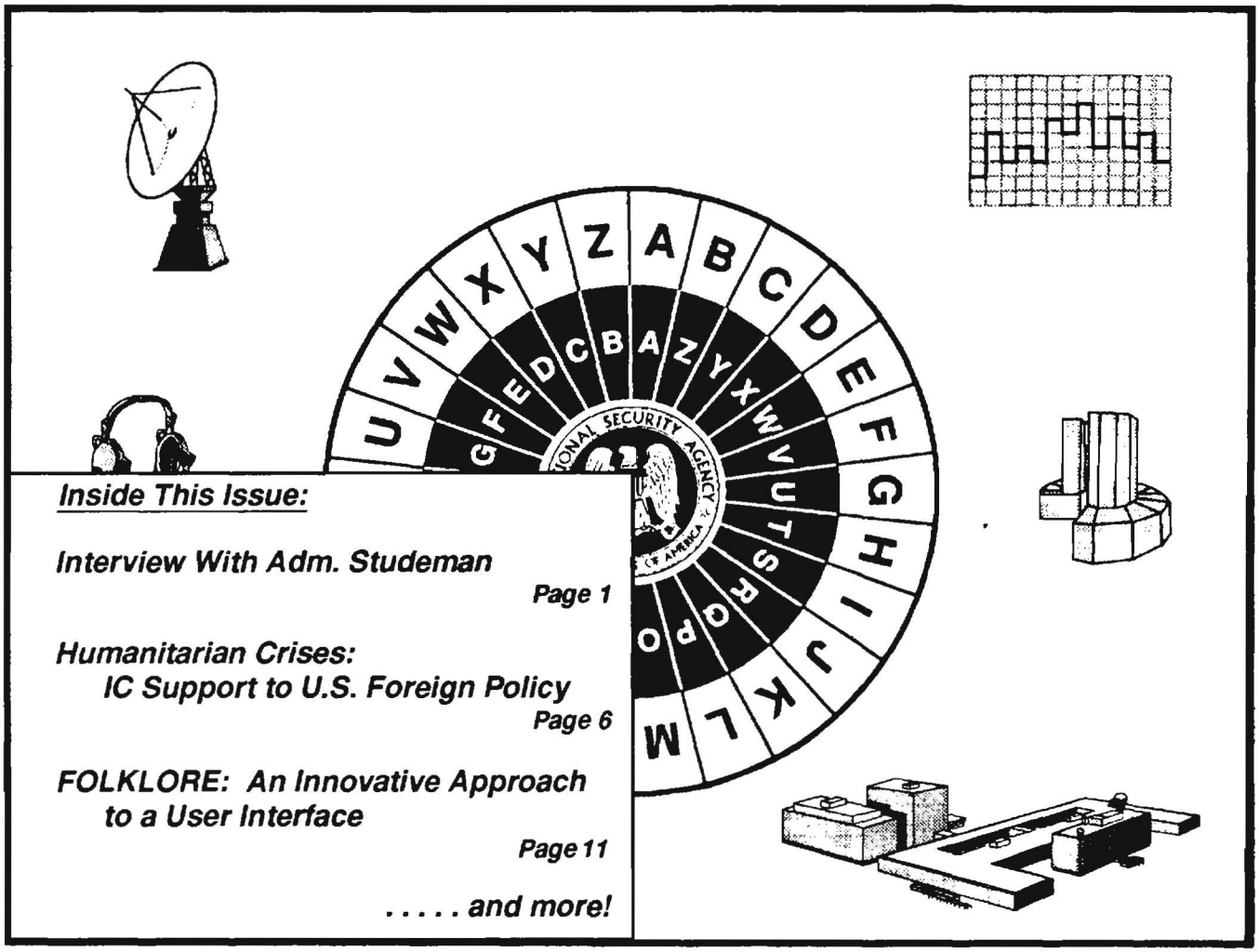
NATIONAL SECURITY AGENCY

CRYPTOLOG

The Journal of Technical Health

Vol. XXII, No. 4 WINTER 1996



Inside This Issue:

~~(FOUO)~~ *As the following article explains, FOLKLORE had its origins in IDASYS, developed back in the 1960's. A small team of operating systems specialists from NSA took IDASYS and molded it into an "industrial-strength" system on NSA's supercomputers of the time, mainly in support of the cryptanalytic community of users. A richly interactive and highly responsive system, it was a standout during an era of batch-oriented systems. It served a specialized set of NSA users for 25 years before the last FOLKLORE system was recently retired. It provided capabilities and features that some believe are still unmatched today, although it finally had to give way to the fast pace of technological change and the considerable resources that industry eventually applied to the supercomputing arena. There are still some old-timers around who can tell war stories from the FOLKLORE era and the successes that it enabled. FOLKLORE is a significant and rich part of NSA's heritage.*

[redacted] *DO Chief Information Officer and chief of E Group (DO Information Technology Applications Development and Support), participated in the development of NSA's supercomputer development; he was the first chief of the division that took over support for and maintained FOLKLORE.*

P.L. 86-36

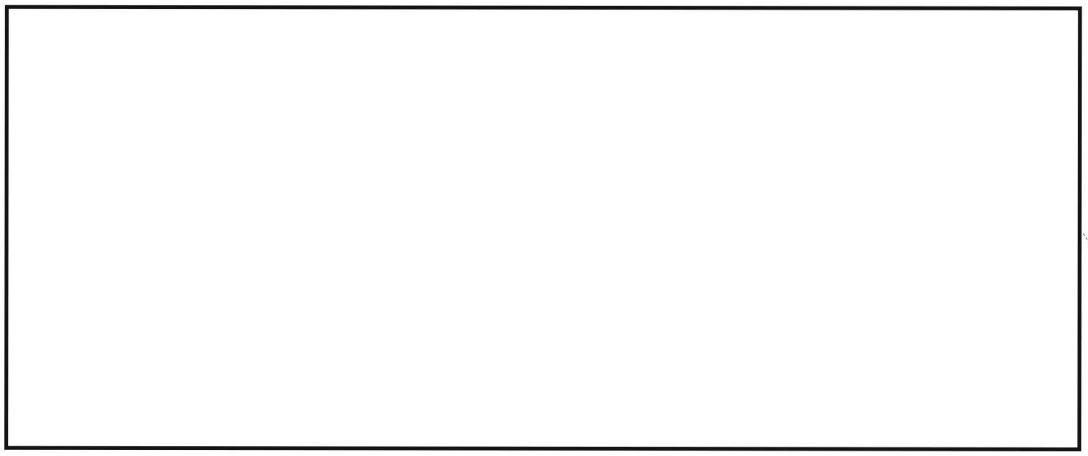
FOLKLORE:
An Innovative Approach To A User Interface (U)

by [redacted]

(U) The purpose of this article is to provide a historical perspective on the user interface characteristics of a 1970's operating system which was very responsive to its customer needs, extremely user-friendly, and anticipated many of the features that are common place today. Most computer operating systems developed prior to 1980 (before windowing systems) presented the user with a command prompt. Editors had to be explicitly executed. In contrast, the FOLKLORE operating system took a different approach. The edit and command modes were one and the same! All interaction was full-screen, not one line at a time!

(U) The IDASYS operating system (as FOLKLORE was known in the beginning) was developed in the late 1960s at the Communications Research Division of the Institute for Defense Analyses (CRD/IDA), Princeton, NJ. It was a highly interactive, multi-user system for the supercomputer of the time (CDC 6600). IDASYS was designed as a supercomputer operating system to provide full supercomputer responsiveness to the user. The target user population was the IDA and NSA cryptanalytic community. IDASYS was renamed FOLKLORE in the late 1970's when NSA took over full support and maintenance of the operating system. FOLKLORE was easy to learn to use for both end-users and software developers. It allowed a lot of flexibility and creativity to be put to productive use rapidly. FOLKLORE survived over twenty-five years. On January 31, 1996, still popular with its users and running on multi-CPU vector processor systems (Cray X-MP), the last FOLKLORE system was powered down.

(U) The user interacted with FOLKLORE through a terminal (CRT and keyboard) using a full screen display for both editing files and executing commands. In the early days this terminal was a directly connected CDC 210 terminal. Soon a Raytheon Programmable Terminal System (PTS) replaced the 210. The PTS terminal concentrators were eventually networked to allow access to multiple systems from one terminal. Finally, in the late 1980s, networked IBM ATs and SUN systems were used with a PTS terminal-emulation window. This last development enabled FOLKLORE for the first time to display high-resolution graphics on the user's terminal.



P.L. 86-36

(U) Since the PTS handled the character display and cursor movement, the supercomputer was free to handle only function key actions. This greatly reduced the number of interrupts that the FOLKLORE operating system had to handle. A FOLKLORE design goal was to deliver the full power (99%) of the system to the users and respond as if it were a single-user system. The PTS contributed a great deal toward meeting this goal.

(U) The window size was a maximum of 22 lines long and 80 characters wide. The lower two lines generally contained information such as file/program name, keyword and line number. This made the effective browsing window 20 lines. The FOLKLORE system input/output (I/O) functions for terminals were designed to make displaying 20 lines of data and/or two lines of information easy. Because the system I/O functions were readily available to FOLKLORE software developers, they could write interactive programs easily at a time when most users were interacting through decks of cards. Terminal I/O was a matter of filling a buffer with the data (which would appear on the screen) and issuing the function call. Other systems which had terminal access were generally graphics- or line-oriented and did not have many interactive programs except a few text editors and applications written by expert programmers.

(U) The FOLKLORE operating system handled all function key stokes simply by storing the value of the function key and some cursor-related information in a table. Programs then checked that table or asked to be interrupted when the relevant table entry was filled. The STOP key generated a program interrupt which could be handled via an error/interrupt handler or, as the default, the system would terminate the program and return the user to the editing state viewing a diagnostic page in a system file. FOLKLORE had a diagnostic file which contained an appropriate message, picture, or instructions with one page for every system error number. The FOLKLORE editor response to the STOP key was to display the beginning of a user-defined default file. Program error handlers often set the user environment to display an error message and allow quick access to the program output file.

(U) FOLKLORE function keys had names, not F1, F2,..., but STOP, GO, +PAGE, -LINE,... This encouraged software developers to use the same key for similar functions. The labels helped the user to remember which key did what. FOLKLORE software developers also made use of simple features which allowed them quick access to source code for almost every program, especially system-provided programs. Also, most programs could easily be called as subroutines. When a function similar to another was needed, it was simple to discover the underlying source, check related documentation for possible parameters, and copy as much or as little as desired. Much FOLKLORE code was reused because it was easy to do so.

(U) Upon successful login to a FOLKLORE terminal, the user generally saw the system news file. New information was placed at the beginning of this file so that the user would see it as soon as the logon process was completed. At this point the user could browse this file, type a file name to browse/edit another file, type a command to execute a program, or use one of the function keys as a shortcut to executing a command or editing a file. Whether a user could actually modify the file that is being viewed depends on whether the user has write-access to the file. FOLKLORE file access control has been covered in a separate paper, *FOLKLORE: One Approach to Security.*[1]

(U) The FOLKLORE editor, program loader and batch command processor were tightly integrated to

1. *Cryptologic Quarterly* Fall 1994, Vol. 13, No. 3.

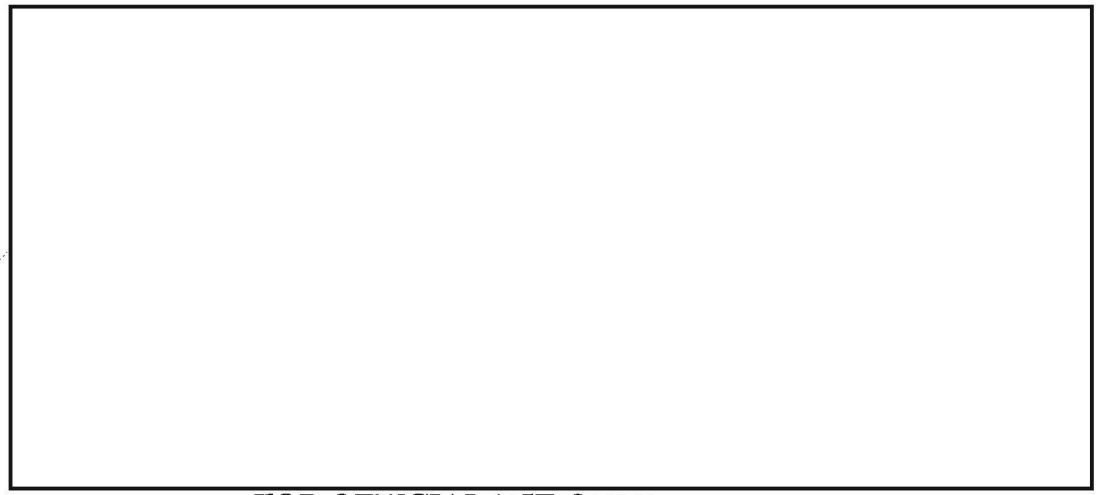
create a unified user interface. Although there were multiple editing processes, they all performed similarly. The actual process in use was determined by the function requested by the user, not by a specific command. Most of the editing services were provided through a single system daemon which handled every terminal (except the system console). For the purposes of this paper, all of the editing processes will be referred to as the FOLKLORE editor. This was the FOLKLORE interface. Every user interacted with it.

(U) The program loader was invisible to the FOLKLORE user. Commands were entered, the GO function key pressed, and the user was back (immediately in most cases) interacting with the editor usually viewing the program output. It did not matter whether the file associated with the command was a fully linked executable, a relocatable version of a main program, or a relocatable subroutine/utility. The program loader "automagically" figured out what to do and it happened quickly.

(U) Initially the FOLKLORE batch command handled only a serial sequence of commands, but it evolved to provide several types of error handling, nested sequences, and the usual logic constructs of today's command script languages such as the UNIX shells. The batch process basically passed each command line to the program loader just as the editor did and the editor returned control to the batch processor when the command completed. The user could tell what was currently being executed because the program loader displayed the program name in the information lines at the bottom of the display window.

(U) FOLKLORE did allow users to customize a few things. The user specified a file to be displayed when the STOP function key was pressed, a default file to be displayed via the ALT function key, a command to execute via the PROG key (to save typing a highly used program name), strides for the -PAGE, -LINE, +LINE, and +PAGE function keys, TAB stop settings, and a file to contain setup information. This setup file contained such information as a list of files to checksum, print header and classification definitions, and lists of files used to build and include program libraries. This customization information could be specified for multiple alteregos, so that the user could change his environment by changing the alterego that he or she was running under. (Alteregos are explained in the previously referenced article, *FOLKLORE: One Approach to Security.*)

(U) A design decision for FOLKLORE allowed unique features to be provided by the editor at very low cost. That is, FOLKLORE text files are contiguous files with an end-of-text string. There were no carriage return or tab characters embedded in the file. These keyboard keys were simply cursor movement keys. The carriage return moved to the beginning of the next line and the tab key moved to the next TAB stop. At the bottom of the display, all cursor movement keys wrapped to the top of the screen. In fact, all FOLKLORE files are contiguous files with no structure except that supplied by the application that produced it. The FOLKLORE editor can be used on any file. Of course, editing an executable file was something best done carefully, but browsing one could be quite useful. Since the FOLKLORE editor does not use carriage returns, but rather blank fills each line to the specified file width (usually 80 characters) some space may be considered wasted. This space was a small price to pay for the cryptanalytic tools that the editor could provide. The block text manipulation feature (BLK) was one very powerful tool. This was implemented in the mid-1980s through a function key and allowed manipulation of text in a rectangle of any dimensions. Common BLK functions were to move, copy, or delete columns of text.



(U) One FOLKLORE function allowed the user to browse a document's index and move directly to a section of the document, in a fashion similar to using today's hyper-linked documents.

Another tool that depended on the fixed width lines was a KEY function which allowed searching quickly down a column. This function was used heavily for searching for left-justified keywords such as message starts. Another use was in browsing the index of a document, moving directly to a section of the document simply by putting the cursor on the line containing the title of interest and pressing the KEY function key. This is similar to using a mouse today with hyper-linked documents, but the cryptanalytic community could do this in 1970!

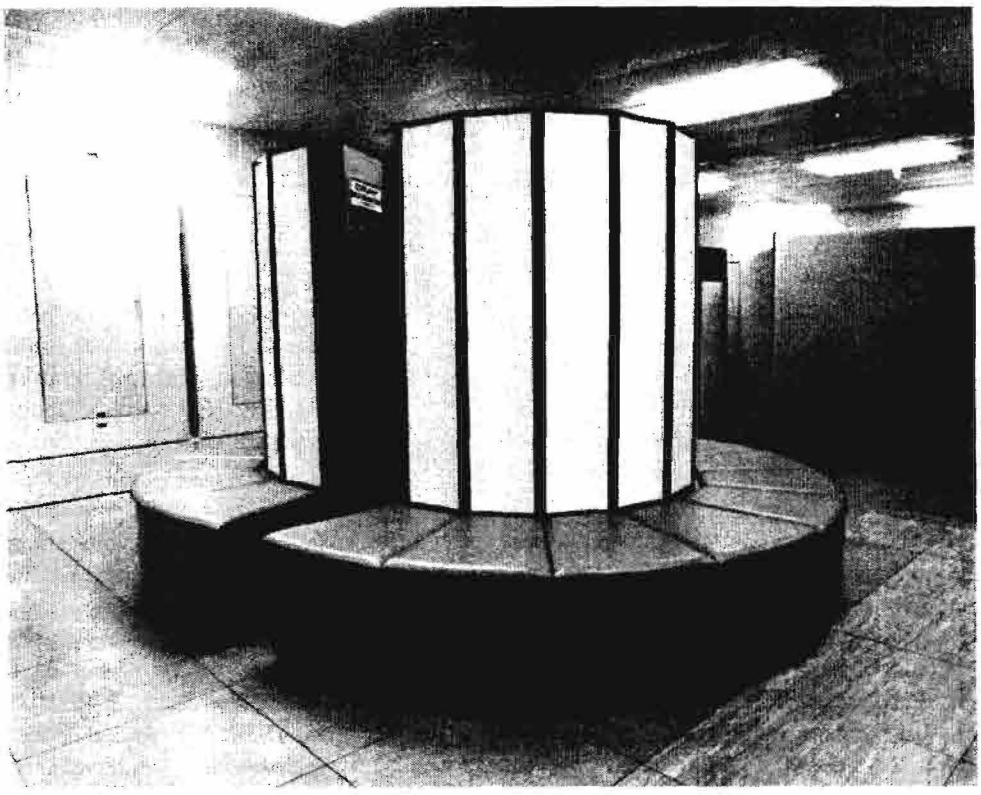
(U) Programs were executed via the GO function key while viewing a file through the editor. The program name and its parameters were parsed from the line containing the cursor up to the first blank character or the end of the screen. Changes to the display were not inserted into the file being displayed until the INSERT key was pressed. This allowed a user to do full screen command line editing. The user could maintain sample command lines in a file, display the file, edit the appropriate command line, execute it, and retain the original version of the command line. Of course the edited command line could be retained by pressing INSERT before GO if desired. The executing program automatically received some information related to the file currently displayed; the file name, the position of the window on that file, and the cursor position within the window. The program controlled the whole window while it executed. When a function key was pushed during execution, the program could find out the function key number, the cursor position within the window, and the four characters immediately preceding the cursor. FOLKLORE did not buffer function keys. Only the information from the last function key pushed was preserved until cleared or read by a program.

(U) FOLKLORE program source files were generally large files containing the source for many programs. The programs could even be written in different languages within the same source file. The only common thing was that each piece of source began and ended with a left-justified '%' character. Special functions took advantage of this. The SEND function key would quickly position the editor at the beginning of a particular piece of source code. Since this function searched for '%'s, it even allowed quick location of data sections which were delimited by '%'s.' Cryptanalytic applications often used this mechanism to maintain many parameter sets within a single flat file. Another feature that this scheme allowed was great for program development. That is the COMP function which could be used from anywhere within the piece of source code. FOLKLORE would automatically find the beginning of the code (previous '%'), determine the appropriate compiler from the keyword following the '%', and compile the program. In fact, if the GO function was used from anywhere within a piece of source code, all of the compile functions were performed, all the necessary relocatable files were located and linked, and then the program was automatically executed. All of this happened within seconds, so the user did not lose track of the real job that needed to be done. It was amazing how little delay the users were willing to tolerate after having developed experience with FOLKLORE for a short while. In fact, just five seconds was considered unacceptable most of the time. Another common practice, especially useful during debugging, was to keep a sample of the command line within comments at the beginning of the program source code. The user placed the cursor somewhere within the source code, pressed COMP to compile, then ALT to return to the top of the source code, moved the cursor down a few lines to the command line and pressed GO to execute it.

(U) Another commonly used FOLKLORE feature made possible by the responsiveness of the FOLKLORE editor was the use of the ALT function to compare files. The ALT function switched between the current file and the alternate file. The current file became the alternate file and the alternate file became the current file. By using the ALT button quickly, a user could easily spot any differences between the two files a whole page at a time. The screen would appear to stay constant if there were no changes and would flicker in the spots where there were differences. It was quite common to align

(U) Another very useful (and very much missed) feature was the command spelling corrector. If a command was executed that was not found, the system loader or editor would suggest a similar command or file name and gave the user several options for continuing.

~~FOUO~~



~~FOUO~~

~~(FOUO)~~ On 31 January 1996, still popular with its users and running on a Cray X-MP [redacted] the last FOLKLORE system was powered down. P.L. 86-36

two files, press ALT several times a second, watch to see if the screen flickered, move to the next page of each file, and repeat the ALT sequence. This process would be useless if the switch between files was not "instantaneous"! In contrast, using a file comparison tool could be a tedious series of executions with a varying offset parameter or it could be even worthless without an offset parameter or ability to work on binary data. Suppose a program was changed to insert a certain string of bits periodically in an output data file. These changes between the old and new output files could be easily spotted and verified using the FOLKLORE ALT method. Even if there would be no change in offset required, the FOLKLORE ALT method would sometimes be faster than typing a command. If there were differences, the user had a full screen of context to interpret the reason for the change. This method worked fine even for binary data. FOLKLORE had a different character displayed for every eight-bit sequence. Therefore, a change in one bit would cause a difference in the character displayed. This change would be seen using the ALT file comparison method. Bit-stream cryptanalysts were familiar enough with the character representation to understand what the difference was and why it was there.

(U) The move to the SUN workstations allowed high resolution graphics for the first time, but some FOLKLORE users edited speech waveforms graphically using the PTS terminals in the early 1980s. The responsiveness of the terminals even allowed some creative analysts to produce animated graphics!

(U) There were at least two important observations made in some research reported from IBM in 1982. First, as system response time decreases, the quickness of human interaction responses increases more than linearly. Second, all skill levels benefit from this effect. Also, as the skill level of the user increases, the benefit also increases. The cryptanalytic community was not surprised! They had been reaping these benefits for more than a decade by then. In fact they complained about losing their train of thought when anything slowed down by even half a second.

(U) What was the power of the combination of the edit and command mode? Commonly used commands could be stored in a file. The user's default/STOP file was often a list of these command lines. The user then simply used STOP to view the file and then selected the appropriate line with the cursor and pressed GO to execute it. Often applications would embed command lines or file names in the output to facilitate running follow-on programs or viewing multiple output files. This feature was very useful to reduce wasted time caused by mistyping or omitting parameters. BATCH sequences, a series of command lines beginning with "BATCH,**" and possibly including some simple control structures, were also stored in source code

P.L. 86-36

sections and executed via the GO key. This also greatly simplified the automation of modified source code installation. BATCH sequences were also developed and maintained in separate files particularly for use in production runs. These were initiated using "BATCH,filename" as the command line. When a BATCH sequence was initiated from within a file, the parameter was '**' and when it was initiated from somewhere outside the file, the parameter was the name of the file containing the command sequence to execute. (Actually there were more parameters, but they are not relevant here). The FOLKLORE convention used '**' to mean to input data from the current file beginning at the current location. A single '*' meant to input data from the current file beginning at the start of the file.

(U) Although having command lines stored in a file reduced a lot of mistyping and wrong parameters, another very useful (and very much missed) feature was the command spelling corrector. If a command was executed that was not found, the system loader or editor would suggest a similar command or file name and gave the user several options for continuing. Much of the time, the suggested command was the correct one, so only one key press was needed to correct the problem.

(U) Having a single edit/command mode simplified the life of the user. Only one set of behaviors had to be assimilated and repetitive actions could be stored for reuse. Efficient use of time for both the user and the computer was the result. The *de facto* standards for software development set by the operating system software and its user environment reduced the amount of deviation that develops between multiple software developers. Although FOLKLORE did not have some of the luxuries provided by today's custom environments and the use of COTS products, a lot less time was needed to set up a new user and to learn to use FOLKLORE effectively and creatively. This coupled with the effects of the extremely rapid response time, gave the cryptanalytic community a very productive quarter of a century.

~~(FOUO)~~ I would like to thank [redacted] for their review and very helpful comments.

[redacted]

[redacted]

P.L. 86-36
EO 1.4.(c)

Kλ