An Automatic Cryptanalysis of Arabic Transposition Ciphers using Compression

Noor R. Al-Kazaz, William J. Teahan
School of Computer Science, Bangor University,
Dean Street, Bangor, Gwynedd
LL57 1UT, UK

***Abstract*—This paper introduces a compression-based method adapted for the automatic cryptanalysis of Arabic transposition ciphers. More specifically, this paper presents how a Prediction by Partial Matching ('PPM') compression scheme, a method that shows a high level of performance when applied to the different natural language processing tasks, can also be used for the automatic decryption of transposition ciphers for the Arabic language. Another well known compression scheme, Gzip, is also investigated in this paper with less efficient performance demonstrated by this method. In order to achieve readability, two further compression based approaches for space insertion are evaluated as well in this paper. The results of our experiments with 125 Arabic cryptograms of different lengths show that 97% of the cryptograms are successfully decrypted without any errors using PPM compression models. As well in a post-processing step, we can effectively segment the output that is produced by the automatic insertion of spaces resulting with only a few errors overall. As far as we know, this is the first work to demonstrate an effective automatic cryptanalysis for transposition ciphers in Arabic.**

***Keywords*—Automatic cryptanalysis; Arabic transposition ciphers; compression; PPM; word segmentation**

I. Introduction

The Arabic language is one of the most widely spoken languages around the world, with as many as 300 million people in Asia and North Africa speaking Arabic. It is the fifth most spoken language [1]. The old manuscripts that have been recently discovered show that the origin of cryptology is much older than first thought, and Arab contributions to it are much more extensive than previously reported. These important findings are largely based on 8th century and later manuscripts written by the prominent Arab cryptologists such us al-Kindi. The newly discovered manuscripts push back the frontiers of the known history of cryptology by more than 500 years. The Arab scientist al-Kindi, the father of cryptology, is the author of the oldest book on cryptology [2].

Despite the large number of Arabic speakers and with Arabs being the first to use encryption systems as well as having success in breaking them, unfortunately encryption systems and cryptanalysis methods for Arabic are very few in comparison to other languages such as English. The purpose of this paper is to generate an Arabic transposition cipher and explore the use of compression models for the automatic cryptanalysis of Arabic text.

Text compression can be defined as a method of eliminating redundant information in a text aiming to minimize space that is required to store it, thereby reducing the time required to transfer this information without losing any information from the original text. There are two main approaches of constructing text compression models, which are dictionary and statistical approaches [3]. A well-known statistical based approach is Prediction by Partial Matching (PPM) while a well-known dictionary based approach is Gzip [4], which uses the Lempel-Ziv algorithm. PPM models can be applied to tasks with performance that emulates that of humans [5].

There are many different approaches used for cryptanalysis especially for the English language. However, using compression schemes adapted for the Arabic language as one way to tackle the Arabic plaintext recognition problem as we do in our paper is a novel approach and not tried before in the literature. The PPM compression system uses a modelling method that performs well at different language processing tasks and can be successfully applied to the automatic cryptanalysis of transposition ciphers when using the English language with a 100% success rate [6]. In this paper, a compression-based approach for the automatic cryptanalysis of transposition ciphers adapted for the Arabic language without any need for human intervention is proposed. In addition, a further algorithm is also introduced to automatically insert spaces into the decrypted texts in order to achieve readability.

Our paper is arranged as follows. Related work is presented in the next section followed by background to transposition ciphers. A description of compression based cryptanalysis and specific details of our method are provided in sections 4 and 5. The experimental results are reviewed in section 6 and the conclusions are provided in the last section.

II. Related Work

Many different cryptanalysis methods have been invented to break transposition ciphers for the English language, such as anagramming [7], using genetic algorithms [8], simulated annealing and Tabu search [9]. Alkazaz et al. [6] presented a new universal automatic cryptanalysis approach of transposition ciphers for the English language. A compression-based approach was used as a basis for breaking the cipher using the PPM and Gzip compression schemes. Ciphertexts with different sizes (from 12 to 600 letters) were tested. They managed to achieve a 100% decryption success rate by using PPM compression models, in addition to the high quality performance achieved in segmenting the decrypted texts.

Despite the fact that the science of cryptology was born among Arabs who also managed to break the mono-alphabetic substitution cipher after many years of its successful use [10],

unfortunately only a few publications have investigated the use of the Arabic language for various encryption/decryption systems [11]–[14] and none for the problem of solving transposition ciphers.

In this paper, we further investigate the English-based work of Al-kazaz et al. [6] to examine if it is applicable to other languages and ascertain whether the Arabic language provides a greater challenge for decryption. We also choose this language specifically because it has characteristics that differentiate it from other languages and because Arabic is non-related to English. Specifically, it is written and read from right to left rather than vice versa. It consists of 28 consonant letters and the vowels are represented by marks below and above the letters. It also has distinctive variations to represent singular, dual and plural forms and to represent male and female forms. Written Arabic also often exhibits triglossia with classical, modern and mixed forms often appearing together. Each letter in the Arabic script denotes a unit of the language. Unlike the English language, there is no upper case or lower case in Arabic and each word comprising of more than one letter are written joined together (cursive writing), with an exception for some letters which are "ز ر ذ د", and "ا" if they come at the beginning. These characteristics, as well as its rich morphology [15], presents challenges for natural language processing and cryptographic systems.

In our approach, Arabic transposition ciphers are generated and then a modified compression based approach for the automatic cryptanalysis of Arabic transposition ciphers is proposed in this paper. Various 125 Arabic ciphertexts with different lengths, from very short ciphers (17 Arabic letters) to ciphers of length 800 letters, are then tested in our experiments described below.

III. Transposition Ciphers

This section will now review transposition ciphers specifically. In a transposition cipher, the content of a message is obscured by rearranging groups of symbols; a transposition is thus a permutation. The concept of transposition is an essential one and has been used in the design of modern ciphersystems [16]. Originally, the message was written into a matrix in row-major order and then read out in column major order. Encoding "أنظمةالتشفير", which means "encryption systems", using a 3×4 matrix is shown in Figure 1. This figure gives "أةشنافظليمتر" if the columns are read off in order.

أ	ن	ظ	م
ة	ا	ل	ت
ش	ف	ي	ر

Figure 1: Example of a matrix transposition cipher.

By reading the columns in a different order, say 3–2–1–4, different ciphertexts can be obtained, such as "ظلينافاةشمتر". The order to read the columns and the size of the matrix constitute the key. The technique can be extended to d dimensions. In transposition ciphers, a plaintext block is encrypted into a ciphertext block using a fixed permutation [17].

Transposition ciphers are a class of ciphers that in conjunction with substitution ciphers form the basis of all modern symmetric algorithms [17]. These algorithms, such as block and stream ciphers also use in conjunction to the above forms more complex transformations, notably for providing diffusion. Although the field of cryptology has undergone a revolution after the introduction of the asymmetric cryptographic cipher in 1976, symmetric ciphers still form the basis for secure data transmission today, because of their superior speed and efficiency [18]. Hence in this paper, we focus on a very basic building block that is inherently present in most symmetric cryptographic systems, namely transposition ciphers.

Transposition ciphers are generally considered as much more difficult to decrypt than the other basic ciphers such as simple substitution ciphers. A number of statistical tools have been developed aiding automated breaking of substitution ciphers while the automatic decrypting of transposition ciphers is notoriously difficult. In general, cryptanalysis of transpositions is highly interventionist and demands some knowledge of the probable contents of the ciphertext to give an insight into the order of rearrangement performed [6], [19].

IV. Compression as a Cryptanalysis Method

The ciphertext only cryptanalysis of simple cipher systems heavily depends on the statistical features of the source language, and it is not a trivial issue to get computers performing this analysis. Although computers have been routinely used for a variety of tasks in cryptanalysis since their invention, the automatic recognition of valid decryptions has been acknowledged as a taxing problem [20]. Several previously published cryptanalysis methods can not run without human intervention or they assume at least known plaintext because of the difficulty of quickly recognizing a valid decrypt in a ciphertext only attack [21], [22].

Having a computer model that is able to predict and model natural language as well as a human is critical for cryptology [5]. The idea of using text compression to break Arabic transposition ciphers stems from the observation that the best PPM models can predict language about as well as expert human subjects [5]. The main idea of our approach depends on using the PPM model as a method for computing the compression 'codelength', which is a measure of the information, for each possible permutation [20]. The smaller the codelength, the more closely the cryptogram resembles the model. In this paper, we have shown how to use this to quickly and automatically recognise the valid decrypt in a ciphertext only attack against transposition ciphers for the Arabic language. We have also examined the use of another well-known compression scheme, which is Gzip. In our experiments, we have determined that the PPM compression method is the most effective to use for the automatic cryptanalysis, with Gzip having significantly less success.

A. PPM Compression Codelength Metric for Arabic

In this section, we describe the PPM method that we devised for our cryptanalysis solution in order to encode Arabic text efficiently and how the codelength metric is calculated. In the PPM compression scheme, the previously transmitted symbols are used to condition the probability of the next symbol using a Markov-based approach. There are

many variants of the basic approach depending on how many symbols are used to predict the next one, whether or not multiple predictions are used, and how shorter context models are used when necessary. The predictions are based on simple frequency counts of the transmission so far. The primary decision to be made is the context length modelled. This technique was first described by Cleary and Witten [23].

The order of a model is the maximum context length used to predict the next symbol. In practice, an order-o model will sometimes base its prediction on less than o symbols. This occurs when the model "escapes" down to the next lower order context if the higher order context has not seen the upcoming symbol that is being predicted. By convention the order(-1) model predicts each symbol with equal probability and the order-0 model predicts each symbol with probability proportional to the number of times it has occurred previously. An order 1 model predicts each symbol based on just the previous symbol, an order 2 model based on the two previous symbols and so on.

In 1990, Moffat [24] devised a simple mechanism that improved compression results further called "update exclusion". This mechanism is based on how the symbol counts for each context model are updated. When encoding with update exclusions, the predicted symbol count is incremented only if it is not already predicted by any higher order context. This means that the counts are updated only for the higher order contexts that are actually used to predict it. Thus, the counts reflect better which symbols are likely to be excluded by the higher-order contexts. This mechanism typically improves the compression rate by 1 or 2% [5]. On the other hand, when encoding without update exclusions, all the counts for all orders of the model are updated. The counts are incremented even if they are already predicted by a higher order context. The use of both of these mechanisms are investigated in our paper: PPM without update exclusions; and PPM with update exclusions (standard PPM).

Several PPM variations have been invented such as PPMA, PPMB, PPMC and PPMD. These variants differ in the way the escape probabilities are assigned. For example, PPMC uses escape method C, and PPMD uses escape method D. When a novel symbol is seen, the escape probability is sent, followed by the prediction using the next shorter context. Several escapes may be made before a context is reached which predicts the symbol. In the worst case the order(-1) model makes the prediction. Also, the maximum order of the context models may be included when the variant is described in the literature; for example, PPMD4 refers to a fixed order 4 PPM model using escape method D.

The PPMC variant was developed by Moffat [24] and has become the benchmark version. The probability estimates for this method (method C) is based on using the number of symbols that have occurred before, called the number of types:

$$e = \frac{t}{n+t} \quad \text{and} \quad p(s) = \frac{c(s)}{n+t}$$

where e represents the probability of the escape symbol, $p(s)$ denotes the probability for symbol s, $c(s)$ is the number of times the context was followed by the symbol s, n is the total number of times that the current context has occurred and t denotes the total number of types.

The PPMD variant was first developed by Howard in 1993 [25]. In most cases, experiments show that the PPMD performs better than the other variants. This variant is similar to the PPMC variant with the exception that each count is incremented by a 1/2:

$$e = \frac{t}{2n} \quad \text{and} \quad p(s) = \frac{2c(s)-1}{2n}.$$

An adaptive PPM compression model for Arabic language has been presented by Alhawiti [26]. This method is called Character Substitution of Arabic for PPM ("CSA-PPM"). It not only showed a considerable improvement in Arabic text compression but also for other texts that use Arabic script such as Persian and Kurdish. There are two main operations in this method, which are the pre-processing and post-processing. Each two-byte Arabic character is substituted with an equivalent number of the UTF-8 encoding scheme in the first operation, and as a result one output file is generated. Contrastingly in the post-processing operation, a reverse operation is performed by replacing the numbers with the original equivalent characters.

The fundamental idea of our approach depends on using a PPM compression method to compute codelength values of each possible permutation. The 'codelength' of a permutation for a cryptogram is the length of the compressed cryptogram, in bits, when it has been compressed using the PPM language model. The smaller the codelength, the more closely the cryptogram resembles the model. By using this metric, it can easily find the valid solutions automatically [6].

The most important step in our implementation is the training step. During the training phase, a large set of training texts is used to prime the models. Training text is chosen that is hopefully representative of the text being compressed and consequently better able to predict it. Experiments with English text show that training can improve performance dramatically as when skipping the training phase, there is not enough and sufficient data at the beginning to effectively compress the texts. In our experiments described below, we use a corpus of many different Arabic books and novels from different topics converted to 36 and 37 character Arabic to train our models (by case-collapsing non-alphabetic sequences to a single space). Classical and modern Arabic texts are used to produce this large corpus (which we have called the 'Mixed Arabic corpus' with details shown in Table I). With the training step, we in essence use static models, unlike standard PPM—that is, once the models have been primed using the training texts, they are not further updated when processing the ciphertexts.

Concerning the Mixed Arabic corpus, it is a large corpus of mixed classical and modern Arabic text. The corpus is a combination of the Bangor Arabic Compression corpus (BACC) [26], Corpus A [27] and a selection of files from the King Saud University Corpus of Classical Arabic (KSUCCA) [28]. The BACC is a 31-million words corpus collected from different sources such as magazines, books and websites. This corpus contains texts from a wide range of genres containing

Table I: Details of Corpora used to Construct the Mixed Arabic Corpus used to Train the PPM Models

Corpus	Bytes	List of topics
BACC	256,867,247	sports, culture, economics, education, art and music, political literature and heritage, history, religion
Corpus A	252,338,294	business, cinema, opinions, conferences, economics, politics
KSUCCA	4,086,066	religion, linguistics, literature, science, sociology, biography

both modern and classical Arabic. Corpus A is a modern corpus recently published with different genres which covers several current modern standard Arabic areas. This corpus is collected from the bilingual newspaper Al-Hayat website, and from the open-source online corpus, OPUS. It consists of 27-million Arabic words. KSUCCA is a corpus made up of classical Arabic texts with the purpose of studying the lexical semantics of the Holy Qu'ran. It is a 50-million word corpus covering several genres. The overall file size of our Mixed Arabic corpus is 513,291,607 bytes encoded using a UTF-8 scheme and consists of 58,068,493 words.

B. Calculating Codelengths Using the Gzip Compression Method

Gzip or GNU zip [4] is one of the most common lossless compression schemes used on the Unix operating system and on the Internet. It uses a dictionary based approach (using the Lempel-Ziv algorithm) whereas PPM is a statistical based approach.

The reason for experimenting with other compression schemes in our paper is to find out which is the most efficient scheme that can be applied to the problem of plaintext recognition using a compression based technique. In our paper, the calculation of the Gzip codelength metric will be based on using a relative entropy calculation which allows us to use 'off-the-shelf' software without the need to re-implement the method itself (as we have done with our PPM models). The codelength metric can be calculated using the relative entropy technique by the following formula [6]: $h_t = h_{T+t} - h_T$, where T denotes the training text (which will usually be large in size), t denotes the testing text, and h_T refers to the size of the compressed file T. Essentially, the codelength for a particular compression scheme is calculated as being the difference between the compressed size of some large training text with testing text concatenated after it compared to the compressed size of just the training text by itself.

We have also investigated using another well-known lossless compression scheme, Bzip2, using our relative entropy method. However, due to the block-sorting nature of the Burrows-Wheeler algorithm, the calculation of some of the relative entropy codelengths ended up being negative and therefore could not be used.

V. Our Method

A full description of our new method for the automated decryption of Arabic transposition ciphertext will be presented in this section. As stated, the fundamental idea of our method is based on using a compression scheme to calculate the codelength value to use as a metric for ranking the quality of each possible permutation [6]. PPMC, PPMD and Gzip are the compression schemes used for the codelength calculations.

First, we generate an Arabic transposition cipher in Unicode. Then, we perform our new cryptanalysis approach. Our new cryptanalysis method consists of two main phases. The first phase (Phase I) is based on trying to automatically decrypt an Arabic ciphertext using a transposition of specified size by exhaustively computing all possible transpositions, while the second phase (Phase II) is based on achieving readability (word segmentation). Achieving readability is gained by automatically adding spaces to the decrypted message that is produced from phase I, as the spaces are omitted from the ciphertext traditionally.

The pseudo code for the first part of our approach is presented in Algorithm 1. The first step in this algorithm focuses on removing all the non-alphabetical Arabic characters (including spaces) from the ciphertext (see line 1). At this stage, the text comprises just 36 Arabic letters. In order to calculate the codelength value for each Arabic permutation, a pre-processing operation is performed using an Arabic-specific encoding method using UTF-8 numbers (see line 4) where each Arabic character is substituted with an equivalent number in the UTF-8 encoding scheme. The algorithm then starts to try all possible key sizes and for each key size a permutation is performed over each cryptogram block trying to get a permutation with a smaller codelength value which represents the correct solution (lines 5 to 14). Each cryptogram is divided into blocks according to the key size (see lines 6 and 7). Then, a permutation is performed over each cryptogram (line 8). For each possible permutation, a codelength value is calculated (line 9 to 12). Two main compression-based models, PPM and Gzip, are used for calculating the codelengths (line 10). Permutations with smaller codelengths are kept in the priority queue as shown in line 11. We have found that the smaller the codelength, the more closely the cryptogram resembles the model.

Algorithm 1: Pseudo code of the main decryption phase 'Phase I'.

```
   Input  : ciphertext
   Output: decrypted text
1  remove all non-Arabic alphabet characters and spaces from the
     ciphertext;
2  maximum size of Q (priority queue) ← 3;
3  maximum key-size of transposition ← 12;
4  perform pre-processing operation by using Arabic-specific
     encoding method on the ciphertext;
5  foreach key-size do
6      if ciphertext-length mod Key-size = 0 then
7          divide the ciphertext into blocks according to key-size;
8          perform a permutation over each ciphertext blocks;
9          foreach possible permutation do
10             calculate codelength value using the CSA-PPM or
                 Gzip compression model;
11             store a permutation with smaller codelength value in
                 Q;
12         end
13     end
14 end
15 return the priority queue 'Q' (the 'decrypted text');
```

The pseudo code for the second phase of our approach

is presented in Algorithm 2. As stated, this phase focuses on achieving readability. The second assessment step is performed in this phase through adding spaces automatically to the decrypted messages that was produced from Phase I, and then assessing the quality of the solutions by computing the codelength values for each possible message. Our solution for Phase II uses the Viterbi algorithm to find the best possible segmentation (see Algorithm 2). For each of the text produced as output from Algorithm 1, the Viterbi algorithm is used to search for the best segmentation sequence and this then is stored in a priority queue (lines 2 to 5) which is used to return the result (line 7). A post-processing operation is performed afterwards, by using the Arabic-specific decoding using UTF-8 numbers. PPM and Gzip compression models are also used again as the method for calculating the codelength values.

Algorithm 2: Pseudo code for Phase II

```
   Input  : the priority queue 'Q' from Phase I
   Output: segmented decrypted text
1  maximum size of Q1 (priority queue) ← 1;
2  foreach text in Q do
3  |   use the Viterbi algorithm to search for the best segmentation
   |     sequences;
4  |   store the text that have the best segmentation which present in
   |     Q1;
5  end
6  perform post-processing operation by using the
     Arabic-specific encoding method on the text in Q1;
7  return the best Arabic segmented decrypted text from Q1;
```

Two modified variants of the PPM modelling system, PPMD and PPMC, have been investigated in our method. Also two forms of these schemes are examined, one with update exclusions (i.e the standard PPMD and PPMC) [5] and one without update exclusions. The use of the Gzip compression scheme is examined as well, in order to explore the most effective method that can be applied to the problem of automatically recognizing the valid decryption.

Experiments with a full range of variations have been conducted (PPMC, or PPMD, with and without update exclusions; Gzip). However, for the purposes of this paper, only the results for three variants are shown in order to illustrate either the best performing schemes or to illustrate interesting results for comparison. In the first variant, called Variant I, order 5 CSA-PPM without update exclusions is used for Phase I and Phase II (using the Viterbi algorithm). For the second variant, Variant II, order 5 CSA-PPM with update exclusions (the standard PPM) have been used for Phase I and Phase II. The Gzip compression scheme is used in the last variant, Variant III.

VI. Experimental Results

The experimental results are discussed in this section. In our experiments, the order-5 CSA-PPM models have been trained on a corpus of many different Arabic books and novels from different topics using 36 and 37 Arabic character (when space is included). As explained above, classical and modern Arabic texts are used to produce this large Mixed Arabic corpus. After this training operation and during cryptanalysis, these different models remain static. Regarding the cryptograms test corpus, 125 different cryptograms were chosen at random from different resources to be used as testing texts. Cryptogram lengths ranged from 17 to almost 800 letters.

A sample of decryption is shown in Table II for the Arabic cryptogram 'ميكقنصدن دكعوكهاو اأرغخمنك'. Compression codelengths are listed in bits with the lowest three results presented for Phase I.

Table II: Output Produced for the Sample Cryptogram 'ميكقنصدن دكعوكهاو اأرغخمنك'.

Phase I	Phase II
Variant I	
صديقكمننهاكوعدوكمنأغراك 108.64	صديقك من نهاك وعدوك من أغراك
نكمنصديقكعدوهاكوراكمنأغ 110.62	يك من نصدقك عدوكها وأراك منغ
يكمننصدقكعدوكهاوأراكمنغ 111.37	نك من صديقك عدوها كورا كمن أغ
Variant II	
منكنصديقدوعكهاكواكرمنأغ 108.15	صديقك من نهاك وعدوك من أغراك
قكمنصدينوعدوهاككغراكمنأ 110.76	قك من صدين وعدوها ككغراكمنأ
صديقكمننهاكوعدوكمنأغراك 110.85	منكن صديق دوعك هاك واكر من أغ

We have created a transposition cipher for Arabic language that depends on using the Unicode encoding. For each run, a random key is generated to encrypt the original message (plaintext). Then, we have applied our cryptanalysis method to automatically break this message (ciphertext). Different permutation key sizes (from 2 to 12) and different cryptograms with different lengths have been examined in our experiments.

The experimental results of Phase I showed that for the first variant I, when the CSA-PPM methods without update exclusions is used, 97% of the cryptograms are successfully decrypted without any errors. For variant II, when the standard CSA-PPM method is used, the results showed that 95% of the ciphertext are solved with no errors. For the last variant, Variant III, using the Gzip scheme, a success rate of only 90% was achieved.

The main idea of the second phase is based on inserting spaces automatically into the cracked messages produced from Phase I. The codelength compression metric is also used in this phase to automatically rank the quality of the solutions to find the correct message. In order to measure the differences between the original texts and the decrypted texts that resulted from our decryption method, the Levenshtein distance metric is used. It is a string metric that is commonly used to calculate the differences (either deletions, insertions or substitution) between any two messages [29]. The results show that for variant I, 97% of the cryptograms are successfully decrypted and segmented. For variant II, which resulted in a 95% decryption success rate from Phase I, now 97% of the ciphertexts are effectively decrypted and segmented as a result of Phase II.

It is important to note that the second phase of our approach has the ability to find better solutions not found by the first phase. For example, referring to Table II, the best two solutions as output from using Variant II shown in column one are not correct (the third solution is the correct one). However, after the second phase has been applied, the best segmentation of the third solution in column one now appears as the best solution in column two which is correct.

Arabic word segmentation is considered a difficult problem [30]. As Arabic language is a rich morphological language, this characteristic represents a real challenge for identifying the

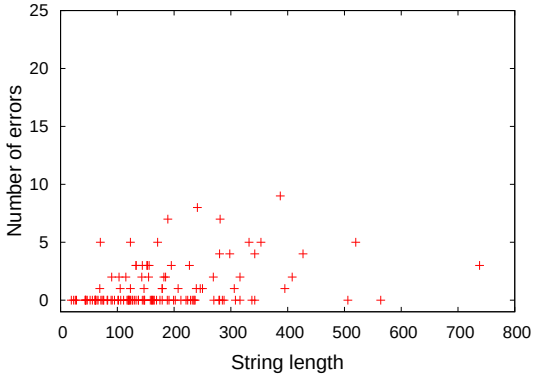

Figure 2: Segmentation errors produced from Variant I.

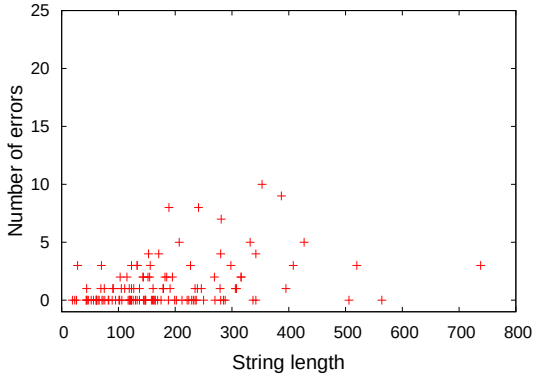

Figure 3: Segmentation errors produced from Variant II.

correct segmentation among many possible corrected alternatives. For example, the word " وردہ" could be segmented into " وردہ " and " و ردہ", which means "rose" and "and his reply". Both of these segmented forms are correct, but with completely different meanings and contexts. However, according to our approach and results, in almost all cases the proper readable solutions were obtained for variants I and II, but not for Variant III (using gzip).

Output examples (with spaces) produced from different variants are exhibited in Table III. For example, this table shows how 'إناللهلاينظرإلىصوركموأموالكمولكنينظرإلىقلوبكموأعمالكم', which is the best result produced from Phase I for a sample cryptogram, is effectively segmented with no errors using both variants I and II, while Variant III shows poor performance with nine segmentation errors.

Table III: Example of Solved Ciphertexts with Spaces by the Three Different Variants

Variants	Number of errors	Decrypted message (with spaces)
Variant I	0	إن الله لا ينظر إلى صوركم وأموالكم ولكن ينظر إلى قلوبكم وأعمالكم
Variant II	0	إن الله لا ينظر إلى صوركم وأموالكم ولكن ينظر إلى قلوبكم وأعمالكم
Variant III	9	إن اللهلاينظرإلىصوركموأموالكمولكنينظرإلىقلوب-كموأعمالكم

Figures 2, 3 and 4 present scatter-plots for the number of errors (on the y axis) versus the string lengths of cryptograms (on the x axis) for the different variants. For example, Figure 2 presents the number of word segmentation errors for each decrypted message that was output from Variant I (when PPMD is used). This figure shows that Variant I offered better performance than the other variants. More than 63% of the testing texts have been correctly segmented with no errors and a quarter of them with three errors or less. Just four cryptograms ended up with seven, eight or nine errors. These errors can be attributed to the rich morphological characteristic of the Arabic language with unusual words being used for various topics such as names of places and people, that were not included in the training text.

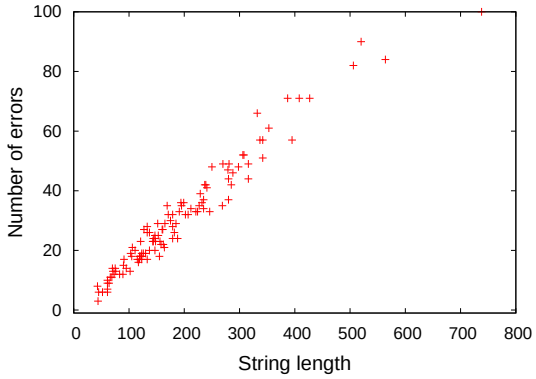

Figure 4: Segmentation errors produced from Variant III.

Variant II showed a slightly worse result than Variant I. Fifty three percent of the cryptograms were effectively segmented with no errors and 40 percent were segmented with four errors or less. Only one cryptogram consisting of 353 letters ended with ten errors. Figure 3 outlines Variant II results. The PPMD model is used in this variant. The number of errors produced from Variant III is shown in Figure 4. It is clear that the number of errors for each decrypted ciphertext in this case is much higher, with most of the errors being greater than 20. Also none of the cryptograms finished with no errors, and just eight of the solved cryptograms were segmented with the number of errors being less than ten.

The average number of space addition errors for the 125 testing texts for each variant using both main models, PPMD and PPMC, is presented in Table IV. The results show that PPMD produces slightly better results than PPMC and that the number of errors for the variants, except Variant III, is quite low with Variant I performing best.

Table IV: Average Number of Errors for the Phase-II Variants. (The PPMD and PPMC Models are used for the First Two Variants)

Variants	I	II	III
PPMD	1.10	1.23	30.43
PPMC	1.18	1.27	30.43

To investigate more about the accuracy of the compression methods that we used in segmenting the 125 Arabic decrypted texts, three main metrics are used: recall rate, precision rate and error rate. The first metric, recall rate, is calculated by dividing the number of successfully segmented words over the number of words in the original testing texts. The second metric, error rate, is calculated by dividing the number of unsuccessfully segmented words by the number of words in the original testing texts. The last metric, precision rate, is calculated by dividing the number of successfully segmented words by the number of words which are correctly and incorrectly segmented [6].

Table V: Three Different Accuracy Rates for Evaluating the Quality of the Word Segmentation

Variants	**Recall rate%**	**Precision rate%**	**Error rate%**
Variant I	96.35	95.94	3.65
Variant II	95.50	95.67	4.50
Variant III	1.53	10.02	98.47

The results presented in Table V indicate that the variants which depend on the CSA-PPMD compression methods were able to achieve very high recall and precision rates, reflecting the quality of the Arabic word segmentation. The error rates resulted from unusual words not included in the training text.

The average time that is needed to recognize the correct solutions for the different Arabic transposition cryptograms with different block sizes is shown in Table VI. This table presents the average elapsed time for the automatic cryptanalysis of three different length cryptograms, for both Phase I and Phase II. For each cryptogram, the average elapsed time in seconds is shown after running the approach ten times. The results show that the average time increases as the key size increases but overall the method is reasonably efficient.

Table VI: Average Time Required (Across 10 runs) to Automatically Break Cryptograms with Different Lengths and Different Keys Size

Ciphertext length (Letter)	**Key size**					
	Time (in seconds)					
	5	**6**	**7**	**8**	**9**	**10**
40	0.85	0.87	0.90	1.08	2.60	12.30
150	1.23	1.27	1.42	1.90	13.93	48.20
300	3.54	3.77	3.89	4.94	23.13	95.50

VII. Conclusions

The Arabic language is one of the most used languages in the world, but unfortunately the techniques used to encrypt and decrypt information in Arabic language are rare with few publications. In particular, we have not been able to find any cryptographic system for transposition ciphers in Arabic. In this paper, an Arabic transposition cipher is investigated and an automatic cryptanalysis of this cipher is also introduced. A success rate of 97% was achieved by using different Arabic compression models. PPM compression schemes were used as a basis for computing the codelength value (which we have found is an accurate way of measuring information in the text). Various Arabic ciphertexts with different lengths have been successfully decrypted and effectively segmented. This efficient method eliminates any need for human intervention.

The PPM compression variants showed better performance than Gzip in both decryption and segmentation processing. According to the main decryption phase, PPM models with no update exclusions, performed better than the standard PPM models with a 97% success decryption rate comparing to 95%. Concerning the second decryption phase, a success rate of 97% was achieved for the different PPM models with a high level of performance in segmenting words.

By using our method for automatically breaking Arabic transposition ciphers, various Arabic ciphertexts with distinct lengths (from 17 to almost 800 Arabic letter) have been examined. Different block lengths were also experimented with (from 2 to 12) and 97% of these ciphers were successfully decrypted with no error. This compares with 100% for the automatic cryptanalysis of transposition ciphers for the English language [6]. Although the success rate for Arabic is slightly worse than English, it still leaves open the question whether Arabic is a more challenging language for cryptanalysis or not. Further investigation needs to be performed regarding other Arabic ciphers and the automatic decryption of these ciphers to ascertain whether this language really provides a greater challenge for cryptanalysis.

References

[1] S. F. Gary and C. D. Fennig, *Ethnologue: Languages of the world (21st edition)*. Dallas, Texas: SIL International, 2018.

[2] I. A. Al-Kadit, "Origins of cryptology: The Arab contributions," *Cryptologia*, vol. 16, no. 2, pp. 97–126, 1992.

[3] T. C. Bell, J. G. Cleary, and I. H. Witten, *Text compression*. Prentice-Hall, Inc., 1990.

[4] Gzip. (2012) The gzip home page. (http://www.gzip.org), (accessed 1 March 2016).

[5] W. J. Teahan, "Modelling English text," Ph.D. dissertation, University of Waikato, New Zealand, 1998.

[6] N. R. Al-Kazaz, S. A. Irvine, and W. J. Teahan, "An automatic cryptanalysis of transposition ciphers using compression," in *the 15th International Conference on Cryptology and Network Securit*. Springer Int. Publishing, 2016, pp. 36–52.

[7] J. Seberry and J. Pieprzyk, *Cryptography: an introduction to computer security*. Prentice-Hall, Inc., 1989.

[8] K. P. Bergmann, R. Scheidler, and C. Jacob, "Cryptanalysis using genetic algorithms," in *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation (GECCO '08)*. New York, USA: ACM, 2008, pp. 1099–1100.

[9] A. Dimovski and D. Gligoroski, "Attacks on the transposition ciphers using optimization heuristics," in *Proceedings of International Scientific Conference on Information, Communication & Energy Systems & Technologies (ICEST)*, Sofia, Bulgaria, 2003, pp. 1–4.

[10] S. Singh, *The code book: the science of secrecy from ancient Egypt to quantum cryptography*. Anchor, 2000.

[11] Y. Alqahtani, P. Kuppuswamy, and S. Shah, "New approach of Arabic encryption/decryption technique using Vigenere cipher on mod 39," *International Journal of Advanced Research in Information Technology and Engineering*, vol. 2, no. 12, pp. 1–9, 2013.

[12] S. D. Rihan and S. E. F. Osma, "Arabic cryptography technique using neural network and genetic algorithm," *International Research Journal of Computer Science, vol. 3, pp. 35-42*, 2016.

[13] R. S. Habeeb, "Arabic text cryptanalysis using genetic algorithm," *Iraqi Journal for Electrical and Electronic Engineering*, vol. 12, no. 2, pp. 161–166, 2016.

[14] A. Muanalifah, "Construction of key echange protocol over max-plus algebra to encrypt and decrypt Arabic documents," *Journal Of Natural Sciences And Mathematics Research*, vol. 1, no. 2, pp. 51–54, 2017.

[15] T. Ng, K. Nguyen, R. Zbib, and L. Nguyen, "Improved morphological decomposition for Arabic broadcast news transcription," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, April 2009, pp. 4309–4312.

[16] M. Stamp and R. M. Low, *Applied cryptanalysis: breaking ciphers in the real world*. John Wiley and Sons, 2007.

[17] J. Giddy and R. Safavi-Naini, "Automated cryptanalysis of transposition ciphers," *The Computer Journal*, vol. 37, no. 5, pp. 429–436, 1994.

[18] W. Grundlingh and J. H. Van Vuuren, "Using genetic algorithms to break a simple cryptographic cipher," 2003, retrieved 31 March 2003 from (http://dip.sun.ac.za/ vuuren/abstracts/abstrgenetic.htm), submitted 2002.

[19] R. Matthews, "The use of genetic algorithms in cryptanalysis," *Cryptologia*, vol. 17, no. 2, pp. 187–201, 1993.

[20] S. A. Irvine, "Compression and cryptology," Ph.D. dissertation, University of Waikato, New Zealand, 1997.

[21] B. Schneier, *Applied cryptography (2nd edition)*. John Wiley and Sons, Inc., New York, 1996.

[22] M. J. Wiener, "Efficient DES key search," in *Advances in Cryptology: CRYPTO'93, Lecture Notes in Computer Science*, vol. 733. Berlin. Springer-Verlag, 1993.

[23] J. G. Cleary and I. H. Witten, "Data compression using adaptive coding and partial string matching," *IEEE Transactions on Communications*, vol. 32, no. 4, pp. 396–402, 1984.

[24] A. Moffat, "Implementing the PPM data compression scheme," *IEEE Transactions on Communications*, vol. 38, no. 11, pp. 1917–1921, 1990.

[25] P. G. Howard, "The design and analysis of efficient lossless data compression systems," Ph.D. dissertation, Brown University, Providence, Rhode Island, 1993.

[26] K. M. Alhawiti, "Adaptive models of Arabic text," Ph.D. dissertation, Bangor University, 2014.

[27] S. Alkahtani, "Building and verifying parallel corpora between Arabic and English," Ph.D. dissertation, Bangor University, 2015.

[28] M. Alrabiah, A. Al-Salman, and E. Atwell, "The design and construction of the 50 million words KSUCCA," in *Proceedings of the Second Workshop on Arabic Corpus Linguistics (WACL)*. The University of Leeds, 2013, pp. 5–8.

[29] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," in *Soviet Physics Doklady*, vol. 10, no. 8, 1966, pp. 707–710.

[30] A. M. Zeki, "The segmentation problem in Arabic character recognition the state of the art," in *the 1st International Conference on Information and Communication Technologies*. IEEE, 2005, pp. 11–26.