# A CRYPTOGRAPHIC TECHNIQUE BASED ON AVL TREE

**Sura N. Abdulla, Alyaa M. N. Al Barrak**
Department Of Computer Science, College Of Science, University Of Baghdad. Baghdad-Iraq.

**Abstract**

Cryptography provides confidentiality and privacy by scrambling information. This paper presents a new cryptographic method using English letters frequencies and AVL tree to provide an increased level of confidence for exchanging information over networks especially internet which is insecure network. Experiment results of the proposed method provide better security than the classical methods depending on number of parts used to build the key in this method.

# تقنية تشفير جديد تعتمد على شجرة الـ AVL

**سرى نجم عبدالله، علياء محمد نوري البراك**
قسم الحاسبات، كلية العلوم، جامعة بغداد. بغداد–العراق.

**الخلاصة**

الهدف الاساسي لأي طريقة مستخدمة في التشفير هو تجهيز المعلومات التي نحتاج الى نقلها من مكان الى اخر بالأمن والخصوصية. هذا البحث يقدم طريقة جديد للتشفير بأستخدام تكرار احرف اللغة الانكليزية و شجرة الـAVL لزيادة الموثقية والامنية وخصوصية المعلومات خلال عملية نقلها على الشبكات خصوصا شبكة الانترنت التي تتميز بكونها شبكة غير امنة وسهلة الاختراق. اوضحت نتائج تجربة هذه الطريقة اعطاء امنية افضل من الطرق التقليدية المعروفة وذلك اعتمادا على الاجزاء التي يتكون منها المفتاح المستخدم للتشفير.

## 1. Introduction

Information revolution is one known concept in today's world, no dout the old or traditional techniques for information movements, storage, and processing, specially during communicat-ions are no longer suitable for today's requirements, or at least can't be efficient, so it can't invest or process that information in speed suitable for today's requirements.

Processing and movement techniques used for information had been improved during the passed few years, and at the same time, attacking and stealing information had been improved too .The full protection of information is impossible or not an easy process, there always exist some gaps that can be used by attackers.

Cryptography, steganography, and operating system techniques methods used to protect files from the intruders [1].

This paper focuses on cryptography which is the science of ensuring that information can't be easily read or modified by unauthorized users [2], or it can be defined as the science of using mathematics to encrypt information. Cryptography is the art and science of scrambling information and writing in secret code and is an ancient art. The first documented use of cryptography in writing dates back to 1900 B.C. [3, 4]. Some experts argue that cryptography appeared sometime after writing was invented, with applications ranging from diplomatic missives to war-time battle plans [5]. It is no surprise then that new forms of cryptography came soon after the wide spread

development of computer communication. In data and telecommunication, cryptography is necessary when communicating over any untrusted medium, which includes just about any network, particularly the Internet.

## 2. Letter Frequency

The paper focuses on English words or letters appearing in the messages or texts that we want to encrypt to find the suitable key. The key depends on the repeated groups of letters that could appear in any message. By studying the English words in general, there are some groups of letters occurs more than others, these groups of letters can be divided into two classes: "words" and "parts of words" (as shown in Figure 1).

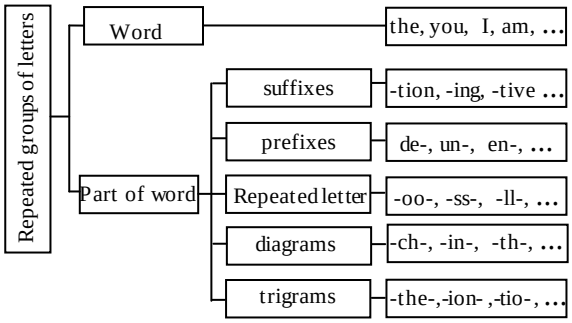



**Figure 1: The main classes of repeated groups of letters**

When an ordinary English dictionary is opened, one can see that a lot of English word that can be divided into three parts (prefix, stem (or root), suffix) [6, 7] (as shown in figure 2):-

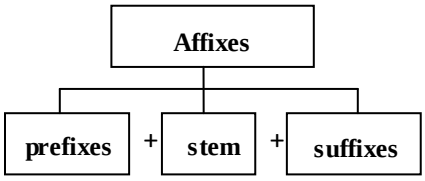



**Figure 2: main structure of an English word**

The words 'prefix' and 'suffix' are good example 'Pre-' means 'before' and '-fix' means 'to fasten' or 'attach', so a quite literally, a prefix is something attached to the beginning of something else, therefore it is what comes before the stem[8]. Consider as an example the prefix "de-" (meaning 'reduce' or 'reverse') in a word like demagnetize (meaning 'to deprive of magnetism'). 'suf-' is a variant of 'sub', 'below' or 'under', so a suffix is something fastened underneath something else, in this case, a suffix is what is attached to the end of the stem, they are standard syllables fastened to the end of a word to clear its meaning and usually its part of speech[10] .Consider as an example the suffix-er (meaning 'someone who') in a word like programmer ('the person who programs').

Both prefixes and suffixes are referred to as affixes because they are attached to a root [6]. Prefixes and suffixes were originally words themselves but they are now groups of letters added to words or roots to create new words [9]. Prefixes usually change the meaning of the word. Suffixes, on the otherhand, change the word from one part of speech to another.

'Frequency of letters' in English text has often been studied for use in cryptography, and frequency analysis in particular. No exact letter frequency distribution underlies a given language, since all writers write slightly differently. More recent analyses show that letter frequencies tend to vary both by writer and by subject. For example, one can't write an essay about x-rays without using frequent Xs.

Since the major methods for breaking crypts are letter and word frequencies [10], then there were many previous studies that results in building tables of the most common used letters patterns and high frequency words that could appear in the English texts (as shown in tables (1) and (2)). Letter patterns consist of distinctive letters, and may contain a repetition of the same letter. Letter patterns are groups of letters that often appear together in lots of different English words [11]. A letter pattern is a sequence of the same letters in the same order which can be found in many words.

## 3. AVL Trees

*AVL tree data structure* is used in this paper to reorder the resulting encrypted text to increase the difficulty of breaking the encrypted text.

In computer science, an *AVL Tree* is a special type of *Binary Search Trees*, as a matter of fact, it is a self-balancing binary search tree, and it is the first such data structure to be invented [12]. Recall that the purpose of a binary search tree is to allow efficient insertion and retrieval of information. Unfortunately, if values are added to binary search tree in sorted order (as we need in our work with the proposal method), then the tree becomes *long and straight* [13]. Insertion and retrieval into this type of tree is O(n), while the insertion and retrieval needs for the method in this paper O(log(n)) which doesn't *stretch* the tree into a single long path. That structure is called *AVL tree data structure* [13].

AVL tree is a binary tree that has the following properties [14, 17]:-

1- Its left subtree and right subtree differ in hight by 1 at the most.
2- Both subtrees themselves are also AVL trees.

This definition allows for trees that are slightly out of balance. The height of an AVL tree is always, even in the worst case, roughly proportional to log (n) where n is the number of nodes in the tree. This guarantees the logarithmic performance for the operations in adding and retrieving nodes of the tree [14, 15].
The criteria that is used to determine the *level* of *balance-ness* is the difference between the heights of subtrees of a root in the tree .The *height* of tree is the *number of levels* in the tree [18, 19] (as shown in Figure (3)).

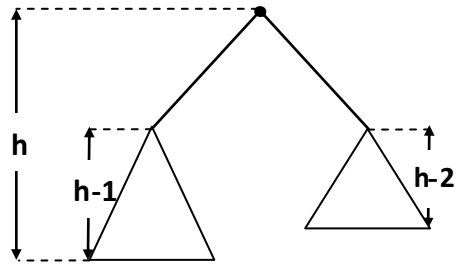



**Figure 3: balance requirement for an AVL tree. The left and right sub-tree differ by at most 1 in hight**

Operations on the AVL tree are essentially the same as on binary tree, with some addition to maintain approximate balance of the tree. If the tree gets out of approximate balance after an insertion or deletion then some additional mechanism will get it back into the required degree of balance. Insertions and deletions may require the tree to be rebalanced by one or more tree rotations which help to restore the height balance of the subtrees.

## 4. The Proposal Method

The AVL tree structure is used as a part of the encryption method, therefore the abbreviation "AVLEM" that refer to AVL encryption method is used as a name of this method.
In AVLEM method, the product cipher was used, to transform the original unencrypted message to an ciphertext .
First a substation mechanism is used to transform μ1, the original unencrypted message, to μ2 which can be defined as *a primary encrypted message* that can be used by a transposition mechanism to get μ3 the final ciphertext (as shown in Figure (4)).
The first part of the AVLEM method is the substitution mechanism, it substitute one or more characters from the original message by one code of 8 bits.

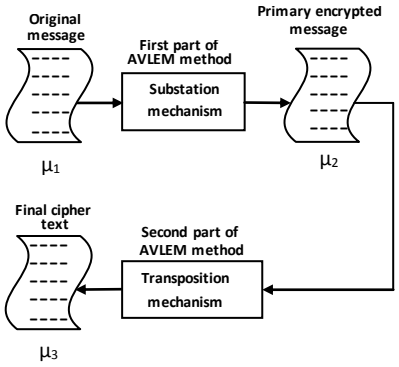



**Figure 4: AVLEM method main structure**

Frequency of letters that would appear in the texts is analyzed, and found that there are some groups of letters appear more than others. These groups of letters can be divided into four classes:-

1- Groups the most common used English words such as: *the*, *for*, *you*, … . These groups of letters have the following property: each group comes after a space and it's followed by another space.
2- Groups of letters that makes a prefix, such as: - *encryption*, *decoding*, … . These groups of letters have the following property: each group comes after a space and followed by an English letters (or suffix), i.e in the beginning of an English word.
3- Groups of letters that makes a suffix, such as: *encryption*, *decoding*…. These groups of letters have the following property: - each group comes after an English letters (or prefix) i.e in the end of an English word and it's followed by a space.
4- Groups of letters that makes one of the following:
   - Repeated letter, such as: *wood*, *follow*, *feed*…
   - Diagrams (i.e., most common used two English letters) such as: *school*, *father*,….
   - Trigrams (i.e., most common used three English letters) such as: *other*, *information*. These groups of letters have the following property: each group comes after an English letters (or prefix) and followed by an English letters (or suffix), i.e. in the middle of an English word.

After that two tables are constructed. One table to associate one code for four groups of letters, each group belong to a different class than the others, i.e .the same code can be used to replace a word and a prefix and a suffix and a (repeated letter or a diagram or a trigram). The second table to associate one code for each English letter or special character which does not belong to any class. Codes used to replace English letters and special characters cannot be used to replace any group of letters because the remaining English letter can have the same property of any one of the four classes.

The table of the four classes and table or remaining English letters and special characters shown in table (1) and (2) respectively.

**Table 1: group code table**

| Code | Suffix | prefix | double | word |
|---|---|---|---|---|
| **00000001** | -once | un- | -he- | will |
| **00000010** | -ence | non- | -an- | by |
| **00000011** | -er | in- | -th- | had |
| **00000100** | -or | dis- | -in- | one |
| **00000101** | -ation | re- | -er- | or |
| **00000110** | -tion | ex- | -on- | from |
| **00000111** | -ist | pre- | -re- | have |
| **00001000** | -yst | dis- | -ed- | this |
| **00001001** | -ness | im- | -nd- | be |
| **00001010** | -ion | il- | -ha- | at |
| **00001011** | -ing | de- | -at- | I |
| **00001100** | -ment | under- | -en- | they |
| **00001101** | -ity | sub- | -es- | his |
| **00001110** | -ian | post- | -of- | with |
| **00001111** | -ism | dec- | -nt- | as |
| **00010000** | -dom | di- | -ea- | are |
| **00010001** | -ship | pro- | -ti- | on |
| **00010010** | -ary | com- | -to- | for |
| **00010011** | -ize | con- | -io- | was |
| **00010100** | -ate | of- | -co- | he |
| **00010101** | -fy | op- | -ee- | it |
| **00010110** | -dom | per- | -tt- | that |
| **00010111** | -ly | suc- | -ff- | you |
| **00011000** | -al | sur- | -ll- | is |
| **00011001** | -ar | at- | -mm- | in |
| **00011010** | -ical | al- | -oo- | to |
| **00011011** | -able | as- | -nn- | a |
| **00011100** | -ous | geo- | -pp- | and |
| **00011101** | -ious | neo- | -cc- | of |
| **00011110** | -ive | co- | -dd- | the |
| **00011111** | -ery | semi- | -ss- | but |
| **00100000** | -tor | mini- | -tha- | not |
| **00100001** | -ify | auto- | -ent- | what |
| **00100010** | -ency | pan- | -ion- | all |
| **00100011** | -sion | maxi- | -tio- | were |
| **00100100** | -ial | tri- | -for- | we |
| **00100101** | -ent | ad- | -has- | when |
| **00100110** | -arly | af- | -nce- | your |
| **00100111** | -arize | cat- | -edt- | can |
| **00101000** | -able | dif- | -the- | there |

**Table 2: character code table**

| code | English letters And special characters |
|---|---|
| **00101001** | a |
| **00101010** | b |
| **00101011** | c |
| **00101100** | d |
| **00101101** | e |
| **00101110** | f |
| **00101111** | g |
| **00110000** | h |
| **00110001** | i |
| **00110010** | j |
| **00110011** | k |
| **00110100** | l |
| **00110101** | m |
| **00110110** | n |
| **00110111** | o |
| **00111000** | p |
| **00111001** | q |
| **00111010** | r |
| **00111011** | s |
| **00111100** | t |
| **00111101** | u |
| **00111110** | v |
| **00111111** | w |
| **01000000** | x |
| **01000001** | y |
| **01000010** | z |
| **01000011** | Speace |
| **01000100** | : |
| **01000101** | , |
| **01000110** | ; |

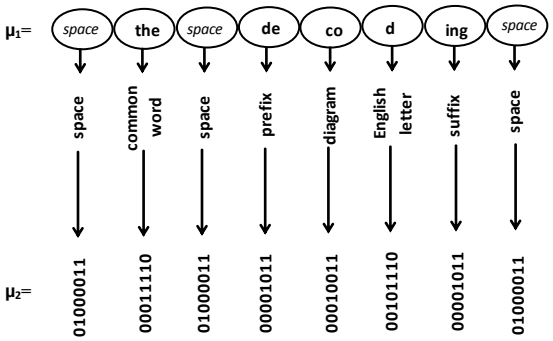
## 5. Results

This section demonstrates how the proposed method AVLEM was operated.

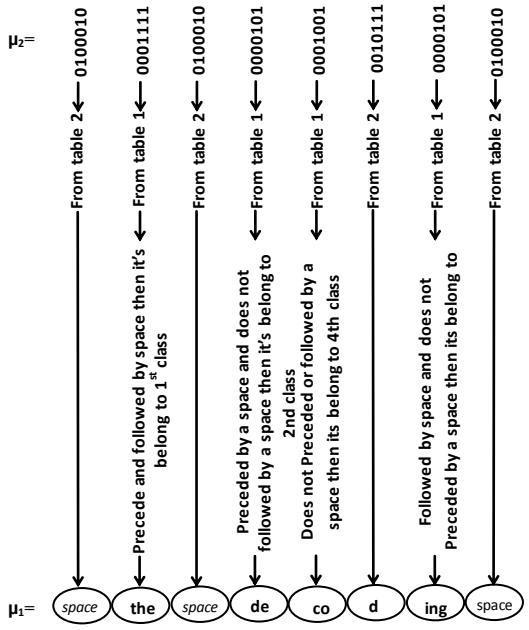
Example: Original message containing the following two words:

μ1= the decoding

Then, encrypt $\mu_1$ using table (1) and table (2) to obtain $\mu_2$:




If we need to reverse that work and decrypt $\mu_2$ to get $\mu_1$ by using the same tables then the result is:




The second part of AVLEM method is the transposition part:-

The result from processing part 1 of AVLEM method on the original message $\mu_1$ (i.e. $\mu_2$), was rearranged to get the final ciphertext $\mu_3$.

To reorder the bits of $\mu_2$, AVL tree structure is used. Notice that $\mu_2$ is an array of one dimension, each element in this array has a value either 0 or 1, and has an index which determines its position in the array. These indexes are used to reorder the values in the array. An AVL tree is built where each node in the tree contains an index of a bit in the array $\mu_2$.

In general, AVL tree is a self balanced binary tree, where each node has the following information (as in figure 5):

1. V :value of this node
2. L:pointer to the left son of this node.
3. R:pointer to the right son of this node.
4. H :balance factor, which is either -1, 0, or +1, equal to the height of right subtree minus the height of the left subtree .This factor is important during insertion process to keep the balance of the tree.

The proposed method needs to add another pointer to each node, N: next link, a pointer to the next value to be displayed during traversing process for the tree.

From figure (6) pointer N is pointing to the next node at the same level of this node, and if this node is the last node in its level then it will point to the first node (from the right side of the tree) in the next level. Pointer N is use to traverse the AVL tree, for example, traversing the tree shown in figure (6) will result the following order of the values:-

21, 15, 32, 9, 18, 28, 38, 7, 16, 24, 36, 41, 37

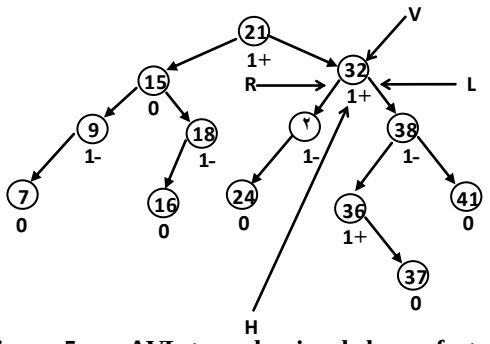



**Figure 5: an AVL tree showing balance factors**

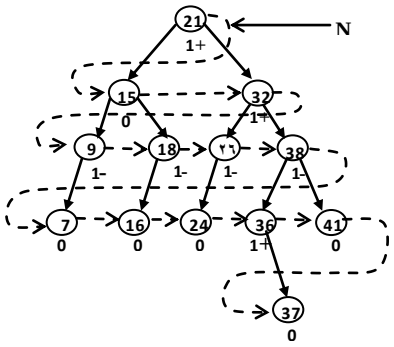



**Figure 6: an AVL tree with next links**

These are the main lines of part 2 of AVLEM method, $\mu_2$ (array of bits), the AVL tree is builted where each value V in the tree is an index in $\mu_2$ then we traverse the AVL tree to get the new order of the indexes and we use this new order to reorder the elements of $\mu_2$ to get $\mu_3$ the ciphertext .

For example suppose that $\mu_2$ is as shown in figure (7):-

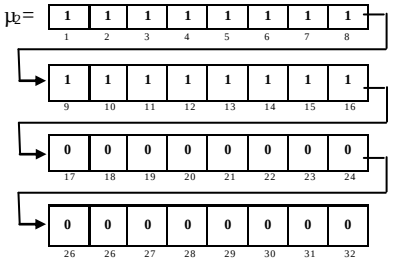



**Figure 7: $\mu_2$**

Which means that $\mu_2$ is an array of 32 bit, then the indexes of elements of this array is 1, 2, 3, 4,…, 29, 30, 31, 32. These indexes are used to build an AVL tree as illustrates in (figure 8):-

Traversing tree in (figure 8) will result in:-

16, 8, 24, 4, 12, 20, 28, 2, 5, 10, 14, 18, 22, 26, 30, 1, 3, 6, 7, 11, 13, 15, 17, 19, 21, 26, 27, 30, 31, 32

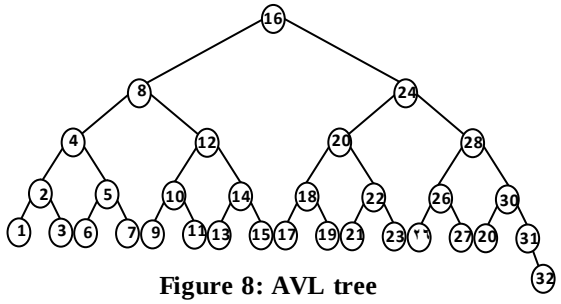


**Figure 8: AVL tree**

$\mu_2$ is reordered to build $\mu_3$ as in (figure 9). The first bit in $\mu_2$ will be the 16th bit in $\mu_3$, the second bit in $\mu_2$ will be the 8th bit in $\mu_3$, and the third element in $\mu_2$ is the 17th in $\mu_3$, and so on.

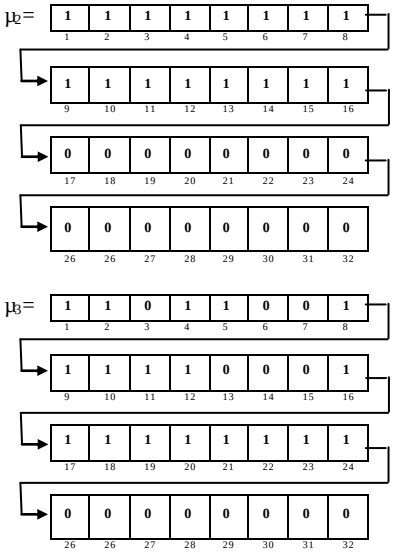


**Figure 9: building $\mu_3$ from $\mu_2$**

At the decryption process, all we have to do is to reorder the elements of $\mu_3$ to get $\mu_2$ by using the same AVL tree that we build at the encryption process because number of elements in $\mu_3$ is equal to number of elements in $\mu_2$.

For example, suppose we have the following ciphertext (figure 10).

We notice that number of elements in array is 64 then the AVL tree for this array as in figure (11):

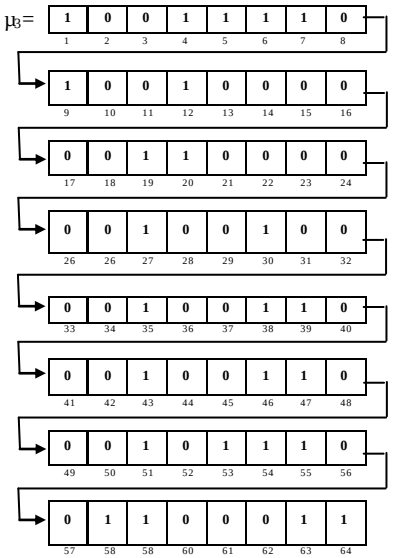


**Figure 10: array of ciphertext**

The result of traversing tree in (figure 11) will be:-

32, 16, 48, 8, 24, 40, 56, 4, 12, 20, 28, 36, 44, 52, 60, 2, 6, 10, 14, 18, 22, 26, 30, 34, 38, 42, 46, 50, 54, 58, 60, 1, 3, 5,7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41, 43, 45, 45, 47, 49, 51, 53, 55, 57, 59, 61, 63, 64

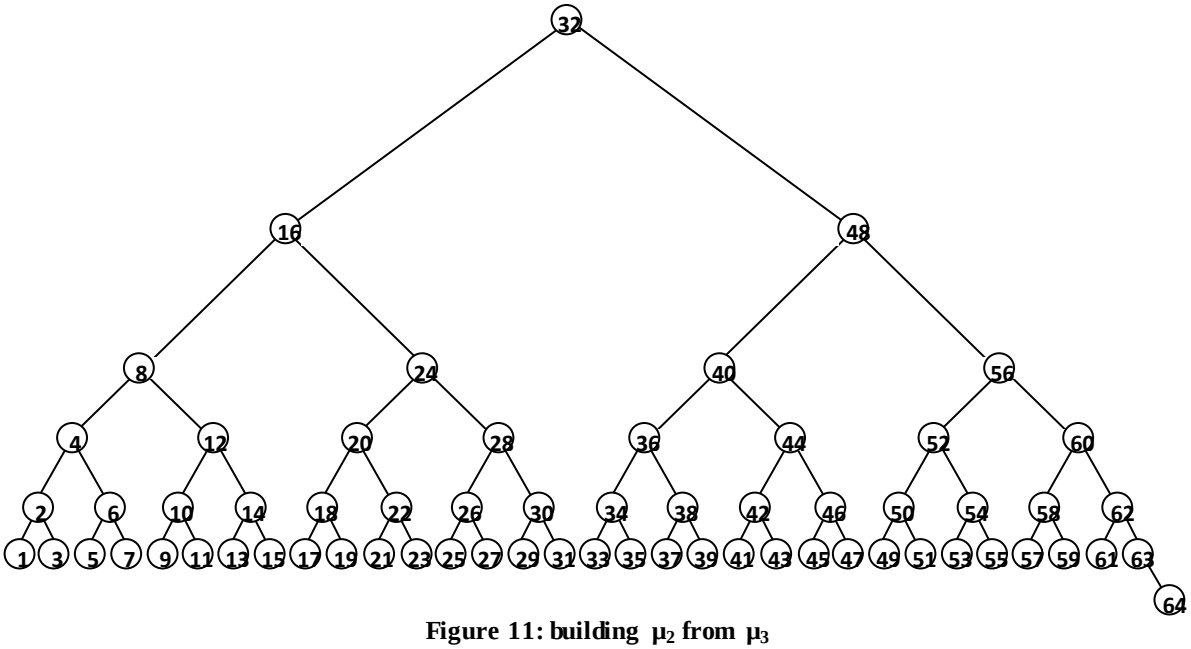


**Figure 11: building $\mu_2$ from $\mu_3$**

Which means that the first element in $\mu_3$ is the 32 element in $\mu_2$, and the second element in $\mu_3$ is the 16 element in $\mu_2$, and the third element in $\mu_3$ is the 40 element in $\mu_2$, and so on. Then we can reorder $\mu_3$ to get $\mu_2$ as in figure (12):-

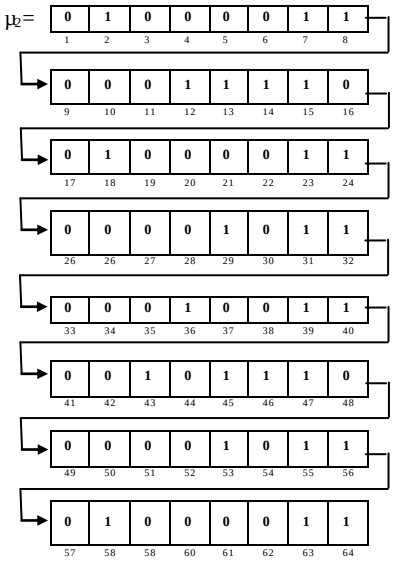



**Figure 12: array of decrypted message**

If we return to tables 1 and 2 then we will get $\mu_1$ (the original message) from the above $\mu_2$ which will be:-

$\mu_1$ = the decoding

## 5. Discussion

Good cryptographic systems should always be designed so that they are as difficult to break as possible [18, 19]. In theory, any cryptographic method with a key can be broken by trying all possible keys in sequence [18].

Cryptanalysis is the art of deciphering encrypted communications without knowing the proper keys [2], or we can defined it as: the science of recovering the plaintext of a message without access to the key [18].

The goal of cryptanalysis is to find some weaknesses or insecurity in a cryptographic scheme [1, 3]. Cryptanalysis might be undertaken by a hostile attacker, attempting to subvert a system; or by the system's designer (or others) wishing to evaluate whether a system is secure .

The major cryptanalysis methods used for breaking cryptographic systems are [20]:

1- "Brute force":- Try all possible keys.

2- "Statistical":- Look at letter and word frequencies. The major methods for breaking crypts are:-

   a. Letter Frequencies: Check for the high frequency letters which corresponding to the most used in the English words: E, T, A, O, N, and I. Look for doubled letters . Check for the final letters of the cipher's words.
   b. Vowel Placement: - Try to identify vowels by checking letters frequency.
   c. Pattern and High Frequency Words: - Check for one letter words, two letter words and three letter words.

3- "Cribs":-Look for known parts of message, e.g., html header, date, time, spaces.

4- A combination: Try all likely combinations.

According to the above mentioned major cryptanalysis methods, strength of the proposed method AVLEM is analyzed as follows:-

1- The key of AVLEM is constructed according the following:-

   I. Table of suffixes, prefixes, most common words, and most used doubled letters, and table of English letters and special characters..
   II. An AVL tree that has a depth depending on number of codes in the middle ciphertext $\mu_2$ and traversed level after another.

   These two parts of the key makes it difficult for attackers to find the exact key to decipher or attack the ciphertext because the list of all possible keys will be a very large list. Moreover if the original plaintext consists of more than one paragraph, then the resulting ciphertext will contain more than one paragraph too, and each paragraph in that ciphertext has code order different than other paragraphs because each paragraph has its own AVL tree. According to that, cryptanalysis of a ciphertext to find all possible keys is difficult since the key differs from one ciphertext to another, and from one paragraph of the ciphertext to another paragraph of the same ciphertext.

2- Looking at letters and words frequencies in the ciphertext resulting from AVLEM is useless since the proposal method is built to give code for each high frequency letters or words, and the same code can be used to represent more than one group of letters.

3- Each part of the message (such as date, time, html header) can be ciphered using AVLEM which makes it difficult to attack it. Moreover, space character has a code in AVLEM, and by ordering the ciphertext according to the corresponding AVL tree, the code that corresponds to the space

character will not appear between words, but will appears in random places.

4- Number of English letters is 26 small letter, and if we add the digital characters (0, 1, …, 9) and some special characters that could appear in the plaintext such as (,), (’), (:), then the number of possible characters that could appear in the plaintext will be a mostly 39 characters. In the resulting ciphertext, the number of codes is equal to the sum L+39 where L is the longest code table, this means that number of possible keys for (part 1 only) of the proposed method is greater than (L+39)!.

## 6. Conclusion

AVLEM used to encrypt a text using the ordinary English prefixes, suffixes, repeated letters and most used English word, then reordering the AVLEM has the following characteristics.

1- The resulting encrypted text is shorter than the original text. The difference between length of the two texts is arbitrary depending on the sequences of characters appearing in the original text.

2- The order of the characters that appears in the resulting encrypted depends on the length of the encrypted text paragraph resulting from the first part of the method.

3- Each character appearing in the resulting encrypted text could corresponds to one character or a set of different characters which means that the relationship between original text characters and resulting encrypting text characters is many – to – one relationship (i.e., more than one character in the original text can be corresponding to only one character in the resulting encrypted text).

4- Number of possible keys for this method is a very large number (greater than (L+39)!).

## References


1. Tanenbaum, A. S. **2001**. *Modern Operating System*, Prentice-Hall international edition, London.
2. Gerald, R. and Martin, S. **2002**. *Principles of Cryptography And System Security*, 1st edition, Tax. Llc.
3. Kessler, G. C. **1998**. *An Overview of Cryptography*, Handbook on Local Area Networks, Auerbach, P.3.
4. Lowrie, B. **1996**. *Introduction to Cryptography*, available at:-http//Williamstallings.com/ extras/security-nots/ledures/classical.html.
5. Lee, E. S. **1999**. *Essays About Computer Security*, center for communications systems research, Cambridge, P.177.
6. Vocabulary Workshop, *Vocabularly: Roots, Prefixes And Suffixes*, available at www.southhampton. liunet. edu/ acadmic/ pau/courc/websel.htm.
7. Mullen, M.D. and, Brown, P.C. **1987**., *English Computer Science*, 2nd edition, Oxford University Press.
8. Fowler, H. R. and Aron J. **2004**. *The Little Brown Handbook*, 9th edition, Longman, USA.
9. Vocabulary Workshop, *Vocabularly: Prefixes*, available at:-www.southhampton. liunet.edu /acadmic/ pau/ courc/ webpre. htm.
10. Cryptopop’s Hints, **2007** available at:-http: // www. freewebs. com/gidusko/cryptopop/ cryptopop_hints.html.
11. **2001**, *Skills wise Fact sheet*, available at:- http://www.bbc.co.uk/skillswise/words/spilli ng/recognising/letterpatterns/factsheet.shtml.
12. Sedgewick, R. **1983**. *Algorithms*, 2 illustrated, reprint, revised, Addison-Wesley.
13. Horowitz, E. and Sahni, S. **1984**. *Fundamentals of Data Structures in Pascal*, Computer Science Press, London.
14. Bratko, V. **1988**. *Prolog Programming For Artificial Intelligence*, Addison-Wesley, New York.
15. Miller, N. E. **1987**. *File Structure Using Pascal*, Benjamin Cummings,
16. Appleton, B. **1997**. *AVL trees: Tutorial and C++ Implementation*, Free EBook available at:http://www.cmcrossrroad.com/ftp/sre/libs/ C++/AVLtrees.html.
17. Drozdek, A. **2001**. *Data Structure and Algorithms in C++*, Brooks/Cole, 2nd edition, Magrow-Hill, New York.
18. Schneier, B. **1996**. *Applied Cryptograph: protocols, algorithms, and source code in C*, 2nd Edition, John Wiley and sons, London, P.5.
19. Swiecki, R. **2004**. *Principles of cryptograph*, document available at: www. minelinks. Com / supercode / index .html
20. **2003**, *Creative Cryptography*, available at:- www.enchantedmined.com/html/creativity/t echniques/creative_cryptography.html.