225 000 ASCII characters (7-bit) was encoded. No source modelling was employed, and source statistics were determined by simply counting the frequency of occurrence of each character. The validity of this approach is based on the assumption that the source statistics remain fairly constant throughout the entire file. The results in Fig. 5 show that this is approximately true. For comparison the estimated source entropy of is also shown.

Table 5. Results of statistical tests.

Probabilities	$P(0)$	$P(1)$	$P(00)$	$P(01)$	$P(10)$	$P(11)$
Uncoded Data	0.562	0.438	0.2780	0.3150	0.2519	0.1551
Homophonic Coding	0.499	0.501	0.2495	0.2500	0.2498	0.2507

Note that the per-symbol entropy after homophonic coding is increased by less than 2 bits, according to theory. The resulting encoded file is smaller than the original file, which implies that source coding (data compression) has taken place. It is instructive to ascertain whether the binary output of the arithmetic coder is indeed uniformly distributed. Table 5 shows the probability of occurrence of single bits and pairs of bits. The improvement in distribution of the coded data over the uncoded data is immediately evident.

6 Homophonic Coding Versus Source Coding

In [2] Boyd investigated three source coding (data compression) schemes, viz. Huffman coding, Lempel-Ziv coding and arithmetic coding, and calculated the

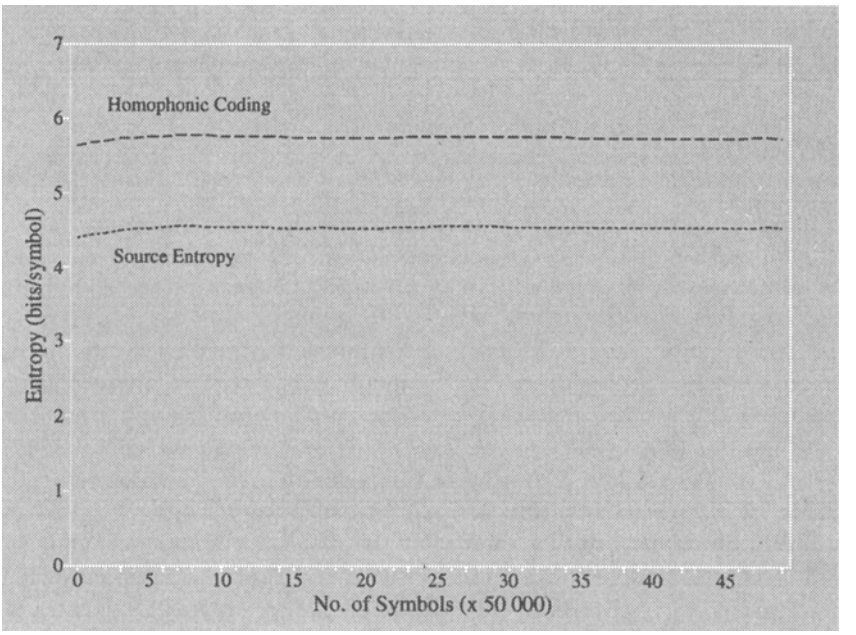

Fig. 5. Simulation results for homophonic coding of ASCII text file.

enhancement in security provided by these three techniques. On the basis of theoretical results and published information he estimated how far these source coding schemes can increase the unicity distance of a cryptosystem. Furthermore, he also posed the question whether homophonic coding offers any advantage above source coding, since it does not increase the value of the unicity distance beyond that provided by source coding.

To answer this question, consider again Fig. 2, and suppose that one of the data compression schemes investigated by Boyd is used in place of the Homophonic Coder. Depending on the accuracy of source modelling, the sequence $X_1, X_2, \ldots$ which is input to the Encrypter will resemble a completely random sequence, and therefore the resulting ciphertext $Y_1, Y_2, \ldots$ will also be random. Essentially the same argument applies when the source coder is replaced by a homophonic coder. Hence, source compression and homophonic coding both provide protection against a *ciphertext-only* attack.

However, in the case of a *known plaintext attack*, or a *chosen plaintext attack*, source coding provides no protection at all, since there exists a *deterministic mapping* between the message source symbols $U_1, U_2, \ldots$ and the output $X_1, X_2, \ldots$ of the source coder. Homophonic coding, on the other hand, produces a *probabilistic mapping* between the source symbols and the output $X_1, X_2, \ldots$.

Therefore, if an encryption system merely needs to be protected against

ciphertext-only attacks, it is sufficient to apply source coding. However, if the system is to be protected against known/chosen plaintext attacks as well, additional protection is required which is provided by homophonic coding.

7 Conclusion

This paper introduces a practical homophonic coding algorithm based on arithmetic coding. Homophonic coding provides protection against ciphertext-only attacks as well as chosen/known-plaintext attacks, whereas source coding merely protects against ciphertext-only attacks. It is shown that an optimum homophonic coder can be designed for any given message source by dyadic decomposition of the source probabilities into distinct negative powers of two. The dyadic decomposition leads to a particularly simple implementation of arithmetic coding, termed the *shift-and-add algorithm*, which requires no multiplications, only shifts and additions. The operation of this algorithm was demonstrated by the encoding of an ASCII text file, making use of a very simple form of source estimation. Simulation results show that the resulting binary output is indeed uniformly distributed. Furthermore, it was found that the shift-and-add algorithm is about 25% faster than the arithmetic coding algorithm given in [1].

Acknowledgement
The author would like to thank Professor G J Kühn from the University of Pretoria for helpful discussions and comments in the preparation of this paper.

References

1. T C Bell, J G Cleary and I H Witten, *Text Compression.* Prentice Hall, 1990.
2. C Boyd, "Enhancing secrecy by data compression: theoretical and practical aspects", *Advances in Cryptology – Eurocrypt '91*, LNCS no. 547, Springer-Verlag, pp. 266-280, 1991.
3. T M Cover and J A Thomas, *Elements of Information Theory.* Wiley, New York, 1991.
4. C Günther, "A universal algorithm for homophonic coding", *Advances in Cryptology — Eurocrypt '88*, LNCS no. 330, Springer-Verlag, pp. 405-41, 1988.
5. H N Jendal, Y J B Kuhn and J L Massey, "An information-theoretic treatment of homophonic substitution", *Advances in Cryptology — Eurocrypt '89*, LNCS no. 434, Springer-Verlag, pp. 382-394, 1990.
6. D Kahn, *The Codebreakers: The Story of Secret Writing.* Weidenfeld and Nicolson, London, 1967.
7. G Langdon, "An introduction to arithmetic coding", *IBM J. Res. Develop.*, vol. 28, no. 2, pp. 135-149, March 1984.
8. G Langdon and J Rissanen, "Compression of black-white images with arithmetic coding", *IEEE Trans. Commun.*, vol. COM-29, pp. 858-867, June 1981.
9. J L Massey, "On probabilistic encipherment", *1987 IEEE Information Theory Workshop*, Bellagio, Italy.

10. J L Massey, "Some applications of source coding in cryptography", *European Transactions on Telecommunications*, vol. 5, no. 4, pp. 7/421-15/429, July-August 1994.
11. J Rissanen and G Langdon, "Arithmetic coding", *IBM J. Res. Develop.*, vol. 23, no. 2, pp. 149-162, March 1979.
12. C E Shannon, "Communication theory of secrecy systems", *Bell Syst. Tech. J.*, vol. 28, pp. 656-715, Oct. 1949.