APL Mechanization of Indirect Symmetry

BY WILLIAM E. MAY

Unclassified

This article discusses the use of APL computer language as a tool in the reconstruction of alphabets when the components are unknown sequences.

STATUTORILY EXEMPT

INTRODUCTION

In the Fall 1971, issue of the *Journal* there appeared two sequential articles. "A Cryptanalyst's Nightmare" by L. D. Callimahos and "An Introduction to APL" by [ ] The former, divested of its esculent esoterica, described the recovery of components in a rather complicated polyalphabetic system; the latter was an excellent exposition of a relatively new and very flexible computer programming language. This editorial cohabitation, plus numerous discussions with the author of "Nightmare," led by a rather devious route to some interesting experiments in the use of the APL system as a sort of desk aid in the reconstruction of alphabets in polyalphabetic systems. APL appeared to be particularly well suited for this type of operation, since the volume of input and output was small, and the analyst could "talk" to the computer. This would enable him to stop at any intermediate point when the output was deemed sufficient for his purpose, even though there might be additional work which the computer could do. This will be amply demonstrated in the course of this paper.

BACKGROUND

The theory and application of the principles of indirect symmetry, a technique for exploiting latent relationships between alphabets derived from the same primary component, have been well documented, most recently in Chapters VI and VII of *Military Cryptanalytics, Part II.* Some examples from the text have been used, with permission, to illustrate various points in this paper. Perhaps the best place to begin would be to restate a few definitions which will be used in subsequent paragraphs:

  a. *Primary Component*—A sequence used in conjunction with itself or with a different sequence to encipher plain text. In this paper we concern ourselves with mixed sequences only, i.e., those derived from keywords or by random selection or other means. In most cases

13   UNCLASSIFIED

the sequences are inscribed on sliding strips for ease in manipulation. To encipher a message, an *index letter* (usually A_p or the first letter of the plain component) is set opposite a *key letter* in the cipher component, and the cipher letter then falls opposite the plaintext letter. Designating the plain component 1 and the cipher component 2, we express the enciphering (or deciphering) relationship by the Vigenère equation

$$\theta_{k/2} = \theta_{i/1} \; ; \theta_{p/1} = \theta_{c/2}$$

b. Secondary Alphabet—For any pair of components a number of secondary alphabets can be generated corresponding to the length of the primary components. The cipher component is set against the plain component at each position, and the plain component is then rearranged in normal alphabetic order with the corresponding cipher letters under the plain. To illustrate, suppose the components are the same mixed sequence based on QUESTIONABLY and are aligned with $Q_p = I_c$:

```
P: QUESTIONABLYCDFGHJKMPRVWXZ
C: IONABLYCDFGHJKMPRVWXZQUEST
```

Rearranging the plain component:

```
P: ABCDEFGHIJKLMNOPQRSTUVWXYZ
C: DFJKNMPRLVWGXCYZIQABOUESHT
```

Fig. 1

It can be seen that the realignment destroys most of the similarity between the two components, rendering *latent* those relationships which were previously *patent*. When recovery of unknown components is under way, it is normal to think in terms of the standard alphabet. It is therefore inevitable that assumptions based on letter frequencies or probable words will develop relationships which reveal little or nothing as to the nature of the components, since those relationships are fragments of secondary alphabets. The same remarks hold true when the components are different sequences; the difference is that *only the cipher sequences will be interrelated.* This will be illustrated below.

c. Equivalent Primary Component—A sequence which is cryptographically identical to the original primary component is called an *Equivalent Primary Component* (abbreviated EPC). When both components are known to be the same mixed sequence, an EPC may be formed by chaining together the plain/cipher pairs of any secondary alphabet. For a 26-element sequence this process can yield the following: one of 12 possible 26-letter sequences if the interval between

the plain/cipher letters in the *original* component was odd (except for interval 13); thirteen 2-letter chains if the interval was 13; or two 13-letter chains if the interval was even. Using the example in Fig. 1 and chaining together the pairs $AD \rightarrow DK \rightarrow KW \rightarrow \ldots \rightarrow MX \rightarrow XS \rightarrow SA$, we obtain the sequence

ADKWENCJVUOYHRQILGPZTBFMXS

which yields the QUESTIONABLY...Z sequence when decimated at an interval of −5. But if the two components are different sequences, it is fruitless to chain plain/cipher pairs. Note the following, with a plain component based on HYDRAULIC and the cipher component based on QUESTIONABLY:

```
P:  HYDRAULICBEFGJKMNOPQSTVWXZ
C1: QUESTIONABLYCDFGHJKMPRVWXZ
C2: BLYCDFGHJKMPRVWXZQUESTIONA
```

When rearranged in secondary alphabet form:

```
P:  ABCDEFGHIJKLMNOPQRSTUVWXYZ
C1: TBAELYCQNDFOGHJKMSPRIVWXUZ
C2: DKJYMPRBHVWGXZQUECSTFIONLA
```

A cursory glance will show that there are numerous plain/cipher identities if chaining is attempted; however, the two *cipher* sequences may be chained together to derive the sequence

TVDIFWOGXNHZAJQBKULMEYPSCR

which will decimate to the QUESTIONABLY sequence at an interval of 3. In a nutshell, when in doubt as to whether or not the two components are related, look for symmetry "inside the matrix." Pausing here for a moment to split a hair, we note that unrelated plain and cipher alphabets have one chance in twenty-six of being chainable, but the probability of recovering either of the two original keywords by decimating the resulting sequence is too small to worry about.

APPROACHING THE PROBLEM

When one reads the literature on indirect symmetry, the problem of reducing to machine terms what appears to be the ability to visualize a literal relationship looks almost hopeless, especially when one considers the ramifications introduced by the graphical method of chaining. The analyst's eye is a link to his brain which is not easy to simulate. As a simple example, given the fragment GKQV, the analyst would search other sequences for relationships involving not only GK, KQ and QV but also for G-Q, K-V and G-V. Compare this to the bookkeeping complexities which would arise if one attempted

to mechanize this approach. It was therefore decided that only the simplest relationships would be considered, i.e., adjacent letters in paired form. The decision to abandon the more complex procedure was rationalized by conjecturing that the additional "soup" which might result would hardly be worth the man-hours and machine time required for its production. Somewhat surprisingly, in all problems tested (involving both 26- and 36-element sequences). when the program was run to completion—that is, when no EPC could be improved upon—there was very little more which could be done by hand except to assume the placement of missing letters or to recognize some or all of the keyword. The decision to go the simplest route was thereby rendered wise ex post facto.

It was also realized that some means would have to be devised for stopping the chaining process; otherwise the computer would add redundant pairs to each chain until someone pulled the plug. If the sequence length N were a prime number, only *complete* N-letter chains could be formed, and a simple counting procedure would suffice. However, when N is composite, its factors produce chains of factor lengths; for $N = 36$, for example, there exist chains of lengths 2, 3, 4, 6, 9, 12, 18 and 36. To deal with the 26-letter case, we drew up a table listing the types of chains which had to be recognized to direct the course of the program:

    a. One chain of 27 with letters 1 and 27 identical;

    b. Two chains of 14, with letters 1 and 14 in each identical;

    c. Thirteen chains of 3, with first and third letters in each identical;

    d. One chain of type *b*, and one or more chains of fewer than 13 letters;

    e. One or more chains of type *c*, and one or more chains of 2 letters;

    f. One or more chains of various lengths with no initial and final letter identities.

If types *a, b* or *c* obtain, there is no point in adding to them, since they are already complete. In types *d* and *e*, the complete chains can be ignored, but the others are still subject to possible accretion. In type *f*, no holds are barred, unless or until it evolves into one of the other types.

Another early consideration was how to format the data. As Mr. [   ] pointed out in his article. APL is especially flexible in its ability to handle arrays. It was decided that the most convenient way to enter the relatively small amount of input was to present it in the form of a literal vector. Each fragmented EPC was given a name tag consisting arbitrarily of a letter and two digits, and each EPC was terminated by a plus sign. For the first problem tested (Problem C1 on page 594, *Military Cryptanalytics, Part II*), the set of five EPC's was labeled XXX and the literal vector appeared as follows:

X01 IAZ DO EW TQ NK+X02 AR PL ETNH SIF OM+X03 POW
TU ERC+X04 SC RNX EIV OB+X05 IX RG CP TH NY VL UJ+

Fig. 2

Parenthetically it should be noted that one small bit of help was given to the program in the matter of data preparation. Pairs which were obviously chainable, such as IA and AZ, were linked prior to input; after all, how lazy can you get?

To generalize the program, we decided that processing parameters such as lengths of the various EPC's, number and length of partial chains within each EPC and number and location of spaces within each EPC should be made calculable rather than entered via the keyboard. Thanks to the indexing capability of APL, all requisite information of this nature could be derived from the basic literal vector, as will be shown.

THE PLOT THICKENS

Before delving into the murky depths of the program proper, it may be helpful to list a few of the APL operators which appear throughout the program, with illustrative examples as needed:

ιX—Generate consecutive numbers from 1 to X inclusive

$X\phi Y$—Left rotate Y by X places; $2\phi 1\ 2\ 3\ 4 = 3\ 4\ 1\ 2$

$^{-}X\phi Y$—Right rotate Y by X places

$X\uparrow Y$—Take X elements of Y; $3\uparrow 1\ 2\ 3\ 4\ 5 = 1\ 2\ 3$

$X\downarrow Y$—Drop X elements of Y; $3\downarrow 1\ 2\ 3\ 4\ 5 = 4\ 5$

$\sim X$—Not X

$X\epsilon Y$—Membership of X in set Y; answer to $2\epsilon 1\ 2\ 3$ is "yes"

$X\iota Y$—Location of Y in X; $2\ 7\ 5\ 3\ 9\ \iota 3$ is 4

$+/X$—Summation of X

ρX—Number of elements in X

$X\wedge Y$—Both X and Y

$X\vee Y$—Either X or Y

$|X$—Absolute value of X

$X[Y]$—Yth element of X

$X\leftarrow Y$—Store the value of Y in X

There are many operators which can be used to advantage in combination to provide information necessary for processing. For example. suppose X specifies the location in memory of a literal vector containing an entire message. The analyst may wish to know the number and location of all E's in the message; he can obtain this information by typing $Z\leftarrow\rho Y\leftarrow(X = 'E')/\iota\rho X$. The vector Y will contain the numerical position of all E's in the message, and Z will contain the number thereof. The number of ways in which such combinations may be

17   UNCLASSIFIED

used is large, in many instances enabling the analyst to contain a subroutine in one program statement.

A short preparatory routine is required to establish the initial conditions for the symmetry search. It is shown below in its entirety:

```
            START SET
|1|   K←ρX←(T='+')/ι ρT←SET
|2|   I←B←1
[3]   A←(B−1)φι X
[4]   E←T[ι 3]
[5]   AA:E←E,T[X[I]+ι 3]
|6|   →(K=I←I+1)/AB
|7|   →AA
|8|   AB:HR←I21 .
|9|   F← ι 0
|10|  SYM
```

Fig. 3

The first statement provides two variables needed for processing, viz., K, the number of sequences in the set, and a table, X, which specifies the numerical locations within the literal vector of the plus signs separating the EPC's. From this table can be derived the starting points within the vector of all EPC's as well as their lengths. The third statement builds a table, A, of the sequence numbers which is rotated during the course of processing so that the number of the basic EPC—that is, the one which is being improved—is always the first one in the table. The loop containing statements 5, 6 and 7 builds a table, E, which holds the tags for each EPC in the set. The last statement calls in the main program, and we are off to the races.

SEARCH AND COALESCENCE

From this point on we will use the sequence set XXX (Fig. 2) and show the progress of reconstruction of the sequences at various stages. The first few statements in the main program, SYM, build the sequence-length and chain-length tables, and convert the first EPC into numerical equivalents according to the positions of the letters in the normal alphabet, with 27 as space and 99 added as a terminator. These tables are shown below. TX is the sequence locator table; its entries, when incremented by one, specify the positions of the first tag letters in the literal vector. D is the table of sequence lengths, including tags and separators. UA contains the numerical equivalents of the letters in the basic EPC, and UW shows the lengths of the chains within the basic EPC, i.e., the distances between spaces:

TX	0	20	42	57	75												
D	20	22	15	18	25												
UA	9	1	26	27	4	15	27	5	23	27	20	17	27	14	11	27	99
UW	3	2	2	2	2												

Since the length of UA is subject to increase by the addition of more data during the course of the program, these tables must be regenerated at the beginning of each pass.

Following table generation is the routine which tests for the existence of completed chains within the basic EPC:

[8] $BA:\rightarrow(\sim NX[N]\epsilon UW)/BF$

[9] $\rightarrow(1\neq G\leftarrow_\rho UP\leftarrow((UW\geq NX[N])/\iota_\rho UW))/BB$

[10] $\rightarrow(NX[N]+3=G)/ADJ$

[11] $NJ\leftarrow L\leftarrow 1$

[12] $\rightarrow BD$

[13] $BB:NJ\leftarrow 1$

[14] $BC:L\leftarrow UP[NJ]$

[15] $BD:\rightarrow NX[N]\neq UW[L]/BE$

[16] $Q\leftarrow UV[L]+\iota NX[N]$

[17] $UC\leftarrow UA[Q]$

[18] $\rightarrow(UC[I]\neq UC[NX[N]])/BE$

[19] $UA[Q]\leftarrow SP$

[20] $Y\leftarrow Y,Q$

[21] $H\leftarrow H,UC$

[22] $BE:\rightarrow(G<NJ\leftarrow NJ+1)/BG$

[23] $\rightarrow BC$

[24] $BF:\rightarrow(3<N\leftarrow N+1)/BG$

[25] $\rightarrow BA$

Those familiar with APL may wish to dissect the statements and follow the progress of the operation. For the benefit of others, the routine checks for completion of a chain, and, if one is found, places its elements in temporary storage, labeled UC. The positions in the EPC vector occupied by those elements are then filled with spaces. Since spaces are recognized and ignored during the search process, the completed chains are removed from competition for machine time.

Having thus set aside the completed chains, we are now ready to search the rest of the EPC's for elements to be added to the basic EPC by means of a forward and backward search in the routine below:

[26] $BG:I\leftarrow 1\ 2$

[27] $VB\leftarrow(VA\neq SP)/VA\leftarrow(((_\rho VX)-1)\uparrow VX\leftarrow ALF\iota 4$
$\downarrow T[TX[A[C]]+\iota D[A[C]]]),SP,99$

[28] $BH:UB\leftarrow UA[I]$

```
[29]    →(99εUB)/BN
[30]    →((∼(UBεVB))v(SPεUB))/BM
[31]    X←1+Z←VA ι UB
[32]  BI:→((SPεVA[X])v(2=+/VA[X]εUC))/BO
[33]    →(2=+/VA[X]εUA)/BJ
[34]    →((0=+/VA[X]εUA)v(1=+/VA[X]εUA))/BK
[35]    →BM
[36]  BJ:→(1=|((UA ι VA[X[1]])−(UA ι [X[2]])))/BL
[37]  BK:→(2=+/VA[X]εW)/BL
[38]    W←W,VA[X],SP
[39]  BL:X←X+1
[40]    →(100>|HR−121)/BI
[41]    HR←121,0ρ⎕
[42]    →BI
[43]  BM:I←I+1
[44]    →BH
[45]  BN:→(K<C←C+1)/BT
[46]    →(100>|HR−121)/BG
[47]    HR←121,0ρ⎕
[48]    →BG
[49]  BO:X←Z−1
[50]  BP:→(0εX)/BM
[51]    →((SPεVA[X])v(2=+/VA[X]εUC))/BM
[52]    →(2=+/VA[X]εUA)/BQ
[53]    →((0=+/VA[X]εUA)v(1=+/VA[X]εUA))/BR
[54]    →BS
[55]  BQ:→(1=|((UA ι VA[X[1]])−(UA ι VA[X[2]])))/BS
[56]  BR:→(2=+/VA[X]εW)/BS
[57]    W←W,VA[X],SP
[58]  BS:X←X−1
[59]    →(100>|HR−121)/BP
[60]    HR←121,0ρ⎕
[61]    →BP
[62]  BT:→(0=ρH)/BU
[63]    UA[Y]←H
[64]  BU:→(0=ρW)/ADJ
[65]    F←F,B
[66]    UA←¯1ϕ(1↓(¯1ϕUA)),W
```

As an illustration of what happens during this routine, consider the first EPC. Stripped of its tag and terminator, its basic pairs IA, AZ. DO, EW, TQ and NK are sought in the other four EPC's. When both elements of a pair are found, the elements immediately following or preceding are scrutinized. If one or both of the new elements does not already appear anywhere in the basic EPC, this new relationship is added to a storage vector, W, in which is accumulated all supplemental data for the basic EPC. In this instance only one new pair of letters is found, based on the EPC pair IA. We note that in the second EPC (AR PL ETNH SIF OM), F and R are adjacent, in a forward direction, to I and A: so the pair FR is stored in the vector W. At the end of the first pass we thus have

(EPC 1)
$$UA \quad IAZ \ DO \ EW \ TQ \ NK$$
$$W \quad FR$$

to pass along to the coalescence routine. In a similar manner, with the other EPC's in turn as basics, we obtain the following:

(EPC 2)
$$UA \quad AR \ PL \ ETNH \ SIF \ OM$$
$$W \quad RU \ CV \ HY \ WQ \ QK$$

(EPC 3)
$$UA \quad POW \ TU \ ERC$$
$$W \quad IN \ VX \ HJ \ GP \ DE \ LM \ MQ \ UV$$

(EPC 4)
$$UA \quad SC \ RNX \ EIV \ OB$$
$$W \quad GX \ XL \ WA \ UH \ AT \ TF$$

(EPC 5)
$$UA \quad IX \ RG \ CP \ TH \ NY \ VL \ UJ$$
$$W \quad EN \ XM \ AU$$

The coalescence routine takes care of linking new pairs to the basic EPC by means of common letters or as added members (chains). Linkage is accomplished by recognizing the fact that the first letter of a new chain is the same as the last letter of an existing chain, or that the last letter of a new chain is the same as the first letter of an existing chain. Any additions to the basic EPC mean that the vector must be expanded. This is done by "ditting out" the superseded material and adding the expanded portion at the beginning of the revised EPC together with the tag. The coalescence routine follows:

[67] $CA: \rightarrow (I = \rho P \leftarrow (UA = SP) / \zeta \ \rho UA / CG$

[68] $PD \leftarrow I + PC \leftarrow I + (PB \leftarrow ((1 \downarrow P - 1), \rho UA)) - PA \leftarrow 2 + PZ \leftarrow P - 1$

```
[69]    J—1+I—1
[70]  CB:→(UA[PA[I]]=UA[PB[J]])/CC
[71]     →(UA[PA[J]]=UA[PB[I]])/CD
[72]     →((ρPA)<J←J+1)/CF
[73]     →(100>|HR—121)/CB
[74]     HR←121,0ρ⌈⍳⌉
[75]     →CB
[76]  CC:M←(UA[P[J]+⍳PC[J]]),1↓UA[P[I]+⍳PC[I]]
[77]     ←CE
[78]  CD:M←(UA[P[I]+⍳PC[I]]),1↓UA[P[J]+⍳PC[J]]
[79]  CE:PQ←(PZ[I]+⍳PD[I]),PZ[J]+⍳PD[J]
[80]     UA[PQ]←0
[81]     UA←SP,M,(UA≠0)/UA
[82]     →CA
[83]  CF:→((ρPA)<J←1+I—I+1)/CG
[84]     →CB
[85]  CG:U←E[((B—1)×3)+⍳3],ALF[UA[,'⌐']
[86]     T[TX[B]+⍳D[B]]←'—'
[87]     T←TX[B]ΦT
[88]     →((ρU)=D[B])/CI
[89]     →((ρU)<D[B])/CH
[90]     T←(((ρU)—D[B])ρ'—'),T
[91]     →CI
[92]  CH:T←(D[B]—ρU)↓T
[93]  CI:T←(—TX[B])ΦT
[94]  T[(T='—')/⍳ρT]←U
```

A final subroutine tests for completion of the job and the status of sequence improvement, making adjustments as necessary for continuing or stopping. If all passes have not been completed, the program returns to the aforementioned table regeneration subroutine and the processing cycle is repeated.

```
[95]    ADJ:→(K≥B←B+1)/DB
[96]    'PASS DONE'
[97]    →(0≠F)/DA
[98]    'NO IMPROVEMENT'
[99]    →0
[100]   DA:'SEQU';F;' IMPROVED'
[101]   →0
```

$\lfloor 102 \rfloor$ $DB:A \leftarrow (B-1)\phi \, \iota \, K$
$\lfloor 103 \rfloor$ $I \leftarrow I \; 2$
$\lfloor 104 \rfloor$ $\rightarrow AZ$

DISPLAYING RESULTS

A run is finished when the program 'has attempted to improve each EPC in turn. Results are not printed, although in the first version of the program this was done. It was felt that in most instances a print-out of results at the end of each run was a waste of time and paper, since there is often very little improvement between one run and the next. It was therefore decided to limit the printed output to either

     PASS DONE
     SEQU 2 5 6 8 IMPROVED

or

     PASS DONE
     NO IMPROVEMENT

The latter may indicate either completion or frustration.

To supplement the basic program, a small supplementary program was written which could be used to print the sequences after the completion of a run. It had been observed that during the series of program runs the number of improvable sequences would begin to decline, indicating that the point of diminishing returns had been reached. This usually was the optimum time for the analyst to request a print-out of the results, since it was probable that any further improvement could best be made by reverting to manual methods, if in fact the original alphabet could not be completed merely by recognizing all or part of the keyword. The extra complications that would be introduced by requiring the basic program to assume placement of missing letters are not worth considering.

Since the literal vector. T. which is the source vehicle for the first run, is also the result vehicle, it is possible to initiate subsequent runs by using T as the SET in the header line of the preliminary program. This procedure can best be seen in Fig. 4. which is a set of runs (to completion) on the original XXX set of partial sequences (Fig. 2). The printout of the results is also shown for each run. It can be seen that the analyst could have taken over manually after the second run, since the original sequence is obvious.

FINAL REMARKS

It is almost certain that APL students who feel so inclined could find many ways to improve upon the foregoing program. It should be emphasized that this was an embodiment of an experiment in

illustrating a technique and that it was not intended to be either a perfect job or a production project. It does show that a programming system which is usually thought of as a mathematical tool can also be used to advantage in data manipulation. The purist who rises to this bait and avers that data manipulation *is* a mathematical process is probably right, but the cryppie who sits down with his cross-section paper and pencil to assemble a jumble of letters into a cryptographic entity would find this small comfort.

Using the APL system, we have explored many other types of analytic aid jobs, ranging from matching of transposition columns by means of digraphic weights to generation and search of synoptic tables for placement of generatrices in strip systems. All have proved interesting, and some have provided rather startling results. Some may be worth describing in future articles.

START XXX

PASS DONE
SEQU 1 2 3 4 5 IMPROVED
        PRINT T
X01   IAZ DO EW TQ NK FR+
X02   WQK ETNHY ARU PL SIF OM CV+
X03   LMQ DERC TUVX GPOW IN HJ+
X04   WATF RNXL SC EIV OB GY UH+
X05   AUJ ENY IXM RG CP TH VL+

        START T
PASS DONE
SEQU 1 2 3 4 5 IMPROVED
        PRINT T
X01   VTQ IAZ DO EW NK FR XU MJ+
X02   SIFX ARUGJ WQK ETNHY PL OM CV+
X03   BAK INGPOW STUVXY DERCFHJLMQ+
X04   KUHOBSCGYM ZQRNXL EIVJWATFP+
X05   DIXM ENYQCPBTHWKVL ZSFOAUJ RG+

        START T
PASS DONE
SEQU 1 2 3 4 5 IMPROVED
        PRINT T
X01   XUSMJFRDOGIAZ YVTQLHCEWPNKB+
X02   LBETNHYWQKCVPL DSIFXOMARUGJZ+
X03   WDERCFHJLMQSTUVXYZBAKINGPOW+
X04   LDKUHOBSCGYMEIVJWATFPZQRNXL+
X05   LENYQCPBTHWKVL MRGZSFOAUJDIXM+

        START T
PASS DONE
SEQU 1 2 IMPROVED
        PRINT T
X01   BYVTQLHCEWPNKB ZXUSMJFRDOGIAZ+
X02   ZDSIFXOMARUGJZ LBETNHYWQKCVPL+
X03   WDERCFHJLMQSTUVXYZBAKINGPOW+
X04   LDKUHOBSCGYMEIVJWATFPZQRNXL+
X05   LENYQCPBTHWKVL MRGZSFOAUJDIXM+

        START T

PASS DONE
NO IMPROVEMENT

Fig. 4