How the Dutch broke the Japanese Blue Code in the late 1930s

by Joost Rijneveld

Bachelor Thesis in Computer Science
Supervised by Prof. Dr. Bart Jacobs
Radboud University Nijmegen
joostrijneveld@student.ru.nl
July 2013

Abstract

In a prelude to the Second World War, Japan sought to expand its territories. The Japanese navy used several code systems to encrypt the contents of their communication telegrams, most notably a code the intelligence department of the Dutch East Indies navy referred to as the '.' code. Through breaking this code, Johannes Frans Willem Nuboer was able to provide the Dutch with a valuable tactical insight in Japanese naval communications in the late thirties. This thesis will explain how the code worked and how the Dutch intelligence department broke the '.' code. Through various examples, the inner workings of the code will be made clear. Additionally, an implementation of the algorithm that was used to break the cipher will be provided. Furthermore, evidence will be provided that indicates that the '.' code is the very same code that the United States Navy refers to as the Blue Book or Blue Code.

Acknowledgements

First and foremost I would like to thank my supervisor, Bart Jacobs, for providing me with advice and feedback throughout the process of writing this thesis, as well as valuable new leads to pursue. Furthermore, I would like to express my gratitude to the kind people at the Netherlands Institute of Military History for their time and support, and the historians at the Center for Cryptologic History of the NSA for guiding me to useful new sources.

To conclude, I want to mention Johannes Frans Willem Nuboer for the excellent documentation and notes he left behind in the archives, for me to explore.

Contents

1 Introduction

In the late nineteen thirties, the Japanese Imperial Navy was using an intricate cryptographic system to secure their communications — all the while, the Dutch were reading along. In this thesis, we will see how.

In times of war, it can provide an immense tactical advantage to be able to listen in on the channels of the enemy, especially when this enemy considers his communications private and secure. In the prelude to war, it is perhaps even more important to know what is going on in the other camp.

Cryptography and cryptanalysis seem to form a constant contrast in the work of any intelligence service, in both war and peacetime. While the two fields of study are so opposite in their goals (that is, securing communication and uncovering hidden content, respectively), they are clearly strongly intertwined. In this thesis we will study the work of an officer of the Dutch Royal Navy, tasked with setting up the intelligence department of the navy in Batavia, the capital of the Dutch East Indies. This officer is Johannes Frans Willem Nuboer. While also playing an influential role in defining the usage of cryptography to protect the Dutch communications, his most significant work was done in cryptanalysis. In the years leading up to the Second World War, it was Nuboer's work that allowed the Dutch to read a large part of the Japanese telegrams. This work, as available in Nuboer's archives at the Netherlands Institute of Military History, has provided the raw material for this thesis. Based on the descriptions and notes that Nuboer has left behind, we will unravel how the Japanese code worked, what its weaknesses were and how Nuboer was able to exploit these.

As we will see in sections 4 and 7, the Japanese used several different cryptography systems. One of these is head and shoulders above the others when it comes to usage and tactical importance: a code that Nuboer calls the '.' code, as each of the lines ends in a dot. This code will be our main focus, as it was Nuboer's. In section 9, we will provide strong (but not irrefutable) evidence that suggests that this is the very same code that the United States Navy broke and referred to as the Blue Code. Throughout the rest of this thesis we will refer to the code by the name Nuboer gave it.

Before discussing all of the above in more depth, let us first begin by placing the story of Nuboer and the '.' code in its historical context.

2 Historic context

2.1 Tension in the Far East

In the eastern part of the world, the Second World War was preluded by a series of events that built up the tension in the pacific region. The most significant of these is the Second Sino-Japanese War of 1937, which merged into the Second World War with the bombing of Pearl Harbour. But let us back up to the start of the decade.

2.1.1 A shortage of resources

The Japanese economy had been growing steadily throughout the first thirty years of the twentieth century. At this time, the export mainly consisted of raw silk and cotton fabrics – products of light industries. Until the rise of militarism in 1930, natural (and mineral) resources were never a pressing issue (Yasuba, 1996). In 1931, the political and military opinion (and with it, the public opinion) dictated that there would soon be a lack of fuel and raw materials to fulfil the domestic need. This, along with the 1929 Great Depression, presumably formed the major reason for the Japanese military expansionism. This military build-up, as well as the related heavy industries, did require more natural resources. This caused the previously non-existent shortage of oil and coal to become very real (Yasuba, 1996). The only apparent solution was the occupation of the resource-rich regions of Manchuria (northern China) and Mongolia.

2.1.2 Manchuria and the Second Sino-Japanese War

On September 18$^{\text{th}}$, 1931, a Japanese lieutenant set off an explosion on the tracks of the Japanese-owned South Manchurian Railway, in the suburbs of Mukden – this event would be known as the Mukden Incident. The explosion was intended to damage the tracks and derail the incoming train, but no harm was done. Nonetheless, the Japanese accused the Chinese of the bombing, and used this staged event as a pretext for invading Manchuria a day later (Duus, 1989). Manchuria was then renamed Manchukuo and turned into a state under Japanese control. The occupation would last until the end of the Second World War.

From 1932 to 1937, the Japanese continued to fight the Chinese on numerous occasions, both for the sake of expansion and for access to natural resources. The increased tensions were finally ignited in 1937, when on July 7$^{\text{th}}$ Chinese and Japanese troops exchanged fire near Marco Polo. This skirmish, referred to as the Marco Polo Bridge Incident, is often recognised as the beginning of the Second Sino-Japanese War (Crowley, 1963). This war would last until 1945, after becoming a part of the greater Pacific War in 1941. In the process, the Chinese capital of Nanking was captured and both China and Japan suffered great losses. The implications of conflict for the Dutch (and Dutch intelligence) will be discussed in more detail in section 8.

2.1.3 Dutch East Indies

The relevance of the hostilities between China and Japan for the Dutch East Indies is not immediately clear. It is to be found in the reason for the Japanese expansionism: the Japanese were after natural resources, and the Indies had large depots of oil, rice, sugar and tin. The developing conflict with China increased this demand, and an expansion southwards would greatly strengthen the Japanese international military position. Additionally, the inhabitants of the Indies formed a large potential market for Japanese export (Haslach, 1985). With so much to gain, it was unlikely that the strong empire of Japan would sit idly by.

The army of the Dutch East Indies was poorly staffed and poorly equipped, and the large borders of the Indies were considered to be very difficult to defend in case of a all-out war. This forced the Dutch to seek refuge in a strictly organised and well-informed diplomatic campaign to remain neutral and limit Japanese (societal and economical) influence (Haslach, 1985). In order to be able to derail the Japanese propaganda campaigns and keep an eye out for military threats while at the same time maintain diplomatic neutrality, the Dutch had to rely on an entirely different form of warfare. Signal intelligence.

2.2 Founding of Department 1, Batavia

In 1933, the Dutch navy went through an immense organisational overhaul, shaping it into three foundations. Department 1, Intelligence, Department 2, Operations and Department 3, Organisation. This model was being used by the French navy at the time, designed by admiral Raoul Castex (Nuboer, 1977g). A student at the Netherlands Naval War College was selected to set up the intelligence department in Batavia, the capital of the Dutch East Indies. This student was Johannes Nuboer.

The newly formed Department 1 was tasked with gaining insight in the plans and actions of the naval forces that were of strategic importance to the Dutch position in the Far East: the British Royal Navy, the U.S. Navy, the French National Navy, and the Imperial Japanese Navy (Nuboer, 1977i). The intelligence department would focus mainly on the Japanese, as Department 1 of the naval staff in The Hague would study the Western naval forces. Given the growing tension in the area, as described above, as well as the Japanese role in these matters, this was not going to be an easy task.

2.3 Johannes Frans Willem Nuboer

Johannes Frans Willem Nuboer was born in the Dutch East Indies city of Pontianak on October 2nd, 1901 (Bosscher, 1994). He moved to the Netherlands during his childhood, where he enrolled in the Naval Academy at age 17, graduating as a naval officer in 1921. He specialised in the field of the liaison service, and followed a course in cryptography in 1930, taught by major Henri Koot. He ended up playing a significant role in the navy's adoption of the Enigma cipher machine in 1931. From 1932 to 1934, Nuboer followed a course that involved

working in various departments of the navy. Bosscher (1994) mentions that it was typical for Nuboer to rapport his unvarnished critique on the war-readiness of the various parts of the navy. In 1934, Nuboer was dispatched to the Dutch East Indies with the task of founding Department 1.

For the next four years, Nuboer would play an important role in breaking Japanese cryptography, as will be described in the following sections.

In 1938 he returned to the Netherlands, where he took on the position of a teacher and acting director at the Higher Naval Academy (*Hoogere Marine Krijgsschool*). However, before and during the Second World War, Nuboer was attached to the naval staff and, from London, served as an advisor to Admiral Furstner and Prime Minister Gerbrandy (Visser, 2005).

At his discharge in 1953 he was decorated for his remarkable service to the navy and awarded the title of *schout-bij-nacht* (equivalent to that of rear admiral). Nuboer was likeable and behaved himself with modesty. Although he often made a shy impression, his intelligence, perseverance and high moral standards demanded respect of his co-workers (Bosscher, 1994).

2.4 Gathering intelligence

In command of the newly formed Department 1, Nuboer intended to supply the other departments of the navy with monthly reports of his findings. The distribution of this intelligence material would be limited only to the Commander of the Navy, the Chief of the Naval staff in The Hague, several local military authorities as well as the head of the general intelligence service for East Asia (DOAZ), Antonius Lovink, who provided intelligence to the Governor-General (Nuboer, 1977g). As Japanese pressure increased and started influencing the political agenda, Lovink and his staff shifted more and more focus towards Japan (Meijer, 2002) and the information provided by Department 1. These reports consisted of information gathered from a wide variety of sources.

The lack of human intelligence operations is quite striking. The use of human espionage was considered to be impossible from a practical point of view, as the Japanese society was very closed and private at the time. A foreign visitor would be noticed immediately, especially when found wandering off the beaten track or when exhibiting interest in military sites and affairs. This made Department 1 especially dependent on other ways of gathering intelligence (Nuboer, 1981).

2.4.1 Press information and the diplomatic mission

The staff in Batavia kept a close watch on the information that was published in the Japanese press. Additionally, the Dutch diplomatic mission in Tokyo often sent letters to Batavia in which they would provide background and context to international news. Lieutenant at sea J.A.L. Muller, the Dutch adjunct marine attaché to the diplomatic mission in Tokyo, would often send along noteworthy fragments of Japanese newspapers describing changes in the military organisation (Nuboer, 1977g). What might seem like public information to the

Japanese was valuable (and hard to obtain) for the intelligence service back in Batavia.

2.4.2 Photographs of Japanese vessels

The work of Department 1 was greatly aided by photographs of various Japanese military vessels. Nuboer did not want to actively involve civil ships that sailed the Java-China-Japan route in order to keep them from trouble, but when, in 1935, a private captain approached him with photographs of the aircraft carrier Ryujo, Nuboer happily obliged. These photographs showed a remarkable level of detail. In 1934, lieutenant Muller, still in Tokyo, had even managed to send Nuboer a book with a complete collection of photographs of Japanese ships. This book is further mentioned in section 5.5, as it provided ship names that were very useful cribs while breaking the code.

2.4.3 Room 14 and Japanese naval telegrams

In Bandung, the General Staff employed its own intelligence bureau. Room 14, as it was called, housed two cryptographers, Lt. Col. J. A. Verkuyl and Lt. Col. W. van der Beek. Both had been instructed by Henri Koot, who had also taught Nuboer. The cryptographic efforts of Room 14 focussed on Japanese diplomatic ciphers that were used by the Japanese ministry of Foreign Affairs and their consulate in Batavia. This would provide valuable diplomatic context to the Dutch Indies' military leader, and Nuboer was happy to include it in the reports.

The telegraphic interception station at Batavia mainly monitored naval intercepts. These intercepts were sent to Room 14 in Bandung, but were generally filed and archived unprocessed, as the small staff was immensely occupied with the diplomatic communication. For Nuboer, as we are to see in the next sections, these marine telegrams formed the utmost important source of information, and many of his reports were centred around these findings. (Nuboer, 1977i)

Now is as good a place as any to introduce lieutenant at sea J.M. Schalkwyk, who was to join the cryptanalysts of Room 14 in June 1935. Trained in the Netherlands, he was recruited by Department 1 and immediately detached to Room 14 in Bandung. With Schalkwyk's detachment, the interest of Room 14 in naval telegrams grew significantly. Much of the work was still done in Batavia, but Schalkwyk made great progress in preprocessing the data. Schalkwyk would turn out to be a valuable colleague to Nuboer.

3 Japanese telegrams

As the telegraphists became more familiar with the telegrams, several were intercepted each night. The transmissions originated from the Tokyo radio station as well as the stations of Bako and Formosa and warships in the area. This information was readily available, as the address headers were always transmitted in plain text, even when the message itself was enciphered. This changed August 1937, when Japan started hostilities towards China and changed to wartime standards (as is mentioned in more detail in section 8) (Nuboer, 1981).

The telegrams were fitted into grids of ten by ten, filling a page per grid. Each such grid was called a 'hone' and multiple grids were used for longer telegrams, labelled 'hone 2', 'hone 3' etcetera. For enciphered telegrams, the last square of each row contained an indicator symbol signifying which code was used – most commonly a '.', but also 'Ni', 'Sa' and 'I' occurred. The telegrams listed in appendix B and C adhere to this format and were encoded using the '.' cipher. While decrypting them, Nuboer referred to the code systems by these symbols.

3.1 Address headers

The address header always started with 'ate', which is loosely equivalent to 'to'. Then followed the call sign of the addressed recipients, optionally followed by a quotation mark (or hyphen) and the call signs of any other recipients that received the message for information (much like a carbon copy in present-day email). This was then followed by an equals sign and the call sign of the sender (Nuboer, 1977i). See figure 1 for a comparison to email headers.

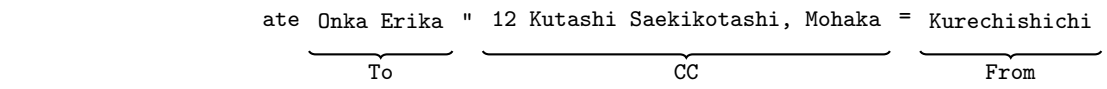

Figure 1: Illustrating the telegram header format by comparison

Based on the contents of readable telegrams, as well as the ever increasing level of insight in the organisational structure of the Imperial Navy, the Dutch were able to grasp the meaning of most of the call signs. For example, `Da` was used by the Chief of the Naval staff and `Reichi` indicated the Commander in Chief of the Combined Fleet. These call signs could also be combined by concatenation: `Shika` meant commander and `Ren` was used for the training squadron, so `Renshika` was the commander of the training squadron. For a more complete list, see appendix A.

In his memoirs, Nuboer provides an example to show the relevance of the address headers. On March 29th 1935, four encrypted telegrams were intercepted, with the following headers: (Nuboer, 1977g)

```
ate Renshika " Yokochi, Kurechi, Sahochi, 1, 2, 3, Shichi, Omiyo, Maiyo,
Chinyo, Bayo, Riyoyo, Chimaka Shika, Kigachi, Hegachi = Da

ate Natoka " Hieka, Sunoka, Chichimuchi, Naza = Yokochishichi

ate Onka, Erika " 12 Kutashi, Saekikotashi, Mohaka = Kurechishichi
```

```
ate Notoroka, 27 Sesutashi " Tomosuka, Sataka, Setaika, Sesutashi,
Hakuroka, Araka, Nachika = Sahochishichi
```

From the headers it can be seen that this was a message sent by the Chief of the Naval staff (`Da`) to the bases of Yokosuka (`Yokochi`), Kure (`Kurechi`) and Sasebo (`Sahochi`) – the `-chi` postfix incidates a base. The Commanders in Chief of the respective bases then sent a message to their ships, likely forwarding the original message (see the 'senders' of the last three headers: `Yokochishichi`, `Kurechishichi` and `Sahochishichi`).

Amongst the stations that were directly addressed (i.e. not just for information) were the training squadron (`Renshika`) and the tankers Ondo (`Onka`) and Erimo (`Erika`), all of which were abroad. The training squadron would soon arrive in Singapore (on April 1st) and the tankers were transporting oil from California. It was thus likely that the message somehow concerned ships that were abroad.

From this, the Dutch concluded that the other addressed ships were also on a mission abroad (Nuboer, 1977g). This included the cruiser Natori, the aircraft carrier Notoro and the 27th submarine division. Other than the fact that this provided valuable tactical information, these messages also gave further insight in the organisational structure of the Japanese Navy. Most notably, they showed which ships were not included in numbered fleets, and which ships were under the command of Yokosuka, Kure or Sasebo.

3.2 Telegram numbers

Every telegram started with a group of nine digits that uniquely identified each telegram and allowed a receiver to order them chronologically (Nuboer, 1977i).

The first three digits indicated the sequence number of the telegram with respect to the sender. Each sender started with sequence number zero on January 1st, midnight. The fourth digit indicated a sequence number that could be used when multiple telegrams were used to transmit a large message. This was usually zero. The fifth and sixth digit indicated the day of the month. Interestingly, the month was not included in the code, indicating that every sender would have to transmit at least monthly for the full date to be deducible from the code. Due to the speed of telegraphy (as opposed to letter-writing), specifying the month was generally considered redundant. Digit seven to nine indicated the time in hours and tens of minutes – 024 would indicate 2 hours and 40 minutes, or 02:40 in the morning.

For illustration purposes, let us apply this to the numbers of the telegrams listed in appendix B and appendix C. The April 1st telegrams were marked 028101093 and 029201093. Grouping the numbers according to the aforementioned groups, we get [028][1][01][093] and [029][2][01][093]. This shows that the telegrams were sent sequentially and were part of some larger message, and both telegrams were sent on the first day of the month, at 09:30.

The April 5th telegram was marked [031][0][05][173]. Apparently the sender transmitted one other telegram (number 030) between April 1st and April 5th, and this telegram was not part of a larger message. It was transmitted at 17:30.

4 The Ni code

Telegrams in which the last symbol of each line was `Ni`, were encoded using a code book system that was not encrypted, other than by the obscurity the code book provided. Such a code book scheme relied greatly on the secrecy of the system, but was broken efficiently by analysing the message structure, finding often recurring combinations of code groups and, perhaps most significantly, by comparing the contents of telegrams with the actual occurance of real-life events and military actions (Nuboer, 1977i).

The code consisted of groups of three Kana symbols (a Japanese alphabet of forty-four symbols – see appendix D), of which the third symbol was a control symbol – in case one of the three characters was garbled in transmission, it could be reconstructed from the other two. This meant there was only room for 1936 (44^2) unique groups. The code contained systematic tables of numbers, dates, timestamps and names of ships – including British, American and French navy ships stationed in the Eastern-Asiatic area. It also contained codes signifying single (or pairs of) characters that allowed the user to spell words for which there was no code available (Nuboer, 1977i). All of the aforementioned left only a small amount of groups for other expressions, mostly build up of codes indicating naval movements. Accordingly, the contents of most messages for which this code was used concerned the movement of British, American and French vessels in China. Because of the large number of structured codes, the Dutch were able to reconstruct nearly the entire code book by filling in gaps in the tables and data structures (Nuboer, 1981).

4.1 Reciphered telegrams

Sometimes `Ni` telegrams were received that did not adhere to the codes of the code book. Using frequency analysis, Schalkwyk concluded that these telegrams contained a recipherment of the original `Ni` code. The frequencies with which the symbols occurred matched those of the original code groups, indicating a transposition cipher (Nuboer, 1977g).

In 1936, Nuboer and Schalkwyk were able to uncover the workings of this recipherment. The code groups were copied into a figure along the diagonals, after which the columns were read in a specified order. Certain pieces of the figure were left empty, as to further conceal the code groups. The figure also included fields of which the contents would be omitted entirely during decryption, serving as noise. This recipherment is very similar to the recipherment system detailed in section 6, and could be considered a variation.

After decryption, it turned out that many of the re-encrypted messages concerned movements of the Japanese Imperial Navy (rather than information about the British or French navies, which made up the content of most of the original `Ni` messages) (Nuboer, 1977g). Unfortunately, no actual `Ni` code telegrams or recipherment figures appear to have survived.

5 Breaking the '.' code

5.1 The breakthrough

On February 20th, 1935, a Japanese training squadron left port at Yokosuka. After brief visits to Keelung, Hongkong, Manilla and Bangkok, it visited Singapore on March 28th. The squadron left Singapore on April 1st and docked in Batavia from April 3rd to April 5th. It would continue its journey southwards, sojourning in Melbourne and Sydney before heading east towards Honolulu and back to the ports of Yokosuka late July, 1935 (Nuboer, 1977i).

After leaving the port of Singapore on April 1st, and again when leaving the port of Batavia on April 5th, the squadron sent out a remarkable encrypted telegram (see appendix B and appendix C for both complete telegrams). These telegrams were especially intriguing, as they contained many repetitions.

After studying the telegrams thoroughly, Nuboer and the staff at Bandung noticed that the last line of hone 1 of the April 1st telegram (028101093) and the fourth line of hone 2 of the April 5th telegram (031005173) contained eight identical symbols (Nuboer, 1977a). These lines read:

```
April 1st: Hi  Shi He  Tsu So  Yu  Ke  Ta  Wa
April 5th: Ta  Ke  Hi  Shi He  Wa  Se  Tsu So
```

5.2 Permuting the columns

All symbols occur in both lines, except for the `Yu` in the first line and `Se` in the second. This could indicate that a cipher system was used that permutes nine columns of plaintext. The `Yu` symbol can be either the first or last symbol of the plaintext line, grouping the other eight symbols together. As the same permutation is applied to both lines of text and the `Yu` and `Se` symbols are in different positions, one of the two has to be the first symbol while the other has to be the last.

In his memoirs, Nuboer does not explicitly specify how they found the permutation that leads to an identical ordered sequence of symbols for both lines. Mentioning that he arbitrarily selected the case where the `Yu` symbol gets placed at the back, he declares finding the permutation `5-2-6-3-7-9-1-4-8` (signifying that the first symbol is mapped to position five, the second to position two, the third to position six, etcetera) (Nuboer, 1977a). Applying this permutation to both of the aforementioned lines of ciphertext results in the following plaintext sequences:

```
April 1st: Ke  Shi Tsu Ta  Hi  He  So  Wa  Yu
April 5th: Se  Ke  Shi Tsu Ta  Hi  He  So  Wa
```

After Nuboer had found a candidate permutation that might have been used to encipher the telegrams, he applied it to the entire ciphertext. He soon found that his suspicions had been correct: suddenly, groups of symbols started recurring throughout the ciphertext at a spectacular rate. At first it seemed like the symbols were paired together, but upon closer inspection the repetitions turned

out to affect groups of four Kana symbols, indicating a recipherment of existing code groups (Nuboer, 1977a).

5.3 Confirming the permutation

Using computer systems available today, finding a permutation of nine symbols that results in a common sequence of eight symbols is fairly trivial. Simply trying all permutations, albeit resulting in an algorithm of complexity $\mathcal{O}(n!)$ (as there are $9 \cdot 8 \cdot ... \cdot 1$ permutations), can be done within seconds. Using Python, it might look something like this:

```
from itertools import permutations

line1 = ['Hi', 'Shi', 'He', 'Tsu', 'So', 'Yu', 'Ke', 'Ta', 'Wa']
line2 = ['Ta', 'Ke', 'Hi', 'Shi', 'He', 'Wa', 'Se', 'Tsu', 'So']

for p in permutations(range(1,10)):
    line1p = list(line1)
    line2p = list(line2)
    for i, a, b in zip(p, line1, line2):
        line1p[i - 1], line2p[i - 1] = a, b
    if line1p[1:9] == line2p[0:8] or line1p[0:8] == line2p[1:9]:
        print p
```

Running this algorithm presents us with two permutations, `5-8-4-7-3-1-9-6-2` and `5-2-6-3-7-9-1-4-8`, exactly matching the two cases described earlier – one where the `Yu` symbol is mapped to `1`, and one where it is mapped to `9`. Note that for this to work, we first had to observe two lines of ciphertext that contained nearly identical sets of symbols.

5.4 Finding the permutation manually

It is clear that simply verifying all permutations in a brute-force manner was an infeasible approach back in 1935. It is that unlikely Nuboer found the permutation by guessing either, as he would have to try nearly a hundred thousand permutations before this would become likely. He must have used a more structured approach.

Let $\mathcal{A} = (\alpha_1, \alpha_2, ..., \alpha_9)$ and $\mathcal{B} = (\beta_1, \beta_2, ..., \beta_9)$ be 9-tuples representing the lines of ciphertext, and $A = (a_1, a_2, ..., a_9)$ and $B = (b_1, b_2, ..., b_9)$ 9-tuples representing the respective plaintexts, where each α_i, β_i a_i and b_i represent individual Kana symbols. Eight out of nine symbols occur in both $\mathcal{A}$ and $\mathcal{B}$. Define a symbol to be *unique* when it only occurs in $\mathcal{A}$ or $\mathcal{B}$, but not in both. Nuboer distinguishes between the cases where the unique symbol in $\mathcal{A}$ is permuted to the front or the end of the plaintext and arbitrarily chooses the latter case. The other case can be handled analogously.

Let $\pi = (\pi_1, \pi_2, ..., \pi_9)$ be a permutation. We can write this permutation interchangeably as a bijective function that maps positions in the ciphertext to positions in the plaintext $\pi : n \mapsto \pi_n$. Thus $\alpha_i = a_{\pi(i)}$ and $\beta_i = b_{\pi(i)}$.

Observing the Kana symbols above, α_6 is not contained in $\mathcal{B}$ and β_7 is not contained in $\mathcal{A}$. Because of this uniqueness of α_6 and β_7, these symbols can only

be in position 1 and 9. In any other case, the sequence of 8 identical symbols would be broken. Since α_6 and β_7 are in different positions, they cannot both be mapped to position 9 – the permutation is bijective. From this we can conclude that, since α_6 was originally in the last position (by Nuboer's assumption), β_7 was originally in the first position. Thus $\pi_6 = 9$ and $\pi_7 = 1$.

From the above observation, it follows that the symbols in A are shifted one position to the left with respect to their occurrences in B. Thus $a_i = b_{i+1}$ for $1 \leq i \leq 8$ (as a_9 is the unique symbol `Yu`).

Now the repetition of symbols in $\mathcal{A}$ and $\mathcal{B}$ is important. We see that α_1 is `Hi`, and β_3 is `Hi` as well. Thus $\alpha_1 = \beta_3$, and from $\alpha_i = a_{\pi(i)}$ and $\beta = b_{\pi(i)}$ it now follows that $a_{\pi(1)} = b_{\pi(3)}$. From $a_i = b_{i+1}$, we then see that $\pi_3 = \pi_1 + 1$.

Applying this course of action to the entire tuples $\mathcal{A}$ and $\mathcal{B}$, we find the relations as listed in table 5.1 below (as `Yu` only occurs in $\mathcal{A}$, there is no matching position in $\mathcal{B}$). The `Hi` column matches the example above.

	`Hi`	`Shi`	`He`	`Tsu`	`So`	`Yu`	`Ke`	`Ta`	`Wa`
$\mathcal{A}$	1	2	3	4	5	6	7	8	9
$\mathcal{B}$	3	4	5	8	9	n.a.	2	1	6

Table 5.1: Symbol positions in ciphertext

Earlier we have seen that $\pi_6 = 9$. As the table shows that $\pi_6 = \pi_9 + 1$ (see the `Wa` column), we know that $\pi_9 = 8$. Continuing along the table, we now see that $\pi_9 = \pi_5 + 1$, thus $\pi_5 = 7$, etcetera. Now the entire permutation π can be deduced:

$$\pi = (5, 2, 6, 3, 7, 9, 1, 4, 8)$$

5.5 The '.' code book

After discovering the permutation that lay at the base of the '.' code, the cryptoanalysts of Department 1 were able to decipher an immense mass of material that had been intercepted over the past months. This allowed for the reconstruction of a significant part of the code book (Nuboer, 1981).

For the '.' code, the Japanese used the Romaji alphabet in lexicographical order. The Kana character set was used with omission of the `N` and `Nu` symbols. The Kana sequence is as follows: `A-E-Ha-He-Hi-Ho-Fu-I-Ka-Ke-Ki-Ko-Ku-Ma-Me-Mi-Mo-Mu-Na-Ne-Ni-No-O-Ra-Re-Ri-Ro-Ru-Sa-Se-Shi-So-Su-Ta-Te-Chi-To-Tsu-U-Wa-Wi-Ya-Yo-Yu`.

The '.' code is formed by groups of four Kana symbols, of which the fourth symbol is a control symbol. The control symbol was found by adding the numerical value of each Kana symbol (`A` = 1, `E` = 2, etcetera), subtracting one and converting the value back to a Kana symbol (modulo 44). Analogous to the control symbol as used in the Ni-code, this checksum system allowed for the reconstruction of the group in case one of the symbols was lost (Nuboer, 1977c). A table of the Kana symbols and their numerical values is included in appendix D.

Let us look at an example, to illustrate this checksum property. Observe the code group `So Me Mi`. Converted to numerals, these symbols have the value 32, 15 and 16. Their sum is 63. After subtracting 1, 62 remains. This is equal to 18 modulo 44, so the checksum symbol is `Mu`. Thus, the complete code group is `So Me Mi (Mu)` (which respresents the Kamoi ship).

As each code group is effectively defined by three of its symbols, the code is significantly larger than the Ni-code: there is room for roughly 85.000 (44^3) groups. To use this space effectively, the code contains several table structures. A large part of the code has been reserved for dates, timestamps, number and character groups, as well as individual characters. This encompassed the groups from `A A A (E)` to `Se Yu Yu (Sa)`. The rest of the book was reserved for various common words and phrases .

The codes for ships start with `So`. Light cruisers are indicated with `So Ku` and codes for heavy cruisers start with `So Ko`. `So Ra`, `So Ri` and `So Ro` indicate destroyers, and `So Se` is used for submarines. Coincidentally, Lieutenant Muller, stationed in Tokyo, had sent Nuboer a book with photographs of Japanese marine vessels (see section 2.4.2). The order soon turned out to match that of the ships listed in the `So`-table, and showed that the ships were ordered according to the date of their commissioning (Nuboer, 1977c).

The code also contains a table of authorities, starting with `Shi`, and geographical names, starting with `To` and `Tsu`. The geographical table is structured according to the actual geographical locations of the sites – starting from Tokyo, traversing around the Japanese islands and southbound along the coast of Eastern Asia, through the Dutch Indies towards the mandated territories (Nuboer, 1977c).

As more of the '.'-telegrams were decrypted, hundreds of other code groups were uncovered, including groups signifying numbers, dates and timestamps. For a tabular overview of a part of the code groups, see appendix E.

6 The reciphered '.' code

In August 1935, all '.' code telegrams suddenly became completely undecipherable. When applying the permutation as shown before, the resulting messages would contain unknown code groups and combinations of known code groups that did not form a coherent message. However, based on frequency analysis, Schalkwyk and Nuboer concluded that it concerned yet another transposition of the same base code.

6.1 A toy example, for the readers' benefit

The following subsection is not in line with the chronological decryption process that will be described in the rest of this section, but is perhaps a valuable shortcut to a basic understanding of the reciphered '.' code cryptosystem. This example will be provided without historical context or notes illustrating how or why the system was unravelled. To be better able to understand how the system was broken, it is convenient to be familiar with the way the system was used for encryption and decryption.

6.1.1 Inventory

The cryptosystem we will be exploring uses a combination of two keys to encrypt and decrypt: a permutation and a grid figure. For our example, we will use the permutation `2-5-1-3-4` and the five by five grid as shown in table 6.1 below. The grid contains three types of fields: (empty) fields, blanked out fields and fields with negation (`N`) symbols. For convenience, we can combine the two keys by writing the permutation below the figure, effectively labelling the columns.

		N		
	N			
2	5	1	3	4

Table 6.1: The key figure

6.1.2 Encryption

We will now use the figure and permutation as defined above to encrypt the plaintext "The cake is a lie". We begin by filling the plaintext into the empty fields of the figure, from left to right and top to bottom. In fields with a negation symbol, we fill in any arbitrary symbol – for this example, we choose `X` and `Y`. See the figure with the plaintext included in table 6.2, below.

	T	X	H	
E	Y		C	A
	K	E		
I			S	A
L	I			E
2	5	1	3	4

Table 6.2: The filled figure

We now read off the columns in the order defined by the permutation. As there is a 1 underneath the third column, we start with that one. We then continue to the first column, the fourth, the fifth and finally the second. Reading off the columns from top to bottom in this order, we find the strings `XE`, `EIL`, `HCS`, `AAE`, `TYKI`. We concatenate these to form the ciphertext: `XEEILHCSAAETYKI`.

6.1.3 Decryption

Decryption is quite literally the inverse procedure of encryption. We take the same key figure as shown in table 6.1, and we fill the ciphertext into the figure vertically. Start with the column labelled '1', then proceed to fill the remainder into the column labelled '2', etcetera. This time we do fill the fields with the negation symbols. We should end up with the same filled figure as we have seen earlier, in table 6.2. If we have done it right, the symbols `X` and `Y` should end up exactly in the negated fields – after all, they were not part of the original plaintext.

Now we can simply read the plaintext from the figure starting at the top-left and reading down along rows horizontally, ignoring the symbols in the negated fields. Doing this, we find the strings `TH`, `ECA`, `KE`, `ISA` and `LIE`. Concatenating these strings gives us our original plaintext: `THECAKEISALIE`.

6.2 Another breakthrough

In early September 1935, the Dutch intercepted two nearly identical '.' code telegrams, transmitted by a ship near Formosa that had apparently made an error and sought to correct it. From this message, Nuboer was able to conclude that the recipherment most likely consisted of ten or eleven columns that were transcribed vertically (Nuboer, 1977g). As Nuboer was otherwise occupied in Batavia at the time, he would not be able to further investigate the code until he arrived in Bandung in November. As luck would have it, the interception station had received an interesting new series of telegrams just days earlier, again originating from a ship off the coast of Formosa. It transmitted a sequence of telegrams at thirty minute intervals, showing many repetitions (Nuboer, 1977i, 1977d).

These telegrams are included fully in appendix F.

6.3 Exploiting repetition

As Nuboer and Schalkwyk had correctly established, the recipherment was a transposition cipher. Specifically, this meant that strings of symbols of equal length and position would be transpositioned in an identical way. After sorting the telegrams by length and similarity, and writing them down in a grid on one line each, Nuboer noticed an interesting pattern. By connecting repeated symbols with a red pencil line, a pattern of vertical lines emerged that seemed to make jumps to the right (Nuboer, 1977d). As the earlier telegrams had aroused suspicions that the columns were transcribed vertically, it was justifiable to assume that the groups that emerged were columns in a figure. Wherever a jump occurred, a column ended.

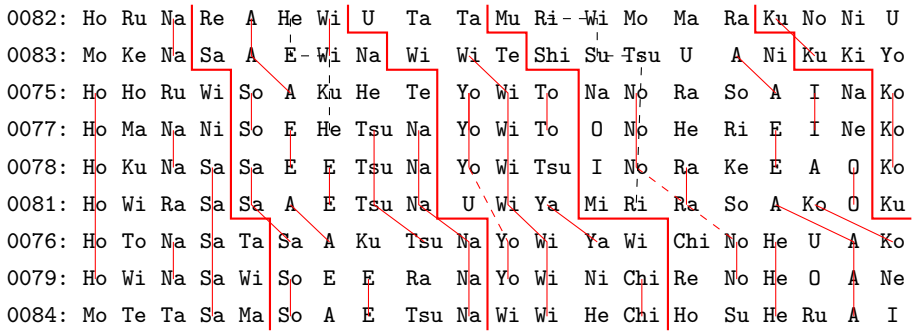

Table 6.3: Connecting identical symbols. Completed in appendix G

The table above (table 6.3) contains the first twenty symbols of the telegrams, along with annotations showing the columns. Thin lines indicate identical symbols, while thick lines indicate where columns end. A more complete annotated version is included in appendix G. Some more obscure connections, albeit reaffirming the pattern, have been omitted for the sake of clarity. This includes connecting the `He` in telegram `0082` on position six to the `He` on position seven in telegram `0077`, and the `Ri` symbols in telegram `0082` and `0081`. These two examples have been indicated with dashed grey lines in the table above.

For the sake of completeness, it is perhaps worth mentioning that Nuboer's assumption that the ciphertexts were build up of transcribed columns was not unanimously supported. Schalwyk was approaching the issue from a different vantage point, assuming a more arbitrary transposition that combined the Kana symbols to known code groups. He found a transposition for the first forty symbols of the telegrams that formed the symbols into a remarkable amount of valid code groups. Schalkwyk was so convinced of the correctness of this discovery (and Nuboer's columnar theory had been stuck in a rut) that Nuboer accepted it for the time being (Nuboer, 1977g). The pair ended up working on it fruitlessly for months. The staff of Room 14 had not yet gathered sufficient insight in the code to realise that the decrypted texts were not at all well-formed code.

6.4 Permuting columns once more

As mentioned above, Nuboer believed that the groups of symbols represented columns. As the telegrams grew larger, the lines that separated the columns made jumps. These jumps indicated to which columns an extra symbol was appended. It soon appeared that the columns did not grow in the order in which they were included in the text, but according to some permutation (Nuboer, 1977d).

Nuboer marked the symbols in positions that were newly filled as the telegrams grew larger. Such a symbol is found by comparing the length of a column to the length of the same column in one of the slightly shorter telegrams – preferably one group shorter. For the first twenty characters, these are marked in table 6.4. Again the entire marked telegrams can be found in appendix G.

0082:	Ho	Ru	Na	Re	A	He	Wi	U	Ta	Ta	Mu	Ri	Wi	Mo	Ma	Ra	Ku	No	Ni	U
0083:	Mo	Ke	Na	Sa	A	E	Wi	Na	Wi	Wi	Te	Shi	Su	Tsu	U	A	Ni	Ku	Ki	Yo
0075:	Ho	Ho	Ru	Wi	So	A	Ku	He	Te	Yo	Wi	To	Na	No	Ra	So	A	I	Na	Ko
0077:	Ho	Ma	Na	Ni	So	E	He	Tsu	Na	Yo	Wi	To	O	No	He	Ri	E	I	Ne	Ko
0078:	Ho	Ku	Na	Sa	Sa	E	E	Tsu	Na	Yo	Wi	Tsu	I	No	Ra	Ke	E	A	O	Ko
0081:	Ho	Wi	Ra	Sa	Sa	A	E	Tsu	Na	U	Wi	Ya	Mi	Ri	Ra	So	A	Ko	O	Ku
0076:	Ho	To	Na	Sa	Ta	Sa	A	Ku	Tsu	Na	Yo	Wi	Ya	Wi	Chi	No	He	U	A	Ko
0079:	Ho	Wi	Na	Sa	Wi	So	E	E	Ra	Na	Yo	Wi	Ni	Chi	Re	No	He	O	A	Ne
0084:	Mo	Te	Ta	Sa	Ma	So	A	E	Tsu	Na	Wi	Wi	He	Chi	Ho	Su	He	Ru	A	I

Table 6.4: Marking appended symbols. Completed in appendix G

By marking the symbols, Nuboer saw that in the transition from 50 to 54 symbols (i.e. the difference between telegram `0082` and `0083`), the symbols `Na`, `He`, `Wi`, `Na` were added to the columns 2, 6, 7 and 9 respectively. At the end of the same columns in the slightly larger telegram `0075`, Nuboer found the symbols `Te`, `Ka`, `Ni` and `Ne` (marked with dashes in appendix G). If Nuboer's suspicions were correct, these symbols were to form a known code group. Indeed, arranging the columns in the order `6-7-2-9` formed two known code groups: `He Wi Na (Na)` and `Ka Ni Te (Ne)`.

Similarly, in the transition from 54 to 58 symbols, the columns 1, 4, 5 and 11 were extended. The ordering `5-1-11-4` revealed known code groups as well. This involved the groups `He Wi Na (Na)`, `Ka Ni Te (Ne)`, `Ma Sa Re (O)` and `Ma Sa No (Ne)`.

In the transition from 58 to 62 symbols, the columns 6, 7, 8 and 10 were extended. Two remarkable things happened here. The appended symbols were identical in all telegrams, again forming the group `He Wi Na (Na)` and the ordering `10-8-6-7`. This ordering overlaps with the one found in the first transition, combining to an ordering of `10-8-6-7-2-9`.

The fourth extension from 62 to 66 symbols completed the puzzle. The columns 1, 3, 5 and 9 were extended and, after permuting them in the ordering `9-5-1-3`, showed the groups `Ru Ra Ta (Wi)` and `Ro Ku Wi (Chi)`. Merging the `9-5-1-3`

and 5-1-11-4 sequences to 9-5-1-3-11-4 can be done by naive trial and error based on the formed groups, as there are only several combinations. This brings together the earlier found sequences to form a complete permutation: 10-8-6-7-2-9-5-1-3-11-4.

It is worth noting that finding these orderings can be made much easier by taking the checksum property into account (as discussed in section 5.5). Let us have a look at the symbols Te, Ka, Ni and Ne. Translating these into numbers according to the Kana alphabet (see appendix D) gives us 35, 9, 21 and 20 respectively. Recall that computing the checksum is done by summing the numerical values of three symbols and reducing the result by one, modulo 44. One can now easily observe that Ne is the only possible valid checksum symbol:

$$\begin{aligned} 20 + 9 + 21 - 1 &\neq 35 \pmod{44} \\ 35 + 20 + 21 - 1 &\neq 9 \pmod{44} \\ 35 + 9 + 20 - 1 &\neq 21 \pmod{44} \\ 35 + 9 + 21 - 1 &= 20 \pmod{44} \end{aligned}$$

Now only six code groups need to be considered, rather than twenty-four. It should be noted that this result is purely academic, as Nuboer had long since memorised the common code groups by studying the earlier version of the '.' code. He would have solved the anagram of Ka Ni Te (Ne) by recognition rather than computation.

6.5 Figuring it out

Nuboer then provides the example of telegram 0076 after permuting the columns according to the newly found permutation (Nuboer, 1977d). This example is included in table 6.5. The columns show a remarkable difference length, and Nuboer notes there were only few in-tact code groups to be observed. This indicates that the underlying figure with which the telegrams were transcribed contains many blank fields, causing much obscurity.

10	8	6	7	2	9	5	1	3	11	4
Ru	Mu	Mi	Ne	Sa	I	Ko	Ho	Yo	Re	Chi
Yo	Ko	Ri	So	A	Ne	Ku	To	Wi	Mi	No
Ro	Na	A	He	Ku	Yu	To	Na	Ya	Ku	He
Ma	Wa	No	Wi	Tsu	Ha	A	Sa	Wi	Tsu	U
Ha	Ke	No	Na	Na	No	Wa	Ta		Ki	A
He	Wi	He			Na	Ma			A	Ko
		No			Ru	Ra			No	Ne

Table 6.5: Telegram 0076 in re-ordered columns

Unfortunately, there seems to be little information available as to how the encryption figure was then found. Nuboer notes that "through comparison with other telegrams" the original text of the 0076 telegram was put together, and

the figure was reconstructed. This figure is included in table 6.6 below. `N`-symbols (negation symbols) indicate fields where apparently random symbols were inserted. This, combined with the obscurity created by the blanked out fields, made the code groups especially hard to isolate.

However, in one of Nuboer's manuscripts the above table is heavily annotated. This sheds a slight ray of light on the process of constructing the figure. It seems that by attempting to align common code groups, Nuboer was able to draw conclusions about the relative positions of the symbols. An example is included in the table above, where the group `Ne Mi Sa (Ne)` is marked. Notice that the group starts with `Ne`, but it is in a column after `Mi`. This means that `Ne` must have appeared at least one row above the other symbols, and there cannot have been other symbols in between. Indeed, this is confirmed by applying the (slightly altered – see below) permutation and filling in the group in table 6.6. The `Ne` symbol is in the first row, while the rest of the group is spread over lower rows.

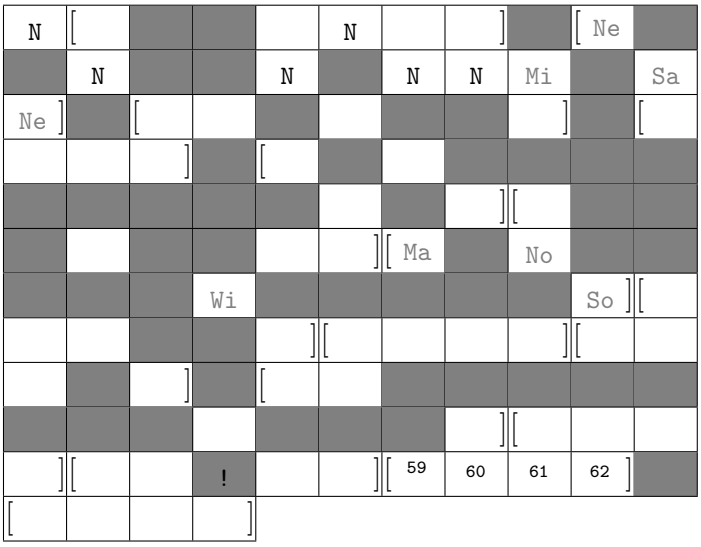

Table 6.6: Encryption figure (November 1935)

The groups were often much less straightforward to observe, as is shown by the `Ma No Wi (So)` group that is marked in table 6.5 with a dashed line, and included in table 6.6. The difference in column length caused vertical offsets, and the many blank fields tore the groups apart horizontally. This makes the fact that Nuboer managed to crack this tremendous puzzle even more astonishing.

Note: to avoid overloading the terms *group* and *column*, from here on the sequences of symbols identified in a telegram as a column (as separated by red lines in table 6.3 and appendix G) will be called an *array*. This is done to avoid confusion with the columns in the figure described below, and the code groups used as words in the '.' code (i.e. four Kana symbols).

The permutation found above turned out to be slightly off. One might notice that the columns of table 6.5 do not fit in the columns of table 6.6. Although Nuboer does this implicitly in his notes, it requires a little explanation to get

correctly. Recall the permutation `10-8-6-7-2-9-5-1-3-11-4`. Rather than filling the tenth array of symbols into the first column of the figure, one must start off with the seventh column. The eighth array goes into the eighth column, the sixth array into the ninth, the seventh into the tenth, the second into the eleventh, after which we loop around to place the ninth array into the first column, etcetera. This is a consequence of the fact that the code group that was used to identify the first four arrays of the sequence (`10-8-6-7`) encompassed fields 59 to 62, which are found in column seven to ten (see table 6.6). This effectively changes the permutation to `9-5-1-3-11-4-10-8-6-7-2`. Applying the permutation becomes more clear in section 6.6 or the toy example in section 6.1, where we will look at an example decryption using the figure above.

The figure in table 6.6 has a particularly large number of available fields in the last few rows – the rows where the symbols are added that caused the telegrams to be of distinct lengths. More importantly, the last few rows contain relatively large sequences of fields that are left uninterrupted by any blanked out fields. This, above all, made the analysis that was described in section 6.4 possible. Had there been more blanked out fields, Nuboer would most likely have required more telegrams to find the initial permutation. Recall that there was a slight conflict when merging two parts of the permutation, `5-1-11-4` and `9-5-1-3`. We can now link this conflict to the blank field in column four, as this where the third array was to be filled in. The sequence `5-1-11-4` was formed on the penultimate row (fields 55 to 58), skipping the third array in column four, as this field is blanked out (marked in table 6.6 with an **!**). The sequence `9-5-1-3` does include the third array, as it is a consequence of symbols in the last row, where the field in column four is not blanked out. This shows how just one blanked out field can cause a severe disruption. One might imagine what more blanks would have done for the difficulty of finding the permutation.

6.6 Exploring an example

The above analysis is quite theoretical. To get a better understanding of the actual workings of the system, it helps to work through a real decryption. In the sections below, this decryption process is illustrated. Note that a smaller example of both decryption and encryption was already included in section 6.1, but several complications were dropped that will now be included.

Let us use telegram `008209133`, as shown below – any of the other telegrams will behave analogous, but this is one of the shorter telegrams. This prevents it from filling up the entire figure, adding a step to the decryption process. The other telegrams are included fully in appendix F. We assume knowledge of the permutation that was figured out earlier (`9-5-1-3-11-4-10-8-6-7-2`, after correcting for the starting position), as well as the figure (see table 6.6).

```
008209133 Ho Ru Na Re A He Wi U Ta Ta Mu Ri Wi Mo Ma Ra Ku No Ni U
Na Mu Ri E I Ne No Ya He A Mi So No Ma To Ro Fu Wi Na Ru Mo No A Sa
Hi I Ma To Na Ru
```

6.6.1 Determining the lengths

The first thing we need to do is separate the telegram into arrays that can later be fitted into the figure. In section 6.3, this was done by comparing it to other telegrams and drawing a red line along the pattern. However, rather than relying on other similar telegrams, we will now use the figure. We simply count the length of the telegram – 50 symbols – and fill as many fields in the figure (see below). Note that we also count the fields that contain a negation symbol (N), as the symbols these fields produced during encryption have not been filtered from the ciphertext yet.

N[1]	2			3	N[4]	5	6		7	
	N[8]			N[9]		N[10]	N[11]	12		13
14		15	16		17			18		19
20	21	22		23		24				
					25		26	27		
	28			29	30	31		32		
			33						34	35
36	37			38	39	40	41	42	43	44
45		46		47	48					
			49				50			
9	5	1	3	11	4	10	8	6	7	2

Table 6.7: Determining array lengths

By writing the permutation along the bottom of the figure, we can now easily find the lengths of the arrays by counting the filled fields in the associated columns. Table 6.8 shows these totals, sorted according to the array numbers.

Array No.	1	2	3	4	5	6	7	8	9	10	11
Length	3	4	3	6	5	5	3	5	5	5	6

Table 6.8: Determining array lengths

Now that we know the array lengths, we can split the telegram accordingly. The ending of each array is marked with a slash (/) character. Note that this is the same division as found in table 6.3, but this time the figure was used to find it, rather than having to depend on similarities with other telegrams.

```
008209133 Ho Ru Na / Re A He Wi / U Ta Ta / Mu Ri Wi Mo Ma Ra / Ku
No Ni U Na / Mu Ri E I Ne / No Ya He / A Mi So No Ma / To Ro Fu Wi
Na / Ru Mo No A Sa / Hi I Ma To Na Ru
```

6.6.2 Filling the figure

The arrays of symbols can now be filled into the figure – keep in mind that we are using the permutation `9-5-1-3-11-4-10-8-6-7-2`. This indicates that the ninth array is to be copied into the first column, the fifth array into the second column, the tenth array into the third column, etcetera. See table 6.9 for the fully filled figure, below. Symbols that were placed in a field with a negation symbol are greyed out rather than omitted entirely in order to better illustrate which arrays of symbols were placed in which columns.

To	[Ku			Hi	Mu	Ru	A]		[No	
	No			I		Mo	Mi	Mu		Re
Ro]		[Ho	U		Ri			Ri]		[A
Fu	Ni	Ru]		[Ma		No				
					Wi		So]	[E		
	U			To	Mo]	[A		I		
			Ta						Ya]	[He
Wi	Na			Na]	[Ma	Sa	No	Ne]	[He	Wi
Na		Na]		[Ru	Ra					
			Ta				Ma]			
9	5	1	3	11	4	10	8	6	7	2

Table 6.9: Filling the figure

6.6.3 Reading the plaintext

After filling in the figure, the plaintext appears along the rows. It is now simply a matter of reading the symbols in groups of four. One must be careful to ignore the greyed out symbols, as these were random symbols added during the encryption process rather than part of the original plaintext. The telegram now reads:

```
008209133 Ku Hi Ru (A) - No Mu Re (Ro) - Ho U Ri (Ri) - A Fu Ni (Ru)
- Ma No Wi (So) - E U To (Mo) - A I Ta (Ya) - He Wi Na (Na) - Ma Sa
No (Ne) - He Wi Na (Na) - Ru Ra Ta (Ma)
```

To further increase the confidence in the validity of this plaintext, one can verify the checksum property for each of the code groups (as described in section 5.5, using appendix D).

6.7 Generically finding permutations

After finding the figure and permutation in early 1936, Nuboer noticed that they were not able to decrypt all messages that had been accumulated over the months. The figure only worked for the odd months in 1935 (since August, as that was when the system was adopted), and the permutation only let to well-formed code groups once every fifteen days. For telegrams sent after January 1st, 1936, the figure stopped working altogether. Apparently the Japanese were using two different grids for the even and the odd months, and updated the figures altogether every six months. Additionally, they replaced the fifteen different column permutations[1] (Nuboer, 1981). All these changes let to great difficulty in decrypting. However, while finding an altogether new figure was still a tedious process, finding the new permutation using an available figure was a feat Nuboer soon mastered (Nuboer, 1977i).

To find a new permutation, Nuboer would require two sufficiently long telegrams that were encrypted using the same permutation – either sent on the same day, or fifteen days apart. Both telegrams would need to have at least enough symbols to completely fill one figure. This guarantees that during encryption, the same transposition is applied to the first parts of both telegrams.

In his notes, Nuboer presents a brief example of how two such telegrams allowed him to find the permutation that was used on July 1st (and thus on July 16th and July 31st) (Nuboer, 1977f). In this section, his example will be elaborated upon and used to illustrate the underlying algorithm.

6.7.1 Identifying groups

Let us first start by having a look at the available resources. First and foremost, this includes the relevant encryption figure (see table 6.10) as found in Nuboer's notes on the July telegrams (Nuboer, 1977e).

By counting the groups, we see that the figure has space for 84 symbols (21 groups of four), and contains 6 negation symbols. This means that the first 90 symbols of both telegrams have been transposed by a single instance of the figure, and we can crop any symbols after this limit. For reference purposes, the full telegrams are included in appendix H. The relevant prefixes are listed below in tables 6.11 and 6.12.

Earlier, we have established that both telegrams have been transposed in the exact same way. This means that the symbols that made up the first code group of telegram `301` were transposed to the same positions in the ciphertext as the symbols of the first code group of telegram `304`. This allows us to search for a combination of symbols that form a known code group in one telegram, and see if the symbols at the same positions in the other telegram form a valid code group as well.

[1] From Nuboer's manuscript notes, it appears that the same initial permutation was re-used after every two months (i.e. on the 1st of July and the 1st of November), but the entire series was replaced every six months along with the figures.

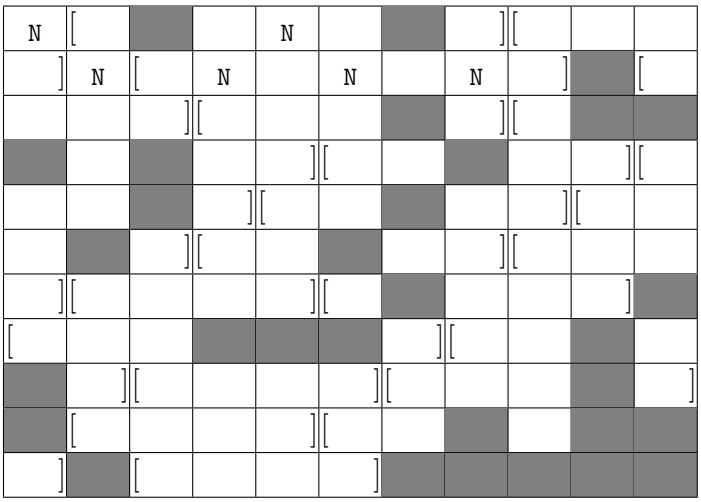

Table 6.10: Encryption figure (July 1936)

Ki	A	Ni	Hi	Ho	A	Ta	Na	Mi	So	Ma	Ke	Hi	Se	Hi	Re	Se	Ki
Mu	Shi	Ru	Ha	Wa	Yu	Se	Wa	Hi	Na	E	A	I	Ya	Ni	Mo	O	A
E	Ku	Me	Shi	Ma	No	Na	Tsu	E	Ya	A	Ru	So	Te	So	Sa	Na	O
Mi	Mi	Ru	A	Ya	Ra	Na	Ta	Na	Sa	Ru	Wi	Ho	Ni	Fu	Ro	Fu	Yu
Me	He	He	Mi	Ro	Shi	Se	A	A	Ri	Ma	Sa	Ho	Yo	Mo	No	I	Mo

Table 6.11: Ninety symbol prefix of 301001071

Once again Nuboer's familiarity with the common code groups proves essential. He notes that trying Mi Ra A (Wa), Ki Ke A (Ni) and Mo Na Ta (Re) did not work out well. Although telegram 301 does contain those symbols, the symbols at the same positions in telegram 304 did not line up to form a code group. Then, when trying Mi I Wa (Na), Nuboer strikes gold (Nuboer, 1977f).

The symbols occur multiple times throughout the telegrams, and are marked with different dashed boxes in table 6.11. The corresponding symbols in telegram 304 are marked with identical boxes in table 6.12. Nuboer forms the group Mo Na Ta (Re) from these symbols – another common group. Based on this, we can now assume that telegram 301 contains the group Mi I Wa (Na), and

Ri	Ni	Me	Shi	Ra	Tsu	Ri	Me	Ri	Ke	Te	Ru	Ya	Ki	Ka	Ha	Te	Mi
Ki	Me	Mi	Tsu	Ta	Me	Me	Ha	Ko	Ki	Ha	Ne	Na	Tsu	Ko	E	I	I
Ko	Tsu	Tsu	Wa	To	U	Tsu	Re	Ho	Mu	Ha	Me	Ra	Ni	I	Ra	Yu	Ma
Mo	Mo	Yo	Ru	Ne	Me	Ke	U	Re	Ka	Yo	Yu	Sa	Mo	Ho	Se	Mo	Re
Ri	Re	Na	Ru	Ho	Ho	Mo	Ma	Ki	Ka	Ho	So	Wa	Se	Hi	Ka	Ro	Ne

Table 6.12: Ninety symbol prefix of 304001200

telegram 304 contains Mo Na Ta (Re) at the exact same position. The symbols included in these groups have been shaded grey. Notice that there are five marked symbols in each telegram – coincidentally, there are two pairs of Mo and Mi symbols. Only one of them contributed to the groups we have just identified, but it is still unclear which one.

6.7.2 Placing the groups

Now that we know that the Mi I Wa (Na) group is included, only the position of the group remains. As can be seen in table 6.10, there are twenty-one candidate positions. At this point either one of them might have been the Mi I Wa (Na) group we are trying to locate.

The telegram (table 6.11) reveals a bit more about the positioning, though. By counting the symbols, we see that the relevant Wa is at position 23, I is at 31, Mi is at 55 or 56 and Na is at 63. This means that Wa and I are only 8 symbols apart, and that Mi and Na are 7 or 8 symbols apart depending on which Mi symbol is part of the group (recall that there are still two candidates). Combining this with the fact that the figure is to be read off vertically, the position of the group is quite restricted.

For each candidate group, we test if the distance from the third to the second symbol is 8, and if the distance from the first and last symbol is 7 or 8. In table 6.13 we see an example of calculating the distance from the Wa to the I symbol, given that they are placed in the seventeenth group. Note that we count the empty fields as well as the fields with a negation symbol, as these all contained symbols that were transcribed to form the ciphertext.

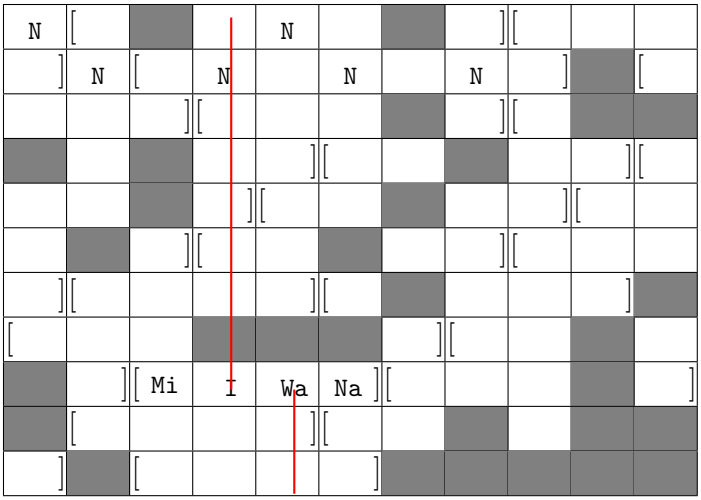

Table 6.13: The distance between Wa and I is 10, not 8

From the example above, we see that the group cannot have been in the seventeenth position. Similarly, many of the groups can be eliminated by checking

the distance from Wa to I and from Mi to Na. After this filter, however, there are still several candidates to choose from.

After systematically working our way through all the groups and checking the distance from Wa to I, only the seventh, eighth and nineteenth groups are valid candidates. All the other groups showed a distance unequal to 8. Note that for the groups with a smaller distance, adding another column in between the one containing the Wa and I symbols never works: the shortest column contains 5 fields, while the shortest possible distance from Wa to I already takes 6 fields (i.e. using the twelfth and twentieth groups). The sum of these distances, 11, is easily larger than the required 8 field distance.

Now that the position has been narrowed down to the seventh, eighth and nineteenth group, checking the distance between the Mi and Na symbols narrows it down further to only the seventh group. Here the distance is 7, while the eighth and nineteenth groups both result in a distance of 9 fields. Thus we can be certain that the Mi I Wa (Na) code group is the seventh group in the figure. As the distance from Mi to Na ended up being 7 rather than 8, we can now be certain that the Mi symbol at position 56 in the telegram (see table 6.11) was included in the code group rather than the one at position 55.

6.7.3 Deduction

In the previous section we have managed to place a code group in the figure. As the columns have been transcribed vertically to obtain the ciphertext, we can now fill in the entire columns using the symbols that appear in telegram 301001071 around the symbols of the located code group. This results in the figure as shown in table 6.14, below.

N	[			N	Na		]	[Shi	Ta	
]	N	[	N		O	A	N	Ru]		[
		]	[		Mi		]	[Ha		
				]	[Mi	I		Wa	Na]	[
			]	[	Ru			Yu]	[Sa	
		]	[			Ya	]	[Se	Ru	
]	[			]	[A			Wa	Wi]	
[						Ni]	[	Hi		
	]	[			Ya]	[Mo		Na		]
	[			]	[Ra	O		E		
]		[			Na]					

Table 6.14: Filling columns six, seven, nine and ten

For the next step it is helpful to have a clear overview of the parts of the telegram that have already been used. The four columns inserted in the figure above have been coloured grey in the copy of the 301 telegram, in table 6.15, below.

Ki	A	Ni	Hi	Ho	A	Ta	Na	Mi	So	Ma	Ke	Hi	Se	Hi	Re	Se	Ki
Mu	[Shi	Ru	Ha	Wa	Yu	Se	Wa	Hi	Na	E]	[A	I	Ya	Ni	Mo	O]	A
E	Ku	Me	Shi	Ma	No	Na	Tsu	E	Ya	A	Ru	So	Te	So	Sa	[Na	O
Mi	Mi	Ru	A	Ya	Ra	Na]	[Ta	Na	Sa	Ru	Wi]	Ho	Ni	Fu	Ro	Fu	Yu
Me	He	He	Mi	Ro	Shi	Se	A	A	Ri	Ma	Sa	Ho	Yo	Mo	No	I	Mo

Table 6.15: Ninety symbol prefix of 301001071

By deduction, we will now attempt to fill the rest of the figure. The next steps might appear vaguely similar to solving a sudoku puzzle.

When looking at the figure above, we observe two code groups that only miss one symbol: the fourteenth and twentieth group. One can find these symbols using the checksum property (the sum of the first three symbols minus one is equal to the last symbol modulo 44 – see section 5.5). The fourteenth group is `A ? Wa (Wi)`. Converting this to numerals (see appendix D) gives us the equation $1 + x + 40 - 1 \equiv 41 \pmod{44}$. Thus the missing value is 1, indicating the Kana symbol `A`. This forms the group `A A Wa (Wi)`. Similarly we find that the twentieth group (`Ra O E ?`) misses the `He` symbol, forming `Ra O E (He)`.

This shows that the last symbol of the first column is `He`, but there are two `He` symbols included in the telegram, in position 74 and 75. However, assuming that the symbol in position 75 corresponds to the last field of column one orphans the `Ho` symbol in position 67. This is impossible, as the columns should adjoin seamlessly in the ciphertext telegram. This implies that the first `He` symbol should be selected, and we can fill in the first column.

A similar strategy can be applied to the `A` symbol inserted in the fourteenth group. This scenario is made slightly more complicated by the fact that there are still six candidate `A` symbols available. However, the ones in position 2 and 36 do not have a large enough prefix of available symbols to fit the eighth column. The `A` symbols in position 47 and 81 are also unsuitable, as they would orphan a small amount of symbols in front or behind them. This leaves only the symbols in position 6 and 80. Electing the symbol in position 6 would lead to a slightly less obvious problem. The required array of symbols would be followed by eleven available symbols, which, given the lengths of the remaining columns, is too large to fit one column and too small to fit two. And so `A` symbol in position 80 is the only remaining candidate.

Updating the figure and telegram results in the situation as shown in tables 6.16 and 6.17.

Ki	A	Ni	Hi	Ho	A	Ta	Na	Mi	So	Ma	Ke	Hi	Se	Hi	Re	Se	Ki
Mu	[Shi	Ru	Ha	Wa	Yu	Se	Wa	Hi	Na	E]	[A	I	Ya	Ni	Mo	O]	A
E	Ku	Me	Shi	Ma	No	Na	Tsu	E	Ya	A	Ru	So	Te	So	Sa	[Na	O
Mi	Mi	Ru	A	Ya	Ra	Na]	[Ta	Na	Sa	Ru	Wi]	[Ho	Ni	Fu	Ro	Fu	Yu
Me	He]	[He	Mi	Ro	Shi	Se	A	A	Ri]	Ma	Sa	Ho	Yo	Mo	No	I	Mo

Table 6.16: Ninety symbol prefix of 301001071

Ho	[			N	Na		He]	[Shi	Ta	
Ni]	N	[	N		O	A	Mi	Ru]		[
Fu		]	[		Mi		Ro]	[Ha		
				]	[Mi	I		Wa	Na]	[
Ro			]	[	Ru		Shi	Yu]	[Sa	
Fu		]	[			Ya	Se]	[Se	Ru	
Yu]	[			]	[A		A	Wa	Wi]	
[Me						Ni]	[A	Hi		
	]	[			Ya]	[Mo	Ri	Na		]
	[			]	[Ra	O		E		
He]		[			Na]					

Table 6.17: Filling the first and eighth columns

The next step is to proceed with another iteration of the same technique. Now the twelfth group misses just one symbol: `Se Ru ? (Yu)`. The missing symbol is `Shi`, as $30+28+31-1 \equiv 44 \pmod{44}$. This allows us to fill in the last column immediately, as there is only one available `Shi` symbol left in the telegram.

The ninth group is the exact same code group – it is `? Ru Shi (Yu)`, and by comparing it to the twelfth group, we see that the missing symbol has to be `Se`. There are two available `Se` symbols left in the telegram, but the symbol in position 17 does not have sufficient postfix space. Thus the fifth column can also be filled in.

This just leaves the second, third and fourth column unsolved. Respectively the sixteenth, third and eighth code groups provide handles to solve these in an identical fashion, by validating checksums. None of these lead to any new situations or complications and further details will be omitted for the sake of brevity. It is worth noting that in this specific case the columns can be found even faster, as the previous iteration left three non-neighbouring groups of unique lengths in the ciphertext. These can be filled into the correct columns immediately, as each only fits in one of the columns. The final figure and telegram now look as shown in table 6.18 and table , below.

[Ki	A	Ni	Hi	Ho	A	Ta	Na	Mi]	[So	Ma	Ke	Hi	Se	Hi	Re	Se	Ki
Mu]	[Shi	Ru	Ha	Wa	Yu	Se	Wa	Hi	Na	E]	[A	I	Ya	Ni	Mo	O]	[A
E	Ku	Me	Shi	Ma	No]	[Na	Tsu	E	Ya	A	Ru	So	Te	So	Sa]	[Na	O
Mi	Mi	Ru	A	Ya	Ra	Na]	[Ta	Na	Sa	Ru	Wi]	[Ho	Ni	Fu	Ro	Fu	Yu
Me	He]	[He	Mi	Ro	Shi	Se	A	A	Ri]	[Ma	Sa	Ho	Yo	Mo	No	I	Mo]

Table 6.18: Ninety symbol prefix of `301001071`

Ho	[Ki		Na	So	Na		He]	[Shi	Ta	A
Ni]	A	[Ma	Tsu	Ma	O	A	Mi	Ru]		[E
Fu	Ni	Sa]	[E	Ke	Mi		Ro]	[Ha		
	Hi		Ya	Hi]	[Mi	I		Wa	Na]	[Ku
Ro	Ho		A]	[Se	Ru		Shi	Yu]	[Sa	Me
Fu		Ho]	[Ru	Hi		Ya	Se]	[Se	Ru	Shi
Yu]	[A	Yo	So	Re]	[A		A	Wa	Wi]	
[Me	Ta	Mo				Ni]	[A	Hi		Ma
	Na]	[No	Te	Se	Ya]	[Mo	Ri	Na		No]
	[Mi	I	So	Ki]	[Ra	O		E		
He]		[Mo	Sa	Mu	Na]					

Table 6.19: Filling the first and eighth columns

6.7.4 Finding the permutation

Spelling out the permutation is now a matter of checking which array of symbols from the ciphertext telegram is transcribed along each column in the figure. It is easily verified that the permutation that can be used to decrypt this message is `9-1-11-6-2-7-4-10-3-8-5`. This indicates that the ninth array is to be placed in first column, the first array in the second column, etcetera. Note that it is important not to confuse this style of specifying a permutation with a format that Nuboer occasionally used in his notes, where the numerals specify for each array to which column it is mapped. This would result in the permutation `2-5-9-7-11-4-6-10-1-8-3`, specifying that the first array is placed in the second column, the second array in the fifth column, etcetera. There is only a minimal difference between the definitions, but the distinction is crucial in order to correctly interpret a specified permutation.

Now we have found the permutation that was used on July 1st. As the permutations were rotated every fifteen days (see section 6.7), the same permutation applies to July 16th and 31st.

6.8 Finding the other permutations

In section 6.5 we discussed how Nuboer managed to find new encryption figures, and in section 6.7 we have examined a generic method that can be used to find a permutation for a given encryption figure. The example that was used to illustrate this generic method left us with one of the fifteen permutations. However, by applying the method that Nuboer described, we should be able to find the other permutations. The approach is clearly divided in three subtasks, as shown in 6.7, dictating sequential procedures. In this section several aspects of a (Python) program based on this algorithmic approach will be illustrated. We will then discuss the results of attempting to use this program to find the other permutations.

The program is included in appendix I and is also accessible on GitHub at `https://github.com/joostrijneveld/blue-code-permutations`. For future reference, note that the GitHub repository will contain the latest version of the program. The repository also includes ciphertext telegrams that can be and have been used to reveal the permutations, as well as telegrams for which the permutation is unknown.

6.8.1 The input

Nuboer's archives include many original ciphertext intercepts. Unfortunately, many of these originate from before August 1936, when the '.' code was reciphered. The only collection of telegrams encrypted with the reciphered code that appear to be available are dated between the 1^{st} and the 9^{th} of July. Additionally, not all dates are as well-represented. Recall that the example in section 6.7 required at least two telegrams of sufficient length.

Fortunately, telegrams of 180 symbols or more encompass multiple figures, as each sequence of ninety symbols was encrypted using a new copy of the figure. This allows us to split large telegrams into groups of ninety symbols, effectively creating more ciphertext telegrams to work with.

The exact procedure to finding the encryption figure is still somewhat uncertain, as Nuboer's notes only provided hints at the methods he used to solve these intricate puzzles (see section 6.5). However, this becomes irrelevant when we take into account that all the available ciphertext was constructed using the July 1936 figure as depicted in table 6.10. Thus we can safely abstract from this part of the decryption algorithm without loss of information, and rely on the figure that Nuboer provided. As only the two figures that we have seen in the section above were included, even with the availability of ciphertext from other months we would still be unable to find any other permutations.

6.8.2 Recognising common groups

The example of section 6.7 starts off by searching the ciphertext telegram for code groups that Nuboer knew would occur with high frequency, and he had learned to recognise them while working with the code. As there is no algorithmic way to determine if a code group is common or not, and the meaning of

the code groups is largely unknown, we will have to rely on a different source. The pre-1936 telegrams provide a decent basis for a frequency count, as the plaintext of these telegrams was formed using the same code groups. Section 5 shows how these telegrams were decrypted using a single permutation as the key. Other than the ciphertext, Nuboer also conveniently provided several pages of decrypted telegrams from early 1935 (Nuboer, 1977b). To further aid the recognition of common groups, we can use the list of fifteen common code groups that Nuboer singles out in his notes (Nuboer, 1977c). Most of these groups coincide with the groups that are already found in the plaintext telegrams used for training the memory, but the list can be used to add extra weight and priority.

Python provides a very suitable structure to keep track of code group frequencies while reading through the plaintext telegrams: the `collections.Counter` data type. Adding n groups can be done in a time complexity of $\mathcal{O}(n)$, provided the groups are added using some sort of hashable[2] format (as this is necessary to be able to use them as dictionary keys). A convenient way to achieve this is by converting each group to a tuple of integers. This is preferable to working with strings, as later parts of the algorithm will force us to integers (e.g. to compute checksum symbols). For consistency, we might as well convert the Kana symbols to their numerical values (see appendix D) as soon as possible and work with these values throughout.

6.8.3 Precomputing column combinations

The figure contains eleven columns of varying length – that is, each column contains eleven fields, but a varying amount of these fields is blank. As there are only eleven columns, only a limited set of lengths are available. Thus, for a given piece of ciphertext it is non-trivial to see if there is a combination of columns that provide the exact length that is required to span it. When trying to validate if a position in the figure is valid for a given code group, these combinations are used quite intensively. As there are $2^11 = 2048$ different subsets of columns, working our way through this set every time it is required quickly gets out of hand. It is therefore useful and required to pre-compute a map that provides a relation between possible lengths and sequences of columns that can be used to construct it. See the `precompute_colcombs` function.

6.8.4 Finding a valid group

After having found a candidate group, an essential step in finding the permutation is positioning this group in the figure. By elimination, each of the groups has to be checked. This is done by seeing if there exists a column sequence that provides the relative distances between each of the symbols of the group. This is fairly trivial for the cases where the distance in the telegram is either smaller than or equal to the vertical distance between the symbols when there are no

[2] An object is *hashable* when there is a method available to compute a fixed-length *hash value* such that equal objects have equal hash values. This generally allows for dictionary keys to be computed in constant time rather than searched for in linear time, which makes checking for containment much more efficient.

other columns between the two symbols; the group can immediately be eliminated when the telegram distance is smaller. When the distance is larger than the distance provided by simply concatenating the two columns, different sets of columns have to be inspected to see if the required distance can be filled without creating impossible situations such as orphaning symbols in between columns. See the functions `valid_group`, `valid_colset_comb` and `valid_colset_pair`.

6.8.5 Deduction

The deduction procedure is the most straight-forward part of the process. For each proposed position of the 'common' group, we fill it into the figure. The symbols included in such a group immediately reveal the position of four columns. Then we iterate over the same deduction process as described in section 6.7.3; we check to see if there is a code group of which only one symbol is missing and then attempt to find candidate symbols in the telegram. If exactly one candidate is valid (again constraints such as orphaning a symbol or overlapping other columns limit validity), we can fill in another column. Repeating this process either results in an empty queue of code groups to try, or a filled figure. See the functions `deduce`, `fill_col` and `valid_symbol`.

6.8.6 Results

Running the program turned out to be less straight-forward, mostly due to the nature of the ciphertext telegrams. As these telegrams were handwritten, many of them contain manuscript errors or illegible symbols. As Nuboer's notes included plaintext for several of these, it was possible to reconstruct the ciphertext manually by filling in the figure and matching the columns with the ciphertext, which allowed for correcting some errors. This is a very precise and tedious process. Additionally, as hinted at in section 6.8.1, there was a lack of available ciphertext for several dates.

Table 6.20 below contains the permutations that were found using the program, as well as a reference to the telegrams that produced the results. These can all be found in the repository mentioned in section 6.8. When a telegram number is appended with a `b` or `c`, this signifies that the second or third block of ninety symbols was used.

Date	Permutation	Telegrams used
July 1st 1936	`[2,5,9,7,11,4,6,10,1,8,3]`	`301001071`, `304001200`
July 2nd 1936	`[7,2,4,11,10,8,5,1,9,3,6]`	`409002114`, `307202231b`
July 3rd 1936	`[4,9,6,2,5,1,3,8,11,7,10]`	`310003155`, `297003153`
July 5th 1936	`[10,3,7,1,9,5,11,8,6,4,2]`	`029005170`, `320005191`

Table 6.20: Permutations found by the program

6.8.7 Work in progress..

When it comes to a program such as the one described above, there are always improvements to be made, bugs to be fixed and exceptional cases to be handled. In its current form, the program is quite strict on finding a perfect match. As mentioned above, many of the telegrams contain some form of error. Improvements to the program might be able to make it more fault-tolerant, or additional modules could be included that allow the user to quickly locate and fix the errors (rather than doing this manually). This should be considered an initial attempt, and new versions should improve it significantly.

There is much ciphertext available from July 8th and some from July 9th, so hopefully it will be possible to perfect the program to find these permutations as well. Additionally, as section 9 will illustrate the suspicion that the '.' code and the Blue Book are the same code, it could prove to be interesting to see if the program is able to solve Blue Book ciphertext. Any and every contribution in the form of additional ciphertext is very welcome and can be submitted to the GitHub repository as listed in section 6.8.

6.8.8 Scraping Nuboer's notes

Nuboer did not only leave a series of ciphertext telegrams, but also included many pieces of corresponding plaintext. Additionally, he listed various permutations (Nuboer, 1977e), several of which the program was not (yet) able to find. For reference purposes, these are included in table 6.21 below.

Date	Permutation
July 8st 1936	`[8,4,10,2,9,6,5,1,3,7,11]`
July 9nd 1936	`[3,11,7,5,2,9,10,8,1,6,4]`
July 14rd 1936	`[10,6,4,2,8,11,9,7,3,1,5]`

Table 6.21: Permutations listed in Nuboer's notes

7 Other ciphers

As mentioned in section 3, Nuboer and Schalkwyk came across two other codes, other than the `Ni` and '.' codes described above. Telegrams written in these codes were marked with `Sa` and `I` symbols in their bottom-right corner. As these telegrams occurred much less frequently, less value was initially attached to decryption schemes (Nuboer, 1981).

7.1 The Sa code

In 1935, Schalkwyk focussed on the telegrams that were transmitted in `Sa` code. He soon recovered the underlying code, which had been encrypted using a similar column-based scheme. Nuboer concluded that they had come across a technical code, as the address headers indicated technical support stations and made mention of technical staff. In his memoirs, Nuboer goes on to remark that they were not well acquainted enough with the Japanese language to try to recover the meaning of the technical jargon, and the '.' code demanded their full attention at the time (Nuboer, 1977h). Unfortunately, no details about the workings of the `Sa` code appear to have been preserved.

7.2 The I code

In early 1937[3], Dutch cryptanalysts discovered that the telegrams written in `I` code were encrypted using the same scheme as the reciphered '.' code, albeit using yet another different encryption figure. By the time this was discovered, several large `I` telegrams had been intercepted (Nuboer, 1981). Most significantly, a message from the Chief of the Naval staff addressed to the 12^{th} Squadron was decrypted. The 12^{th} Battle Squadron of the Imperial navy was well-equipped, consisting of minelayer Okinoshima, seaplane tender Kamoi and destroyers Asanagi and Yunagi, as well as several seaplanes. The destroyers Oite and Hayate were also part of the 12^{th} Squadron (Hackett & Kingsepp, 2011), but did not appear to have played a role in the mission at hand – none which Nuboer was aware of, anyway.

The message contained instructions describing a surveillance journey through the mandated territories, seeking new military support sites in the southern districts (Nuboer, 1977h). The decryption of these telegrams allowed the Dutch naval forces to be on full alert for this important journey. This, combined with the low frequency with which the `I` cipher system was used, let the Dutch to believe that the cipher was meant for especially secret information (Nuboer, 1981). This was further established by observing the strictly limited group of telegraphic stations that were mentioned in the address headers, including only the highest authorities of the Imperial Navy.

[3]Nuboer is inconclusive in his memoirs whether it was 1936 or 1937, but movement records of the Okinoshima (Hackett & Kingsepp, 2011) and Kamoi (Hackett, Kingsepp, Alsleben, & Cundall, 2006) ships show that the 12^{th} Battle Squadron had only been formed in December 1936, and the ships left Yokosuka late January 1937.

8 Wartime results and postlude

Nuboer stresses that the main result of the Dutch cryptanalytic efforts is the amount of general tactical data that was gathered about the Japanese Imperial Navy. This included organisational structures of the various squadrons and fleets, but also positioning and even production information concerning the various Japanese vessels. In 1936, a vast part of all Japanese Naval communications was being read and carefully indexed by Department 1 in Batavia. On multiple occasions when the Dutch diplomatic mission in Tokyo warned for a possible Japanese act of aggression, Department 1 was able to show that the fleet was in port, preventing unnecessary tumult (Nuboer, 1977i).

In early 1937, Nuboer and his staff noticed a build-up of tension with the West. This manifested itself in the provocative participation of the recently modernised cruiser Ashigara in the Coronation Review at Spithead, England (Nuboer, 1977i). During its journey homewards, the Japanese army provoked the war that had been looming for months by advancing upon the Chinese army in what is known as the Marco Polo Bridge incident (July 7$^{\text{th}}$, 1937) (Crowley, 1963). Judging by the intercepted communications, Nuboer concluded that the Japanese Imperial Navy had long been reluctant to enter war. When the incident resulted in the Japanese invasion of China and a full-scale attack on Shanghai on August 13$^{\text{th}}$, the navy found itself forced to go along (Nuboer, 1981).

As mentioned earlier in section 3, the hostilities with China led to a complete overhaul of the telegraphic and cryptographic organisational protocols. The known list of telegraphic call signs was replaced by a more secret and obscured list and plaintext telegrams (previously used for unimportant messages) no longer occurred at all. The keys changed more frequency, and Nuboer recalls that a new encryption figure was used every month. The keys were made longer and the figures more intricate. This greatly inconvenienced the Dutch cryptanalytic effort, but decryption was continued steadily with increased manpower and by working even longer shifts (Nuboer, 1977h).

The mobilisation of the Japanese navy progressed swiftly, but the Dutch were reading along with the instructions as soon as they were sent. The telegram describing the advances upon Shanghai was decrypted promptly. In particular, this made it possible to send the destroyer HNLMS Van Galen to Shanghai and execute an evacuation of the Dutch staff before hostilities started (Nuboer, 1977i, 1977g). The decryption also provided insight in the composition and plans of the attack forces gathered in the newly established base on the Chasun islands and on Formosa, as well as the progress of the blockade along the French-Indonesian border. When the Tenth Army, having gathered on Formosa, landed at Hangzhou Bay in November 1937 and effectively surrounded Shanghai in a pincer grip (Wakeman, 1996), the Dutch were long gone.

All in all, it is safe to say that the main goal of Department 1, to provide insight in Japanese Naval movements and plans, had been accomplished formidably. As for Johannes Nuboer, his job was done; in January 1938 he was transferred to the HNLMS Sumatra – thoroughly worn out and exhausted. In June later that year the ship set sail homeward (Nuboer, 1977h).

9 The American effort

As early as World War I, the Americans were employing some form of crypto-analytic work. This was done in a so-called Black Chamber (Pelletier, 1996), trying to solve Japanese communications from 1919 and onwards. A famous example of the work of the Black Chamber is the insight it provided in the Japanese communications during the Washington naval conference in 1921-22, under Herbert O. Yardley (Kahn, 2004).

In the late twenties, the American army and navy began to see the crucial importance of cryptanalytic work and the tactical advantages that could be gained. The American crypto-analytic effort was chronologically slightly ahead of the Dutch – a formal cryptanalysis 'short course' was established as early as 1926, and a school for intercept operators was formed in 1928 on the roof of 'Main Navy' (accordingly, graduates were nicknamed the 'On the Roof Gang'). As the Americans were also aware of the expansionism that ruled the Japanese military plans (see section 2.1), several interception sites were set up from 1928 to 1932, overlooking the western Pacific area from Peking, Guam and the Philippines (Parker, 1994).

The results of the signals intelligence and cryptanalytic group of the US Navy, OP-20-G, were very similar to those of the Nuboer's staff. They too were able to follow the capabilities and movements of the Japanese navy. This revealed Japan's intentions to invade Manchuria and, more worryingly, the fleet's strategic capability to wage a successful full-scale battle against the U.S. pacific fleet (Parker, 1994). Seemingly minor details could make great tactical differences – using Blue Book decrypts, the US Navy found that the Japanese battleships had a speed of 26 knots, faster than the US ships by 2 knots. To be able to outrun the Japanese, the navy immediately upgraded to 27 knots (Pearl Harbor Review, 2009). Also much like in Nuboer's situation, the intelligence operations were not quite able to cope with the enormous task at hand; lack of staff and budget were greatly inhibiting the progress of OP-20-G. The demand grew explosively, but education and logistics were unable to keep up.

While the organisational structures of the American military intelligence services (and the disputes between the US Navy and the US Army's signal intelligence groups) are sufficiently interesting to provide for several additional sections, it is irrelevant for the cryptographic side of the story. Let us instead focus on the Japanese codes that the American cryptanalysts were breaking.

9.1 The Red Book

Early on in his book, Parker (1994) makes mention of a Red Book cipher – also called the *'Japanese Navy Secret Operations Code 1918'*. An initial solution to this Japanese code book cipher was found in 1928 by the mathematician Agnes Meyer Driscoll, discovering what is referred to as 'transposition forms' (Parker, 1994). This cryptanalytic feat followed earlier successes in what Pelletier (1996) refers to as 'practical cryptanalysis': breaking and entering into the Japanese consulate in New York City and making photographs of the code book. The

actual breaking of the code was greatly delayed by the lack of intercepted telegrams, but once accomplished, proved to be a great source of tactical information. Lt. Wenger, credited with breaking the first key to the transposition cipher, is quoted saying *"The messages gave a comprehensive outline of the Japanese-American war plan, and showed that the Japanese had made an excellent estimate of our war plan."* (Pelletier, 1996). Pelletier claims at that the RED BOOK is in some respects the most important Japanese cryptographic system that was ever broken, as it not only foreshadowed Japanese military intentions for years to come, but also provided great incentive to establish a strong cryptanalytic force.

Unfortunately, the technical details of RED BOOK code do not match any of the ciphers discussed in this thesis. The code consisted of 97,366 groups of three Kana symbols, and the entire alphabet of 46 symbols was used (in contrast to earlier ciphers we have seen, where two symbols were omitted) (Pelletier, 1996). The code mostly consisted of tables of geographical locations and ship names, and the books included an instruction stating that the code was never to be used without additional encipherment – it called for a substitution or additive cipher, but Pelletier (1996) notes that in practice only transpositions were used. The RED BOOK was abandoned in November 1930, being superseded by the BLUE BOOK (Parker, 1994).

9.2 The Blue Book

Pelletier (1996) notes that when they were trying to bring the RED BOOK up to date in September 1931, Mrs. Driscoll wandered by and quickly found that the new code consisted of groups of four Kana symbols rather than three. This allowed the lieutenant working on the code to solve the transposition. However, this single transposition did not amount to the full breaking of the code. There were two transpositions; one relatively simple and the other significantly more difficult. Albert Pelletier writes about the latter: *"It employed a relatively sophisticated columnar transposition involving both nulls and blanks. The garble table, or differential feature, reduced the number of code groups to 85,184, nearly the same as the Red book."* (Pelletier, 1996). Using frequency analysis based on the RED BOOK, the cryptanalysts were able to make guesses about the meaning of the most common groups in the new BLUE CODE.

What is especially striking about the above quotation is that the system described by Pelletier (1996) is extremely similar to the '.' code that has been decrypted by Nuboer and has been the main focus of most of the earlier sections of this thesis. Not only did both codes consist of groups of four symbols out of a 44-symbol Kana alphabet and were identified by three of these (accounting for the $44^3 = 85,184$ code groups), but as we have seen in section 6, the (more difficult) reciphered variant of the '.' code is indeed also based on columnar transposition with nulls and blanks.

The names of such code systems were entirely arbitrary. Recall that Nuboer named the code after dot that appeared at the end of each line (see section 3). The BLUE BOOK was named as such after two blue-covered loose leaf binders that were used to collect the known fragments of the book as it was being recon-

structed (Pelletier, 1996). Even though the initial code had been discovered, the progress was slow and difficult. The Navy lacked budget and was not sufficiently staffed to work on decrypting both the new BLUE BOOK and keep up working on the legacy RED BOOK material, and after long negotiations a settlement with the Army was reached and the labour divided (Parker, 1994).

In 1933, the BLUE BOOK cipher was solved by Safford, Dyer and Driscoll. Parker (1994) writes that *".. their success had followed what was possibly the most difficult cryptanalytic task ever undertaken by the United States up to that time."* and also that *"In Safford's opinion, Driscoll's work in solving the system may have been even more brilliant than the Army's subsequent solution of the Purple machine."* The decryption was also helped greatly by the IBM tabulating machines (Parker, 1994). If it is indeed the same system, this puts the work of Nuboer in a remarkable perspective, especially considering the fact that the Dutch did not have access to the RED BOOK. This breakthrough also matches the time-frame of Nuboer's discoveries, as the '.' code was initially recovered in 1935 (see section 5.1).

Another mention of the BLUE BOOK in the article of Pelletier (1996) adds to the suspicion that Nuboer and Safford et al. were dealing with the same code. A certain Arnold Conant was tasked with the duty of recovering the transposition each day, indicating that the transposition was indeed changed every day, as Nuboer had observed.

The cipher consisted of a grid, and the code groups were written in the grid's blank spaces from left to right. After this the Kana symbols were read off from top to bottom (Pelletier, 1996). Pelletier then writes that the grid changed daily. Nuboer's view is a bit more nuanced, observing that the grid was the same for a longer period of time and only the columns were permuted differently. This slight difference can be accounted for by noting that Pelletier, as it appears that at this point his task was limited to performing the decryption, might not have had a complete overview of the entire system.

The exact date when the BLUE CODE was discontinued is slightly unclear: Pelletier (1996) mentions June 1939 while Parker (1994) says it was replaced in October 1938. In any case this confirms to Nuboer's memoirs, where he mentions the work on the '.' code still continued when he left for the Netherlands in the summer of 1938 (Nuboer, 1977g).

While the above similarities between the BLUE BOOK and the '.' code should not be interpreted as complete and fully conclusive evidence, it provides for serious indications that Nuboer and the Americans were dealing with the same cryptographic system. In other words, this would imply that the BLUE CODE and the '.' code are indeed different names for the very same code.

10 Conclusions and discussion

The main focus of this thesis was to show how the Dutch broke the Japanese naval codes in the decade preluding the Second World War. It soon became clear that most work was done on the two variants of the so-called '.' code. J.F.W. Nuboer, the lead cryptographer of the Dutch Indies Navy (see section 2.3), also worked on several of the other cryptosystems that were used by the Japanese Imperial Navy at the time, as discussed in sections 4 and 7. However, the '.' code was of far more importance, as it was the de facto standard code for Japanese naval communications.

In sections 5 and 6 we have seen how this code worked (both the standard and reciphered version), and we have explored the techniques that were used to break the cipher. While on first glance the code might appear rather simple, it proved to be a serious challenge (and achievement) to break. The code is twofold. First, the message is encoded using a code book system, translating words or names to predefined code groups of four Kana symbols (the latter of which was a checksum). This code book was a closely guarded secret and provided for an extra layer of obscurity. Then a transposition cipher was applied. Initially, the transposition consisted of writing the message in a grid of nine columns, and permuting the columns along a certain permutation – it never changed, so once Nuboer had uncovered this permutation, all messages could be decrypted. In the summer of 1935, the Japanese changed the transposition cipher and replaced it with a much more difficult system. This system consisted of a grid with nulls and blanks, into which the plaintext was written after being encoded into code groups. Then the columns of the grid were permuted, and were read off vertically. On top of all this, the permutation changed daily and several different grids were circulated. To break these systems, Nuboer heavily relied on the repetition of common code groups in the plaintexts of the telegrams. By comparing the ciphertext telegrams that showed only slight differences, much could be learned about the transpositions that had been applied.

It is this last system that the Americans are most likely calling the Blue Book and Blue Code. In section 9 we have seen evidence that shows the American Navy was breaking a very similar Japanese code system at the time. The proof is not complete and should not be regarded as such, but it provides strong indications that make it justifiable to assume that Nuboer and the American Navy were in fact dealing with the same system. If this is the case, we are better able to assess the work Nuboer has done by viewing it in a broader context. The American Navy operated on a much larger scale than Department 1 could ever hope to achieve, and even then it took several years to break the Blue Code – only slightly ahead of Nuboer and his staff. Perhaps even more noteworthy, as mentioned in section 9, is the fact that the Americans indeed considered it to be *"..possibly the most difficult cryptanalytic task ever undertaken by the United States up to that time."* (Parker, 1994). Taking into account the fact that the Americans had access to a predecessor code (the Red Book) and were able to use IBM machines makes Nuboer's work perhaps even more impressive by comparison.

Even without comparing it to the American assessments, the high level of obscurity of the system is quite evident. First and foremost, the Japanese language

caused a great practical issue, as the difference between plaintext and ciphertext is not even immediately clear for someone unfamiliar with the language. The translation into code groups provided an additional hurdle, after which the obscure and unstructured figure distorted the message even further. And then there is the frequent changing of the keys. It is truly remarkable how such a system can be unravelled by a slight tug on the correct loose ends.

The value of cracking the '.' code has been clarified in section 8. The information gathered from the telegrams formed a great source of tactical data with regards to the Japanese naval movements and were essential in anticipating these advancements upon China, forming a defence plan and practising diplomatic neutrality. The value that was attached to his achievements is also partially reflected in the decoration Nuboer received upon leaving the navy; that of titular Rear Admiral (Bosscher, 1994).

10.1 Improvements and future work

While the evidence provides very strong indications, it is still not fully proven that the '.' code and the Blue Book are in fact the same system. As there seems to be little information available on the subject of the Blue Book, it proved difficult to find actual solid proof. Finding a section of the actual Blue Book or a ciphertext or plaintext telegram written in the code could go a long way as to delivering this proof.

In section 6.8, we have briefly discussed a program (see appendix I) that attempts to automates the algorithm that Nuboer used to find new permutations for the '.' code. While this shows that it is possible (and feasible) to automate the algorithm as used manually by Nuboer, several of the permutations that were uncovered manually were not found by the program. This indicates there is room for improvement. Additionally, it might prove possible to devise other, more computationally intensive but more complete methods to achieve better results.

Even though the Dutch considered their East Asian intelligence service (DOAZ) to be one of the leading intelligence organisations in the Far East, their grasp on the communication in the Pacific theatre was far from complete. It is conceivable that neighbouring countries must have had similar cryptanalytic ambitions. Did they, too, focus on the Japanese navy, in fear of expansion?

In a conflict of this proportion, it is unlikely that only one party had been listening in on the enemy-to-be, tactically preparing for an outburst of war. Haslach (1985) describes the human intelligence employed by the Japanese (mostly from viewpoint of the Dutch counter-espionage) (Haslach, 1985), but the signal intelligence effort seems less documented. What did the Japanese know? What codes did the Dutch use, and to what extend were they broken?

Topography

Bandung

The present-day capital of the province of West Java, Bandung lies just south of Batavia. In the early twenties, the Dutch Indies Government planned to move the capital from Batavia to Bandung, as its location provides a much better natural defence system. These plans were cut short by the preliminaries of the second World War (Ashworth, 2009). It housed the General Staff, including the illustrious 'Room 14'.

Batavia

The city of Jakarta, the present-day capital of Indonesia, was known as Batavia from 1619 to 1942. It served as the capital of the Dutch East Indies and housed the Dutch Indies Naval staff, amongst which was Department 1 (Intelligence). This is where Nuboer did most of the cryptographic work described above.

Formosa

The island and archipelago of Formosa is currently known as Taiwan. Formosa was under Japanese rule between 1895 and 1945, serving as a tactical vantage point for the Imperial Navy and a supply of foot soldiers for the Imperial Army. Formosa suffered great losses during the war, as many Formosan youth were killed while serving for the Japanese, as well as in Allied bombing raids.

Manchuria

The north-east coastal region of China (just north of Beijing, spanning the present-day provinces of Heilongjiang, Jilin and Liaoning) is historically referred to as Manchuria. It is the east-most part of China, closest to Japan. This tactical juncture of Russia, Japan and China has been subject to much dispute over the years. By some definitions, parts of Mongolia and Russia are included under the same name.

Yokosuka

Yokosuka is an ocean-side city just south of Tokyo, at the mouth of the Tokyo Bay. The Yokosuka Naval District was one of the first administrative naval districts, the others centring around Sasebo, Kure and Maizuru (Evans & Peattie, 1997). The port was a valuable headquarters for the Japanese Imperial Navy, having grown into this position because of its strategic vantage point in the bay, as well as a centre for naval education and telegraphy.

References

Ashworth, G. J. (2009). The Imperial capital that almost was: Bandung's colonial heritage and what to do with it. (Sharing Cultures conference)

Bosscher, Ph. M. (1994). Nuboer, Johannes Frans Willem (1901-1984). In *Biografisch Woordenboek van Nederland.* Huygens ING - Den Haag. `http://www.historici.nl/Onderzoek/Projecten/BWN/lemmata/bwn4/nuboer`.

Crowley, J. B. (1963). A reconsideration of the Marco Polo Bridge Incident. *The Journal of Asian Studies*, *22*(3), 277–291.

Duus, P. (1989). *The Cambridge History of Japan* (Vol. 6). Cambridge University Press.

Evans, D. C., & Peattie, M. R. (1997). *Kaigun: Strategy, tactics, and technology in the Imperial Japanese Navy, 1887-1941.* Naval Institute Press.

Hackett, B., & Kingsepp, S. (2011). *IJN minelayer Okinoshima: Tabular record of movement.* `http://www.combinedfleet.com/Okinoshima_t.htm`. (Timeline of the movements of the battleship Okinoshima, revision 4)

Hackett, B., Kingsepp, S., Alsleben, A., & Cundall, P. (2006). *IJN seaplane tender/oiler Kamoi: Tabular record of movement.* `http://www.combinedfleet.com/Kamoi_t.htm`. (Timeline of the movements of the seaplane tender Kamoi)

Haslach, R. D. (1985). *Nishi no kaze, hare. Nederlands-Indische inlichtingendienst contra agressor Japan.* Weesp: Van Kampen.

Kahn, D. (2004). *The Reader of Gentlemen's Mail: Herbert O. Yardley and the Birth of American Codebreaking.* Yale University Press.

Meijer, H. (2002). Lovink, Antonius Hermanus Johannes (1902-1995). In *Biografisch Woordenboek van Nederland.* Huygens ING - Den Haag. `http://www.historici.nl/Onderzoek/Projecten/BWN/lemmata/bwn5/lovink`.

Nuboer, J. F. W. (1977a). Bijlage 2: Het vinden van de hervercyfering van de Japanse marine-code 1 in juni 1935. In *collectie 070, inv.nr. 4.18.* Nederlands Instituut voor Militaire Historie, Den Haag.

Nuboer, J. F. W. (1977b). Bijlage 3: Tot open code ontcyferde telegrammen uit begin 1935. In *collectie 070, inv.nr. 4.18.* Nederlands Instituut voor Militaire Historie, Den Haag.

Nuboer, J. F. W. (1977c). Bijlage 4: De Japanse marinecode no. 1. In *collectie 070, inv.nr. 4.18.* Nederlands Instituut voor Militaire Historie, Den Haag.

Nuboer, J. F. W. (1977d). Bijlage 5: De ontcyfering van de sleutels van augustus 1935. In *collectie 070, inv.nr. 4.18.* Nederlands Instituut voor Militaire Historie, Den Haag.

Nuboer, J. F. W. (1977e). Bijlage 6. In *collectie 070, inv.nr. 4.18.* Nederlands Instituut voor Militaire Historie, Den Haag. (Figuur juli 1936 en permutatiesleutels)

Nuboer, J. F. W. (1977f). Bijlage 7: Het vinden van de formule voor de volgorde van de kolommen van de hervercyfering by een bekende hervercyferingsfiguur. In *collectie 070, inv.nr. 4.18.* Nederlands Instituut voor Militaire Historie, Den Haag.

Nuboer, J. F. W. (1977g). Geschiedenis van Afdeling 1 (Inlichtingen) Marinestaf

Batavia van augustus 1934 tot januari 1938. In *collectie 070, inv.nr. 4.18.* Nederlands Instituut voor Militaire Historie, Den Haag.

Nuboer, J. F. W. (1977h). Memoirs J. F. W. Nuboer. In *collectie 070, inv.nr. 4.17* (pp. 48–55). Nederlands Instituut voor Militaire Historie, Den Haag.

Nuboer, J. F. W. (1977i). Oprichting en beginjaren van de Afdeling 1 van de Marinestaf te Batavia. In *collectie 070, inv.nr. 4.17.* Nederlands Instituut voor Militaire Historie, Den Haag.

Nuboer, J. F. W. (1981). A history of Afdeling I (Intelligence), Naval staff, Batavia, Netherlands East Indies, from August 1934 to January 1938. In (Vol. 47). The Cryptogram.

Parker, F. D. (1994). *Pearl Harbor revisited: United States navy communications intelligence 1924-1941* (Vol. 6). Center for cryptologic history, National Security Agency.

Pearl Harbor review - early Japanese systems. (2009). `http://www.nsa.gov/about/cryptologic_heritage/center_crypt_history/pearl_harbor_review/early_japanese.shtml`. National Security Agency - Center for Cryptologic History.

Pelletier, A. (1996). Cryptography – Target Japan. In *U.S. Naval Cryptologic Veterans Association* (p. 27-32). Turner Publishing Company.

Visser, J. (2005). *Rear-admiral J.F.W. Nuboer.* `http://www.netherlandsnavy.nl/Men_nuboer.htm`.

Wakeman, F. (1996). *The Shanghai badlands. Wartime terrorism and urban crime (1937-1941).* Cambridge University Press.

Yasuba, Y. (1996). Did Japan ever suffer from a shortage of natural resources before World War II? *The Journal of Economic History*, *56*(03), 543–560.

A Overview of telegraphic call signs

The call signs listed below were used in the address headers of Japanese naval telegrams to indicate the sending and receiving parties. See section 3.1.

`Da`	Chief of the Naval staff
`Re`	Combined Fleet
`Reichi`	Commander in Chief of the Combined Fleet
`Kata`	Fleet
`Sen`	Squadron
`Kosen`	Aero squadron
`Susen`	Destroyer squadron
`Sesen`	Submarine squadron
`Kutashi`	Destroyer division
`Sesutashi`	Submarine division
`Sutashi`	Torpedo boat division
`Sotashi`	Minesweeper division
`Kotashi`	Aircraft division
`Ren`	Training squadron
`Chi`	Base
`Yo`	Support station
`Ka`	Ship
`Kuka`	Destroyer
`Seka`	Submarine
`Suka`	Torpedo boat
`Shichi`	Commander in Chief
`Shika`	Commander
`Sachi`	Chief of staff
`Fuka`	Adjutant
`Shike`	Paymaster
`Za`	Navy representative
`Muchi`	Telegraphic station

B Telegram 028101093, 029201093 (April 1st, 1935)

This telegram was transmitted by the Japanese educational squadron upon departing from the port of Singapore. Its interception was essential for the decryption of the '.' code, as detailed in section 5.1.

```
028101093 ate Da - Reichi = Renshika

hone 1:
He  Mi  Ne  No  E   No  Mi  Ka  Re  .
Ke  Mu  E   No  No  U   Ro  Ki  Tsu .
Mi  He  Yu  Sa  Ha  Fu  Mi  Yu  I   .
Se  Ha  Mo  Yu  Na  Ke  Ho  Ru  Ta  .
Ho  A   Chi No  Re  Ka  Se  I   Ru  .
Ke  Ko  A   Fu  Ni  Ho  Shi Ki  Mi  .
Ha  Me  Chi A   Ke  Ya  Tsu Su  E   .
Na  Mi  Ko  I   Mo  Se  Ka  Wa  E   .
Hi  Shi He  Tsu So  Yu  Ke  Ta  Wa  .      <-

hone 2:
Se  Na  Yo  Ta  E   Mo  Mo  Re  Se  .
Ru  Ru  U   Ko  So  Shi Ko  Ke  No  .
He  Chi Ne  Hi  Ki  Chi Ru  Ko  Se  .
Na  Shi Ke  Ku  No  Yo  So  Ne  Ko  .
Ha  Su  Su  Ma  Mu  No  Na  Ni  Ka  .
To  Na  To  Ko  Chi Fu  Mi  Fu  Ka  .
Mi  Chi Mu  Ka  Me  Ku  Ni  Tsu Yo  .
Mu  Ku  Ne  Ho  Ya  He  Ri  Yo  Shi .
Fu  Te  Me  So  Ku  Ne  Ri  He  Ta  .
Te  Se  Ku  Ke  Na  Ra  Hi  Mi  Shi .

hone 3:
Ma  E   I   Chi Ki  Fu  Ke  Ha  So  .
Yo  Mi  Ku  Ru  Ri  To  Ho  Me  Ka  .
Ru  He  Ki  I   Mi  Se  Ha  Ne  I   .
O   Ne  Ra  Ma  So  Te  Ka  Ta  Ra  .
Mi  E   I   Te  Wa  He  Me  Fu  Na  .
Me  O   Tsu Ne  So  Ya  Tsu Ra  Na  .
Ro  I   I   Ra  Ho  Tsu Re  Te  Me  .
Re  Mo  Ka  Na  Fu  Ro  Ma  Ta  Ko  .
Ni  Te  Ke  Ya  Hi  Ma  Ko  Yu  Te  .
Ne  Ro  Ya  Se  Mu  Fu  Ta  A   Ho  .

hone 4:
Ru  Ki  Ra  So  Te  Ki  Ni  Na  Ya  .
Sa  Yo  E   Ri  Shi Ka  Mo  A   Fu  .
So  Tsu He  Ri  Ra  Mu  O   Te  U   .
Wa  Mi  Ko  Ra  Fu  Re  Chi A   Fu  .
```

Ho Mu Ma Wa Mo Mi Ni Ta Chi .
Ha E He I Mu Se Ra Ko Ki .
Ke So No Sa Ra Ka Chi Ru O .
Ni Na Ma Su I So Ki Ma Ki .
Mi Yo Ra E Ha Se Se Se Ya .
Fu Ke Yo Te Ni Yu Wa Mi No .

hone 5:
Shi Ta I Se Te Ri Ma Su Sa .
He Me Yo Ta Shi Na Tsu Sa Ro .
Te Sa Se Mi Me Mi Ru Ra To .
Ne Me Mi A Te Mu O Hi Ko .
Na To To Yo Ya I Se Ni Ka .
Yo Ya Ta Ma Ka

029201093 ate Da -Reichi = Renshika

hone 1:
Ko Te Ta Sa He Mi No I Hi .
Tsu He Yu Yo Na Ka Ra Ri Ki .
O Re A Mi Ha Ru Ho Fu Yu .
Se Mi Ri O Tsu Na Se Chi Yu .
Re I Fu Shi Ho Mi Mi Ke To .
A Yo No Ni I Chi Fu Se Ho .
Ko No Fu Ka Ki A Re Shi Ke .
A I Ra Ho Ta Te Ni So No .
Fu He Ke Ma E No Sa E Mu .

hone 2:
Mu Yu He Ku Ka Se Ke No Se .
Yu A Ya Wa A Ya Ke He Se .
Ha I Ri Ma I Ni Re Ke Ro .
Ma Na Ho So Chi No Ki Ku Re .
A Ro Mu Na Ya Na Ho Fu Mi .
Wa Re Ku So No No Su Ma Me .
Ma Se Yo Ni Ri To Tsu Ka To .
Ha Ne Sa Ya A Ta Ki Se Hi .
I Mo Ru Ni Mo Mo Ka E I .
Mi Ta Ha Re Ho I Na So Yu .

C Telegram 031005173 (April 5th, 1935)

This telegram was transmitted by the Japanese educational squadron when departing from Batavia, after a three-day visit. This telegram was also critical for the decryption of the '.' code, as mentioned in section 5.1.

```
031005173 Da - Reichi = Renshika

hone 1:
He  Mi  Ne  No  E   No  Mi  Ka  Re  .
Ke  Na  E   O   No  I   Ri  Ki  Tsu .
Mi  Ta  Yu  Sa  Ha  Fu  Yu  Yu  I   .
U   Ra  Ha  Ku  Su  Ka  Ho  Ha  Mu  .
Ho  A   Chi No  Re  Ka  Se  I   No  .
Mi  Ko  Ma  Fu  Mi  Wa  Shi Shi Tsu .
Ne  Ne  Hi  A   I   Yu  Shi Sa  Yo  .
Ni  Ki  Se  Ke  A   Ko  Ho  A   Ri  .
Ko  Ku  Ki  Sa  U   Yo  Ka  Ho  Mo  .

hone 2:
Mi  Mi  Ra  Mo  A   Mi  Shi Na  Wa  .
Su  Tsu Ha  Me  Chi E   Ho  A   Ka  .
Wa  Ka  Na  Mi  Ko  E   Ya  I   Mo  .
Ta  Ke  Hi  Shi He  Wa  Se  Tsu So  .      <-
Tsu Yo  Yo  Ku  E   Mo  Me  Ri  Se  .
Ru  Ru  U   Ko  So  Na  Ko  Ke  Mo  .
Mi  Re  Ni  Mo  I   Ya  Ta  Su  Ma  .
Fu  Ka  Ri  Shi Ro  Mi  Na  Ko  I   .
Shi Ha  Ku  He  A   Ro  Ko  Mu  Yu  .
No  So  Ko  Ma  Yo  Ro  Yu  Ke  To  .

hone 3:
A   Su  Hi  Ma  Ma  Ha  Na  Ni  Na  .
Ya  Te  Mu  Mo  Ne  Ro  Ri  Hi  Re  .
Ha  Mi  Shi Se  He  I   Ka  Yo  Ri  .
Me  Mi  No  Wa  Ro  Mi  To  He  Sa  .
Tsu Tsu U   Yu  Mi  Sa  Ri  Na  He  .
I   Sa  Yu  Mo  Ta  He  He  Tsu Ka  .
No  Ne  Ka  Tsu Ru  E   Ka  Tsu Ma  .
Tsu Mi  Mi  Chi Re  Su  Yo  Shi To  .
Ra  I   So  U   Ra  E   Ru  Se  Te  .
Mu  Ta  Shi He  Mo  Mi  Ku  Ku  E   .
Ta  Mu  Ra  Na
```

D Kana alphabet and numerical values

This table provides a translation between Kana symbols and numerical values, to be used to compute the checksum of a three symbol code group. This is explained in section 5.5.

A	1	I	8	Me	15	No	22	Sa	29	Chi	36	Yo	43
E	2	Ka	9	Mi	16	O	23	Se	30	To	37	Yu	44
Ha	3	Ke	10	Mo	17	Ra	24	Shi	31	Tsu	38		
He	4	Ki	11	Mu	18	Re	25	So	32	U	39		
Hi	5	Ko	12	Na	19	Ri	26	Su	33	Wa	40		
Ho	6	Ku	13	Ne	20	Ro	27	Ta	34	Wi	41		
Fu	7	Ma	14	Ni	21	Ru	28	Te	35	Ya	42		

E Overview of the '.' code book

Naval vessels

So	*	*	(*)	Various ships
So	Ku	*	(*)	Light cruisers
So	Ku	Ku	(Ku)	Cruiser Tatsuta
So	Ku	Mi	(Mi)	Cruiser Kuma
So	Ko	*	(*)	Heavy cruisers
So	Ma	He	(Hi)	Gunboat Saga
So	Me	Mi	(Mu)	Seaplane tender Kamoi
So	Mu	To	(Ya)	Oiler Tsurumi
So	Ra	*	(*)	Destroyers
So	Ri	*	(*)	
So	Ro	*	(*)	
So	Ra	To	(He)	Destroyer Minazuki
So	Se	Hi	(No)	Submarine I-59 or I-60
Ma	I	Ki	(So)	-maru suffix

Geography

Ta	*	*	(*)	Various Japanese locations
Te	*	*	(*)	
Chi	*	*	(*)	
To	*	*	(*)	Formosa (Taiwan) area
To	A	So	(Ri)	Bako (Magong)
To	He	So	(Ru)	Takao (Kaohsiung)
Tsu	*	*	(*)	Chinese and southern locations
Tsu	Tsu	No	(Ka)	Bangkok
Tsu	U	Mi	(He)	Singapore
Tsu	Wi	Yu	(Ta)	Batavia
U	*	*	(*)	Philippines and the mandated territories
U	O	Nu	(U)	Davao
U	O	Ke	(Ro)	Cebu

Authorities

Shi	*	*	(*)	Various authorities
Shi	Ta	A	(Ni)	Commander in Chief of the Combined Fleet
Shi	Ta	Ku	(Su)	Commander in Chief of the 2nd Fleet
Shi	Wa	Fu	(Su)	Commander in Chief of the 3rd Air Force Squadron

Time

Ko	Ku	Te	(Ha)	10:00
Ko	Re	Ri	(Mu)	11:00
Ko	To	Ni	(Re)	12:00
Ku	Hi	Ru	(A)	13:00
Ku	Mo	Ku	(Yo)	14:00
Ne	Mi	Sa	(Ne)	0 minutes
Ne	Mu	So	(Ro)	30 minutes

Numbers

A	A	Tsu	(U)	1
A	A	Wa	(Wi)	
A	Ha	Wa	(Yo)	2
A	He	Na	(O)	
A	He	Ni	(Re)	
A	Hi	Ma	(Na)	3
A	Fu	Na	(Ri)	5
A	Fu	Ni	(Ru)	
A	I	Ta	(Ya)	6
A	I	To	(A)	
A	Ka	O	(So)	7
A	Ko	Ya	(Ke)	8
A	Mi	Ma	(Se)	11
A	Mo	Mu	(Te)	12
A	Ni	Ni	(Ya)	13
A	No	To	(Me)	14
A	Wi	Ka	(Ho)	37
A	Yu	To	(To)	40

F Formosa telegrams, November 9th, 1935

This series of telegrams was transmitted at thirty minute intervals by a ship off the coast of Formosa, early November 1935. The large amount of repetition in these telegrams led to the initial breaking of the reciphered '.' code. This is explained in section 6. Unforunately, the address headers were lost. As the ten-column hone structure was disregarded by the cipher system, the telegrams can be written sequentially without loss of information.

007509103 Ho Ho Ru Wi So A Ku He Te Yo Wi To Na No Ra So A I Na Ko
Mo Ka Ka Wa He Mu Ri A No Wi Ka No So Mu Ni Ha So Na Se A Chi Ro Wi
Ha Fu Ne Te E Ro Ma Ki Ku Ki Ku Ka Ki A Na

007609110 Ho To Na Sa Ta Sa A Ku Tsu Na Yo Wi Ya Wi Chi No He U A Ko
Ne Ko Ku To A Wa Ma Ra Mi Ri A No No He Na Ne So He Wi Na Mu Ko Na
Wa Ke Wi I Ne Yu Ha No Na Ru Ru Yo Ro Ma Ha He Re Mi Ku Tsu Ki A No

007709113 Ho Ma Na Ni So E He Tsu Na Yo Wi To O No He Ri E I Ne Ko
Ke Ki Fu To Ka Mu Ri A No E He Na No So He Wi Na Mu Ko Ru No A Wi To
Wa E U No Na Ri Shi Fu Ma Ho He Re U Mu Na Su Ke Te

007809120 Ho Ku Na Sa Sa E E Tsu Na Yo Wi Tsu I No Ra Ke E A O Ko Chi
Fu Ko To Ma Mi Ri A No Ma He Na Ne So He Wi Na Re Yo Na Ki U Wi Mi
Ne Hi U No Na Ni Ko Ro Ma Mi He To Ku No Su A Re

007909123 Ho Wi Na Sa Wi So E E Ra Na Yo Wi Ni Chi Re No He O A Ne
Ne Ko Ta So He To Ma Ku Mu Ri A No Fu He Na No So Ku Wi Na Re O Ru
Ka Ru Wi Yo Ro I U Ro Na Ro Ni Me Fu Ma Ya He To Shi Mu Na Su A No

008109130 Ho Wi Ra Sa Sa A E Tsu Na U Wi Ya Mi Ri Ra So A Ko O Ku Shi
Wa Ka To Ma Mi Ri A No Ke He .. Ne So Wi .. Na A Ko Na Mu .. Ma Yo
.. A U No .. Ne Yo .. .. Tsu He Hi Ta Ku O Su A Re

008209133 Ho Ru Na Re A He Wi U Ta Ta Mu Ri Wi Mo Ma Ra Ku No Ni U
Na Mu Ri E I Ne No Ya He A Mi So No Ma To Ro Fu Wi Na Ru Mo No A Sa
Hi I Ma To Na Ru

008309140 Mo Ke Na Sa A E Wi Na Wi Wi Te Shi Su Tsu U A Ni Ku Ki Yo
A To Mi E A No Ro He Ne So He Wi Yo Tsu Na Wi Ne Ka .. Ko Ki Na Na
Ku Ku No Ma Se Mo Na Ke He Tsu Su

008409143 Mo Te Ta Sa Ma So A E Tsu Na Wi Wi He Chi Ho Su He Ru A I
Ne Ku Mu Mu Fu To Ma Ku Mu E A No Yo He Na No So Ro He Na Ya Ta Na
Na A Wi I Ro Mo Ku No Na Ro Ku Ku Ro Ma Re He Mo Ka Ku Ni Su A No

G Annotated Formosa telegrams

H Telegrams 301001071, 304001200 (July 1st, 1936)

These telegrams were used as examples in section 6.7, to illustrate the algorithm of finding a new permutation using sufficiently long ciphertext telegrams.

```
301001071 ate Kosentsusa Kamotsu " Meu Ao Haru Kakutsu = Retsusa

hone 1:
Ki  A   Ni  Hi  Ho  A   Ta  Na  Mi  .
So  Ma  Ke  Hi  Se  Hi  Re  Se  Ki  .
Mu  Shi Ru  Ha  Wa  Yu  Se  Wa  Hi  .
Na  E   A   I   Ya  Ni  Mo  O   A   .
E   Ku  Me  Shi Ma  Mo  Na  Tsu E   .
Ya  A   Ru  So  Te  So  Sa  Na  O   .
Mi  Mi  Ru  A   Ya  Ra  Na  Ta  Na  .
Sa  Ru  Wi  Ho  Ni  Fu  Ro  Fu  Yu  .
Me  He  He  Mi  Ro  Shi Se  A   A   .

hone 2:
Ri  Ma  Sa  Ho  Yo  Mo  No  I   Mo  .
Fu  Mu  Na  Ro  Yo  Ya  Yo  Ra  Na  .
Se  Re  Se  Ne  Ri  Ka  Shi A   Na  .
Tsu No  Mi  Fu  Hi  Ki  Ne  To  Ni  .
Ru  Hi  Fu  Yu  Fu  Ro  Ki  A   So  .
A   Ri


304001200 ate Bayotsusa " 2 Tsusa = Retsusa

hone 1:
Ri  Ni  Me  Shi Ra  Tsu Ri  Me  Ri  .
Ke  Te  Ru  Ya  Ki  Ka  Ha  Te  Mi  .
Ki  Me  Mi  Tsu Ta  Me  Me  Ha  Ko  .
Ki  Ha  Ne  Na  Tsu Ko  E   I   I   .
Ko  Tsu Tsu Wa  To  U   Tsu Re  Ho  .
Mu  Ha  Me  Ra  Ni  I   Ra  Yu  Ma  .
Mo  Mo  Yo  Ru  Ne  Me  Ke  U   Re  .
Ka  Yo  Yu  Sa  Mo  Ho  Se  Mo  Re  .
Ri  Re  Na  Ru  Ho  Ho  Mo  Ma  Ki  .

hone 2:
Ka  Ho  So  Wa  Se  Hi  Ka  Ro  Ne  .
Mo  Ni  Yo  Te  Ru  So  Ne  He  Se  .
So  Ri  Su  No  Su  Mo  Ha  Wi  Ka  .
Se  So  Wa  Ha  I   Ni  Ya  Ya  Su  .
Yu  Ni  Wa  Na  Ma  Ra  No
```

I Python program to find permutations

The program listed below is the program that was described in section 6.8 and was used to find the permutations mentioned in 6.8.6. It is also available on GitHub at `https://github.com/joostrijneveld/blue-code-permutations`. Note that the program will possibly be improved over time, and the GitHub repository will contain the latest version. The repository also includes the ciphertext telegrams.

```
1  #! /usr/bin/env python
2
3  from itertools import combinations, izip_longest, product, chain,
       count
4  from copy import deepcopy
5  from collections import Counter
6  import glob
7  import time
8
9  def powerset(s):
10     """ Itertools recipe. Returns the powerset of a list. """
11     return chain.from_iterable(combinations(s, r) for r in range(
           len(s) + 1))
12
13 def grouper(iterable, n, fillvalue=None):
14     """Itertools recipe. Collect data into fixed-length chunks or
           blocks"""
15     args = [iter(iterable)] * n
16     return izip_longest(fillvalue=fillvalue, *args)
17
18 class TwoWayDict(dict):
19
20     def __init__(self, iterable):
21         dict.__init__(self)
22         for (c, n) in zip(iterable, count(1)):
23             self.__setitem__(c, n)
24             self.__setitem__(n, c)
25
26     def __len__(self):
27         return dict.__len__(self) / 2
28
29     def __setitem__(self, key, value):
30         dict.__setitem__(self, key, value)
31         dict.__setitem__(self, value, key)
32
33 MEMORYPATH = "plaintext/*"  # telegrams that provide code groups
       for frequency count
34 TELEGRAMPATHS = ("telegrams/2 301001071", "telegrams/3 304001200")
       # Finds July 1st permutation
35 # TELEGRAMPATHS = ("telegrams/8 409002114", "telegrams/4 3072022316
       ") # Finds 02-09
36 # TELEGRAMPATHS = ("telegrams/13 310003155", "telegrams/14
       297003153") # Finds 03-09
37 # TELEGRAMPATHS = ("telegrams/17 029005170", "telegrams/20
       320005191") # Finds 05-09
38
39 ALPHABET = TwoWayDict(('A', 'E', 'Ha', 'He', 'Hi', 'Ho', 'Fu', 'I',
       'Ka', 'Ke', 'Ki',
40                      'Ko', 'Ku', 'Ma', 'Me', 'Mi', 'Mo', 'Mu', 'Na', 'Ne', '
                            Ni', 'No',
41                      'O', 'Ra', 'Re', 'Ri', 'Ro', 'Ru', 'Sa', 'Se', 'Shi', '
                            So', 'Su',
```

```
42              'Ta', 'Te', 'Chi', 'To', 'Tsu', 'U', 'Wa', 'Wi', 'Ya',
                    'Yo', 'Yu'))
43  ALPHACOUNT = 44
44
45  NUBOERGROUPS = (('Sa', 'He', 'Sa', 'Mo'),
46                  ('Ka', 'Ni', 'Ta', 'Na'),
47                  ('Ki', 'Ke', 'A', 'Ni'),
48                  ('Mi', 'I', 'Wa', 'Na'),
49                  ('Fu', 'Me', 'Ku', 'Ta'),
50                  ('Mo', 'Na', 'Ta', 'Re'),
51                  ('Mi', 'Ru', 'No', 'Ni'),
52                  ('Mo', 'Ko', 'Ru', 'Ko'),
53                  ('A', 'Shi', 'A', 'So'),
54                  ('Mi', 'I', 'Ka', 'So'),
55                  ('Ha', 'Wi', 'Fu', 'Ho'),
56                  ('Ro', 'Ne', 'Ha', 'Hi'),
57                  ('Ma', 'Sa', 'No', 'Ne'),
58                  ('Ra', 'So', 'Ra', 'Te'))
59
60  fT =    [['n',' ','x',' ','n',' ','x',' ',' ',' ',' ',],
61           [' ','n',' ','n',' ','n',' ','n',' ','x',' ',],
62           [' ',' ',' ',' ',' ',' ',' ','x',' ',' ','x','x',],
63           ['x',' ','x',' ',' ',' ',' ',' ','x',' ',' ',' ',],
64           [' ',' ','x',' ',' ',' ',' ','x',' ',' ',' ',' ',],
65           [' ','x',' ',' ',' ',' ','x',' ',' ',' ',' ',' ',],
66           [' ',' ',' ',' ',' ',' ',' ',' ','x',' ',' ',' ','x',],
67           [' ',' ',' ',' ','x','x','x',' ',' ',' ',' ','x',' ',],
68           ['x',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','x',' ',],
69           ['x',' ',' ',' ',' ',' ',' ',' ',' ','x',' ',' ','x','x',],
70           [' ','x',' ',' ',' ',' ',' ','x','x','x','x','x',]]
71
72  f = [list(col) for col in zip(*fT)]  # much easier to work with in
        columns as opposed to rows
73  memory = Counter()
74
75  # computing several constants based on the figure
76  COLNUM = len(f)
77  ROWNUM = len(fT)
78  COLLEN = [ROWNUM - col.count('x') for col in f]
79  FIGLEN = sum(col.count(' ') for col in f)
80  SPACES = FIGLEN + sum(col.count('n') for col in f)
81  GCOUNT = FIGLEN / 4
82
83  def precompute_colcombs():
84      d = dict()
85      for cols in powerset(zip(range(COLNUM), COLLEN)):
86          colsum = sum([b for _, b in cols])
87          if colsum not in d:
88              d[colsum] = []
89          d[colsum].append(set([a for a, _ in cols]))
90      return d
91
92  COLCOMBS = precompute_colcombs()
93
94  def kana2int(c):
95      return 0 if c == "?" or '.' in c else ALPHABET[c]
96
97  def int2kana(i):
98      return '..' if i == 0 else ALPHABET[i]
99
100 def symbol_space(col):
101     return col.count(' ') + col.count('n')
```

```
102
103 def checksum(a, b, c):
104     x = (a + b + c - 1) % 44
105     return x if x > 0 else 44  # since 0 isn't valid, but 44 is
106
107 def missing_symbol(a, b, x):
108     """ Computes one of the three code group symbols based on the
            other two (a, b) and checksum x """
109     c = x - a - b + 1
110     while c <= 0:
111         c += 44
112     return c
113
114 def find_groups():
115     i = 0
116     groups = []
117     for g in xrange(1, GCOUNT + 1):
118         group = []
119         while len(group) < 4:
120             x, y = i % COLNUM , i / COLNUM
121             if f[x][y] == 'x' or f[x][y] == 'n':
122                 i += 1
123             else:
124                 group.append((x, y))
125                 i += 1
126         groups.append((g, group))
127     return groups
128
129 ##### Functions for recognising common groups
130
131 def remember_groups(f):
132     for l in f.readlines():
133         for t in grouper(l.split(), 4):
134             if ".." not in t:
135                 a, b, c, x = map(kana2int, t)
136                 if checksum(a, b, c) is x:
137                     memory.update([(a, b, c, x)])  # list of tuples
                            , as iterable is required
138
139 def parse_telegrams():
140     telegrams = []
141     tfiles = [open(fname, 'r') for fname in TELEGRAMPATHS]
142     for tgram in tfiles:
143         content = tgram.readline().split()
144         if len(content) < SPACES:
145             raise Exception("Supplied telegram is too short")
146         telegrams.append(map(kana2int, content[:SPACES]))
147     return telegrams
148
149 def find_positions(telegram):
150     positions = [[] for i in range(ALPHACOUNT)]
151     for (n, i) in zip(telegram, xrange(SPACES)):
152         positions[n % 44].append(i)
153     return positions
154
155 def identify_group(telegram, positions):
156     groups = [a for a, _ in memory.most_common()]
157     for (g, x) in memory.most_common():
158         p = [positions[n%44] for n in g]
159         for pos in product(*p):  # all combinations of positions
160             if checksum(*[telegram[i] for i in pos[:3]]) ==
                    telegram[pos[3]]:
```

```
161                 if tuple([telegram[i] for i in pos[:4]]) in groups:
162                     yield pos
163
164 ##### End recognising common groups
165 ##### Functions for finding valid groups
166
167 def valid_colset_pair((cset1, cset2), pairedcols):
168     """ Tests if two pairs of column-sets are simultaniously
        possible
169     cset1, cset2 -- tuples of form (c1, [ci], cn) where c1 and cn
        are edges
170     pairedcols   -- iterable containing pairs of neighbouring
        columns """
171     c1a, cia, cna = cset1
172     c1b, cib, cnb = cset2
173     seta = set([c1a, cna]) | cia
174     setb = set([c1b, cnb]) | cib
175     if not seta.isdisjoint(setb):
176         for a, b in pairedcols:
177             if cna == a and c1b == b:
178                 return False
179             if cnb == a and c1a == b:
180                 return False
181         if c1a in setb and cna in setb and not seta <= setb:
182             return False
183         if c1b in seta and cnb in seta and not setb <= seta:
184             return False
185         if cna not in set([cnb]) | cib and cnb not in set([cna]) |
            cia:
186             return False
187         if c1a not in set([c1b]) | cib and c1b not in set([c1a]) |
            cia:
188             return False
189     return True
190
191 def valid_colset_comb(prod, pairedcols):
192     """ Checks if a series of column-sets are simultaniously
        possible
193     prod       -- iterable of form [(c1, [ci], cn)] where c1 and cn
         are edges
194     pairedcols -- iterable containing pairs of neighbouring columns
         """
195     for c1, ci, cn in prod:
196         cols = set([c1, cn]) | ci
197         for a, b in pairedcols:
198             if (a in ci and b not in ci and b != cn) or (b in ci
                and a not in ci and a != c1):
199                 return False
200             if (a == c1 and b not in ci) or (a == cn and b in cols)
                :
201                 return False
202             if (b == c1 and a in cols) or (b == cn and a not in ci)
                :
203                 return False
204     return all(valid_colset_pair(x, pairedcols) for x in
        combinations(prod, 2))
205
206 def valid_group(group, pos):
207     """Checks if the supplied group conforms to distances of pos,
        given the figure
208     group -- tuple of four (x,y) coordinates
209     pos   -- tuple of four positions within the telegram"""
```

```
210         pairs = []
211         for (s1, s2) in combinations((0, 1, 2, 3), 2):
212             if pos[s2] < pos[s1]:
213                 s1, s2 = s2, s1
214             pairs.append((pos[s2] - pos[s1], s1, s2))
215         hardpairs = []
216         pairedcols = []
217         for (d, s1, s2) in pairs:
218             col1 = f[group[s1][0]]
219             col2 = f[group[s2][0]]
220             if group[s1][0] == group[s2][0]:  # if it concerns symbols
                    in the same column
221                 dist = symbol_space(col1[group[s1][1]:group[s2][1]])
222                 if dist != d:
223                     return False
224                 continue
225             dist = symbol_space(col1[group[s1][1]:]) + symbol_space(
                    col2[:group[s2][1]])
226             if dist > d:
227                 return False  # if the actual distance is larger than
                        required, it is impossible
228             elif dist < d:
229                 if d - dist not in COLCOMBS:
230                     return False
231                 hardpairs.append((d - dist, group[s1][0], group[s2][0])
                        )  # put off the difficult ones for later
232             else:
233                 pairedcols.append((group[s1][0], group[s2][0]))  #
                        perfect fit, need to be neighbouring sequences
234         if not hardpairs:
235             return True
236         usedcols = []
237         for (d, c1, c2) in hardpairs:
238             colsets = [(c1, x, c2) for x in COLCOMBS[d] if c1 not in x
                    and c2 not in x]
239             if not colsets:
240                 return False
241             usedcols.append(colsets)
242         return any(valid_colset_comb(prod, pairedcols) for prod in
                product(*usedcols))
243
244     ##### End finding valid groups
245     ##### Functions for deduction
246
247     def fill_colpart(fig, x, y, v, d, lim, used):
248         c = 0
249         for j in xrange(lim):
250             if fig[x][y + j * d] != 'x':
251                 fig[x][y + j * d] = v + c * d
252                 used[v + c * d] = True
253                 c += 1
254
255     def fill_col(fig, x, y, v, used, usedcols):
256         usedcols[x] = True
257         fill_colpart(fig, x, y, v, 1, ROWNUM - y, used)
258         fill_colpart(fig, x, y, v, -1, y + 1, used)
259
260     def valid_symbol(n, na, nb, used, usedcols):
261         """ Checks to see if the symbol at position n in the telegram
                is a valid candidate w.r.t. the column it is in
262         n        -- the position in the telegram
263         na       -- the required space before the symbol
```

```
264         nb       -- the required space after the symbol
265         used     -- list denoting used/unused, per symbols in the
                telegram
266         usedcols -- list denoting which columns have been filled"""
267         if n - na < 0 or n + nb >= SPACES:
268             return False
269         if any([used[x] for x in range(n - na, n + nb + 1)]):
270             return False
271         ca, cb = 0, 0
272         for x in range(n - na - 1, -1, -1):
273             if used[x]:
274                 break
275             ca += 1
276         for x in range(n + nb + 1, SPACES):
277             if used[x]:
278                 break
279             cb += 1
280         if (ca > 0 and ca not in COLCOMBS) or (cb > 0 and cb not in
                COLCOMBS):
281             return False
282         if ca > 0 and not [x for x in COLCOMBS[ca] if not any(usedcols[
                c] for c in x)]:
283             return False
284         if cb > 0 and not [x for x in COLCOMBS[cb] if not any(usedcols[
                c] for c in x)]:
285             return False
286         return True
287
288     def print_fig(fig):
289         fix_len = lambda x: str(x) + ' ' * (2 - len(str(x))) if
                isinstance(x, int) else x + ' '
290         for a in [[fix_len(x) for x in row] for row in zip(*fig)]:
291             print a
292
293     def find_permutation(fig):
294         """ Extracts the permutation from a filled figure"""
295         permutation = zip([next(x for x in xs if isinstance(x, int))
                for xs in fig], count(1))
296         permutation.sort()
297         return [x for _, x in permutation]
298
299     def deduce(fig, pos, validgroups, groups, telegram):
300         """ Tries to fill the figure by deduction
301         fig          -- an unfilled copy of the figure
302         pos          -- the position in the telegram of the supposed '
                common' group
303         validgroups -- positions where it can be positioned
304         groups       -- a list of positions of groups in the figure
305         telegram     -- the reference telegram, as a list of symbols """
306         for _, l in validgroups:
307             used = [False] * SPACES
308             usedcols = [False] * COLNUM
309             for i in range(4):
310                 fig[l[i][0]][l[i][1]] = pos[i]
311                 fill_col(fig, l[i][0], l[i][1], pos[i], used, usedcols)
312                 for x in filter(lambda x: isinstance(x, int), fig[l[i
                        ][0]]):
313                     used[x] = True
314             todo_flag = True
315             while todo_flag:
316                 todo_flag = False
317                 queue = []
```

```
318             for _, group in groups:
319                 ps = filter(lambda (x, y): isinstance(fig[x][y],
                        int), group)
320                 if len(ps) == 3:
321                     missing = [i for (x, y), i in zip(group, count
                            ()) if not isinstance(fig[x][y], int)][0]
322                     if missing == 3:
323                         z = checksum(*[telegram[fig[x][y]] for (x,
                                y) in ps])
324                     else:
325                         z = missing_symbol(*[telegram[fig[x][y]]
                                for (x, y) in ps])
326                     queue.append((group[missing], z))
327             while queue:
328                 for ((x, y), z) in queue:
329                     candidates = [n for n in range(SPACES) if
                            telegram[n] == z]
330                     na = symbol_space(f[x][:y])
331                     nb = symbol_space(f[x][y + 1:])
332                     candidates = filter(lambda n: valid_symbol(n,
                            na, nb, used, usedcols), candidates)
333                     if len(candidates) == 1:
334                         fig[x][y] = candidates[0]
335                         fill_col(fig, x, y, candidates[0], used,
                                usedcols)
336                         todo_flag = True
337                         break
338                 else:
339                     break
340         print_fig(fig)
341         if any([' ' in col for col in fig]):
342             print "Figure incomplete"
343         else:
344             print "Permutation found!", find_permutation(fig)
345             break
346
347 ##### End deduction
348
349 def main():
350     t0 = time.clock()
351     for fname in glob.glob(MEMORYPATH):
352         remember_groups(open(fname, 'r'))
353     d = dict((tuple(map(kana2int, x)), 50) for x in NUBOERGROUPS)
354     memory.update(d)
355     telegrams = parse_telegrams()
356     for t in range(2):
357         print "Using telegram " + str(t + 1) + " as a base.."
358         positions = find_positions(telegrams[t])
359         group_gen = identify_group(telegrams[1 - t], positions)
360         groups = find_groups()
361         for pos in group_gen:
362             validgroups = [(g, group) for (g, group) in groups if
                    valid_group(group, pos)]
363             print "  Trying", pos
364             if validgroups:
365                 fig = deepcopy(f)
366                 deduce(fig, pos, validgroups, groups, telegrams[t])
367     print time.clock() - t0, "seconds process time"
368
369 if __name__ == "__main__":
370     main()
```