How We Solved the $100,000 Decipher Puzzle (16 Hours Too Late)

Robert W. Baldwin*
Tandem Computers, Inc.
19333 Valco Parkway
Cupertino, California 95014

Alan T. Sherman*
Computer Science Department
University of Maryland Baltimore County
Baltimore, Maryland 21228

July 25, 1989
(revised December 15, 1989)

Abstract

On Saturday March 30, 1985, we solved a cryptography puzzle that had remained unsolved for over two years. Unfortunately for us, we missed by one day the deadline for sharing what had become a $117,000 prize. This paper describes the puzzle and how we solved it.

The $100,000 Decipher Puzzle consists of a simple two-sided jigsaw puzzle that contains a sequence of 376 codenumbers, ranging from 1 to 1252, which were encrypted by a multiple-substitution cipher similar to the Beale cipher. Clues state that the key was derived from some keytext in the public domain. We solved the puzzle by building a Zetalisp program that tested candidate keytexts by trying a variety of ways to extract candidate keys from each keytext and by checking the resulting candidate plaintexts for English. Our program exploited a novel "windowing" technique that detected when part of a candidate key was correct and an effective test for English based on digraph frequencies.

Keywords
Beale cipher, contests, cryptanalysis, cryptography, cryptology, homophonic cipher, multiple-substitution cipher, $100,000 Decipher Puzzle, puzzles, Warren Holland.

1 Introduction

In 1983 Warren Holland Jr. introduced an intriguing cryptographic contest known as the *$100,000 Decipher Puzzle*. The puzzle consists of a simple two-sided jigsaw puzzle that contains a sequence of 376 codenumbers, each an integer in the range 1 to 1252 inclusive. See figure 1. The instructions explain that the ciphertext was produced by a multiple-substitution cipher whose key was extracted from some document in the public domain. Holland offered a

*This work was done while the authors were graduate students in the Department of Electrical Engineering and Computer Science at the Massachusetts Institute of Technology.

Side 1:
909, 736, 71, 55, 651, 269, 24,
973, 342, 46, 682, 462, 26,
326, 323, 857, 765, 397, 203,
111, 1207, 691, 477, 559, 338, 337,
74, 1041, 705, 598, 453, 209, 856,
674, 8, 154, 246, 634, 505, 1249, 161,
1252, 414, 579, 287, 318, 75, 84, 366,
812, 494, 587, 131, 721, 904, 8, 32,
1001, 814, 812, 224, 339, 20, 65,
1082, 910, 669, 492, 319, 12, 152,
144, 588, 860, 986, 969, 425, 624,
202, 149, 333, 663, 671, 473, 266,
780, 655, 757, 779, 777, 569, 438,
294, 1214, 950, 865, 408, 312, 332,
563, 598, 876, 1002, 112, 70, 110,
368, 826, 238, 1126, 918, 867, 63,
113, 1196, 928, 762, 724, 805, 83, 18,
327, 417, 26, 140, 1032, 195, 798,
676, 1102, 867, 523, 491, 560, 435,
67, 177, 180, 6, 1201, 1, 78, 334, 194,
717, 450, 526, 1011, 963, 964, 639,
294, 1096, 727, 807, 775, 661, 545,
350, 253, 34, 455, 735, 644, 780,
750, 631, 434, 201, 1132, 1166, 780,
819, 729, 293, 18, 107, 913,
710, 839, 519, 342, 393, 208, 4,
876, 1040, 295, 369, 784, 757,

Side 2:
1239, 744, 6, 106, 751, 591,
107, 907, 740, 477, 601, 342,
268, 797, 1086, 342, 206, 16,
111, 489, 448, 440, 887, 788, 97, 492,
625, 627, 746, 163, 546, 608, 547,
143, 518, 773, 658, 1109, 1067, 1102,
431, 911, 749, 465, 6, 173, 48, 636,
668, 752, 899, 1084, 1016, 412, 796,
93, 29, 1029, 771, 652, 26, 100, 84,
497, 1117, 281, 654, 867, 877, 860,
506, 297, 209, 357, 876, 40, 84, 418,
342, 294, 633, 1141, 972, 231, 605,
652, 1215, 348, 104, 1002, 939, 696,
1137, 933, 935, 934, 432, 643, 1065,
45, 445, 584, 1053, 176, 59, 395,
234, 815, 782, 214, 294, 979, 768,
384, 331, 256, 875, 987, 876, 1034,
81, 243, 321, 1019, 937, 1006, 713,
1238, 1124, 1103, 944, 1171, 66, 360,
229, 1021, 288, 406, 1244, 638, 408,
209, 609, 557, 1113, 1230, 786, 300,
485, 386, 441, 229, 166, 748, 1198,
735, 749, 447, 1092, 52, 64, 715,
1241, 925, 1026, 108, 494, 833, 759,
403, 54, 37, 545, 539, 342,
1161, 1218, 657, 797, 684,
147, 179, 1171, 749, 844.

Figure 1: Ciphertext, arranged in lines as they appear on the jigsaw puzzle.

$100,000 cash prize—insured by a major U.S. insurance company—to be divided equally among all registered contestants who submit the correct solution by March 1, 1984.[1] To register, each contestant had to purchase a copy of the puzzle. The Decipher Puzzle was marketed in a variety of forms; the most successful was an upscale version whose jigsaw-puzzle pieces came in an attractive velveteen bag and which sold in retail stores for $12.

To produce the puzzle, Warren Holland formed a company called Decipher, Inc., located in Norfolk, Virginia. An entrepreneur and native of Portsmouth, Virginia, Holland is an engineering graduate of the Virginia Polytechnical Institute who also holds an MBA in business and finance. Holland's interest in cryptology grew out of a *Smithsonian Magazine* article [3] about the Beale cipher which he read while working as a partner in a small construction company.

In the *multiple-substitution cipher* or *homophonic cipher* [4, pp. 67–73], each plaintext letter

[1] The insurance was to cover the risk that someone might solve the puzzle in the first year before enough puzzles were sold to pay for the prize. Some sources (*e.g.* [14][11, p. 71]) reported that Lloyds of London turned down Holland because Scotland Yard said the puzzle could be easily broken. According to Holland, it is true that Scotland Yard evaluated the puzzle at the request of Lloyds of London and concluded that the puzzle could be solved. The reason why Lloyds of London turned him down, however, was that Lloyds of London could not determine the risk. Holland's policy with the Admiral Insurance Company of New Jersey cost him $4,500.

key	=	F	O	U	R	S	C	O	R	E	A	N	D	S	E	V	E	N	Y	E	A	R	S
codenumbers	=	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22

plaintext	=	S	E	N	D	R	E	D
ciphertext	=	13	19	17	12	8	14	12

Figure 2: A multiple-substitution cipher of the type used in the $100,000 Decipher Puzzle. To encrypt any letter, choose any position number in which the letter appears in the key.

is represented by a set of one or more *codenumbers* or *homophones*. For example, the letter 'A' might be represented by the set $\{10, 20, 23\}$. The plaintext is encrypted character by character: to encrypt each character, any one of the corresponding codenumbers is selected. Thus, the multiple-substitution cipher generalizes the well-known simple substitution cipher in which each letter is represented by a single codenumber. Although each plaintext can be enciphered in possibly many different ways, the codenumber sets are required to be disjoint so that each codenumber represents exactly one letter.

The major advantage of multiple substitution over simple substitution is that it smooths out the statistics in the ciphertext. One drawback, however, is that it expands the message: because there are more codenumbers than plaintext letters, more bits are required to identify the codenumbers in the ciphertext than are required to identify the letters in the plaintext. Because the sender can select which of the possible codenumbers to use at random, the multiple-substitution cipher is a randomized cryptosystem [20]. The most famous example of a multiple-substitution cipher is the Beale cipher, an unbroken portion of which supposedly describes the location of approximately four tons of gold, silver, and jewels allegedly buried in 1819 and 1821 in Virginia by Thomas Jefferson Beale [12, pp. 771–772].

Instructions for the Decipher Puzzle explain that the codenumbers were selected in the following special way. A sequence of letters, which we shall call the *key*, was written down with each letter numbered sequentially. The codenumbers corresponding to each letter are the position numbers of all occurrences of the letter in the key. For example, as shown in figure 2, if the key is "FOUR SCORE AND SEVEN YEARS...", then the codenumbers corresponding to the letter 'S' include 5, 13, and 22. The instructions do not explain how to deal with peculiarities of the keytext such as punctuation, footnotes, figure captions, and special characters.

The key was derived from some text, which we shall refer to as the *keytext*. The instructions hint that the mapping from keytext to key—which we shall call the *key-extraction strategy*—involves some type of unspecified manipulation of the keytext, such as eliminating every other word or eliminating every other vowel. The instructions also hint that other variations, such as inserting nulls in the plaintext or ciphertext, or using more than one key, may be present. In addition, the puzzle came with eight ambiguous clues (see figure 3).

After reading about the contest in the MIT student newspaper [13] in September 1983, Alan Sherman purchased a copy of the puzzle at a local game store and decided to run a mini-course on the puzzle during the January 1984 Independent Activities Period at MIT. Five people signed up for the course, including Robert Baldwin. Each participant signed a pledge

1. The key is in the public domain—you have easy access to it.
2. 3,19.
3. The first place to start may not be the first.
4. A cube is a clue if you are geometrically inclined.
5. 300 is closer to 'A' than 'Z'.
6. One source will do but what will you do with it?
7. Some numbers are repeated but do not be fooled.
8. If you knew it began with 'C', would it help you?

Figure 3: The eight original clues.

stating that if the group solved the puzzle by the March 1, 1984 deadline, all prize money would be used to establish an undergraduate scholarship at MIT. During the course Sherman and the others discussed ways to attack the cipher, refined Sherman's design of a cryptanalysis engine to carry out the attack, and thought about possible keytexts. Baldwin and Sherman implemented an initial version of their cryptanalysis engine and tried several keytexts without success. Although the mini-course did not solve the puzzle, it laid the foundation for our later solution.

The instructions do not explain what is required to win. Is it sufficient to find the plaintext, or is it required to find the entire key? Knowing the plaintext is equivalent to knowing which letters correspond to the 329 distinct codenumbers that appear in the ciphertext; it might be possible to learn the plaintext without finding the entire key. We assumed it was sufficient to discover the plaintext. In any event, the unique "solution" was resting securely in a safe-deposit box in a major bank in New York City, to be opened if no one solved the puzzle by June 30, 1989.

The rest of this paper is organized in four sections. Section 2 calculates the unicity distance of the cipher and analyzes the distribution of codenumbers in order to determine if it is possible to solve the puzzle from the available information. Section 3 describes our approach to the puzzle, including our cryptanalysis engine, our "windowing" technique for checking when part of a candidate key is correct, and our test for English based on digraph frequencies. Section 4 reveals the solution and explains how we applied our approach to solve the puzzle. Finally, section 5 reflects on our experiences with the puzzle.

2 Was the Contest Fair?

Before attempting to solve the puzzle, we first tried to determine if the puzzle was "fair." Was it possible to solve the puzzle from the available information? We answered this question by calculating the *unicity distance* of the cipher, which is the amount of ciphertext required for there to be a unique solution to the cryptogram without regard to the amount of work required to obtain the solution. We also analyzed the distribution of codenumbers, hoping to find statistical regularities that we could exploit. We found that the puzzle can be solved, but only by exploiting the clues. In addition, although the distribution of the codenumbers is almost entirely flat, the repeated codenumbers reveal some information about how the key was extracted from the keytext.

2.1 Unicity Distance Calculations

Modeling the cipher as a "random cipher" the unicity distance [23, 10], measured in number of ciphertext characters, is

$$U = \frac{\lg K}{D}, \tag{1}$$

where K is the number of keys and $D \approx 3.2$ bits/character is the redundancy of English [24].[2] Equation 1 gives the amount of ciphertext required to determine the key. Thus, every cryptanalyst needs at least U characters of ciphertext to determine the entire key; possibly a smaller amount would be sufficient to determine the plaintext. Because the contest did not fully specify what cipher system was used, we calculated the number of keys in three different ways. First, we assumed the key consisted of 1252 letters drawn at random. Second, we assumed the key consisted of 1252 letters drawn from English. Third, we estimated the number of possible keys taking the hints into consideration. Another estimate of the unicity distance can be obtained by modeling the cipher as an arbitrary multiple-substitution cipher [17, pp. 733–740].

Throughout this paper let $L = 1252$ be the number of possible codenumbers, and let $L' = 329$ be the number of distinct codenumbers that appear in the ciphertext. In addition, let $t = 26$ denote the number of letters in the English alphabet.

If the key is simply any sequence of L characters, then the number of keys is t^L and hence by equation 1 the unicity distance is

$$U_1 = \frac{\lg(t^L)}{D} = \frac{L \lg t}{D} \approx \frac{1252 \lg 26}{3.2} \approx 1839. \tag{2}$$

Thus, if the key were any sequence of L characters, then 376 codenumbers would be insufficient to produce a unique key and the puzzle would be unfair. To discover the plaintext only, it would suffice to determine the L' characters of the the key that were actually used; in this case, the unicity distance is

$$U_1' = \frac{\lg(t^{L'})}{D} = \frac{L' \lg t}{D} \approx \frac{329 \lg 26}{3.2} \approx 483, \tag{3}$$

which still exceeds the amount of available ciphertext.

If the key is restricted to being L characters of English, then the number of bits of information in the key is Lr, where $r = (\lg t) - D \approx 1.5$ bits/character is the information rate for English. In this case, the unicity distance is

$$U_2 = \frac{Lr}{D} \approx \frac{(1252)(1.5)}{3.2} \approx 587. \tag{4}$$

This second estimate of the unicity distance also exceeds the amount of available ciphertext.

[2]Throughout this paper let $\lg = \log_2$ and $\ln = \log_e$.

Number of occurrences	Number of codenumbers	List of repeated codenumbers
6	1	342
5	1	876
4	1	294
3	6	6, 26, 84, 209, 749, 780
2	23	8, 18, 107, 111, 229, 312, 366, 408, 477, 492, 494, 545, 598, 652, 735, 757, 797, 812, 860, 867, 1002, 1102, 1171
1	297	—
0	923	—

Figure 4: Distribution of occurrences of codenumbers, with lists of repeated codenumbers.

To estimate the number of possible keys taking the hints into consideration, we reasoned as follows. Since we have "easy access" to the keytext, the keytext is almost certainly limited to at most 10^8 works (the Library of Congress contains approximately 52 million books and manuscripts [16]).[3] Assuming each work has approximately 10^6 characters (a 400 page book with 500 words per page and 5 letters per word has 10^6 letters), there are approximately 10^6 possible starting positions for each work. The key-extraction strategy probably comes from a set of at most 100 relatively simple strategies. Hence, there are at most approximately $10^8 \cdot 10^6 \cdot 10^2 = 10^{16}$ possible keys. Under this assumption, the unicity distance is

$$U_3 \leq \frac{\lg\left(10^{16}\right)}{D} \approx \frac{16 \lg 10}{3.2} \approx 17. \tag{5}$$

Thus it is possible to solve the puzzle, but only by taking advantage of the clues.[4] The puzzle is fair given the clues.

2.2 Analysis of the Distribution of Codenumbers

Unfortunately, as shown in figure 4, the distribution of codenumbers in the ciphertext is almost entirely flat: of the 329 distinct codenumbers in the ciphertext, 297 of them occur exactly once. Only 32 codenumbers are repeated, including 342 which appears 6 times, 876 which appears 5 times, and 294 which appears 4 times.

We wondered what caused this distribution. Why did Holland repeat codenumbers? Did he repeat codenumbers because he was forced to? Holland would have been forced to repeat a codenumber only if the corresponding letter appeared more often in the plaintext than in the key. Assuming that Holland repeated codenumbers because he was forced to, we wondered what key-extraction strategies are consistent with the observed distribution.

[3] Although this conservative upper bound on the number of keytexts suffices for our purposes, the number of possible keytexts is probably much less since many of the works in the Library of Congress are obscure, not in English, or otherwise inappropriate for use as a keytext.

[4] Our unicity distance calculations are based on modeling the cipher as a random cipher. As discussed by Hammer [8], however, it is possible to build a homophonic cipher that can hide two or more meaningful messages.

Any key-extraction strategy that yields letter frequencies different from those expected in English tends to force repeated codenumbers. For example, we expect the strategy of taking the first letter of each word to force repeated codenumbers because some letters such as 'E' appear frequently in English but relatively infrequently as the first letter of a word. Similarly, we expect repeated codenumbers for taking the last letter of each word and for deleting every other vowel. The rest of this section analyzes how well the observed distribution of codenumbers matches the expected distribution for the following three key-extraction strategies: all letters of word, first letter of word, last letter of word. Our analysis assumes that Holland attempted to avoid repeating codenumbers.

Throughout this paper, let $n = 376$ be the length of the plaintext and ciphertext, and let $a_1, a_2, \ldots, a_t$ be the 26 letters of the English alphabet. In addition, for each $1 \leq i \leq t$, let p_i be the probability with which letter a_i appears in English.

For any key-extraction strategy, the expected number of repeated codenumbers for any letter is the expected number of times the letter appears in the plaintext minus the expected number of times the letter appears in the key, provided this difference is nonnegative. Hence, in particular, if the key were drawn from English, then for each $1 \leq i \leq t$ the expected number of repeated codenumbers corresponding to letter a_i would be $\max(0, np_i - Lp_i)$, which equals zero because $L = 1252 > 376 = n$. Therefore, if the plaintext were drawn from standard English, it is unlikely that the key was also drawn from English.

If the key were drawn from initial letters of English words, then for each $1 \leq i \leq t$ the expected number of repeated codenumbers corresponding to a_i would be $\max(0, np_i - Lp_i^f)$, where p_i^f denotes the probability with which letter a_i appears as the first letter of a word in English. Thus, for this case, we would expect repeated codenumbers only for those letters a_i for which $p_i/p_i^f > L/n = 1252/376 \approx 3.3$, that is, only for those a_i which appear approximately three times more frequently in any position than as an initial letter of a word. Using the statistics given by Sinkov [25, pp. 177–178], we determined that the total expected number of repeated codenumbers is approximately 19, with 18 repeats caused by the letter 'E' and 1 repeat caused by the letter 'X'. These numbers are consistent with the observed distribution which contains 32 repeats, suggesting that the key-extraction strategy might be to take the first letter of each word.

Similarly, for taking the last letter of each word, we expect a total of 43 repeats, with 22 repeats for the letter 'I', 8 repeats for the letter 'U', 4 repeats each for the letters 'V' and 'P', 3 repeats for the letter 'C', and 1 repeat each for the letters 'B' and 'Q'. Again, these numbers are consistent with the observed distribution.

Taking our analysis one step further, for each key-extraction strategy it is possible to predict what letters the repeated codenumbers probably stand for. If the key were drawn from first letters of words, then it is likely that repeated codenumbers represent letters with high $np_i - Lp_i^f$ values. Listed in decreasing order of $np_i - Lp_i^f$ values, the letters of the alphabet are 'E', 'X', 'Z', 'Q', 'N', 'Y', 'V', 'K', 'J', 'U', 'R', 'L', 'G', 'D', 'M', 'H', 'F', 'I', 'P', 'B', 'W', 'C', 'O', 'S', 'A', 'T'. Similarly, if the key were drawn from last letters of words, we would expect in decreasing probability the repeated codenumbers to be 'I', 'U', 'V', 'P', 'C', 'B', 'Q', 'J', 'Z', 'X', 'W', 'M', 'A', 'K', 'H', 'O', 'G', 'L', 'R', 'F', 'Y', 'T', 'N', 'D', 'S', 'E'. Regardless of what key-extraction strategy is used, it is likely that the repeated codenumbers represent letters that appear relatively frequently in the plaintext but relatively infrequently in the key. This situation might arise with an unusual plaintext that contained many unpopular letters.

The essentially flat distribution of codenumbers suggests that statistical attacks based on repeated codenumbers will be of virtually no use. Assuming that Holland tried to avoid repeating codenumbers, this distribution, however, does reflect properties of the key-extraction strategy. In particular, the distribution is consistent with taking first or last letters of English words and inconsistent with taking an entire English keytext.

2.3 Discussion

We speculate that Holland intentionally designed the puzzle to be controlled by the clues. Doing so would accomplish the following three objectives. First, by forcing each contestant to solve the puzzle by guessing the keytext, Holland can control the difficulty of the puzzle by judiciously choosing the clues. To maximize public interest in this and future puzzles, the puzzle should be solvable yet the puzzle must not be too easy nor too hard. A puzzle that is too easy would be solved too soon, thereby reducing sales; a puzzle that is too hard might discourage contestants from buying future puzzles. Second, Holland can ensure that the puzzle will eventually be solved by giving away sufficiently many additional clues. Third, by forcing contestants to guess the keytext, lay people and mathematicians are put on a roughly equal footing.

3 Our Approach

Our plan was to search for the key by intelligently guessing possible keytexts and by using a computer to test each candidate keytext. To test each candidate keytext, the computer would try a collection of key-extraction strategies. For each strategy, the computer would try all possible starting positions in the keytext. We reasoned that since the only way to solve the puzzle was to determine the keytext and the key-extraction strategy, our plan made best use of available information.

To help carry out our plan, we designed and implemented a computer program to test candidate keytexts. Our program ran on a Symbolics 3600 Lisp Machine; including comments it consisted of approximately 1,500 lines of Zetalisp [18].

Figure 5 shows a block diagram of our program. The program accepted two inputs—a ciphertext and a candidate keytext. For each possible key-extraction strategy and for each possible starting position in the candidate keytext, the program extracted a corresponding candidate key and tested it. To test each candidate key, the program deciphered the ciphertext under the candidate key and checked the resulting candidate plaintext for English. For each candidate plaintext that scored sufficiently high for English, our program recorded the plaintext together with the corresponding key and key-extraction strategy into a log file. A crucial feature of our program was a "windowing" technique that enabled us to detect when part of a key was correct.

The rest of this section describes our cryptanalysis engine in more detail, focusing on how we extracted candidate keys, how we checked each candidate plaintext, and how we detected when part of a key was correct. We also discuss our ideas for mounting probable-word attacks.

Figure 5: Block diagram of cryptanalysis engine. Each candidate keytext is checked by extracting candidate keys using a variety of key-extraction strategies and all possible starting positions. Each candidate key is checked by deciphering the ciphertext and testing the resulting candidate plaintext for English. Filters define the space of key-extraction strategies to be tried; windows specify subsequences of the key to be checked separately.

3.1 Extracting Candidate Keys

To speed up the extraction of candidate keys and to facilitate the specification of key-extraction strategies, our program exploited two special features. The first feature was a keytext data structure consisting of a network of arrays with pointers to beginnings of sentences, words, and vowels. As shown in figure 5, our program began testing a candidate keytext by reading the keytext from a text file into the keytext data structure. Although each keytext text file contained all punctuation, special characters, and line breaks as they appeared in the original keytext document, the keytext data structure contained only alphabetic characters. All of the key-extraction strategies could be defined in terms of scanning down the keytext data structure. Using this data structure enabled us to extract keys very quickly.

The second feature of our program was a convenient method for describing a large space of key-extraction strategies by specifying a small set of "generators" for the space. Most of the key-extraction strategies we considered were ones like "take the last letter from each word" or "skip every other word, take every letter from each remaining word, then take the first letter after each vowel." Each of these strategies can be defined as a combination of three types of generators. The first generator describes how to filter words from a sequence of words. The second generator describes how to extract characters from a single word. The third generator describes how to filter characters from a sequence of characters. We specified a number of generators of each type; our program tried the set of all key-extraction strategies formed by taking all sensible combinations these generators, one from each type. Using this framework it was easy to define all of the key-extraction strategies that we wanted to test.

In practice we used the following generators. *Word filters:* all words, skip every k^{th} word. *Word-to-character filters:* all letters of word, k^{th} letter of word (this includes last letter of word), skip first k letters of word, first and last letter of word. *Character filters:* all characters, all characters except doublets, skip every other vowel, k^{th} letter after vowel. Some of these generators depend on a parameter k; for each of these generators, our program tried all values of k within a range specified by the generator. For example, in this terminology the second page of the Beale ciphertext is deciphered when the keytext is the Declaration of Independence and the key-extraction strategy is defined by the generator triple $\langle$all words, 1^{st} letter of word, all characters$\rangle$, which describes the strategy of taking the first letter of each word.

3.2 Checking a Candidate Plaintext

To check when we encountered a valid plaintext, we devised an efficient test for English. The test measures how "close" a candidate plaintext is to English by computing a statistic from the observed digraph frequencies. If the value of the statistic falls within a specified threshold around the expected value of the statistic for English, the candidate plaintext is considered valid. This test played a crucial role in automating our cryptanalysis engine so that it could run quickly and unattended for the many hours needed to check numerous keytexts.

We considered two different approaches for checking a candidate plaintext—searching for English words and comparing statistical properties of the plaintext with those expected for English. Although both approaches are promising, we focused on the statistical approach because it is more flexible in coping with nulls and in enabling us to check when part of a candidate key is correct. We used digraph frequencies rather than single-letter frequencies because they provide a more powerful test and because it was necessary to be able to distinguish second-order English from first-order English since many of the incorrect candidate keys generated candidate plaintexts with first-order English statistics.[5] We did not use trigraph frequencies because they do not work well with our method for testing when part of a candidate key is correct and because they are more complicated and time-consuming to deal with than are digraph frequencies.

The basic idea of our test is to examine a collection of digraphs from the candidate plaintext and to determine how likely it is for these digraphs to appear in English. Input to our test is a set of m digraphs $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$. For example, D might consist of all pairs of adjacent letters in the candidate plaintext, or D might be a selected subset of such pairs. By allowing the test to operate on selected digraphs, we were able to detect when part of a candidate key was correct.

One of the simplest statistics that might be tried is the product of the digraph probabilities

$$r' = \prod_{i=1}^{m} \mathrm{Prob}[x_i y_i] \tag{6}$$

or its $\log_e$-transform

$$r = \ln r' = \sum_{i=1}^{m} \ln \mathrm{Prob}[x_i y_i], \tag{7}$$

where $\mathrm{Prob}[xy]$ denotes the probability with which the digraph xy appears in English. The statistic r is the second-order analog of a first-order statistic suggested by Sinkov [25, pp. 76–77]. The advantages of using r over r' are that r is faster to compute and r has a less skewed distribution.

We, however, computed our statistic $\hat{s}$ as follows. First, we computed

$$s = \sum_{i=1}^{m} \ln \mathrm{Prob}[x_i y_i | x_i], \tag{8}$$

where $\mathrm{Prob}[xy|x]$ denotes the probability that the letter y immediately follows the letter x in English, given that x has already appeared. Second, to express our statistic on a convenient scale, we centered and normalized s by computing

$$\hat{s} = \frac{(s/m) - \mu_{s_E}}{\sigma_{s_E}/\sqrt{m}}, \tag{9}$$

where $\mu_{s_E} \approx -2.51$ and $\sigma_{s_E} \approx 0.98$ are, respectively, the mean and standard deviation of s when applied to English.[6] In equation 8, we used the digraph probabilities for English given by Beker and Piper [1, pp. 398–399]; these probabilities were calculated from 132,597 characters of English drawn from various newspapers and novels.

We based our statistic on s rather than on r for two reasons. First, when we originally developed the statistic, we considered its input to be a sequence of *contiguous* letters $(z_1, z_2, \ldots, z_m)$ rather than a set of *independent* digraphs. In second-order English,

[5] By *first-order English* and *second-order English*, we mean a language source that generates letters according to the expected single-letter and digraph English probabilities, respectively.

[6] We computed the values μ_{s_E} and σ_{s_E} using the definitions of mean and standard deviation [15].

$\text{Prob}[z_1z_2...z_m] = \text{Prob}[z_1]\text{Prob}[z_1z_2|z_1]\text{Prob}[z_2z_3|z_2]\cdots\text{Prob}[z_{m-1}z_m|z_{m-1}]$. Second, when we changed the input to be a set of independent digraphs, the statistic still worked well and we never bothered to modify it. Part of the reason why our statistic worked well on sets of independent digraphs is that we usually applied the test to distinguish second-order English from first-order English. For a set of independent digraphs D, $s(D)$ is related to the odds in favor of the candidate plaintext being drawn from second-order English as opposed to being drawn from first-order English given D [7]. Specifically,

$$\frac{\text{Prob}[E_2|D]}{\text{Prob}[E_1|D]} = \frac{\text{Prob}[E_2]\text{Prob}[D|E_2]}{\text{Prob}[E_1]\text{Prob}[D|E_1]} = \frac{\text{Prob}[E_2]}{\text{Prob}[E_1]}\frac{\text{Prob}[x_1y_1|x_1]\text{Prob}[x_2y_2|x_2]\cdots\text{Prob}[x_my_m|x_m]}{\text{Prob}[y_m]}, \tag{10}$$

and hence

$$s = \left(\ln\frac{\text{Prob}[E_2|D]}{\text{Prob}[E_1|D]}\right) - \left(\ln\frac{\text{Prob}[E_2]}{\text{Prob}[E_1]}\right) + \ln\text{Prob}[y_m]. \tag{11}$$

Here, E_1 and E_2 denote the hypotheses that the candidate plaintext was drawn from first-order and second-order English, respectively.

If the candidate key is correct, then we expect the resulting candidate plaintext to be drawn from English. But what should we expect if the candidate key is incorrect? In this case, the expected statistical properties of the resulting candidate plaintext depend on the keytext and on the key-extraction strategy. An incorrect candidate plaintext might resemble anything from random text drawn from a uniform distribution to highly structured text with more distinctive digraph statistics than those for English. For example, if the candidate key consists of all letters from an English keytext, then an incorrect candidate plaintext will resemble a random sampling of letters drawn from English and hence is expected to have first-order English statistics. Similarly, the key-extraction strategy of taking the first letter of each word tends to produce candidate plaintexts in between first-order and second-order English. In practice, since all of our key-extraction strategies ultimately extract letters from English keytexts, our candidate keys—and hence our incorrect candidate plaintexts—tend to resemble first-order English.

Figure 6 shows the probability density function for the s-statistic when computed on English, first-order English, and random text drawn from a uniform distribution. Let μ_{s_1} and σ_{s_1} denote, respectively, the mean and standard deviation of s when applied to first-order English. Since μ_{s_E} is quite far from μ_{s_1} in terms of σ_{s_E} and σ_{s_1} units, the $\hat{s}$-statistic provides an effective tool for distinguishing standard English from first-order English.

Our decision to accept or reject a candidate plaintext was based on how far the observed value of $\hat{s}$ fell from the expected value of $\hat{s}$ when computed on English. Specifically, we accepted a candidate plaintext if and only if $|\hat{s}| \leq \tau$, where τ was a control parameter. By choosing $\tau > 0$ appropriately we could select our desired false-positive and false-negative error rates. In practice we used $\tau = 4$; that is, we accepted all candidate plaintexts whose $\hat{s}$-statistics fell within four standard deviations of their expected values for English, yielding a very small false-negative error rate of less than .01 percent and an acceptable false-positive error rate. We chose this relatively large value of τ to help ensure that we would not miss the correct solution, even if it contained nulls, contained many unusual words, or otherwise had an unusual distribution; at $\tau = 4$, our test might even detect a plaintext written in a foreign Indo-European language.

Figure 6: Probability density functions for the s-statistic when computed on English, first-order English, and random text, yielding the density functions f_{s_E}, f_{s_1}, and f_{s_R}, respectively. The corresponding mean and standard deviation values for s are $\mu_{s_E} \approx -2.51$, $\mu_{s_1} \approx -3.47$, $\mu_{s_R} \approx -4.97$, and $\sigma_{s_E} \approx 0.98$, $\sigma_{s_1} \approx 1.52$, $\sigma_{s_R} \approx 2.35$, respectively.

There are three details about our test worth discussing. First, there is the question of how to deal with unknown input characters, which might be present in a candidate plaintext generated from an incomplete key. We dealt with this problem by eliminating any digraph from consideration that had one or two unknown characters. In addition, we counted the number of eliminated digraphs and rejected the candidate plaintext if it had too high a percentage of eliminated digraphs. The issue of unknown digraphs was not a problem for us in practice because it did not arise except for candidate keys generated from near the end of a candidate keytext.

Second, there is the question of how to deal with impossible digraphs such as 'QZ'. One possibility is to reject any candidate plaintext that contains one or more impossible digraph. This approach, however, is too inflexible: it cannot deal with minor errors in the keytext, key, or ciphertext, and it is easily defeated by interjecting random characters into the plaintext or ciphertext. Therefore, we allowed impossible digraphs but assigned each of them a near-zero probability of $\epsilon > 0$ (in practice we used $\epsilon = 10^{-7}$). Note that the $\hat{s}$-statistic does not allow any digraph probability to be exactly zero.

Third, there is the issue of how the number of digraphs affects the sensitivity of the $\hat{s}$-statistic. We had expected that the larger the input, the better our test would perform. To our surprise, our test nearly rejected some very large English texts. After some thought and experimentation, we came up with the following explanation. As the number of digraphs m increases, the standard deviation of s/m decreases; consequently, the test becomes increasingly selective in what it considers to be English. The trouble is that there are many different types of English—newspaper English, military English, and poetry English, for example. We computed our test using digraph frequencies from one sample of English newspapers and novels.

As the number of digraphs increases, our test becomes increasingly sensitive to the different types of English and hence begins to reject English texts drawn from types of English different from those on which our digraph frequencies were based. In practice the sensitivity issue was not a problem because we used a relatively large threshold value τ and because we tested the plaintext in small pieces.

Although our statistic $\hat{s}$ worked remarkably well in practice, both it and the statistic r have the drawback that they do not measure any variation in the observed digraph frequencies. Therefore, each of these statistics can be fooled by a candidate plaintext that consists solely of a repeated typical digraph. To get around this problem it might be helpful to develop a test for English using the index of coincidence [1, pp. 39–50] to compare the observed digraph frequencies with those expected from English.

3.3 Detecting Partial Solutions

One problem with our initial design was that is was too sensitive to errors in the keytext. For example, a single character error in the candidate keytext might cause all corresponding characters in the candidate key past that error to be wrong, which might cause most of the candidate plaintext to be wrong, which in turn might cause our system to overlook the solution. Similarly, our initial system might not have detected when it was close to the correct key-extraction strategy. To overcome these difficulties we introduced the concept of "windowing," which enabled us to detect when part of a candidate key is correct.

A *window* is a subsequence of the key. For example, codenumbers 1–50 constitute a window. As shown in figure 7, each window determines various characters throughout the plaintext. When a window determines two or more adjacent plaintext characters, the window can be checked separately from the rest of the key by examining the affected adjacent plaintext characters. For example, codenumbers 933–1084 determine the following nine digraphs in the ciphertext: (933, 935), (935, 934), (963, 964), (986 969), (1011, 963), (1002 939), (925, 1026), (1084, 1016), (1082, 910). With the help of a computer, we found several relatively short windows that each determined several plaintext digraphs. Specifically, we used the following five windows: 16–113, 144–294, 243–418, 724–877, 933–1084. Each of these windows produce approximately 10 digraphs. For each candidate key, instead of checking the entire key at once, we separately checked each of its windows using our English test. This windowing technique proved to be an extremely effective way of dealing with errors and ambiguities in the candidate keytexts.

3.4 Probable-Word Attacks

Early in our deliberations we considered the possibility of mounting probable-word attacks. Unfortunately, these attacks have little to offer over our main approach. This section briefly describes some of our ideas.

A probable-word attack can be mounted against either the keytext or the plaintext. Unless the key-extraction strategy preserves all or most of the letters, however, attacking the keytext is unlikely to be of any use. If the key is drawn from English, then probable words and phrases in the keytext can be checked using the windowing technique. Finding a word in the keytext

Figure 7: Windowing strategy. Any subsequence of a candidate key can be separately checked by examining the digraphs it determines in the resulting candidate plaintext.

might help in checking candidate keytexts, but since the number of digraphs affected by a typical probable phrase is very small, this attack is unlikely to be of much use.

Mounting a probable-word attack against the plaintext is very much like using the windowing technique in reverse: each word in the plaintext affects certain key characters, which through the key-extraction strategy affect certain keytext characters. Since the effect of each plaintext word is so spread out over the keytext, such attacks are very weak. Nevertheless, probable-word attacks against the plaintext can be used in checking a candidate keytext or in determining the key-extraction strategy under the assumption that a particular keytext is used. But since our main method already provides an effective way of checking a candidate keytext, probable-word attacks for the purpose of checking a candidate keytext offer little value. Perhaps the greatest value of a probable-word attack lies in completing a partial solution.

One idea we explored briefly was to guess the entire plaintext. Suspecting that the plaintext might be poetry, we searched for a well-known poem or stanza of length exactly 376 characters. Among the poems we considered were "An enigma," "Sonnet—to science," and "A valentine" by Edgar Allan Poe [9, pp. 110, 22, 115]. We suspected Poe because of his interest in cryptology, and we suspected "A valentine" because it encodes the name of its recipient. We never did find a poem of exactly 376 characters.

We compiled lists of probable words, including the following: 'congratulations', 'decipher puzzle', 'one hundred thousand dollars', 'science', 'secret', 'solution', 'winner', 'you have won', and names of famous authors and poets. We also considered the possibility of trying the first lines of all famous poems.

4 The Solution

Sherman quickly assembled the two-sided jigsaw puzzle and began analyzing the distribution of codenumbers. Holland's decision to put the ciphertext on a puzzle was clever—even if the contestant cannot discover the plaintext, he can almost certainly put together the jigsaw puzzle which lets him feel as if he has made significant progress.

In hindsight many features of the puzzle appear straightforward, but before we solved the puzzle we wondered about many basic questions. Is each codenumber a homophone representing one letter? Does the deciphering process involve manipulating individual digits of the codenumbers? Is the plaintext in English? In what order should the codenumbers be read? Is the layout of the codenumbers on the puzzle significant? How many keys were used? Although we briefly experimented with other possibilities, we took as working assumptions that each codenumber represented one plaintext letter and that the plaintext had statistical properties similar to those of English. Throughout our work we left open the possibility that the codenumbers might be read in a nonstandard order (*e.g.* reverse order or taking every every other codenumber) and that different portions of the plaintext might be encrypted using different keys.

To help us find the keytext and the key-extraction strategy, we thought about requirements for these unknowns. Since everyone has "easy access" to the keytext, we reasoned that it was either in most libraries or included in the materials that came with the puzzle. Moreover, it was probably something from which it is easy to extract a reliable sequence of characters—it probably did not have mosaics of words or multiple dissimilar editions. There must a sufficient amount of text to support the key-extraction strategy. As for the key-extraction strategy, we reasoned that is was probably something that could be easily carried out by hand without the use of a computer—many people would consider it unfair to require contestants to use a computer.

From the beginning, we strongly suspected that the key-extraction strategy was either to take the first letter of each word or to take the last letter of each word. These strategies were consistent with the observed distribution of codenumbers (see section 2.2) and they had the appropriate level of difficulty. Moreover, part of the Beale cipher extracted the initial letters of words from the Declaration of Independence. From clue 6 (see figure 3) and from the distribution of the codenumbers, we felt quite strongly that the key did not include every letter of the keytext. Our belief in the importance of the Beale cipher was strengthened when we discovered that Daniloff's [3] article about the Beale cipher, which was referenced in the puzzle instructions, mentioned a Beale cipher enthusiast named Colonel J. J. Holland.

Although the antecedent of 'it' in clue 8 is ambiguous, we thought this clue probably meant the keytext or its name begins with the letter 'C'. This led us to suspect the puzzle instructions, which begin with the word 'Congratulations', and the Constitution. We thought that the numbers '3, 19' in clue 2 probably serve as an index into the keytext (*e.g.* page 3, line 19) or represent the letters 'C' and 'S', which are the third and nineteenth letters of the alphabet. We noted that 'congratulations' begins with 'C' and ends with 'S', which led us to suspect the key-extraction strategy of taking both the first and last letters of each word.

After completing an initial implementation of our cryptanalysis engine in early February 1984, we began checking candidate keytexts. Without success we tried the Declaration of Independence, the Constitution, the contest instructions, and Daniloff's article. We also considered but did not check other famous documents including the Bible, the Articles of Confederation, the Virginia Bill of Rights, and the Gettysburg Address. Our usual mode of operation was to let our engine run overnight unattended and to check its log file in the morning.

One practical problem we faced was how to enter the keytexts. We did not have easy access to an optical scanner, and the available optical scanners at that time required a lot of adjustment to use. We solved this problem by finding volunteers and by hiring a secretary to type in the keytexts.

At times we were concerned about the security of our system: we feared that, if we solved the puzzle, someone might steal our solution. We handled this problem in two ways. First, we introduced a moral barrier. To run our software, the user had to state that he was an authorized user; a dishonest person could say yes, and the system would start working. Second, we agreed that if we solved the puzzle, we would continue to tell everyone (except our girl friends) that we failed to solve the puzzle until after the deadline.

After our initial attempts at solving the puzzle failed, we put the puzzle aside and went back to work on our dissertations. When the March 1, 1984 deadline passed with no one submitting the correct solution, Holland extended the contest for another year under the following modified rules. All registered contestants who submit the correct solution by February 28, 1985 will share the prize. If no one solves the puzzle by February 28, 1985, then the contest will proceed on a monthly basis, with the prize to be shared among all registered contestants who submit the correct solution during the first month in which the puzzle is solved. Holland announced that the keytext was a "popular novel" written by one of twenty-one specified authors, including Woody Allen, William F. Buckley Jr., Norman Mailer, James Michener, and Carl Sagan. In addition, Holland established a "hotline" to distribute additional clues by telephone on a monthly basis beginning May 1, 1984. Two authors would be eliminated from the list each month. To sweeten the prize, Holland promised to increase the prize by \$1,000 plus interest each month. Our plan was to wait until January 1985 to resume work. We also agreed that if we solved the puzzle, we would give half of our winnings to charity.

By January 1985 the list of authors had been narrowed to Carl Sagan and one other author. Because of the 3,19 clue (clue 2), we strongly suspected that Carl Sagan was the author of the keytext. At this point Holland's clues puzzled us: because of the letter 'C' clue (clue 8) we strongly suspected that *Cosmos* [22] was the keytext, but *Cosmos* is not a novel because it is not a work of fiction. Did Holland inadvertently misuse the noun 'novel' in the sense of 'book'? Although we also tried parts of Sagan's only popular novel—*The Dragons of Eden* [21]—we concentrated our efforts on *Cosmos*.[7] Without success we tested chapter 10 (which contains a picture of a cube [22, p. 262]) and chapter 12 (which discusses the Rosetta stone [22, p. 294]). Also without success we experimented with various ways of reordering the codenumbers based on a cube. Had we hired someone to type in *Cosmos* in its entirety, we would have likely solved the puzzle by the end of January.

When no one solved the puzzle by February 28, 1985, Holland announced that the key was a sequence of "first letters" from *Cosmos*, chapter 6. Since there are less than 1252 sentences or lines in chapter 6, we were now practically certain that the key was drawn from first letters

[7] An anonymous referee pointed out that *Cosmos* is copyrighted and hence not in the public domain. In clue 1 did Holland use the phrase "in the public domain" for its colloquial meaning of "publicly available"?

of words. In hindsight we discovered another pointer to chapter 6: for the country Holland the index lists pages 140–143, which are in chapter 6. Confident that we would quickly solve the puzzle, we hired a secretary to type in chapter 6 and went back to more pressing tasks of finishing our dissertations, writing conference papers, and applying for jobs. Since we didn't have much money, we offered the secretary a share of any winnings we might receive.

To our surprise, our engine did not immediately find the solution even though we tried all possible starting positions in chapter 6 for each of several key-extraction strategies including taking the first letters of all words. As we suspected and later confirmed, the problem was that there were some minor errors in our keytext. For example, we included the 'c' in the abbreviation 'c.' for 'circa'; Holland did not. Similarly, we counted the abbreviation 'JPL' as three words; Holland deleted it. There were also other similar questions such as how to treat hyphenated words and whether or not to include footnotes and figure captions. To deal with these errors and ambiguities in the keytext, we developed the windowing technique (see section 3.3).

On Friday March 29, we finished implementing our windowing technique and ran the engine again. This time our English test detected solutions for each of our windows, beginning with different starting positions for each window. With a little more work and with the help of the Webster dictionary program that can find pattern words, Baldwin detected and corrected all of the errors and ambiguities in the keytext. We obtained the correct key from chapter 6 of *Cosmos* by extracting the first letters of all words beginning on page 137 with the phrase "first ages of the world ..." treating hyphenated words as single words and deleting all numerals, abbreviations, footnotes, and figure captions. As shown in figure 8 with word breaks and punctuation added, the solution is a passage from "A poet's advice to students" by E. E. Cummings [6, p. 335].

Interestingly, according to our English test over 99.8 percent of all English texts score closer to English than did the correct plaintext, which has unusual digraph frequencies. For example, the trigraph 'you' appears nine times; the digraph 'no' appears five times; and the digraph 'th' appears only twice. Fortunately, we had set the threshold in our English test sufficiently large to catch even Cummings's English.

We then checked the instructions to review how to submit a solution. Oops! The solutions were due in Virginia on the last *business* day of the month—not on the last *calendar* day as we had assumed.

Having missed the deadline, we thought we might as well try to get some recognition for our efforts. On Monday morning we telephoned the MIT News Office and *The Boston Globe*. At

> Almost anybody can learn to think or believe or know, but not a single human being can be taught to feel. Why? Because whenever you think or you believe or you know, your're a lot of other people: but the moment you feel, you're nobody-but-yourself.
>
> To be nobody-but-yourself—in a world which is doing its best, night and day, to make you everybody else—means to fight the hardest battle which any human being can fight; and never stop fighting.
>
> From "A poet's advice" by E. E. Cummings.

Figure 8: Solution—a passage from "A poet's advice to students" by E. E. Cummings.

first the media doubted our story—Monday was April 1. A few days later *The Boston Globe* [5] and *The Tech* [14] carried the news.

By this time all of the clues except clue 5 made sense to us. We had interpreted clue 5 to mean that codenumber 300 is closer to 'A' than 'Z' in the alphabet, but codenumber 300 represents 'O', which is closer to 'Z' than 'A'.

As expected (see section 2.2), most of the repeated codenumbers represent letters that appear relatively frequently in the plaintext but relatively infrequently in the key. For example, the letter 'Y' appears 18 times in the plaintext but only 5 times in the key, and the codenumber set for 'Y' consists of the repeated codenumbers 26, 342, 780, 867, 876. Not all codenumber repetitions, however, were forced. For example, although the letter 'E' appears 46 times in the plaintext and 51 times in the key, the codenumber set for 'E' contains the unnecessarily repeated codenumbers 8, 18, 294, 312, 545, 757.

It turned out that 36 people submitted the correct solution in March 1985, including a farmer, two coal miners, a college math teacher, a biochemist, a psychologist, a medical student, and a 15-year-old girl [2, 19]. Each winner received a T-shirt and a check for $3,251.71. Although many of the winners used a computer, apparently all of them depended crucially on the massive February 1985 clue. The story of the $100,000 Decipher Puzzle has been reported in numerous newspapers and in a popular book by Paul Hoffman [11, pp. 70–78].

On July 6, 1989, Sherman spoke with Holland by telephone. Holland explained that he designed the clues to have multiple levels of interpretation and to challenge people to think more deeply; for most clues the "obvious" interpretation is incorrect. For example, clue 5 means that codenumber 300 is closer to 'A' than 'Z' not in alphabetical order but in the sense that codenumber 300 represents a vowel rather than consonant. Clue 4, which mentions a cube, refers to chapter 6 since a cube has 6 sides. Holland said that he had used the word 'novel' in the spirit of the clues to mean both 'book' and "new and unusual." The picture of a cube in *Cosmos*, chapter 10 and the references to the country Holland in chapter 6 were simply coincidences. There is no relationship between Warren Holland and Colonel J. J. Holland.

Although Holland would not reveal how many people purchased his puzzle, he did say that he received over 1,000 solutions and that there were approximately 250,000 calls to the hotline. Holland also said that he was surprised the puzzle was not solved within the first year. With the success of the Decipher Puzzle, Holland went on to produce similar Decipher II and Decipher III puzzles. Currently Holland has no plans for a Decipher IV; instead, he is focusing on a variety of new game and gift products, including a dinner party mystery game called "How to host a murder."

5 Conclusion

To solve the puzzle, it was necessary to guess the keytext and the key-extraction strategy. Our approach was to place ourselves in an optimal position to exploit the clues by building a computer program that automatically checks candidate keytexts. The program performed the tedious calculation of trying all possible starting positions for a variety of candidate key-extraction strategies. Two features of our cryptanalysis engine played a crucial role: our windowing technique enabled us to deal with errors and ambiguities in the candidate keytexts, and our test for English enabled us to automate our checking completely.

Mathematically, the most interesting part of our work is our test for English. Although our test worked very well for our application, there is undoubtedly much more to learn about the important problem of language detection and identification.

In addition to describing the Decipher Puzzle and our solution of it, this paper also points out the following weakness of Beale ciphers. If the statistical properties of the keytext are known, then the distribution of codenumbers reveals information about the key-extraction strategy. Conversely, if the key-extraction strategy is known, then the distribution of codenumbers reveals information about keytext. Although our solution did not depend on this weakness, this weakness helped draw our attention to the correct key-extraction strategy.

There are many lessons to be learned from our experience. One of the obvious lessons is "Read instructions carefully!" But beyond this obvious lesson, we would like to reflect on a number of points. First, throughout our cryptanalysis we found ourselves caught in the following dilemma: how can we tell how close we are to the solution? Although the windowing technique detects some near misses, and although interpretations of the clues provide some confirmation, for the most part it was virtually impossible to know whether or not we were near solving the puzzle. This made it difficult for us to decide how much time to dedicate to the puzzle.

Second, the computer's facility for doing tedious calculations is a double-edged resource. On the one hand, the computer was a tremendous aid in calculations such as trying all possible starting positions in a keytext. On the other hand, the computer posed a danger in inviting us immediately to follow the straightforward approach rather than to think about a possibly more effective clever approach. For example, to check a candidate key, the straightforward approach is to use it to decipher the *entire* ciphertext and to check the resulting candidate plaintext for English. But if the entire candidate plaintext is checked for English, poor results are obtained because this checking procedure is too sensitive to errors in the keytext. Someone solving the puzzle by hand would almost certainly check a candidate keytext without generating the entire key and without deciphering the entire ciphertext. By virtue of his limited computational powers, the hand-solver is practically forced to use a superior method that detects when part of a candidate key is correct! We do not wish to imply that straightforward methods are necessarily bad; nor do we wish to imply that computers cannot perform clever feats. We simply point out that computers can make it convenient for their users not to think deeply.

Third, we found ourselves more interested in the *process* of our solution than in the *solution* itself. For example, we preferred to spend our time analyzing our test for English or adding features to our cryptanalysis engine rather than to run our engine on candidate keytexts. This situation contributed to our missing the deadline.

Cryptanalysis is a complex activity involving probability, statistics, algorithms, computer systems, software and hardware engineering, and general problem solving. We hope the reader can learn from and enjoy our experiences in this fascinating area.

Acknowledgments

We would like to express our appreciation to several people who contributed to our solution of the puzzle. As members of the 1984 IAP mini-course 4505, Ravi Boppana, John Chang, Joseph

Marshall, and David Saslov participated in early discussions about the puzzle. Kent M. Pitman wrote some of the input routines for the cryptanalysis engine, and James D. McNamara, David Saslov, and Ken Story typed some of the candidate keytexts. All computer work was carried out on a Symbolics 3600 Lisp Machine that belonged to the Theory of Computation Research Group at the MIT Laboratory for Computer Science. In addition, we thank Jonathan Cox and Tomoko Shimakawa for drawing the diagrams, and we thank Maureen Brown and Tomoko Shimakawa for putting up with us during our codebreaking activities.

References

[1] Beker, H. and F. Piper 1982. *Cipher Systems: The Protection of Communications.* New York: John Wiley.

[2] Bryant, J. 1985. Puzzle players decipher cryptogram. *The Virginian-Pilot.* April 17: D6.

[3] Daniloff, R. 1981. A cipher's the key to the treasure in them thar hills. *Smithsonian Magazine.* April: 126–144.

[4] Denning, D. E. R. 1983. *Cryptography and Data Security.* Reading MA: Addison-Wesley.

[5] Dolnick, E. 1985. Puzzle wasn't what stumped them: Code-crackers let contest fall through a crack. *The Boston Globe.* April 4: 1,15.

[6] Firmage, G. J., ed., 1985. *E. E. Cummings: A Miscellany Revised.* October House, Inc.

[7] Good, I. J. 1965. *The Estimation of Probabilities: An Essay on Modern Bayesian Methods.* Cambridge: MIT Press.

[8] Hammer, C. 1981. Higher-order homophonic ciphers. *Cryptologia.* 5: 231–242.

[9] Harrison, J. A., ed., 1902. *The Complete Works of Edgar Allan Poe*, vol. VII. New York: Thomas Y. Crowell & Co.

[10] Hellman, M. E. 1977. An extension of the Shannon theory approach to cryptography. *IEEE Transactions on Information Theory.* IT–23: 289–294.

[11] Hoffman, P. 1988. *Archimedes' Revenge: The Joys and Perils of Mathematics.* New York: W. W. Norton & Company.

[12] Kahn, D. 1967. *The Codebreakers: The Story of Secret Writing.* New York: MacMillan.

[13] Kaliski, B. S. Jr. 1983. Inventor offers $100,000 to puzzle solver. *The Tech.* September 27: 11.

[14] Kilian, J. 1985. EECS grads decipher puzzle, miss deadline. *The Tech.* April 5: 2.

[15] Larsen, R. J. and M. L. Marx 1981. *An Introduction to Mathematical Statistics and its Applications.* Englewood Cliffs NJ: Prentice-Hall.

[16] 1989. Library of congress. In *The New Encyclopædia Britannia*, vol. 3. Encyclopædia Britannia, Inc. p. 536.

[17] Meyer, C. H. and S. M. Matyas 1982. *Cryptology: A New Dimension in Computer Data Security.* New York: John Wiley.

[18] Moon, D., R. M. Stallman, and D. Weinreb 1984. *Lisp Machine Manual*, 6th edition.

[19] 1985. A puzzling way for 36 to collect $117,000. *The Boston Globe.* April 17.

[20] Rivest, R. L. and A. T. Sherman 1983. Randomized encryption techniques. In *Advances in Cryptology: Proceedings of Crypto 82*, Chaum, Rivest, and Sherman, eds. New York: Plenum Press. pp. 145–163.

[21] Sagan, C. 1977. *The Dragons of Eden: Speculations on the Evolution of Human Intelligence.* Random House.

[22] Sagan, C. 1980. *Cosmos.* Random House.

[23] Shannon, C. E. 1949. Communication theory of secrecy systems. *Bell System Technical Journal.* 28: 656–715.

[24] Shannon, C. E. 1951. Prediction and entropy of printed English. *The Bell System Technical Journal.* January: 50–64.

[25] Sinkov, A. 1966. *Elementary Cryptanalysis: A Mathematical Approach.* Mathematical Association of America.

About the Authors

Robert W. Baldwin is a software architect at Tandem Computers, Inc. in Cupertino, California. He received his Ph.D. in computer science from the Massachusetts Institute of Technology in June 1987, and his S.M. and B.S. in electrical engineering and computer engineering from the Massachusetts Institute of Technology in May 1982. His research interests include information security and cryptography.

Alan T. Sherman is an assistant professor of computer science at the University of Maryland Baltimore County and a member of the University of Maryland Institute for Advanced Computer Studies. He received his Ph.D. in computer science from the Massachusetts Institute of Technology in February 1987, his S.M. in electrical engineering and computer science from the Massachusetts Institute of Technology in June 1981, and his Sc.B. in mathematics, *magna cum laude*, from Brown University in June 1978. His research interests include algorithmics, cryptology, and VLSI layout algorithms.