Computers—The Wailing Wall

BY J. A. MEYER

~~Secret~~

A description of the struggle to accomplish cryptanalysis by machine.

INTRODUCTION

Computers are a new kind of cryptanalytic clerk which have, in the past five years, become the focus of a constellation of ideas, spectacular feats, and misunderstandings at NSA. Popular clichés, ". . . solved by the computer", ". . . read by the computer", subtend the difficult and fascinating process of machine cryptanalysis, and assign a halo of achievement to the dominant concrete element, the computer, as if it were in some way deserving of praise or blame for the results it produces. For people engaged in the facts of mechanization, this illusion does not persist; the computer is seldom more than a convenient scapegoat, and it is the aim of this paper to describe some of the problems, ideas and people involved in the development of a cryptanalytic machine attack, and its consequent machine program.

1. FACTORS IN MECHANIZATION

The machines, the people, and the nature of the cryptanalytic problem determine the success of a mechanization.

1.1 *Machine Characteristics*

A computer is a large electronic machine. Electromechanical devices such as tape readers, typewriters, and magnetic tape units transfer information in and out of the machine. A memory unit stores a set of numbers. All processes and data are represented in a computer as numbers. A switching network provides an arithmetic-logical unit, and control system. A computer functions by reading instructions in its memory, and performing simple operations in its arithmetic-logical unit. Its particular value lies in its ability to perform any sequence of simple operations in any specified order. This distinguishes it from special-purpose machines for which the function and logical organization are fixed at the time of construction. The task of specifying sequences of operations is called programming. The outstanding logical characteristic of digital computers is that they can be programmed. This is also their principal weakness; they *must* be programmed.

1.2 *The Technical People*

When an attempt is made to mechanize a problem, three kinds of specialists are involved: the cryptanalyst, the methods analyst, and the programmer. The cryptanalyst starts a project by conceiving

of an experiment. If the experiment cannot be carried out by hand, he may seek machine aid. The methods analyst, who is introduced to the problem at this point, must decide if the experiment can be done by machine, and if so, he must invent some workable method. A methods analyst is usually a cryptanalyst of ability and experience who also possesses a special flair for transforming vaguely expressed analytic ideas into exact logical procedures which can be represented on a machine. The programmer must invent a way of representing the logical procedure on a computer,[1] and one of his most challenging responsibilities is to discover as quickly as possible an efficient representation, and produce a timing estimate which is not wrong by more than a factor of five or ten. This is frequently a very difficult thing to do, but it may be the most significant single factor in deciding whether to pursue an attack or not.

1.3 *Status of the Problem*

Cryptanalytic problems which are solved usually pass through three stages of development; *diagnosis*, in which the nature of the encipherment is discovered; *recovery*, in which components, wheels, key, code groups, etc. are "earned"; and *setting*, in which the relation of the known elements to a particular message is sought.[2] Because of the laborious routine, which not only bores the analyst, but frequently is too large to carry out by hand, setting has become the classical machine attack, and exhaustive trials the classical method.

[1] By "representing" we do not ordinarily mean constructing or simulating an exact analogue, but rather, abstracting certain characteristics of the problem or mechanism into an information matrix (a set of function tables and relationships) which can be manipulated to give the effect of the object. For example, a wired rotor enciphers by conducting a current through a wire. We do not make an analogue of the wire and simulate the current flow in terms of electrical equations; instead we designate the rotor input as a linear coordinate and the rotor output as a number. The characteristic of rotor encipherment in which we are interested is a transfer from a point in space to a point in space, and this can be represented by a function table. A good representation is simple and accurate.

[2] A simple example of this is monoalphabetic substitution. Starting with raw traffic, and perhaps some knowledge of the underlying plain text, the cryptanalyst determines by frequency counts, or repeats, or some other obvious characteristic of the cipher text, that monoalphabetic substitution is a likely encipherment system. He then tries to recover the substitution, and if he succeeds, this confirms the diagnosis. He may also discover, after solving a few messages, that a small family of substitutions is being used for the day's traffic, and he then tries placing known alphabets into new messages. This last is a setting problem, in which all the components are known. On a higher level, diagnosis may consist in proving that a four-digit code is enciphered by additive key. Recovery may involve earning new code groups and key from messages in depth, and when enough key and code have been recovered by hand, it may be possible to try setting messages by sliding all recovered key against each message, and recognizing good placements.

The machine and programs for setting are usually simple and functional, and are often successful. Recovery is a more difficult problem to mechanize, because exhaustive trials can never be used, and instead, an attack must be devised.[3] Some of the most striking advances in the past few years have been made in the development of machine recoveries, but the attacks and programs are much more sophisticated and require much more thought and effort than the setting techniques.

1.4 *The Idea*

To attempt mechanization of a problem, a state of knowledge must be reached in which exact hypotheses can be formulated. An exact hypothesis is a question which can be accorded a yes-or-no answer after an experiment. It is the cryptanalyst's job to sift the evidence, and evolve such a hypothesis. This requires extremes of observation and insight, and will probably be permanently beyond the power of any machine.

1.5 *The Method*

The development of an iterative system of trials is the essence of mechanized cryptanalysis. Many successful cryptanalysts are unable to express in simple terms the techniques they use for solving problems even though they may, through worksheets and sign language, be able to communicate enough of the idea so that other cryptanalysts are able to derive equivalent methods. In devising a system of trials, the methods analyst cannot rely on simply constructing an analogue of the hand method, because frequently so-called hand methods are really conceptual feats of strength, accompanied by doodling.[4] To suit a machine, an attack must be reducible to a simple, exact routine with a large volume of work.[5] Unless the

[3] The impracticability of exhaustive trials in recovery is easily illustrated by the case of monoalphabetic substitution. The simplest method would be to try 26! alphabets, and recognize the resulting plain text. If the trials were carried out at the rate of 10^6 alphabets/second, it would take over one trillion years to complete the testing on one message, and if a Bayes factor scoring system were used to reject null trials, a large number of random trials would outscore the correct answer. Given fifty groups of cipher text, a hand worker could recover the correct alphabet in a short time.

[4] An example of a non-iterative task is reading a depth. A depth is a pair of messages which are enciphered by the same key. Where neither key nor code is known, the solution requires a knowledge of language, a hunch about context, and a feeling for believable plain text. Extremely good guesses are also a help. For a task like this, the experienced depth reader is unchallengeable.

[5] A classical iteration is slide-running, in which all recovered key is tried against all new messages, and the underlying code, if any, is recognized. The same operations, differencing and recognition, are performed over and over on each message, and the function tables (key and cipher and codebook) permuted exhaustively. In this sort of task, a machine is unequalled.

task is routine, it cannot be done at all, and unless there is a fair volume of work, it may not be worth doing by machine, since the program may be more difficult than the process.

1.6 *The Program*

The outstanding characteristic of a computer program is its inflexibility. Sometimes programs are also very complex, but the function of a program is a purposeful calculation, and a program in operation is always a special-purpose mechanism which uses the components of a general-purpose machine to perform a particular process. A frequency-counting routine will only do counting; it cannot be used for slide-running. A decipherment routine will only do a particular kind of decipherment, such as modular addition, and cannot also do Playfair or Vigenère substitution. Because any features of flexibility in a program require additional thinking and effort to incorporate, programmers customarily write the most inflexible program possible, since this involves the least work and chance of error, and offers great advantages in efficiency and compactness.

A program is usually accomplished in two distinct phases, solution and execution. In solving a program, the programmer investigates a number of different ways of representing processes, and concentrates on the ideas, the statement of the problem, and the invention of "programming tricks" to increase the efficiency or flexibility of the program. It is much easier to change an idea than to rewrite a page of coding, so the programmer makes an effort to avoid exact commitments until he is sure all the planning is completed. Executing the program is a feat of construction in which the ideas are reduced to an exact set of numbers which will direct the machine through a specific process. What the machine lacks, in comparison with the human clerk, is any degree of understanding, i. e., the ability to follow general directions. Hence the programmer must specify every detail exactly, correctly and completely.

The program script is a special obstacle in any machine attack. The cryptanalytic idea, the iterative method, the programming tricks, all represent logical achievements in the attack, and are frequently the result of considerable cunning and insight; but once the ideas have been evolved, writing them down is usually a simple process—at most a few days work. The program script is opposite in the extreme. A program which can be described and understood in a few minutes may require a script of thousands of lines of coding, and the programmer, for months on end must pursue a single, fixed idea. The completion of a large program script is usually accompanied by a feeling of uncertain relief. The minimum requirement for accuracy is absolute perfection, and a coding-error rate of one part in a thousand will

make a program completely unworkable. An uncompromising rereading of the script will remove a large number of "bugs" before the program is tried on the machine, but the programmer can hardly bring himself to go back through all that coding again. At the same time he knows that the bugs he does not remove by hand may frustrate him for weeks on the machine. "Thus conscience doth . . ."

2. A SAMPLE PROBLEM

In order to describe the interplay of the technical and human factors inherent in mechanization, a sample case history has been invented.

2.1 *Description of the Problem*

An electromechanical key generator has been completely recovered except for the current output stecker.[6] A message is enciphered by adding key (mod 26) to the text. The key is generated by a three-wheel cipher machine with a restricted set of input steckers, and a variable output stecker. The cycle of the key generator is 4×10^6 characters, for each input stecker. The messages are 50 groups long, and the first group of each message is steckered key with no text added. The messages use the key in sequence for a month,[7] then make a new start in the key cycle, with a change of output stecker. The beginning key group is used to locate messages correctly in the key cycle in case of transmission errors, or loss of a message. The underlying text is plain language, which is known and fully catalogued.

2.2 *Description of the People*

The cryptanalyst is a unit head, a hand worker[8] who has figured prominently in the solution and recovery of the enciphering device. He has no special mathematical training, and is completely naive about machines and machine methods. He would like computer help, but is skeptical. The methods analyst is a junior mathematician[9] who has been applying formulae to another problem, and taking courses in cryptanalysis for three years. He has acquired a passable vocabulary, taken a machine orientation course, and sponsored a small program for statistical computation which was never completed. This is his first try at a plaintext problem. The programmer holds a BS in mathematics, has been exposed to eight weeks of cryptanalysis in training school, and has been learning programming for six months.

[6] A stecker is a single stage substitution which does not change during a machine setup. It is frequently represented in cipher machines by a 26 x 2 plugboard. The 26 input points are connected by wires to the 26 outputs in some arbitrary sequence, for example, A:X, B:Q, . . . Z:F, and these connections can be changed periodically by the operator. A wired wheel is a packaged substitution unit in which a set of input and output commutator points are connected by wires, like the stecker plugging, but for logical reasons the wiring is usually permanent.

He has done a few small practice routines, but this is his first operational program.

2.3 *Statement of the Problem*

The problem is to set the first message of the month, and recover the output stecker. From this start the rest of the month's traffic can be read by hand.

2.4 *The Idea*

[redacted]

Encipherment is the result of transmitting a current through a number of wheels juxtaposed in series. A current flowing through a three wheel maze travels from the input to the output of the first wheel, through some fixed contact to the input point of the second wheel, and so on in series, until a closed circuit is completed between some input source and some output point. Thus:

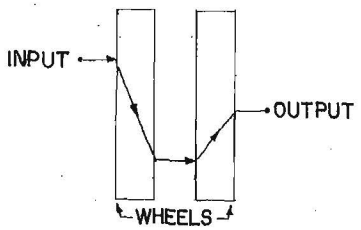


Fig. 1.

The wheels are usually permuted to each new setting by mechanical action, and for this reason they are so constructed that the change of setting does not disable the circuits—as by opening a switch—but changes the effect of the serial substitution. The mechanical action is called *stepping*, and the number of successive steps before the same encipherment effect recurs is called the cycle of the machine. The key generation can be represented as (Input → S M F E→Key) where E designates the output stecker, and S M F (slow, medium, and fast) stand for the three wheels.

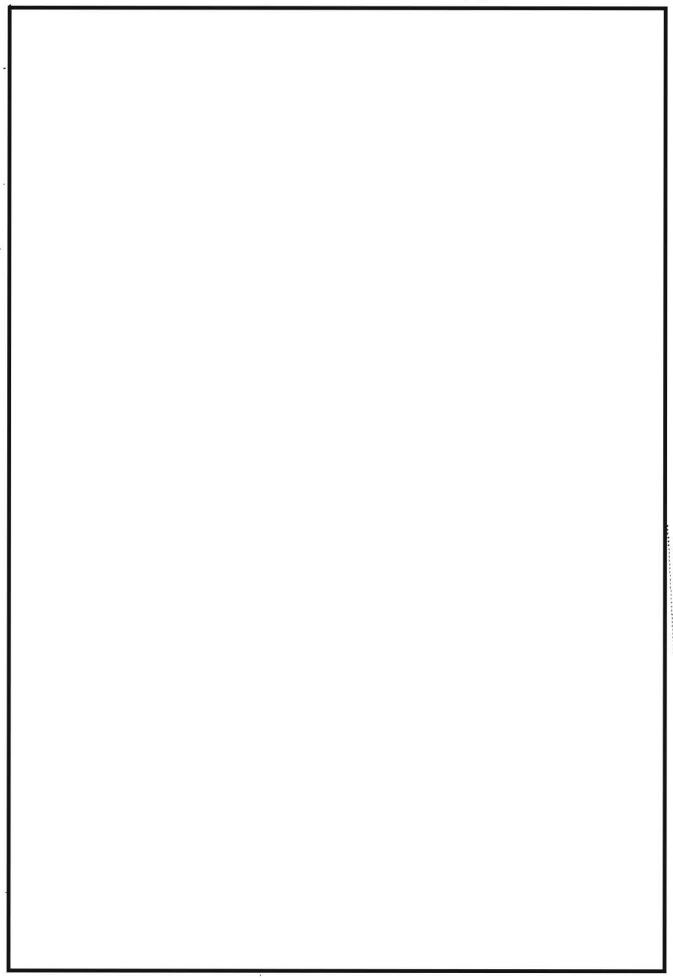
[7] This is called "tailing", the messages following one another without intermission or overlap, like elephants holding each others tails.

[8] The hand cryptanalyst evolves his solutions by heuristic experiment, insight and "cheating". The machine cryptanalyst designs a simple, honest procedure, and when the trials have been carried out, he surveys the results.

[9] A junior mathematician usually arrives with an MS degree, and some specialization in a field of mathematics totally unrelated to cryptanalysis

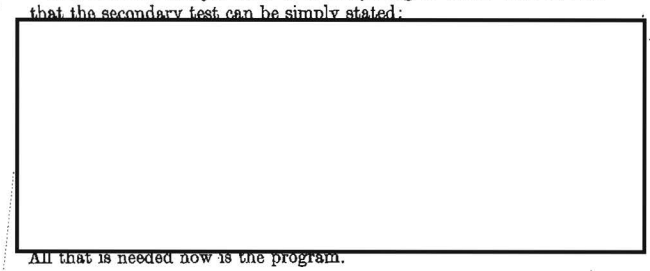
[10] The key can be generated through three wheels, S, M, and F. The actual key used in the message is a substitution of this: $f(E_2, K) = K'$, $K' + P = C$.

[redacted]



2.5.6 *Secondary Testing Procedure*

The methods analyst feels that everything is taken care of and that the secondary test can be simply stated:



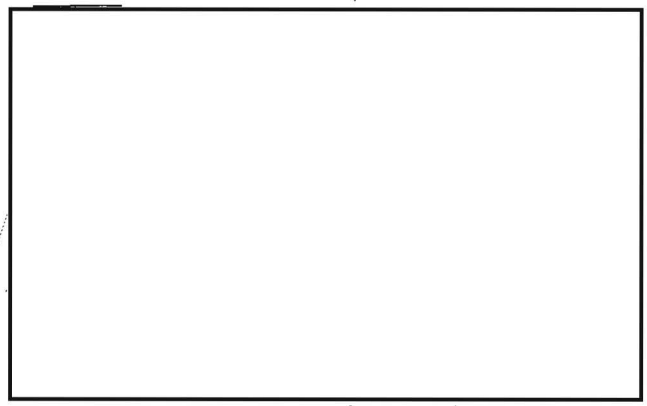
All that is needed now is the program.

2.6 *The Program*

The programmer is usually kept out of a problem until the people concerned with the cryptanalysis feel that all the thinking has been done.

2.6.1 *Communication*

The methods analyst is impatient to see concrete results emerge from his now perfect attack, and he harries the powers that be to have a programmer assigned to the task immediately. In an exceedingly short time, perhaps only two or three weeks, he is notified that a programmer *will* be sent, and only ten days later the programmer is led into the section.



The methods analyst, in the interest of efficiency and quick results, dispenses with any unnecessary explanation of the problem and the idea, and races through a detailed description of the exact process he wants the machine to do. He then asks, "How fast will it run?", and "How soon can you have it done?" and leans back, waiting for an answer. The programmer, feeling he should break the silence in some way, says, "I don't know," and then, without apparent premeditation, asks, "*What do you want the program to do?*"

Under careful grilling the programmer reveals that he does not understand the least bit of the cryptology nor the cryptanalysis, and in fact has never heard terms like "stecker", "three wheel cycle", "isomorphic poker hand", and "wheels not on a true base" in his life. Amid cries of despair, the methods analyst settles down to the task of drilling the ideas into the programmer's head, and he is doing fairly well in explaining the nature of electromechanical encipherment, steckers, and additive arithmetic until one of the hand workers, in a helpful mood, begins explaining to the methods analyst the proper method for explaining wired wheels.

All those present then unite efforts in a semantic log jam, trying to convince each other that a simple device like a wired rotor is best represented as a square, no, a strip, no, an inverse, converse, encipher, decipher table, no, an abstraction of rotating points in space, no, ...

Listening to the raised voices and supercapital crypto-idioms, " . . . you've got to correct the twist . . ", " . . . you can't use strips in a machine . .", " . . . but what happens when you step the *F*?", the programmer is apt to undergo a serious ignorance trauma. He hopes that it will soon be time for lunch . . .[18]

2.6.2 *System Design*

After a few more hours of help the programmer is able to recite back the setting and testing procedure well enough to try writing the program. At his own desk he sketches, in crude pictograms, the design of a system which will handle a [redacted] The workload is to be one message a month, so elapsed time, rather than running time, is the alleged target, and the minimum program which will do the job is intended.

2.6.3 *Timing*

The way the programmer solves the timing problem is to write a routine which will do the primary part of the program, i. e., scan for

[18] This is not really exaggerated. Cryptanalysts are notoriously inarticulate even in their own idiom

[redacted][9] After a few days work he has a set of coding which he *thinks will do the job.* This routine is several hundred lines of coding in size, and, using tables which predict the operating time for each instruction, the programmer calculates the number of microseconds required to go through the basic iteration or "loop" for one character. This basic time is about .001 seconds, so the complete [redacted] can be done in about 20 minutes. This estimate is greeted enthusiastically by the analyst, and in the optimism of the moment the programmer ventures that he can complete the coding in three weeks.

2.6.4 *Strategy and Tactics*

The computer being used has a small (1024 cells) high-speed memory, an auxiliary magnetic drum, and a magnetic tape system. In order to manage his table storage and secondary testing, the programmer decides to build his program in several discrete segments, and to store the major portion of his [redacted] on the magnetic drum. His cycle control bookkeeping will [redacted]

[redacted]

four minutes. There is also an input routine and an output routine to worry about, but these are straightforward, and the programmer defers work on them until the main program is written.

2.6.5 *Writing and Coding*

At this point all the reasonably interesting parts are done. Accurate timing estimates can be given, and the programmer has familiarized himself with the functional penetralia. To complete the job he must execute a great mass of coding: a clerical feat requiring a capacity for monotonous drudgery and uncompromising attention to detail; the kind of exacting toil a person of imagination and intelligence most

[redacted]

abhors.[20] However, there is no other way of getting the program done.

[20] In the craft, this is referred to as "grinding out the program", and is the most prominent factor in discouraging people from pursuing programming as a career. Programming ordinarily does not present the intellectual challenge that cryptanalysis does; it is practiced in an environment of facts, rules, exact logic, endless detail—and generally requires more talent for graphic formalism than abstract reasoning. At the same time the average program seems (to programmers) harder to do than the average cryptanalytic problem. The term "programming" covers a wide range of skills and tasks which vary from machine and systems design to keypunching. Furthermore, the programmer is assigned maximum responsibility for his program and frequently executes it as a single-handed achievement, working for months from the statement of the problem through every detail of the job till the results emerge—an unique and rewarding absolutism. Some aspects of programming, particularly the solution of critical processes within the exact restrictions of machine hardware and logic, are fascinating to everyone who tries them. Other aspects, however, seem to condition programmers into a determined negativism towards their jobs. The dilemma lies in a basic conflict between thinking and coding. In order to convert an abstract idea into an operating mechanism, a programmer must think *and* code.

Most of a programmer's life is devoted to developing and exercising exceptional skills in coding, trivial invention, and memory. The concentration required to produce pages of correct coding precludes reflection, imagination, or inventive fantasy, and so when a programmer, who usually knows for months at a time exactly what he must do, finds himself with a new, strange, and incomprehensible problem, he may cease work entirely, and stare into space. Outside of NSA this is referred to as the "stupor period". The NSA term is "planning". It is during this time that the programmer is trying to think.

This is a justifiable lethargy, since one good idea is worth much more than a month of coding, but sponsors never understand this sort of unproductive delay; all they want is results—immediately—and they usually demand hasty action, which from the sponsor's point of view is the sensible thing to do. Aside from the resulting friction, however, the intellectual hazard is that the programmer, to get quick results, will be forced *not* to invent, but rather to use cliché representations which he is certain will work. It is difficult for sponsors to appreciate the value of originality until they are stalled on a massive problem which cannot be programmed because no one has discovered a sufficiently creative representation. Programmers who argue the merits of good ideas over treadmill coding are regarded as *agents provocateurs*. Nevertheless, after the ideas have been evolved, and are found functional as well as aesthetically pleasing, they must be converted into realities of a program script. Trying to get other people to do the work at this point involves a severe communications problem—the more original the ideas and representations, the more difficult it is to explain them to other people, especially if the recipients are also naive about the analytic ideas.

Creative programming seems practically to exclude the analyst-coder relationship, and it is much easier and faster for the programmer to write the program himself than to turn it over to others, unless they have an advantage of intellectual superiority. This is one of the hard facts of programming, and an ironic side effect is that as a programmer's skill and strength increase, the aspects of programming which require thinking become much easier for him, but the coding tasks retain their intractability.

2.6.6 *Debugging*

When all the instructions and tables have been coded and checked, and the program appears to be finished, there lurk a number of small errors in the script, which are advertised when it is run on the computer. The most obvious of these can usually be corrected in a few weeks and the program may then give the appearance of running through correctly. This is when the real debugging begins.[21] It is a discouraging axiom that a program can never be proved to be completely debugged, and the process of detecting, locating, and correcting the more elusive bugs is an experience which is only appreciated by people who have endured it.

2.6.7 *Checking Out*

A completely correct program may have several variables such as message length, number of input steckers, output options, and so forth, for which the boundary conditions must be proved. The more inflexible the program, the more easily it is checked out.

2.7 *First Results*

Not only is the program completed two weeks late, but in studying the results of the first hurried run, the cryptanalyst notices that all the hits are false, and the key generation is incorrect.[22] The programmer, who has been congratulating himself on finishing the job, turns livid with shock. The methods analyst regards him with chagrin, and the handworkers all nod knowingly. The fault is traced to an incorrect stepping-sequence for one of the wheels, a misunderstanding which occurred when the problem was first explained.

[21] Debugging is a contest of logic and willpower against the idiotic resources of the machine. It is not unlike chess in the formal alternation of successive moves, but the special ability the machine has is the ability to conceal the occurrence as well as the cause of errors. For example, an incorrectly set counter may let a routine exceed its geographic limits, and modify a table, which will in turn cause a wrong decipherment—none of this being detectable to the programmer until a combination of circumstances fortuitously causes him to get a false hit, or lose a correct answer. At this point he has to determine which of his million characters is wrong, and trace back through the logical implications and branchpoints of the program to isolate the incorrectly set counter.

Cryptanalysts and other professionals who face the machine for the first time in programming are usually completely baffled by the problem of detecting errors in a dynamic process. This depends upon interrupting the events of the program at exactly the right point so that the erroneous calculations and all the implications thereof can be traced. At 30,000 computations/second, with millions of characters and bits of data being processed, this pinpointing requires cunning strategy and judgment. The machine designer can help to some degree, but the classical solution depends principally on the programmer's ingenuity and thorough understanding of his program.

[22] Every program should be made to solve a toy problem before completion is advertised.

2.7.1 *Success*

The tide of battle turns, and after three more days of furious debugging, the message is run again, and this time, luckily, a hit is discovered. This is the hour of glory for the methods analyst, and a long parade of supervisors and colleagues salute his ingenuity, while he in turn pays generous tribute to the handworkers. A meeting next day, in the flush of victory, decrees revision of the program to meet new conditions.

2.8 *Changes in the Problem*

[redacted]

2.8.1 *Cribbing Procedure*

[redacted]

2.8.2 *Number of Hits*

[redacted]

2.8.3 *Extension of Secondary Testing*

[redacted]

2.8.4 *Modified Output Format*

[redacted]

2.8.5 *New Specifications*

[redacted]

2.9 *Economy*

The program which was intended as a casual project has mushroomed into a large production job, and the programmer seeks some way to improve the efficiency of his main routine, i. e., generation and scanning.

[redacted]

[redacted]

[redacted]

2.9.4 *Rebugging*

The programmer tackles the job with misgivings, and after a week of coding, he attempts to connect a group of simple routines into the existing program. This, and the writing of the glorified output routine, are the most unpleasant parts of the entire job. In reopening old hostilities, he discovers that he has forgotten many details and implications of the previously debugged routines, and a discouraging succession of trivial and obscure errors have to be isolated and corrected.[30]

2.9.5 *Making the Program Operational*

When the program appears to be working, the programmer must tie up the loose ends and turn it over to the operations group, who will run it on the computer. At this time he must prepare lists of simple instructions for people to follow in preparing the data, running the program and dealing with the answers. Anticipating human and machine errors, he must provide routines to restart the program, check the running of the program, and check all the variables, such as messages, steckers, and cribs, which may be loaded into it. The peripheral routines for input, output, checking, and reset frequently require more time and work to program than the basic cryptanalytic process.

2.10 *Operational Results*

Eventually a program, consisting of a small inefficient core and a patchwork of peripheral routines, is delivered with a sigh of relief. A massive processing of cribs and messages is immediately undertaken in a hurried effort to read something. Hours of running time are logged, hits are punched out by the thousands, and electromatic typewriters run to destruction preparing the worksheets. Great stacks of printouts are shipped to the sponsor, and clerks grind over the answers, trying to force readable text into the reams of inert three-stecker stops. Finally the backlog is processed, and the programmer goes up to the crypt section to see how his answers are faring.

[30] Programs approach perfection asymptotically with respect to time. Rebugging introduces a regression. See Fig. 2.

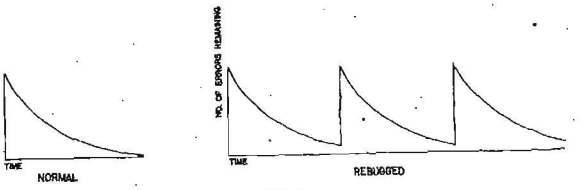


Fig. 2.

He discovers a lone clerk crumpling the pages into a burn-bag, and learns that the problem has crashed to the ground, the machine has been changed, the methods analyst has gone off to another wing, the crypt section is busy with efforts at a new recovery, and the messages he has run are considered too difficult to continue working on when there is a good chance they are totally unreadable. The only residue is the program, which is no longer wanted.

2.11 *Epilogue*

A machine attack which started off well bogged down, and finally disintegrated. Why?

2.11.1 *The Logical Method*

The program failed as a machine project because it did not go far enough. When the number of answers, or rather the work required to evaluate them, becomes too great, the answers are usually given very cursory hand treatment and tossed away.[31] Too many answers are worse than none at all, because the handworkers, if they are not impeded by scanning yards of results looking for "something", may solve the problem by some intuitive method which defies mechanization.

From the exploitation viewpoint, if the key generator had not changed, and a few new hits had turned up, greater effort would have been expended on the answers, but the program would still have been unsatisfactory because of the weakness of the secondary test and the risk of losing answers from fatigue, and falling behind the machine processing. In the case of a success, further work on the secondary testing would be imperative, and a good solution would be a great personal triumph for the analyst. To the programmer, a success

[31] The analyst always assumes that he will be able to recognize the right answer at a glance. Unless the problem is very easy, or the attack is very good, this is not true. The real test of a scoring system is its ability to isolate a causal hypothesis from a great population of trials. As machine speeds increase and trial factors are multiplied, greater exertions must be undertaken to develop adequate extension and scoring systems early in the design of the attack.

would mean, at best, an opportunity to make a really efficient and elegant job of the program to compensate for the dissatisfactions of the existing miscarriage.[32]

2.11.2 *Communications*

After several months of impatient fuming for results, the crypt people have come to regard programmers as very obstinate and undependable people who have no grasp of the problem, bother a great deal about petty details, appear crushed by the slightest change of plans, and never get jobs done on time. This is not only true—at least as to outward manifestations—but a natural consequence of the state of the art. The programmer meeting a new problem is faced by the double hazards of understanding the ideas, and representing them to the machine. Waste, delay and confusion are sure indications of a poorly stated problem. Furthermore, in a completed program the basic cryptanalytic process is usually only a fraction of the job, and the calisthenics of input, output and housekeeping, which the analyst completely disregards, represent the major exertion. The analyst should make some attempt to understand the problems of packaging, system design, machine limitations and efficient representation, because these factors will force his hand as well as the programmer's.

2.11.3 *Modification*

A vital technical problem ignored by the methods analyst was the difficulty of modifying the program. The coding of a program requires exact commitment of thousands of details of logic and storage, and is usually achieved as a feat of sustained concentration and imagination. Successive revision alters the organization of the program and forces the programmer to rebuild in his imagination all the details and implications of his coding and rememorize the intended chain of events, in spite of the persistence of earlier commitments. This is quite difficult if the program is complex, and a secondary effect is that it destroys the programmer's feeling for design; blocking and rejection of the task frequently result. Consequently, a programmer threatened by modifications prefers to start over rather

[32] A programmer can never identify himself with the results of a program, but only with the mechanism; thus the dominant motive which carries him through a burdensome program is pride in his work. The success or failure of the attack is the responsibility of the analyst, and the programmer must execute the program perfectly even when he is certain it is a dead horse. When this pride or identification is absent or thwarted the job becomes a millstone rather than an achievement, and the programmer is usually lost. He may continue to draw pages of numbers, but he no longer creates.

than tamper with an old program,[33] just as an architect would resist building a railway station on the foundation for a doghouse. The changes in the program from the simple poker-hand search to the fruitless cribbing were dictated by the problem rather than by the whim of the analyst, but this did not make it any easier for the programmer.

A completely accurate statement of the problem is the minimum precondition to the writing of a program, yet unfortunately this accuracy is frequently attainable only *after* considerable cut and try.

[33] This is a universal practice when inheriting someone else's half-coded project, or revisiting a forgotten program of one's own. The reason is laziness. A bare page of coding is really an idea recorded in private shorthand, and the reincarnation and understanding of the idea first requires recovery of someone else's shorthand system—a valueless achievement.

One of the harshest sentences a programmer can receive is the task of finishing someone else's fiasco. These homeless programs, for which the sponsor has usually abandoned all hope, have established amazing records in elapsed time between the desire and the delivery. Two or three years delay in finishing a program of a few thousand lines is not considered exceptional; in fact it is considered rather good. Many programs simply wander off and are lost. Conventionally, when a programmer quits or is promoted out of useful service halfway through a program, the greenest member of the group is given the task of ". . . finishing it up." This may be the tyro's first and last program, and there are cryptanalytic sections where the sponsors have indoctrinated new programmers on the same task at monthly intervals.

The long delays are of course trying to the sponsors, who generally exorcise their frustration by presenting each successive challenger with a new list of desired modifications. Few aspects of programming are as thankless or unpopular, and the certain characteristics of a derelict program are that it is uninteresting, messy, and wanted in a great hurry.

The basic problem in hand-me-down programs is communication. A programmer's progress through a job should always be marked by accurate description of the program, the coding, the changes, and the status of the debugging. A workable system, used outside of NSA, requires that one person in addition to the programmer be familiar with and responsible for the coding and logic of the program. Orderly and exacting methods such as this will delay the arrival of first results, but provide greater insurance. Sponsors usually disregard the merits of orderly progress through a large job, and spur the programmers to stake everything on the race for the esteemed first results.

First results are usually wrong or insignificant, and may provoke an avalanche of modifications.

8. SUMMARY

It is much more difficult to do cryptanalysis by machine than by hand, and even simple problems often conceal pitfalls. Significantly, the computer plays no active part in the solution except to carry out a chain of operations for which the programmer has correctly specified every detail. The programming poses some special problems in ingenuity and endurance, but, regardless of the crudeness or elegance of the machine representation, the design of the logical method is the dominant factor in the success of the attack. When the logical method fails, the program sinks with it.