Integrated Encryption in Dynamic Arithmetic Compression[*]

Shmuel T. Klein[a], Dana Shapira[b]

[a]*Department of Computer Science, Bar Ilan University, Ramat Gan, Israel*
tomi@cs.biu.ac.il

[b]*Computer Science Department, Ariel University, Israel*
shapird@g.ariel.ac.il

Abstract

A variant of adaptive arithmetic coding is proposed, adding cryptographic features to this classical compression method. The idea is to perform the updates of the frequency tables for characters of the underlying alphabet selectively, according to some randomly chosen secret key K. We give empirical evidence that with reasonably chosen parameters, the compression performance is not hurt, and discuss also aspects of how to improve the security of the system being used as an encryption method. To keep the paper self-contained, we add a short description of the arithmetic coding algorithm that is necessary to understand the details of the new suggested method.

Keywords: Adaptive data compression, arithmetic coding, cryptography

1. Introduction and previous work

The system suggested herein lies at the crossroads of two disciplines dealing both with the encoding of data, yet with different objectives. Data *compression* focuses on representing its input in as few bits as possible, while data *encryption* concentrates on letting two parties securely exchange some information, by protecting it from being understood by a third, possibly

[*]This is an extended version of a paper that has been presented at the 11th International Conference on Language and Automata Theory and Applications (LATA'17) in 2017, and appeared in its Proceedings [14].

malicious, eavesdropper. By combining compression and encryption techniques, one may address some of the main concerns of communication over a network, namely security, space savings of the transformed information, and processing speed.

Although the objectives and methods are different, the goals of both compression and encryption are achieved by removing redundancies. We refer to a system combining the two disciplines as a *compression cryptosystem*, and suggest basing such a system on *adaptive arithmetic coding*. This yields, on the one hand, an increased data transfer rate in a communication system, by generating a data file of reduced size. On the other hand, since the exact model on which the encoding is based is only known to the communicating parties, the system seems to provide also privacy, as detailed below.

A major purpose of an adversary could be, given a specific *cipertext*, to reveal the corresponding *cleartext*. A more ambitious goal could be *break the code*, that is, trying to reconstruct the *secret key* K, which would enable the adversary not just to decode some specific ciphertext, but in fact all those that have been encoded using the same key K. Breaking the code is indeed possible for coding schemes like static Huffman coding, that use a fixed set of codewords to represent the sequence of symbols in some given input file, see [1] for a simple chosen plaintext attack. For Huffman and similar codes, the secret key is the correspondence between codewords and the encoded elements, characters or words. Fraenkel and Klein [6] suggest methods for increasing the cryptographic security of variable length prefix free codes, and Huffman codes in particular. Their methods are based on the NP-completeness of various decoding problems, such that there is probably no polynomial algorithm for breaking the code.

The ability of decoding when there is only partial knowledge, or equivalently, guessing some codewords in the ciphertext substituting for given characters of the plaintext, is related to the robustness of the given code against errors. Huffman coding schemes are able to cope with communication errors such as bit losses and changes: even if following such an error, the actually decoded text differs from the original encoded one, synchronization is generally regained after a small number of erroneous codewords [13]. Arithmetic coding schemes, however, are rendered inoperable at the loss of a single bit [3].

The fact that arithmetic coding is more sensitive to transmission errors than Huffman coding is a disadvantage as far as the choice of a compression scheme is concerned, but it turns into an advantage when one considers also the cryptographic security of the encoding. Indeed, small fluctuations in the probabilities will often not have enough of an impact to change the set of optimal codewords. Longo and Galasso [15] define a pseudometric on the set of probability distributions over a finite alphabet and derive an upper bound on the distance from any probability distribution to the dyadic distribution yielding the same Huffman tree. A dyadic distribution is one in which all the probabilities are powers of $\frac{1}{2}$. On the other hand, small fluctuations will have a cumulative effect in arithmetic coding, and the dependence of the encoded message upon all the previously transmitted characters ensures that even a minor discrepancy between the model actually used for encoding and that assumed for the decoding will, in the long run, produce nonsensical output.

Simultaneous compression and encryption can be achieved by either embedding compression into encryption algorithms as in [25], or by adding cryptographic features into compression schemes, as we suggest here. The combination of arithmetic coding with data security was already suggested long ago by Jones [9] and Witten and Cleary [23]. Jones' implementation uses fixed source symbol probabilities but is flexible in the choice of source and code alphabets. Bergen and Hogan [1] investigate the security provided by *static* arithmetic coding, and show how the attacker can determine both the ordering of the symbols in the cumulative frequency table, and the actual value of the symbol frequencies, by feeding repeated binary substrings as the input to the algorithm. In [2] they extend their work to a cryptosystem based on *adaptive* arithmetic coding.

Witten and Cleary [23] refer to the model as a very large key, without which decryption is impossible, and claim that an adaptive scheme provides protection of messages from a casual observer and against chosen plaintext attacks.

In [21], a key controls the interval splitting of arithmetic coding, but the scheme is vulnerable by known plaintext attacks [8] based on the fact that the same key is used to encode many messages. Katti and Vosoughi [10] show that even an improved version that uses different keys for encrypting different messages is still insecure under ciphertext-only attacks. Randomized arithmetic coding was proposed in [7], which is inefficient in terms of compression when compared to the traditional "compress-then-encrypt" approach. An example of the latter is Singh and Gilhotra [19], who suggest private key encryption based on static arithmetic coding. Utilizing chaotic systems for arithmetic coding was suggested in [24], where the secret key controls both the position and the direction of the line segments in the piecewise linear chaotic map. Klein et al. [11] suggest several heuristics for using compression as a data encryption method in order to prevent illegal use of copyright material.

A cryptosystem is used in [16] to provide security for mobile SMS communication: the SMS are first compressed and then encrypted using RSA. A similar approach is taken in [18]. A compression cryptosystem especially suited for the secure transmission of medical information such as images, audio, video etc. is proposed in [17], but it is only suitable for short enough messages, as efficiency drops when the length of the message increases. In addition, the system requires a very large public key which makes it very difficult to use in several practical applications.

Cleary et al. [4] consider static arithmetic coding with a binary alphabet and show that for a chosen plaintext attack, $w + 2$ symbols are sufficient to uniquely determine a w-bit probability. Obviously, they deal with a very simplified version of arithmetic coding, while in the method we suggest herein, many more parts of the system remain unknown, besides the single probabilities. Duan et al. [5] propose a dynamic arithmetic coding method based on a Markov model for joint encryption and compression. The ith symbol is encoded based on a Markov chain of order 0 or 1, possibly permuting the relevant conditional probabilities in the model depending on a given secret key. Similarly to their scheme, we also update the model based on a secret key, but we ignore the dependency between consecutive characters.

Although simultaneous arithmetic coding and encryption has already been studied, most of the suggested schemes are found insecure and especially inefficient in terms of compression. Unlike previous research, preliminary empirical results suggest that our proposed algorithm provides security without hurting the compression efficiency. Section 2 describes the proposed method, Section 3 considers possible attacks and suggests appropriate remedies and Section 4 reports the experiments we have performed.

2. Proposed Method

We consider a cryptosystem which we imagine as being superimposed upon an adaptive arithmetic coder. Given is a plaintext $T = t_1 t_2 \cdots t_n$ of length n elements, drawn from an alphabet $\Sigma = \{\sigma_0, \ldots, \sigma_{s-1}\}$ of fixed size s. For simplicity, we shall refer to the elements of Σ as characters, but they may as well be words or more general elements into which the plaintext could be partitioned. The text will be encoded using adaptive arithmetic coding, producing a compressed text $B = b_1 b_2 \cdots b_m$ of length m bits.

Arithmetic coding represents a message to be encoded by a sub-interval $[low, high)$ of $[0, 1)$. For an initially empty string T, the algorithm starts with the basic interval $[0, 1)$, which is increasingly narrowed as more characters from T are processed. The narrowing procedure is based on partitioning the current interval into sub-segments according to the probabilities of the characters in Σ.

For example, if $\Sigma = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$ and the probabilities are 0.2, 0.7 and 0.1, respectively, a possible partition could be into the segments $[0, 0.2)$, $[0.2, 0.9)$ and $[0.9, 1)$. If the first character of T is b, the current interval after processing b is $[0.2, 0.9)$. In subsequent steps, the same procedure is applied after appropriate scaling. So if the second character of T is a, to which the first 20% of the initial interval have been assigned, the current interval after processing ba will be $[0.2, 0.34)$, the first 20% of the current interval $[0.2, 0.9)$. After processing 3 characters bac, the interval would be $[0.326, 0.34)$. Figure 1 is a schematic view of the repeated partition process for this example.

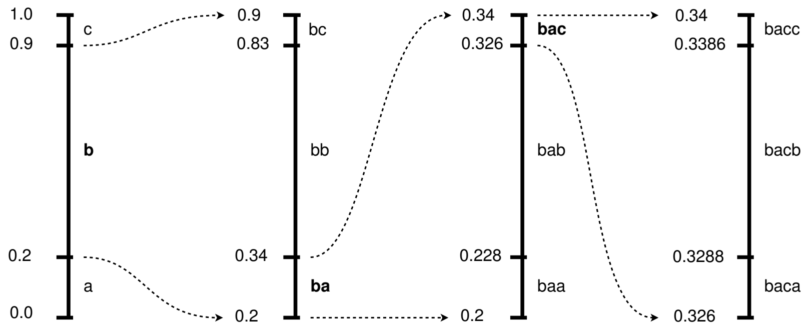


FIGURE 1: *Example of arithmetic coding.*

The longer the text T, the narrower will the corresponding interval be, thus the more bits will usually be needed to represent any real number in it. To save space and since there will often be some overlap between the representations of the real numbers *low* and *high* (in the last example, they both start with 0.3), the final encoding will not be the resulting interval itself, but rather a single real number within it. This suffices to allow decoding by reversing the above procedure, if some external stopping condition is added, like transmitting an end-of-text character or the length of the text. For the above example, one could choose, say, 0.33, or, even better, 0.328125, which is the number with the shortest binary representation, 0.010101_2, in the interval $[0.326, 0.34) = [0.01010011011101\cdots_2, 0.01010111000010\cdots_2)$, where the subscript $_2$ indicates that the fractional number is given in binary. Opting for the minimal number of necessary bits seems natural in a data compression application, but the truth is that the savings are generally just of a small number of bits; for large enough files, this is often negligible, so one could as well choose the number within the final interval at random, which adds some security as it puts an additional burden on a potential adversary.

In the static variant of arithmetic coding just described, the partition of $[0, 1)$ into sub-segments is fixed throughout the process. A dynamic variant calls for updating these segments adaptively, according to the probability distribution of the alphabet within the prefix of the text that has already been processed. In fact, the general step of the dynamic variant consists of two independent actions:

1. compute the new interval as a function of the current one, the current character and the currently assumed distribution of probabilities;
2. update the model by incrementing the frequency of the current character and adjusting the relative sizes of all the intervals in the partition accordingly.

The encryption we suggest is based on the fact that the model updates of the second action above are done selectively, not necessarily at every step. The exact subset of the processing steps at which the model is altered is controlled by a secret key K. A similar idea of selectively updating the model has been suggested in [12], albeit for enhancing processing time rather than improving cryptographic strength.

Specifically, for the current encoding, if $K = k_0k_1 \cdots k_{r-1}$ is the standard binary representation of the key K, where its length r is chosen large enough, say, $r = 512$ or more, then the model will be updated at step i, that is, after encoding the ith character, for $i \geq 1$, if and only if $k_{(i-1) \bmod r} = 1$. The algorithm for encrypting a given message T according to a secret key K is presented in Algorithm 1.

The encoding model is represented here by means of a partition of the interval $[0, 1)$ into sub-intervals $[\ell_j, h_j)$ corresponding, respectively, to characters σ_j, for $0 \leq j < s$. In fact it would suffice to keep only the lower or only the higher bounds of the intervals, since $\ell_0 = 0$, $h_j = \ell_{j+1}$ for $0 \leq j \leq s - 2$ and $h_{s-1} = 1$, but the algorithms is easier to understand when both ℓ_j and h_j are used. A convenient initialization would be to assume a uniform distribution, for example of the $s = 256$ 8-bit characters of the extended ASCII set. The corresponding intervals would then be $[\ell_j, h_j) = \left[\frac{j}{256}, \frac{j+1}{256}\right)$ for $0 \leq j < 256$, which is what we have used in our experiments.

Updating the model consists of: incrementing the frequency of the currently read character t_i; adjusting the cumulative frequencies of the elements

```
encode($T, K$)
1      $[low, high) \longleftarrow [0,1)$
2      initialize the interval partition distribution $\{[\ell_0, h_0), \ldots, [\ell_{s-1}, h_{s-1})\}$
3      for $i \longleftarrow 1$ to $n$
3.1        $range \longleftarrow high - low$
3.2        $high \longleftarrow low + range \cdot h_{t_i}$
3.3        $low \longleftarrow low + range \cdot \ell_{t_i}$
3.4        if $k_{(i-1) \bmod r} = 1$ then
3.4.1          update $\{[\ell_0, h_0), \ldots, [\ell_{s-1}, h_{s-1})\}$
4      return some value in the current interval
```

ALGORITHM 1: *Cryptosystem based on dynamic arithmetic encoding.*

σ_p for $p \geq j$, where j is the index within the alphabet Σ of the current character in the text, that is, $t_i = \sigma_j$; and finally adjusting accordingly the probabilities by calculating the relative cumulative frequencies for all the characters.

ALGORITHM 1 is given here in a simplified form, to convey its main idea. We shall refine it, correcting potential flaws, in the following section dealing with possible attacks.

3. Attacks

3.1. Cryptographic attack 1: guessing the repeatedly used key

One of the parameters of the system is the length r of the secret key. Ideally, r should be as large as possible, and if one could choose $r = 8n$, that is, a key as long as the text in bits, the key could be used as a one-time-pad, providing absolute security. In practice, of course, the key K is limited, and using it cyclically, as suggested in ALGORITHM 1, might be risky, if the adversary finds some way, by a chosen plaintext or other attack, to exploit the repetitive pattern in order to reveal the key itself.

We suggest to avoid this possible weakness of the system by considering K as the seed of some random number generator. The secret to be kept is then only of limited length, as in the initial setting, but the actually used key can be generated to be of unbounded length. For example, use a large constant randomly chosen prime P and a large primitive root of a modulo

P; choose a new K at each iteration by

$$K \leftarrow a^K \bmod P.$$

Even if the prime P is publicly known, and even if an enemy gains somehow knowledge of the value of K at some stage, it might not be possible to deduce the following value of K without knowing the value of a that has been kept secret. For going backwards, assume that the adversary succeeds somehow to reveal the values of a and P after having processed a part of the file. The part from that point B on is then not secured any more, but having broken the code and found the values of a and P does not help to decipher the text preceding B, because it means calculating the discrete log, for which no efficient algorithm is known.

The conclusion is that replacing a fixed key K as suggested in Algorithm 1 by a sequence of generated keys hardly incurs any penalty in compression performance or any addition in encoding or decoding time, but no attack based on the knowledge that a fixed key is used time and again will be possible, since every block of r characters uses a different key.

3.2. Cryptographic attack 2: guessing the key iteratively by chosen plaintext

An adversary might try to reveal the secret key incrementally, bit per bit, taking advantage of one of the weak points of the encoding, its initialization. Indeed, the initial partition of the interval is a parameter of the arithmetic encoding, and as such is known not only to the communicating parties.

The following chosen cleartext attack could be used. Assume, for simplicity, a binary alphabet $\Sigma = \{\mathsf{a}, \mathsf{b}\}$, and that we start with uniform probabilities, so that the corresponding intervals are $[0, \frac{1}{2})$ and $[\frac{1}{2}, 1)$. More precisely, we suppose that we start with fictitious initial frequencies of 1 for both characters, to avoid the zero-frequency problem [22]. Choose as plaintext the text consisting only of the two characters ab. If the first bit of the key is $k_0 = 1$, then the model is updated after having read the first character, so the new frequencies are 2 and 1, and the corresponding intervals are $[0, \frac{2}{3})$ and $[\frac{2}{3}, 1)$. If $k_0 = 0$, the intervals are not changed.

It now depends on which number x is returned in the final interval $[low, high)$. If low is chosen as x, the the system returns $\frac{2}{3}$ if $k_0 = 1$ and $\frac{1}{2}$ if $k_0 = 0$. If the number with shortest binary representation is chosen as x, then x is $0.11_2 = \frac{3}{4}$ in the former case, and $0.1_2 = \frac{1}{2}$ in the latter. The attacker can thus deduce the value of the first bit of the secret key from the ciphertext of the chosen plaintext. If the algorithm returns an arbitrarily chosen value within the final interval, the attacker has a probabilistic procedure to decide the value of k_0: repeat the above test several times; if there is at least one of the returned values within the interval $[\frac{1}{2}, \frac{2}{3})$, then k_0 must be 1, otherwise decide that $k_0 = 0$, though there is a small probability of being wrong.

Once the first bit is known, the attack can be repeated similarly to reveal the second bit, then the third, etc.

To avoid this flaw, the plaintext could be preceded by some known text of a predefined size, say, the thousand first characters of the Bible or some other well known text like *Moby Dick*. This prefix is then used just to tune the encryption and decryption processes to a safe initialization of the boundaries of the intervals. The real encoding only starts when the original plaintext is processed. The rationale behind prepending a long enough known text, is that we expect that after updating the model selectively several times according to a secret key, the chances for the decryption model assumed by the attacker to be perfectly synchronized with the actual encryption model, without the knowledge of the secret key, are negligible, despite the fact that the characters are also known to the adversary. Thus, decoding errors will eventually appear, which in turn will initiate a snowball effect, as sporadic mistakes at the beginning will trigger even more errors subsequently, and the cumulative impact in the long run may destroy any similarity to the original text. The alternative of guessing a random key with 2^{512} potential variants is obviously ruled out.

We thus expect not to hurt the compression efficiency on the one hand, yet to provide strong enough encryption on the other hand in the long run. Indeed, dynamic arithmetic coding in particular, and all adaptive compression methods in general, are based on the assumption that the distribution of the characters in the text starting from the current position onwards will be similar to the distribution in the part of the text preceding this current position. This leads to the intuition that a longer history window will always be preferable to a shorter one. This will, however, not always be the case. It might well happen, in particular for non-homogeneous texts, that basing the prediction of the character probabilities on a random subset of n of the $2n$ most recently read characters may yield a compression performance that is not inferior, and sometimes even better, than using just the last n characters as basis.

It is easy to construct a worst case example in which any deviation from the standard model of considering the full history (or in case of limited memory, a bounded size window with a suffix of the history), will give deteriorated compression performance. Consider, for instance, a text of the form $\mathsf{abababab}\cdots$, consisting of strictly alternating characters of a binary alphabet. An adaptive arithmetic encoder will produce a uniform probability distribution $(\frac{1}{2}, \frac{1}{2})$ for any standard (even) sized history window. However, if the model is based on choosing n of the $2n$ last seen characters, according to some secret key K, then the uniform distribution will be obtained only if exactly half of the chosen characters are a and half are b. Assuming the values of K are randomly chosen, the probability of this event is

$$\frac{\binom{n}{n/2}\binom{n}{n/2}}{\binom{2n}{n}} \simeq \frac{2}{\sqrt{\pi\, n}},$$

where we have used Stirling's approximation. Thus for $k = 512$ corresponding to $n = 256$, only in 7% of cases we will end up with an exactly uniform model. Nevertheless, this worst case behavior is restricted to such artificial texts, and our empirical tests of several real language texts suggest that the loss incurred by turning to a selective updating procedure as suggested is hardly noticeable.

3.3. Cryptographic attack 3: guessing the boundaries of intervals

There might be other possible attacks that could render the method vulnerable. In the following experiment we wanted to check the ability of the adversary to predict the partition of [0,1) into intervals according to the probabilities of the characters, without knowledge of the key. This is important, because in a chosen cleartext attack, an adversary would know at each stage the true distribution of the characters in the already processed prefix of the text, which could possibly help to infer from it the stages at which the model has been updated, and thereby guess the secret key on which the encryption is based. We are therefore interested in measuring, at each stage, the size of the overlapping parts of the subintervals of all characters, between two possible partitions: the one induced by the entire history processed so far, and the one corresponding to the subset of updating steps according to the secret key.

Figure 2 illustrates what we mean by overlap. The interval $[0, 1)$ is shown, partitioned into 5 segments, corresponding to an alphabet $\{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{e}\}$ of 5 characters, but with slightly differing boundaries in the upper and lower parts of the figure. Overlapping parts of segments assigned to the same character are emphasized. Our measure for the overall overlap is the cumulative size of the boldfaced sub-intervals, 0.714 in this example.

The boldfaced upper plot of Figure 3 shows this overlap for the partitions of $[0, 1)$ into sub-intervals according to:

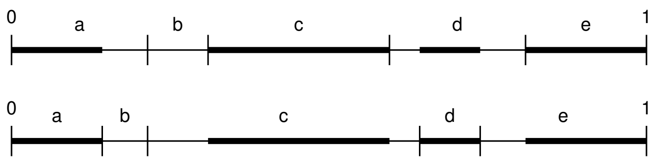


FIGURE 2: *Overlapping intervals.*

1. a model updating the partition after each character read;
2. a model updating the partition selectively, according to some secret key.

The test text for this figure is the beginning of the Bible (King James version) in English, in which the text was stripped of all punctuation signs, which we have used to initiate the frequency table prior to the processing of the cleartext itself. As can be seen, the overlap is large at the beginning, but quickly drops to a level of about 8%, which is reached after roughly 100 characters. Then the overlap slowly rises until reaching a value of about 0.83 for 10000 processed characters. The overlap in the upper plot of Figure 3 will eventually approach 1, as both distributions tend, after processing a large enough number of characters, to the actual distribution of the text. Indeed, the initial prefixes of the text are too short to be representative samples of the general text; but as more characters are accumulated, even the distribution within a sub-sample will be increasingly similar to that of the entire set. A similar phenomenon can be observed when considering several independent sources describing the distribution of the characters in "standard" English. Almost all will agree that the order of the first few characters will be E, T, A, O, I, N, and will give quite similar values for their probabilities.

Returning to the suggested compression cryptosystem, the overlap after 1000 characters is roughly around 0.3, which means that an adversary guessing the partition, or basing it on the full distribution of all the characters in the prefixed Bible sample, will have, for each processed character, a chance of about 0.7 for being wrong. The probability for correct guesses in all of the, say, 10 first attempts is thus less than 0.000006, and a single wrong guess will imply many more subsequently.

3.4. Cryptographic attack 4: guessing the order of the characters

An additional parameter of the system is the order of the characters in the partition. While for Huffman coding, characters have first to be

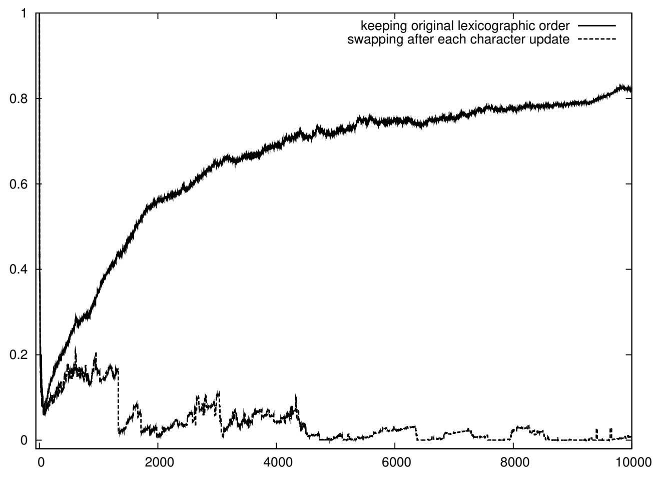


FIGURE 3: *Size of overlap as a function of the number of processed characters.*

ordered by their frequencies, this is not the case for arithmetic coding. The partition of the interval $[0, 1)$ into sub-intervals can be done using any order of the elements, and it is often convenient to adopt a lexicographic ordering of the characters, rather than sorted by probabilities, as we have done in the example of Figure 1. If a well determined and publicly known order is believed to endanger the security, as suggested in [1], a different order could be used, which has the advantage of putting the additional burden on the adversary to guess not only the exact sizes of the intervals, but also their order.

Ideally, the sequence of elements of Σ should be reshuffled after each processed character, and the exact details of how to perform this action should be controlled by the secret key K. This would obviously not change the compression efficiency, but it would be very costly in execution time, for both encoding and decoding. We therefore suggest the following strategy which is much faster to implement, yet seems to be just as secure in the long run.

Denote by π the permutation giving the current order of the characters, that is, the alphabet vector $\Sigma = \Sigma[0], \Sigma[1], \ldots, \Sigma[s-1]$ contains the elements $\sigma_{\pi(0)}, \sigma_{\pi(1)}, \ldots, \sigma_{\pi(s-1)}$, respectively. We start with the identity permutation, $\pi(i) = i$, corresponding to lexicographic order. After each processed character, we suggest to swap it with its neighbor in the current order. More precisely, suppose the current character is σ_j for some $0 \leq j < s$; at this stage, σ_j is stored at index $\pi^{-1}(j)$ in vector Σ. The suggestion is to swap the contents in Σ of the neighboring elements with indices $\pi^{-1}(j)$ and $(\pi^{-1}(j)+1) \bmod s$. Figure 4 is a small example: assume the last seen character is w, whose ASCII value is 119 and that it is currently stored at position 73 in the table Σ. The figure shows a part of Σ before and after the swap.

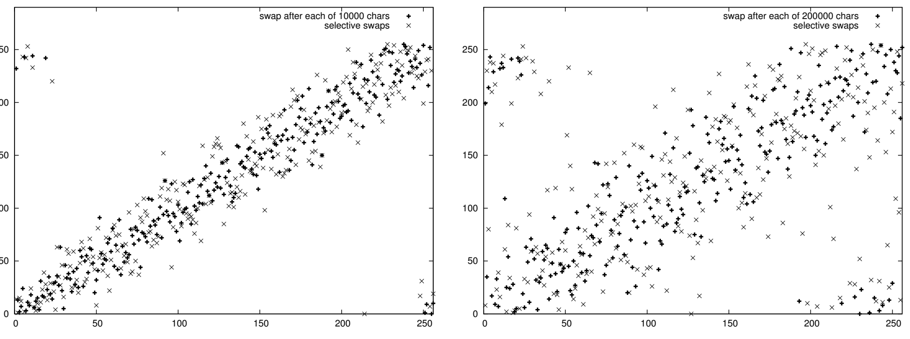
index	···	72	73	74	75	76	···
before swap	···	17	119	56	201	178	···
after swap	···	17	56	119	201	178	···

FIGURE 4: *Example of swapping elements.*

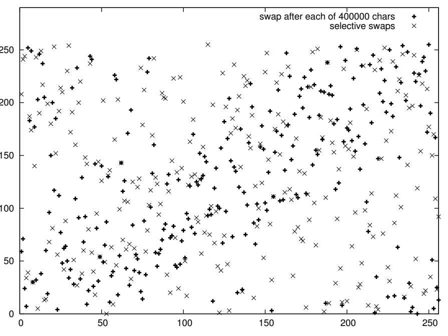
Of course, this swapping strategy might be known to the enemy, so the idea is, once again, to perform the swaps selectively, only at steps indexed i for which $k_{(i+1) \bmod r} = 1$. The effect of such a swap on the interval partition is that all the boundaries remain in place, except the one separating the adjacent intervals that have been interchanged (unless the last interval is swapped with the first one, and then two boundaries are moved, or in case the intervals are equal in size, and then no boundary is moved). The update process is thus significantly faster than if all the elements had to be permuted.

On the other hand, using just a small number of such restricted permutations cannot possibly lead to a uniform spread of the $s!$ different permutations of the alphabet, but after a large enough number of iterations, the distribution should approach uniformity. Figure 5 is an illustrative example visually representing the permutations as dots in an $s \times s$ matrix. The test file is the gzip compressed form of the English Bible mentioned above. The bold faced **+** signs represent the elements of the permutation obtained by swapping in each iteration, the $\times$ signs correspond to the elements of the permutations performed selectively according to the 1-bits of the secret key K. We see that after 10,000 iterations, the elements still seem to be concentrated close to the main diagonal, reflecting the fact that elements have only been moved locally. After 200,000 and 400,000 iterations, the obtained permutations give a less clustered impression. In the three plots, the permutations for full and selective updates are completely different.

Returning to the number of overlaps, the lower plot of Figure 3 depicts this number after the swapping strategy has been applied. We see that while without swapping, the size of the overlap is increasing after some initial phase, and tends to 1, the effect of permuting the alphabet keeps the overlap at the level of a few percent in the long run. We conclude that this



(a) after 10,000 iterations (b) after 200,000 iterations



(c) after 400,000 iterations

FIGURE 5: *Permutations of the lexicographically ordered alphabet after swaps.*

simple strategy may significantly improve the security, at almost no cost.

4. Empirical Results

We considered four texts of different languages and sizes, each encoded as a sequence of characters. *ebib* is the version of the English Bible, mentioned already above; *ftxt* is the French version of the European Union's JOC corpus, a collection of pairs of questions and answers on various topics used in the ARCADE evaluation project [20]; *sources* is formed by C/Java source codes obtained by concatenating .c, .h and .java files of the linux-2.6.11.6 distributions; and *English* is the concatenation of English text files selected from the etext02 to etext05 collections of the Gutenberg Project, from which the headers related to the project were deleted so as to leave just the real text.

4.1. Compression performance

We have assumed that basing the prediction of the character probabilities on a random subset of bits would not hurt the compression performance. We therefore compared our method to traditional dynamic arithmetic coding, which uses the entire portion of the file that has already been processed to predict the current character, and measured their compression performance.

Table 1 presents information on the compression performance of the data files involved. The second column presents the original file size in MB. The third column gives the size of the file, in MB, compressed by adaptive arithmetic coding without any key, that is, using the full history window. The fourth column shows the difference in size, in bytes, of the compressed file, when the updates of the model are done according to a randomly chosen key as suggested. Interestingly, there was a loss, albeit a negligibly small one, in all our tests. The last column gives the ratio of the loss to the size of the file.

File	full size MB	compressed size MB	absolute loss bytes	relative loss
ebib	3.5	1.8	56	3×10^{-5}
ftxt	7.6	4.2	316	7×10^{-5}
sources	200.0	136.6	436	3×10^{-6}
English	1024.0	579.3	437	7×10^{-7}

TABLE 1: *Information about the used datasets*

As can be seen, there is hardly any noticeable difference in size between the files obtained by using the standard adaptive arithmetic code, and that with the selective updating. Timing results for both encoding and decoding are also almost identical for both variants, with, typically, a slight advantage of the selective version, due to the time savings for not updating the model at every step. Table 2 presents the processing times for both compression and decompression, for the standard and selective methods. For each of the test files the encoding and decoding times were averaged over 10 runs. The displayed times are the time averages per MB, given in milliseconds.

File	Compression		Decompression	
	standard	selective	standard	selective
ebib	146	138	511	497
ftxt	141	140	448	446
sources	166	162	389	387
English	154	143	466	462

TABLE 2: *Processing Times in* ms

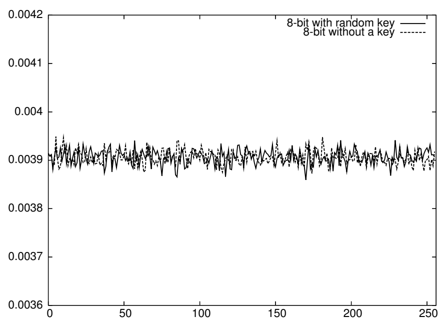
4.2. Uniformity

As to security, any reasonably compressed or encrypted file should consist of a sequence of bits that is not distinguishable from a randomly generated binary sequence. A criterion for such randomness could be that the probability of occurrence, within the compressed file, of any substring of length m bits should be 2^{-m}, for all $m \geq 1$, that is, the probability for 1 or 0 are both 0.5, the probabilities for 00, 01, 10 and 11 are 0.25, etc. We checked this fact on all our compressed test files, for all values of m up to 8, and found distributions that are very close to uniform, with small fluctuations, for both methods of the original arithmetic coding and that with our selective updates. We bring here only the data for the file *ebib*; the results for the other files were practically identical.

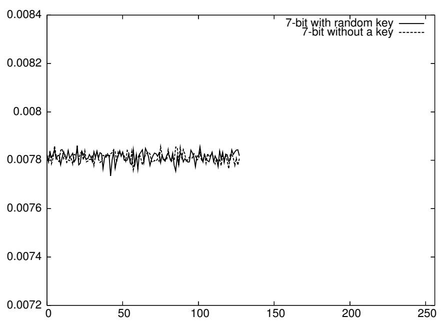
The left part of Figure 6 plots the probability of occurrence of 8-bit strings as a function of their possible values 0 to 255. As expected, the probabilities fluctuate within a narrow interval centered at $\frac{1}{256} = 0.0039$. All the possible bit-positions have been taken into account, so the strings were not necessarily byte aligned. The solid line corresponds to the method with a secret key suggested herein, whereas the broken line is the distribution for the original arithmetic coding. As can be seen, these distributions can be equally considered as uniform, and there seems to be no obvious deterioration of the uniformity due to the selective choice of the update steps.

The right part of Figure 6 repeats the test with 7-bit strings. Here the values are in the range $[0, 127]$, and the probabilities fluctuate around $\frac{1}{128} = 0.0078$. Similar graphs would be produced for the other values of m. For $m \leq 3$, the probabilities are shown in Table 3.

To get a more quantitative judgment of the intuitive impression that these values are evenly spread, we calculated the standard deviation of the distribution of the 2^m values for each value of m. Usually, the standard



Distribution for 8-bit



Distribution for 7-bit

FIGURE 6: *Probability of occurrence of 8- and 7-bit substrings as function of their value.*

deviation σ is of the order of magnitude of the average μ, so their ratio $\frac{\sigma}{\mu}$ may serve as a measure of the skewness of the distribution. Table 4 gives this ratio for $1 \leq m \leq 8$, for both the standard adaptive arithmetic coding with model updates after every encoded character, and for the selective model we proposed. As can be seen, the ratio is very small in all cases, of the order of $\frac{1}{1000}$, suggesting that the distributions are indeed very close to uniform.

4.3. Sensitivity

In the last test, we checked the sensitivity of the system to variations in the secret key. The measure of similarity between two image files proposed in [5] is the normalized number of differing pixels. Since we consider encrypted files which are not images, we shall use the normalized Hamming distance: let $A = a_1 \cdots a_n$ and $B = b_1 \cdots b_m$ be two bitstrings and assume $n \geq m$. First extend B by zeros so that both strings are of the same length n. The normalized Hamming distance is then defined by $\frac{1}{n}\sum_{i=1}^{n}(a_i \text{ XOR } b_i)$. Figure 7 plots these values for prefixes of size n, for $1 \leq n \leq 1000$ of the file *ebib*: the first plot considers two independently generated random keys, for the second and third plots the same key is used, with just the first or last bit flipped. In any case, one sees that the produced cipherfiles are completely different, with the number of differences in corresponding bits rapidly tending to the expected value $\frac{1}{2}$. The values of the ratio for the three tests at the end of the file (after processing 1.8 MB) were 0.499894, 0.500004 and 0.499995, respectively. We conclude that the suggested selective update procedure of the model is extremely sensitive to even small alterations: all

value	standard arithmetic			selective updates		
	$m=3$	$m=2$	$m=1$	$m=3$	$m=2$	$m=1$
0	0.12503	0.25002	0.50011	0.12507	0.25010	0.500005004
1	0.12498	0.25009	0.49989	0.12503	0.24991	0.499994996
2	0.12510	0.25009		0.12491	0.24991	
3	0.12499	0.24981		0.12499	0.25009	
4	0.12498			0.12503		
5	0.12511			0.12488		
6	0.12499			0.12499		
7	0.12482			0.12499		

Table 3: *Probability of occurrence of 3-, 2- and 1-bit substrings as function of their value.*

m	8	7	6	5	4	3	2	1
standard	0.00383	0.00251	0.00164	0.00125	0.00094	0.00072	0.00053	0.00030
selective	0.00135	0.00042	0.00207	0.00182	0.00059	0.00013	0.00003	0.00001

Table 4: *Ratio $\frac{\sigma}{\mu}$ of standard deviation to average within the set of 2^m values for $m = 1, \ldots, 8$.*

produced files pass the above randomness tests, are practically of the same size, and are completely different from each other.

5. Conclusion

A standard way to devise a compression cryptosystem is to compress the text and only then encrypt the compressed form. The alternative of encrypting first and then compressing would not be effective, as a reasonably encrypted file lacks any easily detectable redundancy and thus would not be compressible at all. We have suggested a way to combine the compression and encryption transformations in a single step, by using selective update steps in the maintained adaptive model. Experiments show that neither the compression performance, nor the running time are hurt. Moreover, the randomness of our ciphertext suggests that decryption is hard without the knowledge of the key.

A main feature of our compression cryptosystem is that the encryption is, in fact, given for free, while compressing the data. The advantage of this

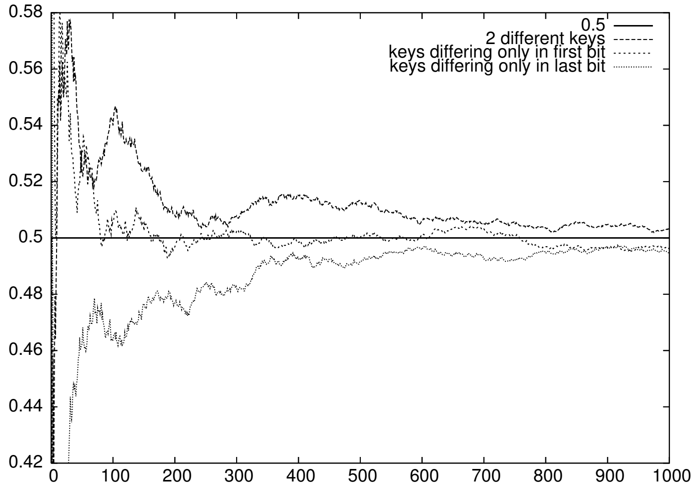


FIGURE 7: *Normalized Hamming distance.*

method is not only saving the time for both encryption and decryption, but also saving the space needed for the intermediate compressed file, which is only needed in the traditional approach. Moreover, storing this compressed, unencrypted, file, increases the risk of leaks of the underlying information. Our method also suggests that this process can be done in a *streaming* manner, without even storing the original file.

Acknowledgement: This work has been supported by grant 4440754669 of the Israeli Ministry of Defense and by the Ariel Cyber Innovation Center in conjunction with the Israel National Cyber directorate in the Prime Minister's Office.

References

[1] Bergen, H. A., & Hogan, J. M. (1992). Data security in a fixed-model arithmetic coding compression algorithm. *Computers & Security*, *11*, 445–461.

[2] Bergen, H. A., & Hogan, J. M. (1993). A chosen plaintext attack on an adaptive arithmetic coding compression algorithm. *Computers & Security*, *12*, 157–167.

[3] Bookstein, A., & Klein, S. T. (1993). Is Huffman coding dead? *Computing*, *50*, 279–296.

[4] Cleary, J. G., Irvine, S. A., & Rinsma-Melchert, I. (1995). On the insecurity of arithmetic coding. *Computers & Security*, *14*, 167–180.

[5] Duan, L., Liao, X., & Xiang, T. (2011). A secure arithmetic coding based on Markov model. *Communications in Nonlinear Science and Numerical Simulation*, *16*, 2554–2562.

[6] Fraenkel, A. S., & Klein, S. T. (1994). Complexity aspects of guessing prefix codes. *Algorithmica*, *12*, 409–419.

[7] Grangetto, M., Magli, E., & Olmo, G. (2006). Multimedia selective encryption by means of randomized arithmetic coding. *IEEE Trans. on Multimedia*, *8*, 905–917.

[8] Jakimoski, G., & Subbalakshmi, K. (2008). Cryptanalysis of some multimedia encryption schemes. *IEEE Trans. on Multimedia*, *10*, 330–338.

[9] Jones, C. B. (1981). An efficient coding system for long source sequences. *IEEE Trans. on Information Theory*, *27*, 280–291.

[10] Katti, R. S., & Vosoughi, A. (2012). On the security of key based interval splitting arithmetic coding with respect to message indistinguishability. *IEEE Trans. on Information Forensics and Security*, *7*, 895–903.

[11] Klein, S. T., Bookstein, A., & Deerwester, S. (1989). Storing text retrieval systems on CD-ROM: Compression and encryption considerations. *ACM Trans. Inf. Syst.*, *7*, 230–245.

[12] Klein, S. T., Opalinsky, E., & Shapira, D. (2019). Selective dynamic compression. In *Data Compression Conference, DCC 2019, Snowbird, UT, USA, March 26-29, 2019* (p. 583).

[13] Klein, S. T., & Shapira, D. (2005). Pattern matching in Huffman encoded texts. *Information Processing & Management*, *41*, 829–841.

[14] Klein, S. T., & Shapira, D. (2017). Integrated encryption in dynamic arithmetic compression. In *Language and Automata Theory and Applications - 11th International Conference, LATA 2017, Umeå, Sweden, March 6-9, 2017, Proceedings* (pp. 143–154).

[15] Longo, G., & Galasso, G. (1982). An application of informational divergence to Huffman codes. *IEEE Trans. Information Theory*, *28*, 36–42.

[16] Mahmoud, T. M., Abdel-Latef, B. A., Ahmed, A. A., & Mahfouz, A. M. (2009). Hybrid compression encryption technique for securing SMS. *International Journal of Computer Science and Security (IJCSS)*, *3*, 473–481.

[17] Ojha, D. B., Sharma, A., Dwivedi, A., Pande, N., & Kumar, A. (2010). Space age approach to transmit medical image with code base cryptosystem over noisy channel. *International Journal of Engineering Science and Technology*, *2*, 7112–7117.

[18] Pizzolante, R., Carpentieri, B., Castiglione, A., Castiglione, A., & Palmieri, F. (2013). Text compression and encryption through smart devices for mobile communication. In *Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, IMIS 2013, Taichung, Taiwan, July 3-5, 2013* (pp. 672–677).

[19] Singh, A., & Gilhotra, R. (2011). Data security using private key encryption system based on arithmetic coding. *International Journal of Network Security & Its Applications (IJNSA)*, *3*, 58–67.

[20] Véronis, J., & Langlais, P. (2000). Evaluation of parallel text alignment systems: The ARCADE project. *Parallel Text Processing*, (pp. 369–388).

[21] Wen, J., Kim, H., & Villasenor, J. (2006). Binary arithmetic coding with key based interval splitting. *IEEE Trans. on Signal Processing Letters*, *13*, 69–72.

[22] Witten, I. H., & Bell, T. C. (2006). The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Trans. Inf. Theor.*, *37*, 1085–1094.

[23] Witten, I. H., & Cleary, J. G. (1988). On the privacy afforded by adaptive text compression. *Comput. Secur.*, *7*, 397–408.

[24] Wong, K., Lin, Q., & Chen, J. (2010). Simultaneous arithmetic coding and encryption using chaotic maps. *IEEE Trans. on Circuits and Systems–II Express Briefs*, *57*, 146–150.

[25] Wong, K., & Yuen, C. H. (2008). Embedding compression in chaos based cryptography. *IEEE Trans. on Circuits and Systems–II Express Briefs*, *55*, 1193–1197.