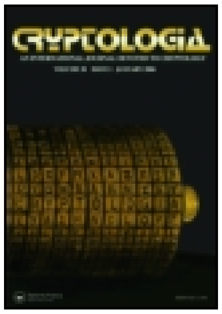

Cryptologia
Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/ucry20

THE AUTOMATED CRYPTANALYSIS OF SUBSTITUTION CIPHERS

JOHN M. CARROLL & STEVE MARTIN

Published online: 04 Jun 2010.

To cite this article: JOHN M. CARROLL & STEVE MARTIN (1986) THE AUTOMATED CRYPTANALYSIS OF SUBSTITUTION CIPHERS, Cryptologia, 10:4, 193-209, DOI: 10.1080/0161-118691861001

To link to this article: http://dx.doi.org/10.1080/0161-118691861001

THE AUTOMATED CRYPTANALYSIS OF SUBSTITUTION CIPHERS

JOHN M. CARROLL AND STEVE MARTIN

ABSTRACT: Expert systems programmed in artificial intelligence languages can break simple substitution ciphers by capturing and employing knowledge about the relative frequencies of single letters and certain combinations of letters in certain positions. These techniques can be extended to fixed-key polyalphabetic ciphers after using regression analysis to separate out the segments of text enciphered in each component alphabet.

INTRODUCTION

Our interest in machine cryptanalysis arose out of a graduate course in Cryptography and Data Security taught by Helmut Jurgensen and J.M. Carroll in Spring 1985.

One student project involved computer aided analysis of simple substitution ciphers; that is, making and displaying frequency counts and implementing trial substitutions. Another group implemented the Peleg and Rosenfeld [7] relaxation algorithm on a VAX 11-780. This group achieved qualified success but tied up the machine for a weekend to do so.

FIRST ATTEMPT

Our first program ran on a TI Professional Computer in MS/BASIC. Its objective was to disambiguate automatically the letters E, T, A, N, O, S, I, and R in cryptograms in English having 50 or more letters, using only the 26 alphabetic characters with spaces left in, incorporating no proper names or initials, and enciphered in a fixed, single-alphabet, random key (i.e. not a Caesar, etc.).

Our program is shown by a structure chart in Figure 1. In Figure 2, the steps included in each symbol box are expanded. The fourteen tests were derived from occurrence frequency characteristics summarized by Caxton Foster [4].

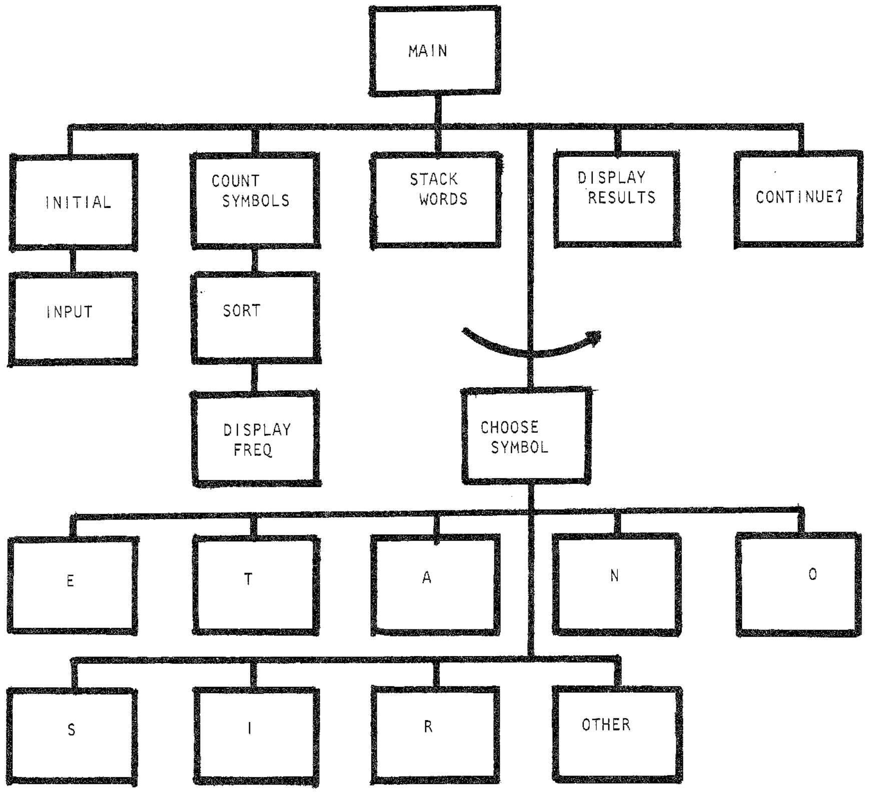

Figure 1. Structure chart of a semi-automated cryptanalysis program.

1. Number of symbol occurrences.
2. Number of occurrences as a one-letter word.
3. Number of occurrences as the initial character of a word.
4. Occurrences as the second character.
5. Occurrences as the third character.
6. Occurrences as the fifth character.
7. Occurrences as the sixth character.
8. Occurrences as the terminal character.
9. Next to last.
10. Third from end.
11. Fourth from end.
12. Fifth from end.
13. Sixth from end.
14. Occurrences as a member of a pair of doubled characters.

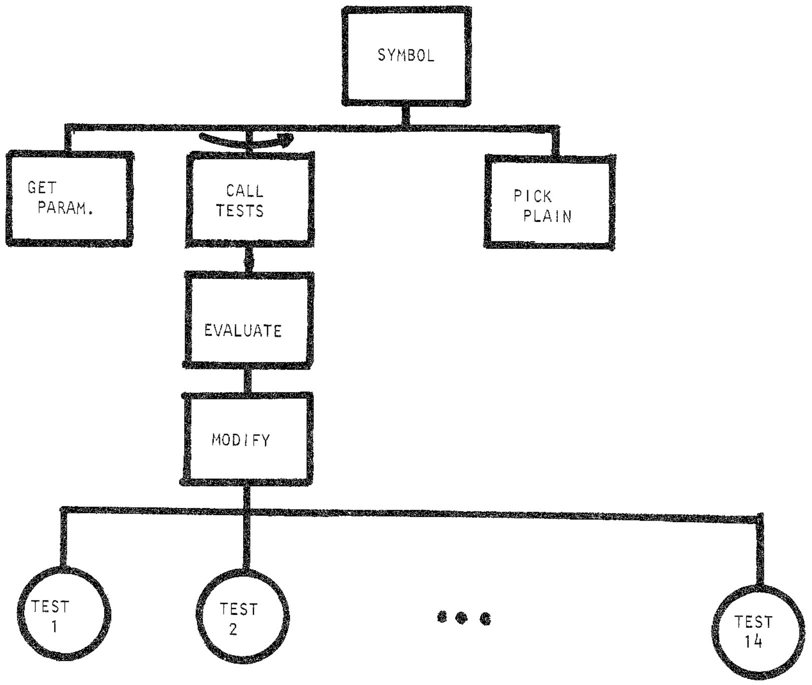

Figure 2. Structure chart expanding the program elements indicated by boxes labeled E, T, A, N, O, S, I, and R.

The program had limited success. We could always disambiguate E and usually T, then solve the crypt manually. Sometimes we solved four or five letters automatically. Running time was on the order of five minutes or less.

EXPERT SYSTEM

Next we tried to automate the process. The data flow diagram (Figure 3) shows main parts of the program. The program was written in the artificial intelligence language MICROPROLOG. It runs on an IBM-PC/AT. The program DECODE sets up a 26-by-26 matrix that encapsulates all the knowledge about the crypt we can acquire from statistical analysis. The program RESTORE-CHOOSE then starts making guesses as to the plain text equivalents of crypt symbols while testing them against "impossible" digraph combinations on one hand (removing bad guesses) and "common" digraph and trigraph combinations on the other (reinforcing good guesses). RESTORE-CHOOSE also allows the user to intervene when the program runs out of guesses (although by that time the content and meaning of the cryptogram are usually evident).

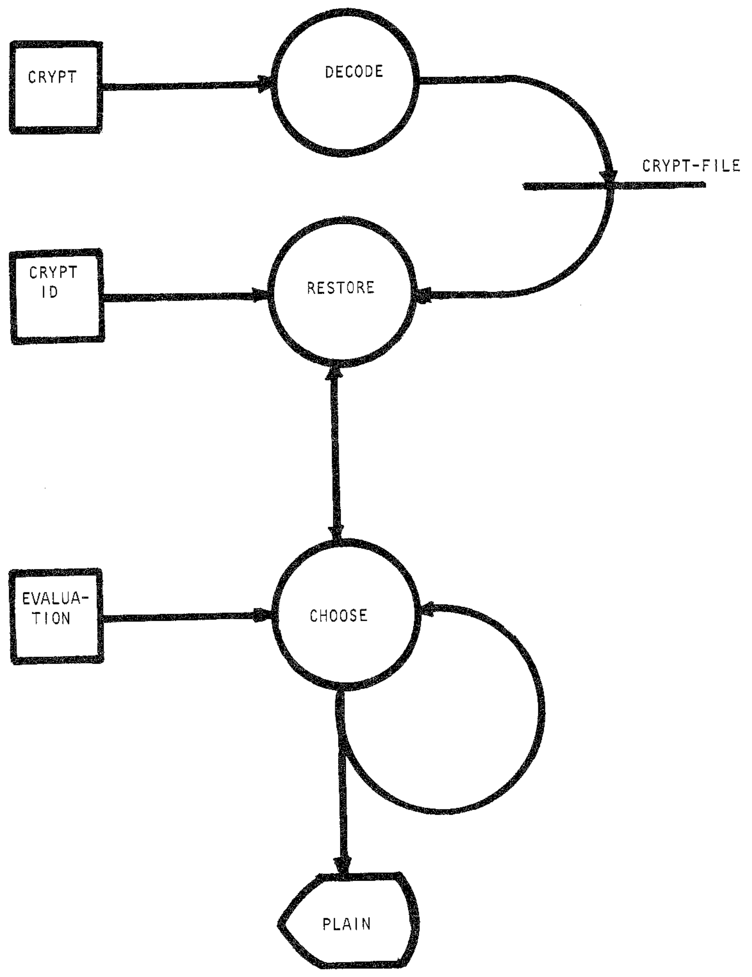

Figure 3. Data flow diagram of the automated cryptanalysis system.

Figure 4 is a structure chart of the DECODE and RESTORE-CHOOSE programs. Figure 5 expands the IMPOSSIBLE and ADJUST_COUNTS modules of the DECODE program. However, the heart of the DECODE program is the GET_FREQUENCY module.

"GET_FREQUENCY" MODULE

Let X stand for any crypt symbol, there are 26 of them; and let y stand for any of the 26 plaintext symbols. Let L stand for the number of symbols in

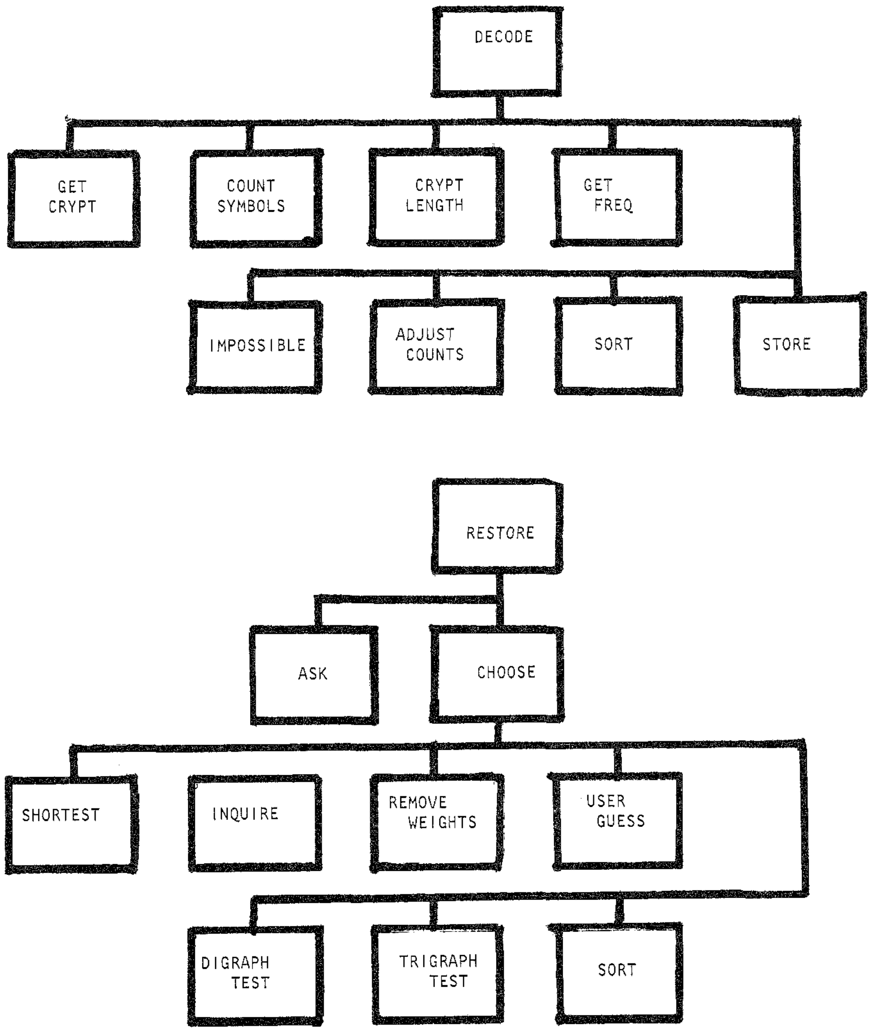

Figure 4. Structure charts of the DECODE and RESTORE_CHOOSE programs.

the cryptogram; F(X) is the number of times a particular crypt symbol appears in the cryptogram; F(y) is the number of times a particular plaintext symbol appears in 1000 words of plaintext (according to Fletcher Pratt [8]). Then W(X,y) is the weight of crypt symbol X evaluated as plaintext symbol y. It is found from:

$$W(X,y) = (F(X)*1000 - F(y)*L)/100$$

If L = 50, F(X) = 7, and y = e, then

$$W(X,e) = (7*1000 - 131*50)/100 = 61$$

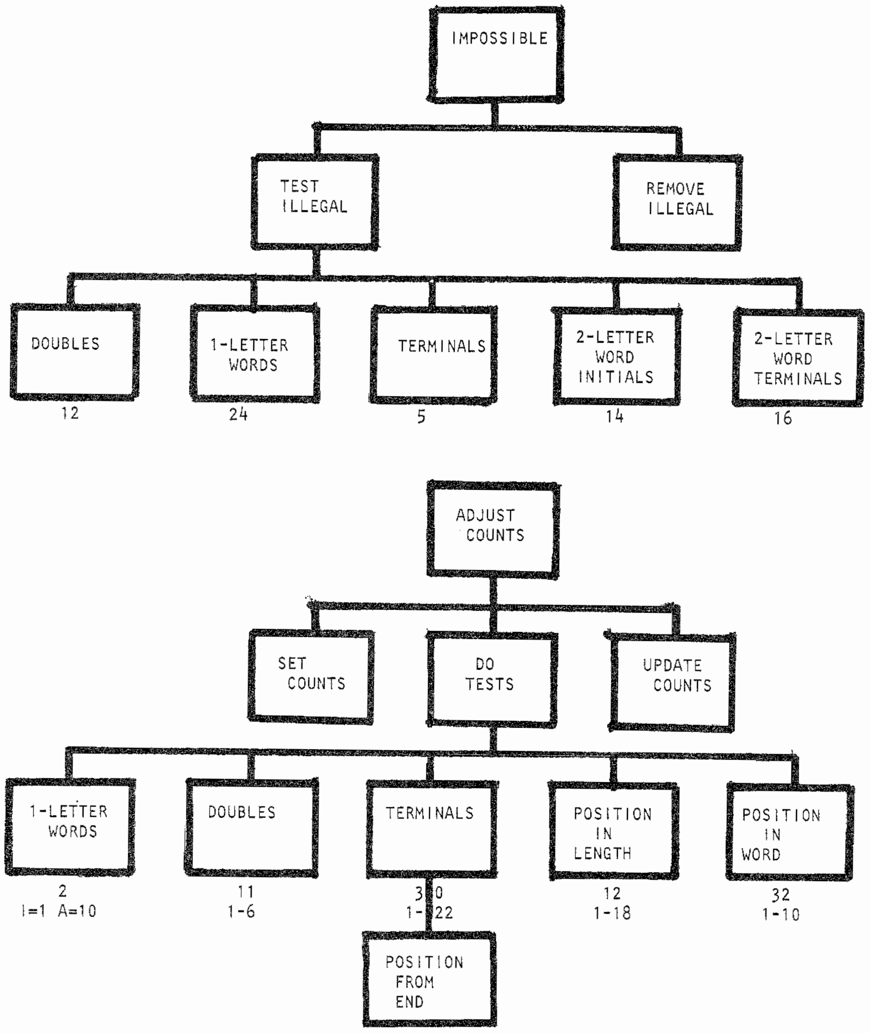

Figure 5. Structure charts expanding the IMPOSSIBLE and ADJUST_COUNTS modules of the DECODE program.

Here is a sample list; there are initially 26 of them, one for every crypt symbol.

y	F(y)	W(X,y)
e	131	61
t	105	34
o	80	10
—	—	—
z	1	-60

The program suggests to the user a plaintext equivalent for the crypt symbol having the shortest list (i.e., the one from which the greatest number of "impossible" equivalences have been removed); on this list the plaintext equivalent is taken as the one with the highest W(X,y) value. In this case, the program would suggest that X equals e.

Figure 6 expands four key modules of the RESTORE-CHOOSE program: INQUIRE, USER_GUESS, DIGRAPH_TEST, and TRIGRAPH_TEST.

In our expert system we made use of character-occurrence statistics from both Pratt [8] and Gaines [5].

We regarded 12 alphabetic characters as being "impossible" doubles; 24 as "impossible" one-letter words; 5 as "impossible" terminals; 14 as "never" occurring as initial characters of two-letter words; and 16 as "never" occurring as terminal characters of two letter words.

A crypt character appearing as a one-letter word was given one added weight count for being an "i" and 10 added weight counts for being an "a". Eleven plaintext equivalents were given added weight counts when crypt characters appeared doubled; counts ranged from 1 to 6.

Added counts for terminals and initials were assigned to 30 plaintext characters (some got points in both cases); counts ranged from 1 to 22. There were 12 cases of positions from the front of a word with counts ranging from 1 to 18; and 32 cases of positions from the end, with counts ranging from 1 to 10.

We regarded 286 digraphs as "illegal" combinations and 163 as "common" digraphs, there were awarded from 1 to 17 bonus points. We regarded 477 trigraphs as "common" and awarded from 1 to 50 bonus points.

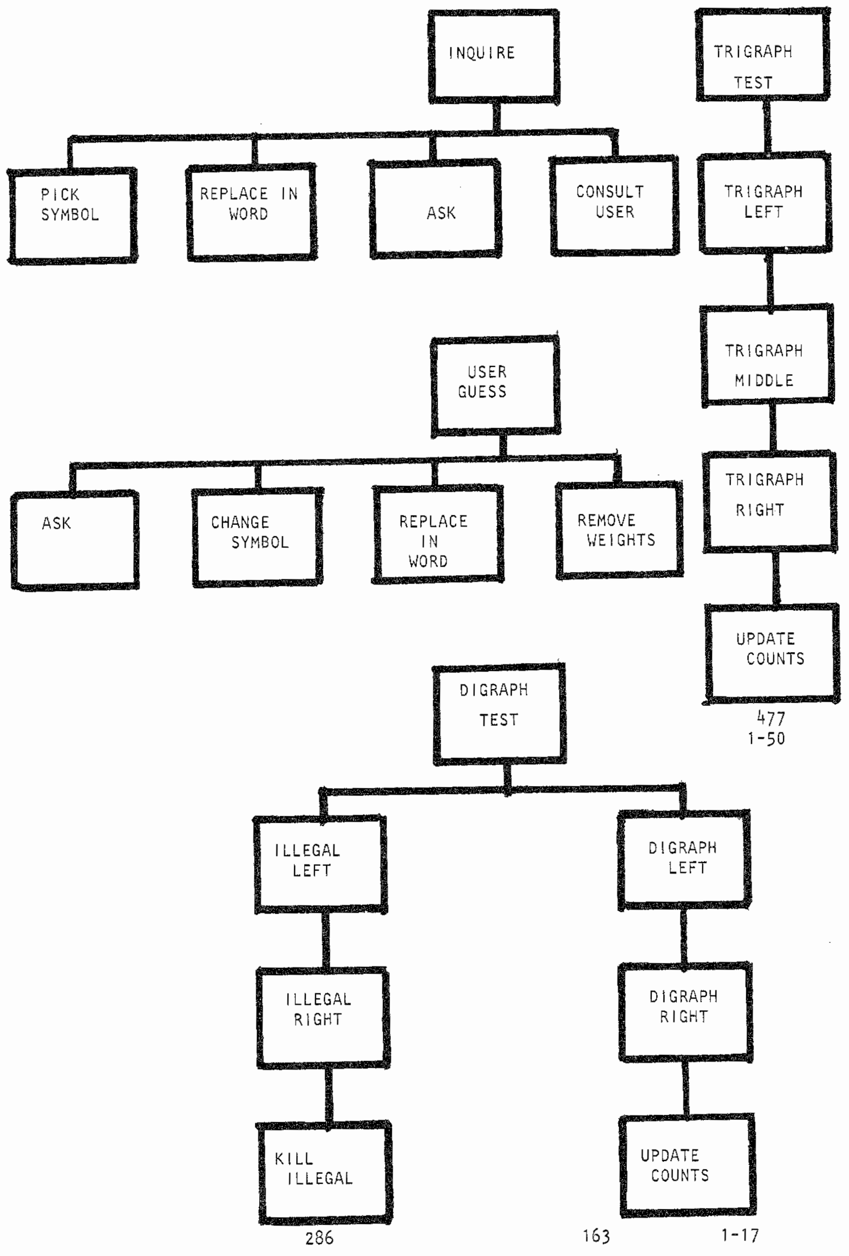

Figure 6. Structure charts expanding the INQUIRE, USER_GUESS, DIGRAPH_TEST, and TRIGRAPH modules of the RESTORE_CHOOSE program.

The weakest part of this procedure is the selection of weights to assign to symbol occurrences and co-occurrences. No set of tables is reliable for all languages nor even for all variants of one language. However, the cryptanalyst can usually rely on exogenous information to suggest what kind of plaintext went into the cryptogram (military, telegraphic, literary, computer programs, etc.); then acquire a corpus of corresponding text and run statistical programs to particularize the schedule of weights.

PROGRAM DETAILS

Decode is a PROLOG program that tries to incorporate the knowledge of a cryptanalyst in breaking simple substitution ciphers. This program uses traditional methods of decryption such as frequency counts, double letters, and letter position. As well, it employs some newer tactics such as common digraphs and trigraphs, and illegal character positions, doubles, and digraphs. The process of decryption is broken into two parts, a static part in which evaluation of the encrypted text occurs, and an interactive part where guesses are made as to the plain text substitutions. In the static part of the decryption process, frequency counts are made and probabilities are calculated as to the likely plaintext identity of each cipher letter. These probabilities are based on the positions in which the cipher letters are found in the crypt text and whether they are found doubled. The interactive part of the program determines the most likely substitutions and presents these to the user. Here the user may either accept them, reject them or substitute on his own. As each substitution is made, the system determines what new information it has gained in the areas of possible trigraphs and digraphs and updates its probabilities.

PROLOG is well suited for implementing a system such as this. It provides features that allow the easy manipulation of strings (lists) and the representation of knowledge about the decryption problem. On the other hand it has the drawback of being relatively inefficient. Because of limitations imposed by the system that was used, we had to simplify the program at the cost of some useful features

DECRYPTION METHOD

The usual method of solving a simple substitution cipher is based on frequency counts and a visual analysis of the crypt text. The cryptanalyst pays special attention to letters that appear doubled or frequently as initial or terminal letters of words. He will proceed by trying to find letters that are easy to identify and will thereby lead to the identification of more obscure letters.

Two easy plaintext letters are 'e' and 't' as they both appear with high frequency as terminal and initial letters respectively. (Note that plaintext letters or substitutions will always be shown in lower case, while the crypt letters will be shown in upper case.) Once some easy letters have been identified it may become easier to find other letters that appear in some combination with them. Examples are 'h' and 'r' which form the common digraphs "th" and "er". Other letters are sometimes found by examining other positions of letters, for example the letter 's' is found commonly as the second last letter of a word. This can be seen in the word endings "st", "se" and "sh". As well, a common second letter is 'h' which is found in words starting with "th", "sh", and "ch". Thus the position of letters in words can give a lot of information as to their identities. A final common clue that is useful in this type of decryption is the occurrence of double letters. A letter doubled in the middle of a four letter word is almost always either an 'o', as in the word "look", or an 'e' as in "keep". This information along with that obtained from the analysis of frequencies and letter positions can usually lead to the identification of at least one or two of the more common letters. Once the cryptanalyst has determined a couple of letters using these methods, it becomes easier to find the identities of the less common ones. This is done by observing that certain letters appear commonly in digraphs or trigraphs.

PROGRAM IMPLEMENTATION

The decryption program tries to accurately represent the process performed by the cryptanalyst. For each crypt letter, it maintains a list of all plaintext letters that it may represent. Paired with each plaintext letter is a value proportional to the probability that it is the proper substitution for the crypt letter. These "probabilities" will be referred to as weights from here on. The weights are initially obtained by counting the frequency of the crypt letter in the crypt text and comparing it to observed frequencies of letters in plaintext. Each crypt letter and its frequency are brought to a common denominator and compared with the observed frequency of every plaintext letter in a string of one thousand characters. A formula involving the relative frequency of the plaintext letter and the frequency of the crypttext letter and the difference between the two is used to determine the initial value of the weight. At the end of the frequency counting portion of the program, there is a list for each crypt letter type in the crypt text. These lists contain 26 entries, one for each plaintext letter of the alphabet. Each entry consists of the plaintext letter and its weight, which is indicative of the probability that it is the proper substitution for the crypt-text letter based on frequency-count evidence alone.

The next part of the process is to remove entries from the weight lists that can not possibly be proper subsitutions for the crypt letters. Entries can be removed for three reasons. The first of these is the most useful; it involves crypt letters that are doubled. Less than half of the letters in the alphabet ever appear doubled. Thus if a crypt letter appears in the crypt text as a double, we can remove all the entries in its weight list that never are doubled in plaintext. The second reason that an entry can be removed from the weight list is similar and involves terminal letters. There are several letters in English that seldom or never appear as terminals and thus may be removed from the weights of some crypt letters. The final case involves letters that appear as single letter words, of which there are two, 'a' and 'i'. Thus if a crypt letter appears as a single letter word, all the entries except those for 'a' and 'i' are removed from its weight list.

Once the impossible letters are removed from the weight lists, the next step is to use more information about the English language that may help in the decryption. This process involves adding weight to some of the entries in the lists to show that one substitution may be more likely than another. The weights are altered according to statistics obtained from studies of English text. These statistics show that certain letters appear in certain places in words with a certain frequency. Thus if a crypt letter is found to occur at the beginning of a word, the weights of each of the plaintext letters are increased by a value that represents the number of times the letter appears as an initial character in English. Thus, for example, if the crypt letter 'B' appears as the first letter in a word of cipher text, the weight for the plain letter 't' will be incremented by a substantial amount reflecting the fact that 't' occurs often as an initial letter. On the other hand, the weight for the plaintext letter 'x' will hardly be increased at all because 'x' is almost never an initial character. All the other weights in the list will also be increased accordingly. The weight lists will also be adjusted in the same manner for crypt letters that appear as terminals, doubles, and at other key locations in words.

Once the static analysis of the crypt text has been performed, the program begins to make suggestions. These suggestions are based on a simple algorithm that tries to determine the most likely substitution. The algorithm defines the best possible guess to be the plaintext letter with the highest weight in the shortest weight list. The reason for this choice is based on the premise that if a substitution is made from the shortest list, it has a greater probability of being correct. Therefore the choice with the greatest weight in the shortest list is the most likely substitution. When the program suggests a substitution to the user, he has the option to either accept it, reject it, or make a substitution of his own. If the suggestion is rejected, the system will try to make the next best suggestion. The process in which the program produces suggestions is part of the PROLOG backtracking procedure. This causes it to produce alternative suggestions and to also possibly repeat previous guesses. This is actually a desirable attribute because it allows the user to review different alternatives while not totally discarding a rejected suggestion.

When the user accepts a guess or makes his own, a number of functions are performed by the system. The first is to remove the weight list for the substituted crypt letter from its database. This has the effect of preventing any further action regarding that guess. In addition, all plaintext weight pairs are removed from the rest of the weight lists for the plaintext letter involved in the substitution. Again this stops the program from suggesting the plaintext letter in any further guesses. At this point the system assumes that a correct substitution has been made and goes on to analyze the new information that it has gained from it. This new information deals with how the guessed plaintext letter effects the weight lists of the crypt letters that are found in proximity to it in the text. This is where the digraphs and trigraphs come into play. If, for example, we have just guessed the plaintext letter 't' and it is found to precede the crypt letter 'X' in the text. We can increase the weight for the plain letter 'h' in the weight list for 'X'. This is done because "th" is a very common digraph. The amount that the weight for 'h' is increased by is based on the frequency of "th" in the English language. The process of updating weights for digraphs and trigraphs is similar to the process used in the static analysis of the text. Another and possibly more important function that is performed here is the removal of plaintext letters from weight lists for crypt letters that would form an illegal digraph with a guessed plaintext letter. This is done because some letters never appear together, such as "tk". Therefore in our previous example the plaintext letter 'k' and it's corresponding weight can be removed from the weight list of the crypt letter 'X'. This is done because 'X' was found to follow the plain letter 't' in the text. Removing entries from weight lists in this fashion has the effect of increasing the probability that one of the remaining letters in the list is the correct guess and reduces the amount of work that must be done by the system. The process of making suggestions and substitutions continues until either all of the crypt suggestions have been substituted, or all possible suggestions have been rejected.

LIMITATIONS

The PROLOG system depends extensively on a run time stack. This allows it to do one of its major features called backtracking. Backtracking happens when the PROLOG system attempts to prove a clause and fails. The system starts to "undo" work that it has already accomplished and "redo" it in an effort to solve the problem in a different way. This can be seen using an analogy of a maze. A person attempting to find his way through a maze will follow an arbitrary path until he finds his way out or he is blocked. If the person is blocked, he must backtrack or retrace his steps until he returns to an intersection. At the intersection, he will take an alternative path and continue until he is again blocked or he is successful. The PROLOG system works much like a person finding his way through a maze. A problem, in this case decryption, is like the maze and the steps or clauses that must be proven are the paths and intersections. As the PROLOG interpreter attempts to solve the problem, it must solve the clauses or follow the paths. When it fails to prove one of the clauses, it must backtrack and follow a different path by proving a clause differently. In order for the system to remember where it has been, it must save a map of the paths that it has followed. This is done by pushing clauses onto the stack as it proceeds. Thus as the system gets deeper and deeper into a problem, the stack grows very large.

In the PROLOG system that this program was implemented on, the stack size was limited to 64,000 bytes of memory. Since the program must continually pass on a large amount of information as it solves a cipher, this amount of space is not nearly enough. Fortunately PROLOG provides a feature, called a cut, which allows the programmer to keep the stack usage at a minimum. This cut has the effect of committing the search to the path it has followed to that point. This can be thought of as putting a sign at a point on the path that says "NO RETURN BEYOND THIS POINT." Thus if we try to backtrack past the sign, we have failed in our search. In the PROLOG system this has the effect of reducing the stack back to the start of the current search. In the decryption program a cut was executed every time a guess was accepted. Thus once we have guessed a letter, we are stuck with it. In general this is not much of an obstacle and even when a wrong guess has been made, it is usually still possible to solve the cipher.

The other major problem that we encountered was that the PROLOG system is very inefficient and time consuming. This is evident in the fact that the static part of the process usually takes about one hour to run for a cipher of about 50 characters. The interactive part can take up to 10 minutes between guesses. Unfortunately it is not possible to improve the performance of the system to any great extent due to the amount of work it must perform to complete the task. However we are happy to report that although the program is slow, it has been very successful at breaking the ciphers that it was designed to do.

POLYALPHABETIC CIPHERS

The first step in cryptanalysis of a polyalphabetic cipher is determining the number of crypto alphabets or length of key. We felt the classical Kasiski analysis as described in Denning [3] was too difficult to program and tended to be indecisive in many instances. Instead we opted to use an adaptation of the log-sort-fit technique that Kahn [6] describes as being used to differentiate between transposition and substitution ciphers.

We assumed one, two, three, four, five etc. alphabets. For all text enciphered in a given alphabet, we performed a symbol frequency count and arranged the counts in order of decreasing value. Then we took the natural logarithm of each value; where the count was zero we left it alone. Next we performed a simple linear Gaussian regression [1] where the X values were the numbers 1-26 and the Y values were the logarithms of the sorted symbol frequency counts.

```
1520 '
1530 ' LINEAR REGRESSION/CORRELATION SUBROUTINE
1540 '
1550 ' Y = RESULT(I,J): X = J
1560 SUMX = 0: SUMY = 0: SUMSQX = 0: SUMSQY = 0: SUMXY = 0
1570 FOR J = 1 TO 26
1580 IF RESULT(I,J) > 0 THEN RESULT(I,J) = LOG(RESULT(I,J))
1590 SUMY = SUMY + RESULT(I,J)
1600 SUMX = SUMX + J
1610 SUMSQY = SUMSQY + RESULT(I,J)*RESULT(I,J)
1620 SUMSQX = SUMSQX + J*J
1630 SUMXY = SUMXY + RESULT(I,J)*J
1640 NEXT J
1650 EXX = SUMX/26: EXY = SUMY/26
1660 SDX = SQR((1/26)*(SUMSQX - 26*EXX*EXX))
1670 SDY = SQR((1/26)*(SUMSQY - 26*EXY*EXY))
1680 COVXY = (1/26)*(SUMXY - 26*EXX*EXY)
1690 R(I) = COVXY/(SDX*SDY)
1700 B(I) = (R(I)*SDY)/SDX
1710 A(I) = EXY - B(I)*EXX
1720 SY(I) = SQR(SDY*SDY*(1 - R(I)*R(I)))
1730 RETURN
```

Figure 7. BASIC code of the LINEAR REGRESSION/CORRELATION subroutine of the POLYALPHABETIC ANALYSIS program.

The analysis program is written in MS/BASIC. It is shown in Figure 7. We calculated all the usual regression parameters: Mean Y (EXY), Standard Deviation Y (SDY), Covariance XY (COVXY), Regression Coefficient (R(I)), Intercept (A(I)), and the Standard Deviation of the estimate of Y (SY(I)). The Slope and Covariance proved to be the most interesting.

Given a 4-alphabet substitution cipher, we obtained the following results:

Guess = 4

Alphabet #	Slope	Covariance
1	-.1380	-7.7646
2	-.1383	-7.7796
3	-.1255	-7.0604
4	-.1364	-7.6718

Guess = 3

Alphabet #	Slope	Covariance
1	-.1071	-6.0224
2	-.0931	-5.2359
3	-.0771	-4.3386

Guess = 5

Alphabet #	Slope	Covariance
1	-.0747	-4.2015
2	-.0908	-5.1103
3	-.1018	-5.7288
4	-.0882	-4.9630
5	-.0844	-4.7500

These results suggest that, given a large amount of crypto text, the slope and covariance will attain minima and exhibit their greatest uniformity (except for guesses 1 and 2) when the cryptanalyst guesses the actual number of cipher alphabets.

Another way to determine the number of cipher alphabets is to use the index of coincidence (IC). See reference [3]. The IC is calculated over the total crypto text so it is more economical to compute than the regression covariance (RC). The IC varies from 0.66 (one cipher alphabet) to 0.38 (infinite key). The plot of the cipher alphabet is roughly a negative exponential. While it is easy to discriminate among 1, 2, or 3 alphabets, this indicator becomes ambiguous for 4 (IC = 0.45), 5 (IC = 0.44) or more alphabets (IC = .041 for 10 alphabets). In such cases, cryptanalysts must rely on Kasiski analysis to resolve doubt.

In a way, this is fortuitous because the IC and RC can work together. The IC can easily resolve the 1 and 2 alphabet cases, in which the RC behaves poorly.

One can postulate another area of cooperation: breaking product ciphers. Suppose a message is enciphered first by substitution using N alphabets then by one of the 32 regular two-dimensional route transposition ciphers (24 matrix, 8 rail fence; neglecting the constants for length and width of matrix and depth of fence). One can use the IC on the total crypto text to estimate the number of cipher alphabets. Then one can reverse each regular transformation and test using the RC to determine which is correct. After that, the cryptogram can be solved using forked processes - possibly incorporating fuzzy digraph analysis.

Solutions can also be achieved for product ciphers using pre-substitution transposition or both pre- and post-substitution transpositions, but each of these embellishments presents a quantitative leap in difficulty with a consequent requirement for a larger quantity of encrypted text on which to work.

CONCLUSION

Given a sufficiently large amount of text it should be possible to reduce any fixed-key polyalphabetic cipher to a finite number of simple substitution ciphers. Then, given sufficiently prodigious computational capability, it should be possible to use a modification of our expert system to solve enough of these simple substitution ciphers to infer the content and meaning of the text; and over time, determine the cipher mechanism and keying schedules.

For this reason, any fixed-key substitution cipher should be regarded as potentially breakable by a cipher-text-only attack. We have had no direct experience with running-key polyalphabetic ciphers, but we tend to distrust them as well. For critical applications, it would seem prudent to rely only on Vernan [3], multiple key ciphers [2], or product ciphers with randomly selectable transpositions, substitutions, and key progressions.

ACKNOWLEDGMENT

This research was supported by the National Scientific and Engineering Research Council (Canada) under Grant #A-7132. Their support is gratefully acknowledged.

REFERENCES

1. Burington, R. S. and D. C. May. 1958. Handbook of Probability and Statistics With Tables. Sandusky OH: Handbook Publishers, Inc.

2. Carroll, J. M. 1984. The Resurrection of Multiple-Key Ciphers. Cryptologia. 8: 262-265.

3. Denning, D. E. R. 1982. Cryptography and Data Security. Reading MA: Addison-Wesley Publishing Co.

4. Foster, C. C. 1982. Cryptanalysis for Microcomputers. Rochelle PArk NJ: Hayden Book Co.

5. Gaines, H. F. 1956. Cryptanalysis. New York: Dover Publications.

6. Kahn, D. 1967. The Codebreakers. New York: MacMillan Publishing Co.

7. Peleg, S. and A. Rosenfeld. 1979. Breaking Substitution Ciphers Using a Relaxation Algorithm. Communications of the ACM. 22: 598-605.

8. Pratt, F. 1938. Secret and Urgent. Garden City NY: Blue Ribbon Books.