How We Cracked the Code Book Ciphers

Fredrik Almgren   Gunnar Andersson   Torbjörn Granlund
Lars Ivansson   Staffan Ulfberg

solvers@codebook.org

Created: October 11, 2000
Modified: November 1, 2000
http://answers.codebook.org

1 Introduction

In 1999, Simon Singh published The Code Book. The book invites its readers to take part in a competition known as the Cipher Challenge. This competition has attracted a large and international audience. A mailing list has for instance been set up as a forum for discussions on the competition.

This document gives a description of how we proceeded to solve the problems posed in The Cipher Challenge. The document gives both a popular description and overview information, as well as more in-depth descriptions of the hunts for solutions for the specific stages. We have tried to organize the document in such a way that it can be read by anyone. Some chapters do assume computer or cryptographic knowledge. When this is the case, the technical description of how each stage was solved is usually in a separate section. The reader who so wishes can skip these sections and still be able to understand the rest of the document.

We warn any reader who still wants to complete the challenge that this document is a major spoiler for all stages. So for a reader who does not want to see any spoilers, it is not a good idea to read this report.

1.1 Authors

When the document uses the term "we" it refers to the following group of people who have been working together:

- Fredrik Almgren
- Gunnar Andersson
- Torbjörn Granlund
- Lars Ivansson
- Staffan Ulfberg

When we entered the challenge, Staffan, Lars and Gunnar were all working on their PhD degrees at the Royal Institute of Technology in Stockholm, Sweden. Staffan and Gunnar have now recently completed their degrees, and Lars will defend his thesis in November.

Torbjörn is the owner of SWOX, a company that develops Open Source software solutions, specializing in the GNU compiler tool chain and high-performance mathematical software.

Fredrik works for Across Wireless, a Swedish company working with mobile Internet solutions.

1.2 Acknowledgments

As always when finally completing something on which work has been ongoing for a long time, one realizes that when time comes to thank everyone that has helped out, much has already been forgotten. We do however have a number of people and institutions to which we would like to express our thanks for assisting us with both computing resources and knowledge.

We are deeply grateful to Johan Håstad of the Royal Institute of Technology for a continuous flow of good advice and encouragement. Many were the times when we asked him to explain the inner workings of the Number Field Sieve.

We would also like to thank Samuel Lundqvist and Christopher Hermansson-Muth, who were keeping our hopes up in the chase for the solution to Stage 5, and also took active part in the solution of Stages 3 and 7. We also thank Mikael Goldmann, who always kindly offered his advice.

We also want to express our thanks to CWI (Centrum voor Wiskunde en Informatica) that wrote the code we used for the solution of Stage 10.

We thank the head of the Department of Numerical Analysis and Computer Science at the Royal Institute of Technology, for letting us use a large number of workstations for the sieving step of Stage 10. Our thanks also go to the system administrators; they endured a lot of problems arising from the heavy IO-requirements of our programs.

Compaq provided access to one of their benchmarking computers with four 667 MHz Alphas and 8 GB of primary memory. This was where we made 90% of the computations necessary for solving a huge system of equations in the final stages of the decryption of Stage 10. We believe that they thus can claim to have provided the hardware for a record factorization without super-computer use.

UMS Medicis 658, Palaiseau, France, provided generous access to their Compaq DS20 dual processor 500 MHz Alpha with 4 GB of primary memory where we performed most of the filtering and verified the parallelization of the Gauss elimination.

We would like to thank the National Supercomputer Centre, Linköping, Sweden, for access to computing resources that made it possible for us to perform the first steps of filtering of the relations in Stage 10, the Swedish Institute of Computer Science, for access to their computers for sieving, and the High Performance Computing Center North, Umeå, Sweden, for sieving and initial execution of a square-root finding program for Stage 10.

We would like to thank Across Wireless and the employees there for using our screen saver for cracking Stage 9. A special thanks goes to Per Höckerman, who slept in late one morning to arrive at 07:00, less than an hour after his computer had reported the correct solution to the server.

1.3 The Order of the Solutions

Like many others working on The Cipher Challenge, we did not solve the stages in the order they are presented in The Code Book. Below follows a list of the stages in the order we solved them.

Stage 1	September 23, 1999
Stage 2	September 24, 1999
Stage 4	September 26, 1999
Stage 3	October 4, 1999
Stage 7	October 10, 1999
Stage 6	October 12, 1999
Stage 8	October 13, 1999
Stage 9	November 12, 1999
Stage 5	May 15, 2000
Stage 10	October 5, 2000

Notable is that after solving Stage 9, it took us almost six months before we managed to break the elusive Stage 5. If one considers that we actually had all the information that Stage 5 could require when we had solved Stages 1–4, it actually took us more than seven months to conquer it.

A more detailed description of Stage 5 follows further down, but the lack of success with solving Stage 5 was very annoying for us.

1.4 Entering into the Contest

The Cipher Challenge is a part of The Code Book, but the first copy of the book was bought without the slightest knowledge of the existence of a competition. As it happened, Fredrik was in London on September 11–18, 1999. During a rainy week in London, Fredrik walked along the bookstore windows that fronted the new book by Simon Singh, The Code Book. Having already read Fermat's Last Theorem by the same author, Fredrik decided to buy this new book as well.

Fredrik went back to Sweden and started reading the book. About half-way through the book, Fredrik flipped through the pages of the book and found a section called The Cipher Challenge. The first cipher looked interesting enough, and he solved it in about twenty minutes. The second was saved for later, not because it seemed difficult, but because the solution was not obvious to him using pen and paper.

On Saturday September 25, Fredrik brought the book to the local juggling practice and afterwards showed it to Staffan. Fredrik knew that Staffan was usually willing to accept a challenge. This time was no exception, and Staffan promptly wanted a book of his own and went off to find an open book store.

As soon as Staffan had bought his own copy, a Swedish translation, they set off for Staffan's work to make copies of the ciphers. The intention was to OCR all the stages so that we had them in electronic form.

Fredrik was going to a birthday party the same evening and had to leave Staffan there. As Fredrik was leaving the party after midnight, he called Staffan who announced that he had solved Stage 4.

He had also completed the OCR work and we now had all the stages in electronic format. Later experiences would show that this process had not been error free, as some parts of the ciphertexts later turned out to be incorrect. These errors never threatened the correct decryption, though.

We now felt that we were ready to make a serious attack on the ciphers.

1.5 Forming a Team

Having cracked Stages 1, 2 and 4 and converted the other ciphertexts into computer files, we felt pretty good about the whole thing.

After the weekend, Staffan showed the book and The Cipher Challenge to his colleagues Gunnar and Lars, and they promptly got books of their own and joined the team. This was actually a great success since it resulted in an order to Amazon for a dozen more copies of the book, since everyone in the university department and some juggling friends wanted books of their own.

Now we were a large and undefined group of people that had just become Cipher Challenge Cracker wannabies. We did not consider this a problem, but instead invited anyone showing interest to work with the group. We felt that time would tell who was in the group and who was not. Thus, while other people in the world much later were discussing the possible formation of teams, we were a team from day one.

As time went by and the solution to Stage 9 had been revealed, leaving us with only Stages 5 and 10 to solve, the group was reduced as some of the original members lost interest. At this stage we also started to operate in a more silent way as we realized that we had quite a large task to complete for the two missing stages.

1.6 Early Success

When the initial stages were solved rather quickly and Stages 7, 6 and 8 shortly afterwards were solved during a hectic week, we proceeded to Stage 9. Writing the computer programs for this stage and running them took only about a month. Shortly before we cracked Stage 9, the American Jim Gillogly announced completing Stage 9, so we were not to the first to get this far. Nevertheless, we felt that we were probably among the first ones to solve this stage.

We were now only missing Stages 5 and 10, and started working on them in parallel, hoping to find the solution to Stage 5 some time before we managed to solve Stage 10. At this time, we felt that being the first ones to crack all ten stages was a realistic goal to strive for.

1.7 E-Groups Mailing List and Siteswaps

Through the Usenet news group `sci.crypt`, we became aware that there was a mailing list about the challenge. We quickly joined in order to keep track of the progress of the others, and to take part in the discussions.

In November, Jim Gillogly mailed and said that there was an update on the official website, which announced his success with Stage 9. Fredrik then replied to him and said that we too had solved Stage 9. However, within the group we had agreed on that we would not announce this fact publicly to the mailing list.

Jim Gillogly kept silent about this, and we managed to stay in what Jim later referred to as "stealth mode" throughout the contest. Some people on the list might have called this "lurking", but we prefer Jim's term. We actually had decided not to openly discuss our progress.

The main reason for not disclosing anything about our progress to the list was that we at a certain point had decided that we were in the contest in order to win it. Once we had decided that, we actually felt that it was a better strategy not to advise the world of how far we had proceeded: Knowledge about how many people have cracked a certain stage can give an indication of how hard it is.

The number of subscribers to the mailing list was steadily growing and the discussion was starting to focus around the stage that was really holding us back, Stage 5. A lot of texts were proposed on the mailing list, but at this point we had tried a large number of texts, and it felt as if we had tried almost all texts proposed, and many others.

At this point, Staffan and Fredrik created Stenils20000 and made him send a joke mail to the list suggesting siteswaps as possible filtering for the Stage 5 codetext. Siteswap is a juggling pattern notation and since three of us juggle, several of the numbers brought up associations of juggling to mind. The only one who commented on the mail was in fact Gunnar, the third juggler in the group. The probable conclusion is that our group might have been the only one with active jugglers taking part, somewhat strange since juggling is popular among programmers and the like.

It is interesting that juggling connects closely to mathematics at times, showing connections to graph theory etc. For the reader interested in more about siteswaps, these two URLs can be a good place to start.

```
http://www.solipsys.co.uk/pub/ssintro.htm
http://www.juggling.org/help/siteswap/faq.html
```

When some groups for attacking Stage 10 started to form in the world, Jim was very much the gentleman and offered to forward the programs that the Dutch mathematician Arjen Lenstra had sent. Since it would have been an outright lie to say that we were no longer in the competition, and it would have been contrary to our agreed strategy to say that we did not need these programs, Fredrik asked Jim to forward them.

1.8 Deciphering the uuencoded messages

The encrypted messages for some stages are in binary form and were therefore given as uuencoded data in the book. This did cause us some problems as this file format is meant to be exchanged by computers, not humans.

In fact, Simon Singh wrongly admitted to an error in one of uuencoded texts. In this case, it had been claimed that the second last line in the file should only contain a back-apostrophe, `` ` ``. Two out of three messages have a `` ` ``, but the third one has a blank line as its second last line. This is rather hard to see for the human eye when the text is printed in a book, but a computer would never miss it.

It should be noted that these two characters are equivalent since applying bitwise AND by 63 yields the same results for 32, the index for the space character in ASCII, and 96, the index for the back-apostrophe `` ` ``. Different versions of `uuencode` may produce different encodings, but the decoding is still consistent. Since a line always starts off with its length count and length zero means that the file ends, both of these mean end-of-file.

2 Descriptions of the Work for each Stage

Here follows a description of how we worked with each individual stage in the competition, the hurdles we faced and the small pieces of success that we achieved.

2.1 Stage 1

Solution found on September 23, 1999.

2.1.1 Initial Knowledge

The type of cipher was clearly stated as a substitution cipher in the book, something which made the deciphering rather straight-forward.

2.1.2 Work Description

This was really just a bedtime pen and paper problem that took about twenty minutes to solve.

We were at first a bit surprised that the cipher was so different in the Swedish version of The Code Book. Then we realized that Stage 1 was in the same language as the rest of the book. This meant that the plaintext was in Swedish in the Swedish version. It was later confirmed on the mailing lists that in most editions of the book, the first stage had been translated to the language in which the book what printed in

However, it turned out to be the same plaintext as the English one but in Swedish translation. This actually was a little bit intriguing since it clearly meant that someone else than the author had knowledge of the ciphers. Would this be true for other stages as well?

2.1.3 How we Found the Solution

The solution was found just shortly before going to sleep using the pillow instead of a table. The plaintext is recognized as the book of Daniel from the Bible, and the codeword for this stage is **OTHELLO**.

2.1.4 Description of Solution Method

This was simply solved with pen and paper.

Even if frequency analysis is honored as the ultimate way of solving this kind of cipher, it was not needed for solving this one.

Out of naivety maybe, it was assumed that the plaintext language was English. Since the word separations were present in the cryptotext, some initial guesses of possible words was all that was needed.

2.2 Stage 2

Solution found on September 24, 1999.

The message is clearly labeled as being a Caesar type encryption. It is therefore the easiest of all stages. As there are only 26 possible shifts to try, the solution can easily be found by hand by trying them all. It can also be done with a computer, for instance using the following Unix shell script:

```
multivac> l=''; while [ ${#l} -lt 26 ];
          do tr $l\A-Z A-ZA-Z < stage2.txt; l=.$l; done | less
```

Going over the 26 candidate plaintexts, it is easy to find the correct plaintext:

```
Faber est suae quisque fortunae.  - Appius Claudius Caecus
Dictum arcanum est neutron.
```

The codeword for this stage is thus **NEUTRON**.

2.3 Stage 3

Solution found on October 4, 1999.

2.3.1 Initial Knowledge

This message was labeled as being homophonic. Our interpretation of this was that the same letter could be encoded as several different letters, or maybe even as several different pairs of letters. Therefore we expected the frequencies of the different characters to be almost equal.

2.3.2 Work Description

Even though the message was homophonic, we did of course use frequency analysis to start with. As we had suspected, this did not give us any good clues. It however became clear that the frequencies in the message exhibited large variations.

The presence of the character * in the message made us feel that there was some trick involving that character. There is also an abundance of the letter X. We quickly developed the theory that each digram in the message coded a plaintext character. However as this gave us nothing, we quickly expanded the possible complexity of the coding scheme.

We now worked a while on the assumption that some cryptotext characters encoded only one plaintext character but that some characters in the cryptotext where magic and always started a digram.

Working along this line of reasoning, Staffan wrote yet another interactive replacement program. This supported interactive replacement of either single characters or digrams into the text. It also showed the frequencies for the most common characters and digraphs as one worked through the message.

Using this program, we spent a number of hours trying to get the message to yield, all to no avail. When explaining the way of reasoning to someone else, he immediately replied that it would be better to assume that the encryption was a little bit simpler than we thought. We were assuming the encryption to be almost diabolical as we invented one possibility after another.

The opening to the whole decryption came when Staffan and Fredrik looked at the message and realized that the distribution of some characters in the messages was very interesting. It turned out that the cryptotext characters M, G, C, Z, N and B appear in a very interesting pattern in the message: Each one of them is only present on a few lines and is very frequent in these lines. This made us assume that they all encoded the same plaintext character.

2.3.3 How we Found the Solution

Using the above knowledge, we went back to the program. We first assumed that the cryptotext character X coded a space and thus discarded these. We then replaced all the characters described above with one character.

It turned out that that character had relative frequency of about 10%. The second most frequent character amounted to about 9%.

Knowing this, we had an internal battle if we should first try Spanish or Italian as plaintext language. Eventually, we decided to give Italian a thorough attempt. The problem was that we knew no Italian. However, finding some samples of Italian on the WWW was no problem so we set out to solve it.

Using information about frequencies and common short words in Italian we used the interactive program to replace characters. After a while, we started feeling that we were actually seeing a real text form before our eyes.

We did not understand the text but using only the first two words as search string in AltaVista quite easily yields a hit for Dante Alighieri: The first five words produces a hit for Canto XXVI de L'Inferno. The codeword for this stage was placed after this text; it is **EQUATOR**.

2.3.4 Description of Solution Method

The interactive program might have made it too easy for us to follow the wrong track for a while when we were working on the digram-theory. However, once we were on the right track, it made it really simple to try different attempts at character substitutions.

We also learned the important lesson that it is smarter to try the simpler solutions before assuming that something is really difficult.

Knowing this, we also realized that the fastest way into Stage 3 is probably to attack it with pen and paper when doing the first frequency analysis. Since we did all that with programs, we missed the suspicious characters for a long time. If that had been done manually, one would quickly have seen the interesting patterns formed by these characters.

Once again the KISS theory had been proven...

2.4 Stage 4

Solution found on September 26, 1999.

2.4.1 Initial Knowledge

It was stated already in the formulation of Stage 4 that the this was a Vigenère cipher.

2.4.2 Work Description

It was very straightforward to solve this stage. Gunnar, Lars and Staffan had taken a course in cryptography some years ago, in which cracking a Vigenère cipher was one of the homework assignments. Programs for solving this kind of cipher were therefore present and could be used immediately. This stage was thus solved in a few minutes.

2.4.3 How we Found the Solution

Below follows the output from the program `vigenere`.

```
monty>./vigenere freqs.txt stage4.txt

Keyword length  1: 0.0437
Keyword length  2: 0.0444
Keyword length  3: 0.0432
```

```
Keyword length  4: 0.0442
Keyword length  5: 0.0786
Keyword length  6: 0.0438
Keyword length  7: 0.0440
Keyword length  8: 0.0430
Keyword length  9: 0.0412
Keyword length 10: 0.0788

Give keyword length (optimal m=10): 5

 1)AKCJI: KGMNWFLHGMJKSEMKWJDWKZGEEWKVWIMAHSYWHJWF
 2)BLDKJ: JFLMVEKGFLIJRDLJVICVJYFDDVJUVHLZGRXVGIVE
 3)CMELK: IEKLUDJFEKHIQCKIUHBUIXECCUITUGKYFQWUFHUD
 4)DNFML: HDJKTCIEDJGHPBJHTGATHWDBBTHSTFJXEPVTEGTC
 5)EOGNM: GCIJSBHDCIFGOAIGSFZSGVCAASGRSEIWDOUSDFSB
 6)FPHON: FBHIRAGCBHEFNZHFREYRFUBZZRFQRDHVCNTRCERA
 7)GQIPO: EAGHQZFBAGDEMYGEQDXQETAYYQEPQCGUBMSQBDQZ
 8)HRJQP: DZFGPYEAZFCDLXFDPCWPDSZXXPDOPBFTALRPACPY
 9)ISKRQ: CYEFOXDZYEBCKWECOBVOCRYWWOCNOAESZKQOZBOX
10)JTLSR: BXDENWCYXDABJVDBNAUNBQXVVNBMNZDRYJPNYANW
11)KUMTS: AWCDMVBXWCZAIUCAMZTMAPWUUMALMYCQXIOMXZMV
12)LVNUT: ZVBCLUAWVBYZHTBZLYSLZOVTTLZKLXBPWHNLWYLU
13)MWOVU: YUABKTZVUAXYGSAYKXRKYNUSSKYJKWAOVGMKVXKT
14)NXPWV: XTZAJSYUTZWXFRZXJWQJXMTRRJXIJVZNUFLJUWJS
15)OYQXW: WSYZIRXTSYVWEQYWIVPIWLSQQIWHIUYMTEKITVIR
16)PZRYX: VRXYHQWSRXUVDPXVHUOHVKRPPHVGHTXLSDJHSUHQ
17)QASZY: UQWXGPVRQWTUCOWUGTNGUJQOOGUFGSWKRCIGRTGP
18)RBTAZ: TPVWFOUQPVSTBNVTFSMFTIPNNFTEFRVJQBHFQSFO
19)SCUBA: SOUVENTPOURSAMUSERLESHOMMESDEQUIPAGEPREN
20)TDVCB: RNTUDMSONTQRZLTRDQKDRGNLLDRCDPTHOZFDOQDM
21)UEWDC: QMSTCLRNMSPQYKSQCPJCQFMKKCQBCOSGNYECNPCL
22)VFXED: PLRSBKQMLROPXJRPBOIBPELJJBPABNRFMXDBMOBK
23)WGYFE: OKQRAJPLKQNOWIQOANHAODKIIAOZAMQELWCALNAJ
24)XHZGF: NJPQZIOKJPMNVHPNZMGZNCJHHZNYZLPDKVBZKMZI
25)YIAHG: MIOPYHNJIOLMUGOMYLFYMBIGGYMXYKOCJUAYJLYH
26)ZJBIH: LHNOXGMIHNKLTFNLXKEXLAHFFXLWXJNBITZXIKXG

Give selected keyword index: 19

SOUVENTPOURSAMUSERLESHOMMESDEQUIPAGEPRENNENTDESALB
ATROSVASTESOISEAUXDESMERSQUISUIVENTINDOLENTSCOMPAG
NONSDEVOYAGELENAVIREGLISSANTSURLESGOUFFRESAMERSAPE
INELESONTILSDEPOSESSURLESPLANCHESQUECESROISDELAZUR
MALADROITSETHONTEUXLAISSENTPITEUSEMENTLEURSGRANDES
AILESBLANCHESCOMMEDESAVIRONSTRAINERACOTEDEUXCEVOYA
GEURAILECOMMEILESTGAUCHEETVEULELUINAGUERESIBEAUQUI
LESTCOMIQUEETLAIDLUNAGACESONBECAVECUNBRULEGUEULELA
UTREMIMEENBOITANTLINFIRMEQUIVOLAITLEPOETEESTSEMBLA
BLEAUPRINCEDESNUEESQUIHANTELATEMPETEETSERITDELARCH
ERBAUDELAIREEXILESURLESOLAUMILIEUDESHUEESLEMOTPOUR
```

```
ETAGEQUATREESTTRAJANSESAILESDEGEANTLEMPECHENTDEMAR
CHER
```

The program is interactive. It first evaluates different keyword lengths. For each length it gives a score whose meaning is described below. The higher the score, the more likely it is that the keyword length is the true keyword length. As we can see, length five is the first candidate with a significantly higher value, so that length is selected.

The program then computes the most probable distances between the letters in the keyword and displays the 26 possible keywords with this letter distances. 25 of the candidates are nonsense, but number 19 both gives a meaningful keyword and an easily recognizable plaintext in French. The plaintext is taken from the poem "L'Albatros" by Charles Baudelaire, and as we can see the codeword for this stage is **TRAJAN**.

2.4.4 Description of Solution Method

There are two major obstacles when deciphering a Vigenère cipher. The first is to find the length of the keyword superposed on the plaintext, and the second is to find the shift between adjacent letters in the keyword.

If the keyword length is five, every fifth letter in the ciphertext have been shifted the same distance. This means that for the offsets 0, 1, 2, 3, 4, the text obtained by reading every fifth word will be an ordinary Caesar shift of the plaintext. Given a string $\mathbf{x}$ of n characters, the index of coincidence of $\mathbf{x}$, denoted $I_c(\mathbf{x})$ is defined as the probability that two random elements of $\mathbf{x}$ are identical. If the frequencies of the 26 letters in $\mathbf{x}$ are denoted $f_0, \ldots, f_{25}$, this probability can be written

$$I_c(\mathbf{x}) = \frac{\sum_{i=0}^{25} f_i(f_i - 1)}{n(n-1)}. \tag{1}$$

For a completely random string this value will be approximately $1/26 \approx 0.038$. However, if we let p_i be the probability that a random letter in English text is i, then for an English text $\mathbf{x}$,

$$I_c(\mathbf{x}) \approx \sum_{i=0}^{25} p_i^2 \approx 0.065. \tag{2}$$

If the different letters in a text are shifted with different shifts this text will look like random text. However, if they are shifted with the same shift the index of coincidence should be that of the plaintext. If we compute the mean of the index of coincidence for the text obtained by reading every fifth letter for the five offsets we thus expect that the index of coincidence will be closer to 0.065 than to 0.038 if five is the correct keyword length. This is also what we observe above.

The next problem is to determine the difference in shifts between the adjacent letters in the text. This is done in a similar way. Let **x** and **y** be two strings with n_x and n_y letters, respectively. The mutual index of coincidence of **x** and **y**, denoted $MI_c(\mathbf{x}, \mathbf{y})$, is defined as the probability that a random element of **x** is identical to a random element of **y**. If $f_0^x, \ldots, f_{25}^x$ and $f_0^y, \ldots, f_{25}^y$ is the frequencies of the 26 letters in **x** and **y**, respectively, this probability can be written

$$MI_c(\mathbf{x}, \mathbf{y}) = \frac{\sum_{i=0}^{25} f_i^x \cdot f_i^y}{n_x \cdot n_y}. \tag{3}$$

Assume once more that the keyword length is five. Let $\mathbf{y}_1, \ldots, \mathbf{y}_5$ be the five substrings obtained by reading every fifth letter for the five different offsets. The substrings $\mathbf{y}_i$ are all obtained by a shift encryption of the plaintext. Assume that $K = (k_1, \ldots, k_5)$ is the keyword. It is quite easy to see that in this case

$$MI_c(\mathbf{y}_i, \mathbf{y}_j) \approx \sum_{h=0}^{25} p_{h-k_i}, p_{h-k_j} = \sum_{h=0}^{25} p_h, p_{h+k_i-k_j} \tag{4}$$

where the subscripts are reduced modulo 26. If we test all possible relative shifts of two strings of English text we will see that when the relative shift is 0, the mutual coincidence will be approximately 0.065; and otherwise it lies between 0.030 and 0.045. This means that if we test all possible values for $k_i - k_j$ for the two substrings $\mathbf{y}_i$ and $\mathbf{y}_j$, the mutual coincidence should be close to 0.065 for the correct relative shift and between 0.030 and 0.045 in the other cases. This will give us an linear system of equations for the five shifts $k_1, \ldots, k_5$ which can be solved and which will yield the relative shifts for the five substrings.

Now we only have 26 possible keywords and plaintexts, and a visual inspection easily reveals the solution.

2.5 Stage 5

Solution found on May 15, 2000.

2.5.1 Initial Knowledge

This stage was haunting us for more than six months. From the very beginning of our efforts we were convinced that this cipher must be some kind of book cipher. As the book cipher was described in The Code Book, this should imply that the numbers given are indices to words in a key text, and the initial letters in these words should give the plaintext. Other versions were proposed as well, e.g. counting backwards, counting every kth word, etc. As the other stages were solved we obtained more clues. The language of the plaintext had a tendency to be correlated with the origin of the cipher. Since book ciphers were exemplified with the Beale ciphers, this would imply that English should be the language of the plaintext.

2.5.2 Work Description

This stage made us more and more desperate. Like most people, our initial essays were made with paper and pencil. Naturally, the Declaration of Independence attracted a lot of attention as well as other similar documents. We found a veritable gold mine filled with this kind of texts at the homepage of the Avalon Project. That The Code Book itself would contain the keytext was proposed early on. The appendices of the book therefore attracted much interest. The presence of "A void" seemed a little out of scope, but also the other appendices were examined carefully.

When all this failed, other ideas emerged. The numbers were between 3 and 215, which made us interested in the crossword shown in The Code Book. The crossword has $15 \times 15 = 225$ squares; surely this cannot be a coincidence! However, no matter how we counted, the 4 times repeated index 109 always became a filled square, and no message emerged.

At this point we decided to stop being intelligent, and start using brute force. Several programs were written. A first version of our program tested all possible offsets in a text. For each offset the program extracted the first, last, or kth (for all possible choices of k) letter in the magic words. We also built a similar program which instead of counting words counted letters (including or excluding whitespace and separators). The possible plaintexts were then written on a file which was scanned with the Unix command `grep` for probable keywords like "stage" and "word". What we learned from this was that given enough amount of keytext, seemingly improbable amounts of candidate plaintexts may emerge. Some of the messages we found scanning the King James version of the Bible would have made Michael Drosnin jump with joy.

Our next step was to make a fully automatic version of the program. To obtain a fully automatic program, we created an evaluation function that evaluated each potential plaintext. The evaluation function uses a large English dictionary. Candidate plaintexts with high enough evaluations were output by the program for visual inspection.

The two programs (the word counter and the letter counter) were then run on various texts. The Project Gutenberg provided us with several hundred MB of text. A small subset of the texts that were tested is:

The Bible
: The book of Daniel was the plaintext in Stage 1.

The Gaelic wars
: Caesar had written the plaintext of Stage 2.

La divina commedia
: The plaintext in Stage 3. Several languages were tested.

Baudelaire
: All we could find electronically was tested. This was after all the author of the plaintext in Stage 4.

Robinson Crusoe
: Written by Daniel Defoe, and the plaintext in Stage 1 ends "now let Daniel be called, and he will show the interpretation."

The adventures of Sherlock Holmes
: Mentioned in The Code Book.

Journey to the center of the earth
: By Jules Verne. Also mentioned in The Code Book. Here we actually found an interesting crib for an early offset.

Treasure island
: A 19th century book about hidden treasures, clearly an eminent choice for the key text.

The Rosetta Stone
: The number of verses is 54, i.e., the same as the number of indices in Stage 5. Surely this cannot be a coincidence!

Not only electronic texts were examined. One group member hunted the Stockholm libraries for the French version of "A void". We also had advanced plans for scanning The Code Book, and "Fermat's last theorem", also by Simon Singh. Singh's PhD thesis was also searched for. After all, the two first solvers that we heard of had connections to Cambridge and might therefore have had access to this thesis.

Alternative solution methods were also discussed. We actually have to admit that during a beer drinking session, we came up with a scam that could be used to get the solution. The CipherChallenge email list was now buzzing with people announcing their success with Stage 5. What if we said that we had solved Stage 5, and volunteered to confirm the solutions of others... Luckily, we never needed to resort to getting the scam, since Lars announced the successful solution some days later.

2.5.3 How we Found the Solution

During the Spring the discussions concerning Stage 5 became more and more vivid on the Cipher Challenge mailing list. Simon Singh himself gave some hints about the nature of the cipher. From Singh's updates a lot of people tried to extract clues which were not there. People having solved the stage encouraged the ignorants to forget all hints and start again with an unprejudiced mind. A hint that was given was to consider who the constructor of the cipher was.

Early on we had thought of the book "Fermat's last theorem" by Simon Singh as the potential keytext. We had also thought about texts by Pierre de Fermat. During a game of the board game Othello between Lars and Gunnar, the famous marginal note by Fermat was suggested as a potential keytext. The objection was raised that this text surely must be too short, but to settle the question we searched for the text on the Internet. It turned out that the text was too short for the word counting approach, but had the right length for letter counting.

After having tried an English and a French version, we stumbled over the text in Latin which turned out to be the original language. This text is as follows.

> Cubem autem in duos cubos, aut quadratoquodratum in duos quadratoquadratos, et generaliter nullam in infinitum ultra quadratum potestatem in duos eiusdem nominis fas est dividere. Cuius rei demonstrationem mirabilem sane detexi hanc marginis exiguitas non careret.

Feeding this text into the letter counting program `cwizard` gave the following result.

```
sandman> cwizard < Fermat.latin.txt > Fermat.latin.out
Building trie...
Done

225: PLAIFAIRCIPHERESELPROXIMONIVELLAPALABRASECRETAESILLIAD
P(0)LAI(3)FAIR(4)CIPHER(6)ESEL(0)PROXIMO(7)NI(0)VELL(4)APA(3)
LABRA(5)SECRETA(7)E(0)SILL(4)IAD(0)
```

We observe that we had a bit of luck here. Even though the plaintext was in Spanish there were enough English words in the text for our program to classify it as a candidate text. If we format the text properly we get

> Playfair cipher es el proximo nivel. La palabra secreta es Illiad.

The keyword for this stage is thus **ILLIAD**.

2.5.4 Description of Solution Method

Here we will give a more detailed description of the program `cwizard` used to solve the book cipher. For a given text all possible offsets are tested, and the characters are counted both from the beginning and from the end of the text. The program considers four ways to count letters in a text.

1. Count all characters as letters.
2. Count separators but not whitespace as letters.
3. Count whitespace but not separators as letters.
4. Count neither whitespace nor separators as letters.

For each text all four ways to count are tested.

Given a hypothetical plaintext we use an automatic sanity check, using an English dictionary. The dictionary is stored in a trie. A trie is a tree-like data structure where each node represents a letter. Any path from the root down the tree thus makes a potential word. If the word obtained by following a path from the root to a node v is in the dictionary, we mark the node v as a word-ending node. With this data structure it is possible to see if a word is present in the dictionary in time linear in the length of the word.

The potential plaintext is evaluated using dynamic programming. A word w is given the score $|w|^2$, if the length of the word $|w| > 2$ and if w is present in the dictionary. Otherwise the score of the word is 0. Our aim is to partition the plaintext into disjoint words so that the sum of the score for each word is maximized. This is easy to do with dynamic programming. Let $S(i)$ be the maximum score for the first i letters in the plaintext, and let $s(i, j)$ be the score for a word beginning at index i and ending at index j in the plaintext. Then

$$S(i) = \max_{1 \leq k \leq i}\{S(k) + s(k, i)\}$$

This function can be computed efficiently in a bottom-up fashion, and the optimal score for the plaintext is given by $S(n)$ where n is the length of the plaintext. The plaintext was finally output if the score $S(n)$ was at least $3.2n$.

The reason for using a quadratic scoring function was that we wanted to reward longer words more than shorter words. Short real words tend to be present, en masse, in nonsense text as well. With a quadratic function and the selected threshold the number of false hits was reasonably small. The hope was that the correct text be detected with this methodology. This was also eventually confirmed.

2.6 Stage 6

Solution found on October 12, 1999.

2.6.1 Initial Knowledge

When we solved this stage we had not yet solved Stage 5, and we did thus not know what type of cipher we had to deal with. However, several properties of the ciphertext suggested that this cipher was a Playfair cipher. For instance,

- The letter J was missing from the ciphertext.
- The frequency distribution of single letters was rather smooth, while the bigram frequency distribution was skewed.
- The chronological order in which the ciphers tend to appear made a Playfair cipher seem probable.

The Playfair cipher seemed connected with Great Britain so our approach was that the plaintext should be written in English.

2.6.2 Work Description

Initially we tried to solve this stage by hand. We made bigram statistics and tried to solve the cipher by considering it as a substitution cipher with $\binom{25}{2} = 300$ letters. Some observations could also be made that should facilitate the substitutions, e.g. that if XY is mapped to AB then YX is probably mapped to BA, etc. Our English skills were however not sufficient for breaking the cipher in this way. We also made some attempts at building the key square in an intelligent fashion. We tried to fill the top of the square with English text and then pad with the remaining letters in order. This was however also made without success. We thus decided to once again resort to brute force.

The idea we used was to make a heuristic search for the true key square. The program we built first evaluated all possible choices of the first six letters in the key square. Then it added one letter at a time and saved the partial keys that got the highest scores to the next round. We also built in the possibility to fix an initial portion of the key square and then proceed as above with the rest of the square. The crucial part of this approach is naturally to construct a good evaluation function.

Our evaluation function was based on bigram statistics. Using a massive amount of English text in which all in-frame double repeats such as AA were substituted with AXA, the frequencies of the different bigrams were counted and the log-probabilities (i.e., the natural logarithms of the probabilities of the different bigrams) were estimated. For a given partial square we first extracted all the plaintext bigrams for which both the plaintext bigram and the corresponding ciphertext bigram were found in the part of the key that was set. The mean of the log-probabilities for all of the pairs with log-probability above -9 were then computed for normalization purposes. Using the partial key, parts of the ciphertext were deciphered to plaintext. The sum of the log-probabilities of all the bigrams in the plaintext fragments were computed, and the score of the key was set to this value subtracted by the computed mean times the number of bigrams found in the plaintext fragments.

2.6.3 How we Found the Solution

In a large run, we assumed that the first letter of the key was E, an assumption that always can be made without loss of generality. We then tried all possible ways to choose the second letter. Using all these possible bigrams as the preset part of the key we used the program described above. When looking at the best candidate keys for the 25 different trials we found some that looked promising. When deciphering the ciphertext with one of those keys we found the fragments SAYFAIRCIRNER and YOUWILLSEXETHATITI in the plaintext. We had thus found a key square which was partially but not entirely correct. After having played around with the bigrams a little bit, we decided that the letters NKEVI should be one row of the key square. We thus fixed these five letters as the five first of the key and ran our program once more. The best key square found was

```
NKEVI
RHCTO
GBALU
PDMFS
ZQWXY
```

which produced the plaintext:

```
THISSTAGEISNUMBERSIXANDITISAPLAYFAIRCIPHERSTOPTHEFOLLOW
INGSTAGEISNUMBERSEVENWITHINTHECONTEXTOFTHISCOMPETITIONA
NDITISENCRYPTEDACXCORDINGTOANADFGVXTYPECIPHERSTOPWHENYO
UATTEMPTXTOCRYPTANALYSESTAGESEVENYOUWILLSEXETHATITISMOR
ECOMPLICATEDTHANASTRAIGHTFORWARDADFGVXCIPHERSTOPUNFORTU
NATELYICANXNOTGIVEYOUANYMORECLUESSTOPNEWPARAGRAPHINTERM
SOFTHISXSTAGETHESHIBBOLETHTHATYOUSHOULDTAKENOTEOFISMOLY
BDENUMNEWPARAGRAPHYOUWILXLPROBABLYHAVENOTICEDBYNOWTHATE
ACHSTAGEISINADIFFERENTCIPHERANDISUSUALLYINANAPPROPRIATE
LANGUAGESTOPSOTHEVIGENERESTAGEWASINFRENCHANDTHISPLAYFAI
RSTAGEISINENGLISHSTOPIFTHEPATXTERNPERSISTSTHENTHENEXTST
AGEWILXLBEINGERMANSTOPBUTDOESTHEPATTERNPERSISTQUESTIONM
ARKQ
```

If we format the text and replace the formatting words present in the plaintext we get the following message.

> This stage is number six and it is a Playfair cipher. The following stage is number seven within the context of this competition and it is encrypted according to an ADFGVX type cipher. When you attempt to cryptanalyse stage seven you will see that it is more complicated than a straightforward ADFGVX cipher. Unfortunately I cannot give you any more clues.
>
> In terms of this stage the shibboleth that you should take note of is molybdenum.
>
> You will probably have noticed by now that each stage is in a different cipher and is usually in an appropriate language. So the Vigenere stage was in French and this Playfair stage is in English. If the pattern persists then the next stage will be in German. But does the pattern persist?

The codeword for this stage is thus **MOLYBDENUM**. We also realize that the key was not created at random. If rotate the columns of the key one step to the left we obtain

```
KEVIN
HCTOR
BALUG
DMFSP
QWXYZ
```

We guessed that the first two rows emerge from "Kevin Hector" and searched for that name to find that it was the name of a front football player in Derby FC during the 1970s. We could not make out the origin of the rest of the key, however. Simon later hinted that the name of the coach of Derby at that time, "Brian Clough," was used for the third row. The fourth row is still a mystery.

2.6.4 Description of Solution Method

For some reason the program used for this stage was called `heavy`. The program builds the key square in a stepwise fashion. The user has the option to specify the initial part of the key. For instance, if we know that the key starts with the letters APE we can tell the program to only try key squares beginning with the three letters APE. Given these preset letters of the key the program tries all possible combinations of the next six letters in the key. The most likely candidates of the possible keys are then stored in a heap of specified size. For each of the keys in the heap all possible choices of the next letter in the key is evaluated, and the keys with the highest score are saved in another heap.

The crucial part of the algorithm is thus to find a good evaluation function. To evaluate a partial key, `heavy` uses a file containing the log-probability of all bigrams occurring in English text, where in-frame repeats AA have been substituted with AXA. Assume that the following key square should be evaluated.

```
THECO
DBK##
#####
#####
#####
```

To eliminate the bias due to the frequencies of the key letters in English text we first compute the mean m of the log-probabilities of the bigrams that may be constructed with the letters in the partial key. For a bigram to be included we required that both the plaintext bigram and the corresponding ciphertext bigram should be a subset of the letters in the partial key. In the example above this means that the bigram DE is included but not OK. Finally, to reduce noise, we only included bigrams with log-probability higher than -9 when computing the mean.

Using the partial key the program then deciphers as much as possible of the ciphertext. The longer the partial key, the larger part of the ciphertext will be deciphered. For all bigrams we then sum the log-probabilities of all bigrams occurring in the text. If we let s be this sum and n the number of bigrams that were the detected, we finally set the score of the partial key to $s - nm$.

2.7 Stage 7

Solution found on October 10, 1999.

2.7.1 Initial Knowledge

We attacked Stage 7 without having cracked Stage 6, so we did not know for certain what type of cipher it was. However, a clone of ADFGVX seemed the most likely guess and the length of the cipher text gave a nice indication to support this guess. The text consisted of 7048 characters, and $7048 = 8 \cdot 881$. Since 881 is prime, we guessed that the code matrix was eight columns wide.

It was pointed out later that this was not necessarily a correct assumption to make since the initial columns could well be of different lengths. However, even after having solved this stage, we felt that that would probably have been to expect the cipher to be more complicated than it was, an assumption which turned out to be true. We had already made the mistake of overestimating the difficulties when trying to solve Stage 3. So even if we had received this warning beforehand, we would have started out with the simpler version.

2.7.2 Work Description

Staffan rather quickly managed to pair the columns two and two to form a new code matrix with only four columns. We now knew these actually represented letters in a 64 character alphabet. The problem was now reduced to finding out which of the possible 24 permutations that was the correct one.

Staffan wrote an interactive program for simultaneously testing replacements in all the 24 possible solutions, and the problem was reduced to a substitution cipher. The problem was to know which one of the ciphertexts to focus on. We hoped that as soon as we got a few characters in place, it would become apparent which one was the correct text. However, the problem did not yield as easily to that approach as we had first thought.

After thinking a couple of days, we focused on the fact that the first four characters would always be the same, and only the order of them would vary. By using the classic technique of frequency analysis, we observed the first four characters' relative frequency.

With this data, we looked at frequencies for different languages and somehow guessed that German seemed to be the only language that could probably satisfy these frequencies and their connection. This also fit quite well with the fact that the ADFGVX cipher was used by Germany during the first world war. Once that guess was made, it was rather easy to deduce which of the 24 alternative crypto-texts was the correct one.

From that on, the rest of the solution was just a simple matter of replacing characters interactively and the whole text was open in a couple of minutes.

In this process, we realized that we recognized the plaintext and pulled out a copy of Fermat's Last Theorem to fill in the blanks. Close to the end of text we found the keyword, **EDISON**, followed by a very interesting paragraph:

```
Der naechste Teil enthaelt eine von einer Enigmamaschine
verschluesselte Mitteilung. Die Abbildung zeigt die Verbindungen
innerhalb jeder Walze und der Umkehrwalze der Maschine. Es ist
dennoch zu beachten dass das Walzenlage falsch wiedergegeben ist.
Eigentlich sollen Walze 1 und Walze 2 gegeneinander ausgetauscht
werden. Die urspruengliche Walzestellung und Ringstellung und
Steckerverbindungenstellung sind unbekannt.
```

Thus spurred by the success in solving the seventh stage, we also jumped to re-running Stage 8, which was attacked in parallel with this stage, armed with the new information that was revealed to us in the plaintext of Stage 7. We started a full run of analysis for Stage 8 with the new information, hoping to crack that one too.

However, Stage 8 still did not yield. At this point, Stages 6–8 had been worked on in parallel, and we hoped that the success with this stage would soon be followed by Stages 6 and 8 yielding. This was also what happened.

2.7.3 Description of Solution Method

The method we used is based very much on the way a legitimate receiver of the message who knows the correct key would decipher it. We assumed the cipher key codeword length (which is the same as the width of the code matrix) to be 8. Decipherment consists of the following steps.

1. Write down the cipher text in an 8×881 matrix.
2. Permute the columns of the matrix according to the cipher key codeword.
3. Group consecutive columns two by two.
4. Apply the simple substitution that is part of the cipher key to the resulting bigrams to obtain the plaintext.

The table for the simple substitution consists of 36 entries for a normal ADFGVX cipher, and of course this table has 64 entries for a CEMOPRTU cipher. We initially guessed that the characters used in the message were 52 lower and upper case letters, 10 digits, space, and full stop. This guess later turned out correct.

Since simple substitution ciphers are generally easy to break, the permutation of columns in the matrix is really the only thing we had to worry about.

After a plain text has been encrypted by a simple substitution, the distribution of characters is still very uneven. Therefore, we knew that some permutations (including the correct one!) of the columns would result in a distinctly uneven distribution of bigrams. This had to hold for all 4 pairs of columns individually, and also, the distribution of bigrams had to be approximately the same for each column pair.

We wrote a program that graphically showed the 28 bigram distributions resulting from all pairs of columns. It was an easy and fast matter to spot the correct pairing of columns from this output.

Notice that if the initial guess for the code matrix width had been wrong, the results at this stage would not have been any good. It could have been, however, that the code matrix width was 2 or 4, but a re-run of the program with this assumption excluded this possibility.

We could now rewrite the matrix as a four-column matrix of bigrams, and we actually substituted the bigrams by the assumed input alphabet at this stage. Since one of the 24 possible permutations of these columns had to be a simple substitution of the plaintext, we decided that the best course of action was to examine each one of the 24 possibilities individually.

2.8 Stage 8

Solution found on October 13, 1999.

2.8.1 Initial Knowledge

Finally a stage where it is obvious what crypto was used to generate the ciphertext. In principle, we could have attacked this stage immediately, but being placed as the 8th problem we thought that it would be too difficult. After some skirmishes with Stage 6 without any success, some team members turned their attention to this problem, and it was pretty much solved in parallel with Stages 6 and 7.

So what do we know, and what information do we need? It is clear from the illustration in Stage 8 that the ciphertext was encrypted with a three-wheel Enigma for which the contents of the three wheels as well as the reflector are known. This means that there are four classes of unknown parameters:

The direction the wheels rotate
: It is not clear from the figure in what direction the wheels rotate.

The initial wheel positions
: For each of the three wheels, there are 26 initial rotations possible.

The ring settings
: The first two wheels contain two rings. Remember that the wheels are rotated after each encrypted character in an odometer-like fashion: The first wheel is always rotated, the second wheel every 26 rotations of the first wheel, and the third wheel every 26 rotations of the second wheel. The rings determine the point of the first rotation of a wheel: If the ring on the first wheel is reached after 10 rotations, then the second wheel is rotated after 10, 36, 62, … rotations. For the first two wheels, there are

26 possible ring settings, but we can actually disregard some placements on the second wheel as the ciphertext is shorter than 676 characters long.

The plugboard connections
The plugboard contains a set of pairs of letters which are exchanged for each other: If E and Z are connected, then every in- or out-going signal which is an E is mapped to a Z, and vice versa. According to Singh, there were at most six pairs of exchanged letters.

This is what we deduced from the problem formulation. In fact, we also introduced another degree of freedom as one team member interpreted the wheel transpositions differently. This later doubled the CPU time needed to find the solution.

2.8.2 Work Description

Clearly the number of possible configurations to try is huge. The number of degrees of freedom is $2 \times 26^5 = 2 \cdot 23762752$ for the rotation direction, wheels and rings, and a whopping $\binom{26}{2} \cdot \binom{24}{2} \cdot \binom{22}{2} \cdot \binom{20}{2} \cdot \binom{18}{2} \cdot \binom{16}{2} = 72282089880000$ for a plugboard with six pairs of exchanges.

After inspecting the construction of the Enigma, the weakest component is clearly the plugboard. Only 12 letters are modified by the plugboard, so when encrypting a random character there is a fair chance, $(14/26)^2 \approx 29\%$, that it is not modified by any of its two passages through the plugboard. This is a glaring weakness, and a key feature of our solution is that we primarily search among the 23762752 wheel and ring settings.

So the main idea is to try all the possible wheel and ring settings, and to try to locate the best one. How should this be done, when we do not even know what the plugboard contains? The idea is as follows. For all the 23762752 possible wheel and ring settings, the following is repeated: Suppose the message has the following boring form: `EEEEEE...` Each E will first pass through the plugboard, and it is there mapped onto one of 26 possible characters. If we knew what character it is mapped on, we can now trace how the E is modified by the plugboard on the way in, then by the wheels, the reflector and the wheels again. Only the final pass through plugboard on the way out is unknown. Now that we have also guessed what E is mapped on, there are $26 \cdot 23762752$ possible strings that can be produced from the supposed plaintext `EEEEEE...` with the given wheels. A key idea is now that E is one of the most common letters in many languages: English, German, French, Swedish etc. It is therefore reasonable to assume that more than 1/26 of the letters in the actual plaintext are Es, so for each fixed guess we determine the plugboard settings which makes the text we get when we encrypt the string `EEEEEE...` coincide as well as possible with the actual ciphertext. This can be done by solving a bipartite matching problem (see any textbook on graph algorithms for an explanation of what this is). The algorithm for this is fairly standard, and as it would be executed a lot of times, some care was taken to make it efficient. As we expect E to be fairly common, the string `EEEEEE...` should behave a like the plaintext in the sense that we expect good plugboard settings to come out of this process.

It may appear as if the approach rests upon the assumption that E is the most common letter in the plaintext. This is really not the case; if E is rare in the plaintext, the approach is still basically sound, but the produced plugboard may be a bit off.

Solving the $2 \cdot 26^6$ different bipartite matching problems may seem like a large computing effort, but the program was capable of doing this in about a CPU day on a modern PC. (This could have been cut by another factor of two if we had not tried two different interpretations of the wheel permutations, but better safe than sorry.)

So for each of the $2 \cdot 26^6$ guesses, we can find the best plugboard. But how can we find the correct guess? Notice that with the plugboard determined from the solution to the bipartite matching problem, everything is known about the state of the Enigma for each guess: The wheel settings, the ring settings, and the plugboard. But this means that we can decrypt the ciphertext for each guess! This gives $2 \cdot 26^6$ candidate plaintexts, and all that remains is to choose the best one of these — Singh's plaintext, to be precise. A human doing this would quickly get bored to the point of falling asleep, but it is an easy task for a computer program. The final design choice is how to measure the likelihood of a candidate plaintext to be the correct one. There were several alternative ways of doing this. As the Enigma was used, we guessed that the plaintext would be in German. By making this assumption, frequency tables for the letters in German text could be used. Another possible approach, slightly less efficient but not making any assumption about what language was being used, is as follows: In all languages, there are some letters which are a lot more frequent than the average 1/26 (e.g. E, T and N in English) and some that are a lot less frequent (e.g. J and Q in English). Thus, the plaintext with the most skewed frequency distribution is likely to be the right one. As the ciphertext was rather short, we decided to use the first approach as it seemed likely to be better at discovering the correct plaintext using the limited information available.

2.8.3 How we Found the Solution

Having decided upon the outlined attack, we tried it by creating an Enigma simulator, encrypting English texts, and letting the program decrypt them. The success rate was fairly high; the wheel and ring settings were always correct, but the plugboard settings were sometimes slightly off.

On October 7, Gunnar and Staffan felt ready to go and let a group of workstations try all the possible settings, storing the 100 most likely decryptions for manual inspection. In the morning, we were disappointed: None of the candidate decryptions looked even remotely like text in any language. What had gone wrong? An intense testing and debugging pass commenced, but no bugs were found.

The explanation came a couple of days later when Stage 7 was cracked: The plaintext for that stage revealed that Singh had been mischievous when constructing this stage: The first two wheels should actually be exchanged! Confident that Stage 8 would succumb immediately, the three team members who found the solution to Stage 7 pulled an all-nighter running the Enigma cracker with the correct wheel settings. Again, nothing sensible was produced by the program.

At this point we knew that we had the correct wheel setting, but our program still failed to find the solution. So what could be wrong? We looked for bugs in the program, but could not find any. Turning to the algorithmic approach, we decided on changing scoring method and removed the assumption that the plaintext was in German, instead trying the less efficient but more general method of assuming that the correct plaintext had the most skewed frequency distribution. Again, the workstations were set to their work during the night.

When Gunnar came to work the next morning, he found the following output from the program.

```
Highest scores:
  10356.00  OUA  FA  E->I  SBTDEFGHINLKUJWPQRACMVOXYZ
  10354.00  OUA  FP  E->I  SBTDEFGHINLKUJWPQRACMVOXYZ
  10232.00  OUA  FA  E->X  SQCDEFGHRNLKUJTPBIAOMVWXYZ
  10220.00  OUA  FP  E->X  SQCDEFGHRNLKUJTPBIAOMVWXYZ
  10206.00  OUB  FB  E->I  SBTDEFGHINLKUJWPQRACMVOXYZ
  10168.00  OUA  EA  E->I  SBTDEFGHINLKUJWPQRACMVOXYZ
  10168.00  OUA  EP  E->I  SBTDEFGHINLKUJWPQRACMVOXYZ
  10150.00  OUB  FC  E->I  SBTDEFGHINLKUJWPQRACMVOXYZ
  10112.00  OUA  EA  E->X  SQCDEFGHRNLKUJTPBIAOMVWXYZ
  10104.00  OUA  EP  E->X  SQCDEFGHRNLKUJTPBIAOMVWXYZ
```

The second column gives the initial position of the wheels, the third column gives the ring settings, the fourth column the mapping of the letter `E`, and the sixth column the rest of the plugboard settings. The best solution corresponds to a plugboard with 7 swaps: `E` $\leftrightarrow$ `I`, `A` $\leftrightarrow$ `S`, `C` $\leftrightarrow$ `T`, `J` $\leftrightarrow$ `N`, `K` $\leftrightarrow$ `L`, `M` $\leftrightarrow$ `U`, `O` $\leftrightarrow$ `W`. When Gunnar used this plugboard to decrypt the ciphertext, he got the following plaintext.

```
DASXATESUNGKOCRWXISWJBLHWCXXSWEXEXNEUIXELWHAELAKEINEHJIWWEIL
KRVXCIEXVIWXXESXEAWKCDIENOXISWXXITHXHABEXVASXLINKSSWEHEMDEXB
YWEXDEXESTHLUESSEPSXENJDETKWXXESXISWIEINSXNIESXGERCXEHNSXZER
MXZERCXEVESXEIOSXEINSXEITHEPRCGRAMMIERWEXDESXUNDXENWDETKWEXD
ASSXDASXOCRWXYEBUGGERXCEDNNESYMIWXOEMXUNWEBSWEHENDENXSTHJUEU
SELXENWKCDIERBXONRDFALJIRESULWAWXDIEXUNWENSWEHENDENXSTHRISWZ
EITHENXHAW
```

As we can see this look almost like German. Some of the plugboard connections are however wrong. If we look carefully we see the "words" `STHLUESSE`, `PRCGRAMMIER`, and `RESULWAW` in the plaintext. It looks as if `C` should not be swapped at all, and that instead `O` and `T` should be swapped. If we then leave `W` unmodified we get 6 swaps altogether: `E` ↔ `I`, `A` ↔ `S`, `J` ↔ `N`, `K` ↔ `L`, `M` ↔ `U`, `O` ↔ `T`.

Using these settings the complete plaintext emerged along with the codeword **PLUTO**:

```
DASXLOESUNGSWORTXISTXPLUTOXXSTUFEXNEUNXENTHAELTXEINEXMITTEIL
UNGXDIEXMITXDESXENTKODIERTXISTXXICHXHABEXDASXLINKSSTEHENDEXB
YTEXDESXSCHLUESSELSXENTDECKTXXESXISTXEINSXEINSXZEROXEINSXZER
OXZEROXEINSXEINSXEINSXXICHXPROGRAMMIERTEXDESXUNDXENTDECKTEXD
ASSXDASXWORTXDEBUGGERXWENNXESXMITXDEMXUNTENSTEHENDENXSCHLUES
SELXENTKODIERTXWIRDXALSXRESULTATXDIEXUNTENSTEHENDENXSCHRIFTZ
EICHENXHAT
```

The plaintext was in German, but the program still could not recognize the right plaintext when using frequency tables for German. What was the explanation for this? Inspecting the plaintext, it is clear that X was used as word delimiter and XX as sequence delimiter. As X is rare in most German texts, the approach based on frequency tables failed to recognize the correct plaintext.

2.9 Stage 9

Solution found on November 12, 1999.

2.9.1 Initial Knowledge

The solution to Stage 8 contains the following clue:

```
STUFE NEUN ENTHAELT EINE MITTEILUNG DIE MIT DES ENTKODIERT IST.
ICH HABE DAS LINKSSTEHENDE BYTE DES SCHLUESSELS ENTDECKT.
ES IST EINS EINS ZERO EINS ZERO ZERO EINS EINS EINS.
```

At first this seems somewhat confusing; how can the first byte of the key contain *nine* bits? The explanation for this is as follows. A DES key is 56 bits, but it is usually split into 8 blocks of 7 bits each, and after each block a parity bit is inserted. This means that the eighth bit in the clue is the parity bit for the first seven bits.

The second part of the solution to Stage 8 provides an explanation to the mysterious sections labelled “Schlüssel” and “Schriftzeichen” in Stage 8:

```
ICH PROGRAMMIERTE DES UND ENTDECKTE DASS DAS WORT DEBUGGER WENN
ES MIT DEM UNTENSTEHENDEN SCHLUESSEL ENTKODIERT WIRD ALS RESULTAT
DIE UNTENSTEHENDEN SCHRIFTZEICHEN HAT.
```

This made it possible for us to verify that our DES implementation worked the same way as Singh's.

2.9.2 Work Description

DES has been around since 1976, and no significant weaknesses have been discovered. No algorithm significantly better than trying all the 2^{56} possible keys have been discovered. With the 8 bits from the clue, the search space is reduced to $2^{48} = 281474976710656$ keys. Brute force seemed to be the only option for this stage, and we immediately started on two tasks:

- Implementing an efficient key tester.
- Creating a distributed system where a central server distributed jobs to a large set of client computers.

Up to now we felt that we had made progress, and in order to start this monster run as soon as possible we split the implementation between three people: Staffan created the server, Fredrik wrote a screensaver client, and Gunnar implemented the key tester.

2.9.3 The key tester

There are several possible modes of operation for DES, it can either work as a stream cipher or as a block cipher. We guessed that the block cipher mode, Electronic Code Book mode, was used as it for the other modes would not suffice to know the key; an initial value block of 64 bits also would have to be found. Attacking these modes of DES seemed too hard for us (and for this competition), so we hoped that ECB was used, in which case "only" 48 unknown key bits would have to be determined.

As we were about to test all the 2^{48} keys, we had to decide upon a way to determine if a key could be the right key. This could of course be done by decrypting the entire message for each key, but this would mean that about a hundred blocks would have to be decrypted with each key. A back-on-the-envelope estimate of the number of keys we could hope to test in a second indicated that this was out of the question. Instead, we made the reasonable assumption that the plaintext only would contain characters from the 7-bit ASCII character set. Only if the first 8-character block of the plaintext we got for a key was solely 7-bit ASCII the next block was tested. This means that for about 255/256 of all keys, only the first block would have to be decrypted; a huge speedup compared to decrypting the entire ciphertext for each key. The complete rules we used for each key were the following:

- The first block must decrypt to 7-bit ASCII.

- At least one of the second and third blocks must decrypt to 7-bit ASCII.
- At least five of blocks 4–12 must decrypt to 7-bit ASCII.

If a key failed one of these tests, it was discarded and the later rules not applied. Based on the assumption that a text encrypted with DES behaves like a random string of binary numbers, we estimated that we would get less than 10 incorrect keys using these rules. On average, only slightly more than one block needed to be decrypted for each key.

The actual decryption of a key was done using a technique called *bit-slicing*: On a 32-bit processor, such as the Intel Pentium, 32 keys can be tried in parallel by letting each bit position correspond to each key. The processor is viewed as a SIMD (Single Instruction Multiple Data) parallel processor with a highly restricted instruction set. This approach restricts the operations possible during the decryption stage to the boolean operators AND, OR, XOR and NOT, but as was observed a couple of years ago by the Israeli researcher Eli Biham, it is possible to implement DES encryption and decryption efficiently using only these operators. By making use of this technique and optimizing the code heavily, we could obtain very high speeds: On the fastest workstation we had available, a DEC Alpha, more than 3 million keys/second could be tested.

2.9.4 The client/server connection

Even with the high speeds achieved, testing all the 2^{48} keys would require several CPU years on a typical PC or workstation. But there was no reason to test all the keys on one single computer, so we decided to parallelize the key testing task. We wrote a key server program whose job it was to keep track of what parts of the key space had been tested. Clients could connect to the server and ask for a job containing a set of keys to test. Each such job contained $2^{30} = 1073741824$ keys, and there were $2^{18} = 262144$ jobs in total. The server was run on a computer always connected to the Internet; this way we would not have to restrict ourselves to computers in a single network.

With a server eager to distribute jobs, all that remained was creating client programs, and finding as many computers to run these clients on as possible. The core of the client code was the key tester described above, but on top of it we put several layers of interface code in order to make the client able to run on as many different platforms as possible. We began by implementing the client as a standalone program, running on Unix workstations. In order to explore more CPU resources, a Windows screen saver version of the program was also created. When installed on a PC, it would automatically connect to the key server, get a portion of the key space, test these keys, request the server for a new set of keys etc. In order for the screen saver to be able to connect to the key server from behind firewalls, we had to make it connect through the HTTP port.

Having created the server and client programs, all we had to do was find as much CPU power as possible, start the programs, and start waiting eagerly. To track the progress, the server automatically updated a webpage with the current status: The fraction of the key space covered, the number of clients reporting jobs as finished during the last hour etc. Last but not least, a very important task for the server was to keep track of potentially correct keys. Any key which passed the tests described above was used to decrypt the entire ciphertext, and the result was then automatically sent by email to all team members.

2.9.5 How we Found the solution

Slowly but steadily, the clients worked their way through the vast key space. After a week or so, the key server sent an email containing an alleged plaintext! Unfortunately, this was a key which passed the tests but did not yield a plaintext which looked reasonable. A couple of days later, another such key was discovered. Then, three weeks after the clients were started, the key server sent the following message:

```
Date: Fri, 12 Nov 1999 06:10:26 +0100 (CET)
From: DES server <des-server@multivac.fatburen.org>
Subject: Good news

This is a message from the server that coordinates the code-breaking
effort on Stage 9.

A client just reported that it might have found the correct DES key.
The key and the ciphertext decrypted with the candidate key follows.

Key: d3cd1694cba126fe

Message:
O!.cursed,.cursed.slave...Whip.me,.ye.devils,.From.the.possessio
n.of.this.heavenly.sight!.Blow.me.about.in.winds!.roast.me.in.su
lphur!.Wash.me.in.steep-down.gulfs.of.liquid.fire!.O.Desdemona!.
Desdemona!.dead!..Only.one.more.phase.to.go...Your.token.for.thi
s.one.is."Armstrong"...The.final.hurdle.contains.two.ciphertexts
...The.longer.one.is.encrypted.with.triple-DES.in.electronic.cod
e.book.mode...It.uses.a.128-bit.EDE.key...The.RSA.public.key.cry
ptosystem.has.been.used.to.encipher.the.triple-DES.key;.this.for
ms.the.shorter.ciphertext...The.public.exponent.is.3735928559.in
.decimal.and.the.public.modulus.is.the.following.155-digit.decim
al.integer:..10742.78829.12665.65907.17841.12799.42116.61266.392
17.94753.29458.88778.17210.35546.41509.80121.87903.38329.26235.2
!D<..t]s06.72083.50494.19964.33143.42555.83344.01855.80898.94268
....‘h...Your.task.is.to.decipher.the.longer.message.and.so.disc
over.the.final.code.word...How.you.do.th.y..R(?..to.you,.but.it.
will.almost.certainly.be.much,.much.easier.to.break.the.RSA.ciph
ertext.by.factoring.the.public.modulus.than.by.any.other.approac
h...Good.luck!..
```

We had found the right key, yielding the codeword **ARMSTRONG** for this stage. Some parts of the message looked a bit strange, though. After checking these sections by hand, we discovered that the OCR scanner software we had used for importing the ciphers from the book into electronic form had made a couple of errors. After having corrected these, all the text looked good. We were aware that scanning errors could have occurred, and we had checked the first 12 blocks carefully as errors there could result in the correct plaintext being discarded.

We were a bit lucky; only about 17% of the key space had been searched when the correct key was found. This corresponded to about a CPU year on the hardware we had available. The client actually running into the correct key was one of the screensavers, running on a rather slow PC, and the job which contained the key would have been aborted if the user of that particular PC had not been a bit later to work than usual that day. If this would have happened, a lot of time could have been lost: Jobs that had been assigned to a client but not finished were placed last in the job queue by the server. Sometimes luck can be as important as a good approach.

2.9.6 Afterpiece

Later in the spring while working on Stage 10 we were a little bit worried about the evolving group efforts to solve Stage 9. We knew we had quite some time to go before we had the solution at hand, and also that the more people having the solution to Stage 9, the greater the competition would be. At this point, Fredrik actually joined the TCBCrackStage9 group in order to stay in the group and get accepted into the TCB_TFF group. He also reported one block since that was necessary to stay in the group and get accepted into the TCB_TFF group.

Since we had the correct key in our possession, we calculated what block it would be in and fed that block number to the TCBCrackStage9 team's DES cracker program. This did not work! We happlily concluded that they would not find the solution using that program, which turned out to be true. As we all know, they later fixed the problem and found the key anyway.

We are sure a very interesting discussion would have taken place within our team if it had turned out that Fredrik had been assigned the correct block. Would we have wanted to report the key?

2.10 Stage 10

Solution found on October 5, 2000.

2.10.1 Initial Knowledge

After solving Stage 9, we had all the input data that was needed to start off on Stage 10. During this time it was actually a little bit amusing to follow the discussions in the mailing lists among the ones who had not seen the plaintext of Stage 9.

Somehow, we had suspected that the final stage would be based on RSA encryption, and had been doing some experiments with the 155 digit number presented as "the short message." It obviously is not an RSA modulus since it is divisible by small factors, and neither is any contiguous substring of the 155-digit string.

Since Torbjörn was known to be interested in factorization and large integer computations in general, we asked him how to best factorize the 155-digit number at hand. Torbjörn said that the only method that could be used for such a large number is the General Number Field Sieve (GNFS), but that it would require on the order of 40 CPU years. Torbjörn had previously factored several numbers using GNFS, but never anything close to a 155-digit number.

2.10.2 Work Description

When we realized that we needed to better understand GNFS, so turned to Johan Håstad and asked him to teach us. It then quickly became apparent to us that GNFS is a very advanced algorithm—it would take us a very long time to implement it ourselves. So, of course we started to investigate the possibility of using the same code that had been used for CWI's record factorization of RSA-155 some months earlier. Fortunately, Nada at KTH where three of us worked had access to CWI's GNFS implementation. Their source code comes with the following warning:

> The code is a research product which is reasonably documented, but it is very advisable to know the (complicated) Number Field Sieve factoring method, in order to use the code successfully.

Indeed, we were quite confused after receiving the code. We started out trying to run the different programs included in the program suite, and eventually managed to factorize a 17-digit number after a full afternoon in Johan's office. We alternately read articles about GNFS, looked at the source code, and tried to factorize larger and larger numbers for well over a month. We also ran the first step of the algorithm, the polynomial finder, during this time. The GNFS algorithm consists of the following steps.

1. Find a pair of irreducible polynomials (f_1, f_2) with certain properties.
2. Select a few parameters.
3. Search for relations between the polynomials where they are both smooth.
4. Combine the relations to find dependencies. This involves filtering and the solving of a large linear system of equations. Each dependency, $a^2 - b^2 = (a+b)(a-b) \equiv 0 \pmod{N}$, is a 50% chance of finding a factorization.
5. Use the dependencies to factor the number.

The parameters in the second step should be chosen so that it is easy to find many relations, but the number of relations that we need also depends on the parameters. Since we did not have much intuition here, we tried varying the parameters until we thought we could finish the sieving step in reasonable time. Actually, it seems that we did quite a poor job since we used almost twice the computing power compared to what CWI used when breaking RSA-155.

The search for relations is carried out by sieving. This task can easily be shared between many computers, where each one checks a different range of a and b values for possible relations. To coordinate the hundreds of computers we had access to at Nada, we modified the client/server solution used for Stage 9. Each task took the client machines a few hours to complete. Since the siever uses over 100 megabytes of memory, we did not want the program to run on computers that were in use, so the program stopped its computation whenever somebody logged on to the machine.

On August 20, we had obtained 75 million relations, which we suspected might be too few, but nonetheless we moved on to processing ("*filtering*") our relations to see how far we had progressed. We were positively surprised in that after the first filtering stage, were we threw away relations that had singleton ideals, there were 37 million relations and 35 million ideals. This is the same as having an linear equation system with 37 million equations and 35 million unknown variables, which implies there are 2 million dependencies between the equations. We had enough relations!

For a successful factorization, one needs just a few dozens of dependencies, so we could just throw away most of the excessive relations. But solving an equation system with 35 million variables—even a very sparse one like this—is a computationally unsurmountable task, so further filtering was necessary. With 2 million redundant equations, we should be able to drastically reduce the equation system size.

It is important to understand that we are looking for an equation ${\alpha_1}^2 - {\alpha_2}^2 = (\alpha_1 + \alpha_2)(\alpha_1 - \alpha_2)$. We really only care about if the multiplicity of an ideal is even or odd—if we can create an equation with even multiplicity for all ideals, we have a dependency that might lead to a factorization. We can therefore compute over GF(2).

The second filter pass means combining sets of relations that have an ideal in common. If we combine all relations for such an ideal, that ideal is reduced from the equation system, and we also get fewer relations, although of higher ideal weight. But since we had 2 million redundant equations, we could here simply throw away those equations that became very heavy!

Filtering is an iterative process, where one tries to select parameters that reduce the equation system as much as possible. We finally had about 8.4 million equations and variables and a total matrix weight of about 500,000,000.

The filtering was performed on a dual processor Alpha DS20 with 4GB main memory. This is an extremely powerful system, but filtering is a daunting task even for this machine. It took about 3 weeks before we had achieved results that we were not able to further improve.

During the long filter computer runs, we started to think about how long it would take to solve the linear equation system, and if we could improve the program. The estimated running time to solve this system on the Alpha was 150 days. Simply too much. We could not afford such a long computer run, and although the computer center at UMS Medicis was very generous, we felt that we could not occupy their most powerful machine for such a long period.

The program we used for this was optimized for running on vector computers, which is what CWI used for their record factorization last year; it had taken them 10 days to solve a slightly smaller equation system on a 16-processor Cray C90. We started to rewrite this program so that it would run better on the hardware available for us, and after a few weeks of hard work, we had a version of the program that used both processors of the computer, and would have to run for 37 days.

At this point, we contacted Compaq to get some help for the computation, since we thought this was great opportunity to show how powerful the Alpha is for scientific computations. Compaq generously let us use one of their quad processor ES40 systems. The total running time on this machine was 13 days, which is almost as good as the 16-processor Cray. Thanks to the Alpha processor, we have thus been able to show that it is possible to factorize 155-digit numbers without using expensive vector computers.

The program produced the result on October 4, at 5:05am, yielding 11 true dependencies. We then only had to complete the last step, which is using the dependencies to get the factor. This involves taking square roots of huge ideals in $\mathbf{Q}_n[\alpha_i]$, and took 11 hours. Each dependency gives the desired result with 50% probability, so we started five simultaneous runs of the program in order to have a reasonable probability of success.

One of the runs gave us the factorization of the RSA modulus:

$$
\begin{aligned}
&10742788291266565907178411279942116612663921794753294588877817210355464150980l \ldots \\
&\ldots\ 21879033832926235281090750672083504941996433143425558334401855808989426892463 = \\
&= \mathrm{p78} \times 128442051653810314916622590289775531989649843239158643682161776470431377654 77
\end{aligned}
$$

2.10.3 How we Found the Solution

In the morning on October 5, the factorization was found. Staffan was the first to discover this, and he immediately called the other team members. Everybody dropped what they were doing and rushed to Staffan's place to be present when the plaintext for the final stage was revealed.

Having found the factorization, it was easy to decrypt the shorter message, which had been encrypted with RSA. This gave us a 128-bit number, which we knew would be a triple-DES key. Tense with anticipation, we fed the longer message and the key into a DES decoder, read the output... only to discover that it was a sequence of random characters, nothing remotely resembling a plaintext. What could have gone wrong? We tried another DES decoder, which we had created ourselves while working on Stage 9, but it gave the same results. We knew that the factorization was correct, and the 128-bit key looked like a proper DES key as it had the right parity properties. Overwhelmed by frustration, a number of schemes were tried: Exchanging the two 64-bit blocks which the key was composed of, reading the bits of the key in reverse order, etc. After an hour of failed attempts, Lars suddenly screamed “Here! It is in P1!” and we all started jumping around with joy. It turned out that the message was not encoded with triple-DES after all: It was only protected by a single DES key, and we had had the file containing the right plaintext without knowing it: Lars discovered that a temporary file output by our DES decoder contained the plaintext:

> For secrets are edged tools and must be kept from children and from fools.
>
> The magic word you need to quote is DRYDEN.
>
> Congratulations, you have solved the tenth and final stage of the challenge.

3 Epilogue

After having been at the Post Office and mailed a letter announcing our success to Simon Singh, we decided to send a fax to the publisher of The Code Book, 4th Estate, as well, in order to give advance warning of the letter. We also called Royal Mail to find out where the magic P.O. Box was located, and to see if we knew anyone who could go there to post a letter that would get there faster. We also called the airlines to see how much it would cost to send one of us to London to send the letter off. In the end, we decided to settle for the combination of the letter, that was supposed to be delivered on Saturday, and the fax.

As the rest of us were all excited, no one noticed that Torbjörn snuck into the bathroom and called his friend Björn. Twenty minutes later, the phone rang and it was rather interesting to watch Staffan's facial expression change as he started jumping and pointing to the phone. Something was happening. It was Singh on the phone! Staffan's expression changed for the worse as he was informed that we were the second ones to submit the correct solution. At this point Torbjörn started screaming "Stop it now Björn! It's enough!" It was his friend Björn performing the practical joke...

Later that evening, the recent messages on the Cipher Challenge mailing list were scrutinized, and some very worrying messages were discovered: One person on the list claimed to have submitted a solution to all stages earlier the same week! Provided this was true, it would mean terrible news for us. What if we lost by a couple days after having worked on the challenge for a year? We tried to calm each other down by saying that it could be a hoax, but doubt still lingered.

On the morning of Friday October 6th, Lars was becoming even more nervous and sent the following mail.

```
From: Lars Ivansson
To: Fredrik, Gunnar, Staffan, Torbjorn
Sent: Friday, October 06, 2000 07:50
Subject: Nervous

Hello!

I would like to say that I am very interested in knowing when/if
Singh contacts any of you. Please send a mail as soon as
possible.  I feel like a child the day before Christmas after a
rumor about the death of Santa Claus and am basically climbing
the walls here.

/ Lars
```

Fredrik, who comes to work several hours after Lars, saw the mail and went for coffee and some work-related discussions. When he got back, a colleague complained that his phone had been ringing a couple of times when he was away.

After a couple of more minutes, the phone rang again and then Fredrik sent a mail with the following wording (translated from Swedish):

```
From: Fredrik Almgren
Subject: RE: Nervous
Date: Fri, 6 Oct 2000 11:07:44 +0200

I have just talked on the phone with an English-speaking
gentleman who said he was Simon Singh. He started off the call
with a short discussion on how he was going to make me believe
that he was indeed Simon Singh. After some rambling from my
side, he said that the first part of the plaintext for Stage 10
consisted of fourteen words and that words 5 and 14 rhyme. At
that point, I felt about ready to accept that he was really who
he claimed.

We then went through all ten stages. Staffan was here yesterday
and helped me put my X-terminal in boxes. So I couldn't connect
to multivac to verify the answers. I got very nervous that he
would say that something I said was wrong. I tried to explain
that I needed to try to remember this from head.

He said that my answers were correct.

IF he is an impostor, he obviously has seen the plaintext for
Stage 10. In that case, the only thing he could gain was to get
one of the earlier code words that he was missing. It doesn't
feel very likely. Besides, he stayed on track all through the
conversation and talked about the stages in a way only an
EXTREMELY good impostor could be expected to manage. He was
however a bit off regarding my claim that there is an error in
the 3-des encoding of the message and said that he would check
it. I did NOT tell what the error was....

He said that he had received a couple of letters from people
claiming to have solved all ten stages before us.



He also said that we were the first ones to have solved all ten
stages!!

He asked us not to announce this yet since he wanted to contact
his British, Swedish and Amercian publishers and get back to us
on Monday for a discussion on how this is going to be presented.
I then told him that we had already sent press releases
```

```
yesterday but that we would not give the go-ahead until we had
spoken to him again.

He also gave me two phone numbers:

<snip>


He explicitly asked me not to give his number to anyone else.


Well guys, it seems like we have won!!!!!

Santa Claus will come this year too!

/Fredrik

Ps. I will disconnect my computer from the network in 55
minutes. So any further discussion must take place without me.
```

To this Lars replied:

```
From: Lars Ivansson
Sent: Friday, October 06, 2000 11:15
Subject: Re: Ping

Thanks Fredrik. You are better than Santa Claus!

/ Lars
```