Automatic Key Structure Extraction

Crina Tudor, Beáta Megyesi
Dept. of Linguistics and Philology
Uppsala University
Sweden
first.last@lingfil.uu.se

Benedek Láng
Dept. of Philosophy and History of Science
Budapest University of Technology
and Economics, Hungary
lang@filozofia.bme.hu

Abstract

Studying original cipher keys constructed throughout history gives important insights into encryption methods and cipher systems. We can study the type of encryption used, the code structure and their corresponding plaintext entities, be it letters, morphemes, words, or named entities. The insights can lead us to better decryption methods, and the understanding of the development of historical ciphers. In this paper, we present a tool for automatic key structure extraction that describes the symbol system and the code structure along with the encoded plaintext features and the mapping between the two. The tool is aimed at the empirical study of historical keys given transcribed keys.

1 Introduction

In historical cryptology, the key (also called clavis), according to the definition of Kahn (1996), "specifies the arrangement of letters within a cipher alphabet". Keys exist — as Kahn goes on — "within a general system and control that system's variable elements." Of course, a key is defined differently in transposition ciphers where it is a pattern of shuffling and in cipher machines where it is a disk alignment. In this article, however, we will concentrate on early modern monoalphabetic, homophonic and polyphonic ciphers, so the above shortened approach will be followed. In such ciphers, the cipher alphabet is often complemented with a nomenclature table, a list of codewords, where ciphertext symbols stand for words, common notions, names, geographical unities, bigrams, etc. Nulls, i.e. cipher characters without meaning, are also often added to the system (Láng, 2018). We will call "key" those —- usually one page —- tables that comprise the cipher alphabet, the code words and the nulls. It should be added, that in classical cryptology, the sender and the receiver must use the same key while in the post WWII era, this requirement has been changed.

Being able to automatically identify and describe key structure can prove to be useful if we want to conduct an extensive study on the structure of keys, for example from a chronological perspective. An automatic method would be much more effective if we are looking into the structure of keys over a longer period of time or if we are targeting a certain era. This way, we could investigate changes in the structure of keys throughout time, and see in which way the means of encryption have evolved.

In this study, we present a tool that automatically extracts the inner structure of historical original keys, taking into account the symbol system used for encryption, the code structure and the encoded plaintext entities as defined in the key.

Despite the vast advances when it comes to computational decipherment techniques, there currently seems to be a lack of large scale systematic studies that focus on keys. We believe that it is vital to start exploring historical cipher keys by means of computational methods, as well as to develop a way to classify keys. Other than Kahn (1996) there are not many other comprehensive cryptological studies, and even his is mostly targeting ciphers rather than keys. Given the fact that historical cryptology is a relatively young discipline, there can still be differences in notation from one study to another. It is for this reason that we will use Kahn's work as a basis for the terminology we employ in this paper.

2 Key Structure Extraction

First, we need to transform the key image into a text file by transcribing the key, then extract the key structure given the transcribed text file. Be-

low we give a brief overview of the transcription of keys, followed by a description of the key structure extraction.

2.1 Transcription

In order to be able to use computational methods for this purpose, we must first establish a transcription standard. This way, we ensure a stable and uniform basis to provide a reliable comparison across keys.

Our proposed method makes use of plain text files (".txt") containing the transcription of the original key document. The transcription replicates the original document as closely as possible, both in terms of its structure as well as its content. In large terms, we follow the same guidelines (Megyesi, 2020) as those used in the DECODE database (Megyesi et al., 2019), and expand on them in order to adapt to the specific key structure.

The header of our transcription file consists of metadata which we extract from the DECODE database (Megyesi et al., 2019). The metadata is preceded by a number sign ("#"), followed by the name of the field (e.g. catalog name, language etc.) and the corresponding information: "#CATALOG NAME: BAV_Barb.lat_6960-17". Each new type of metadata is transcribed on a new line.

We then proceed to transcribe the content of the original key, following its layout, generally top to bottom, left to right. The transcription of key entries consists of `ciphertext - plaintext` pairs, where ciphertext represents the symbols used to encode the plaintext message. For those cases where the method of encryption is either homophonic, i.e. a plaintext entity can be encoded by several codes as depicted in Figure 1, or polyphonic substitution, i.e. ciphers in which one ciphertext symbol is used to encode several plaintext elements, as shown in Figure 2, we use the logical operator "|" ("or") as a way to separate between several ciphertext or plaintext entries, such as illustrated in the following example:

- *72|37 - a* → the letter "a" is encoded by either "72" or "37"
- *24 - a|m* → the number "24" can either encode the letter "a" or "m"

For the transcription of those symbols that are not part of the extended Latin character set, we follow the DECODE standard of transcribing graphic

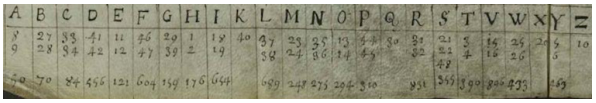

Figure 1: Example of plaintext alphabet encoded by means of homophonic substitution, extracted from a key from the UK (TNA-SP106, 2020c). Plaintext units on the top row, codes on the bottom row.

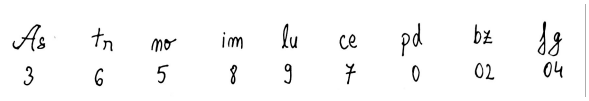

Figure 2: Example of plaintext alphabet encoded by means of polyphonic substitution, inspired by a key from the 16th century (ASV-ARM-XLIV-7, 2016). Plaintext units on the top row, codes on the bottom row.

signs according to their name in the Unicode database (Unicode, 2019).

More advanced keys can also have codes that do not map to any kind of lexical unit, and which are commonly referred to as "nulls" (Kahn, 1996). For these cases, we use the tag "<NULL>" as a placeholder for plaintext (e.g. "73 - <NULL>"). Sometimes keys can be incomplete where we find codes without any plaintext elements attached. These empty plaintext elements are not indicated as nulls, they are just missing. To be able to distinguish those from the intended nulls, we transcribe the empty entity as "<EMPTY>" .

Some keys also contain portions of text that are not part of the encoding scheme, such as section headers or notes from the original writer of the document. We refer to such information as "clear-

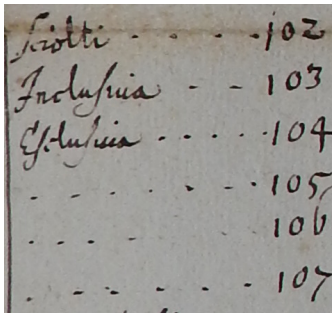

Figure 3: Example of codes mapping to empty plaintext inspired a key from 1692 (ÖstA HHStA Staatzkanzlei Interiora, Kt. 13. Fasc. 20. f. 22., 2020).

text" (Megyesi et al., 2019) and we mark it with its own respective tag, followed by a language ID and the body of text, namely <CLEARTEXT LANGID TEXT>. If the transcriber is able to identify the language of the cleartext, they can use a two-letter ID to mark it in the cleartext tag according to the ISO 639-1 nomenclature (Byrum, 1999). Otherwise, they can replace the language ID with the letters "UN" (i.e. "unknown").

The transcription file can also contain comments from the person who is transcribing the original document. These can be remarks about the quality of the document, such as bleed-through, ink stains or torn paper, or simply general remarks about the transcription process. An example comment can look as follows: "#COMMENT: torn paper, some symbols were lost".

Given the transcription file, we can proceed to investigate the components and the structure of the key.

2.2 Automatic Extraction

In order to be able to automatically extract statistical information from the transcription file, we write a Python script that analyses the text file and returns a detailed analysis of its content. The script can be run in a terminal window, with the file name as its argument, and then it prints out the results in four main parts.

Other than statistical information, the script also returns the metadata present in the file. This generally includes the original file's catalog name, the plaintext language (if recognizable), whether or not the transcription is complete or partial, and the transcription time. Optionally, the metadata can also include information about the type of entities in the nomenclature, should they be provided by the transcriber (e.g. names, towns, common words, morphemes etc.). The transcription file can also include additional comments from the transcriber, regarding the transcription process, the state of the original document or its layout, for example. If the script encounters such comments while processing the file, it will not print them as with the metadata, but it will inform the end user of their existence so that they can check the original file in case the comments might be relevant to them.

2.2.1 Symbol Set

The first major section of our output focuses on the analysis of ciphertext symbols, beginning with the type of symbols used for encryption. Here we differentiate between 3 major types, namely Latin alphabet, digits, and graphic signs. By "Latin alphabet" we refer to those cases where the individual letters are used for encryption of plaintext, as shown in Figure 4.

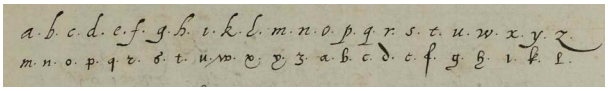

Figure 4: Example of plaintext alphabet encoded by means of simple substitution, extracted from a key from 1596 (TNA-SP106, 2020b). Plaintext units on the top row, codes on the bottom row.

In this context, we use "graphic signs" as an umbrella term that refers to any kind of symbol representation that is not part of the Latin alphabet (a-z, A-Z) or digits (0-9). These symbols can be Roman numerals (I-X), zodiac symbols, alchemical signs or any other symbols. When detecting such symbols, the script can further categorize and identify specific sets, as grouped in the Unicode standard (Unicode, 2019). Any other miscellaneous symbols will simply be referred to as "graphic signs" in the output.

2.2.2 Code Structure

The next section of the output looks more in-depth into the internal structure of the ciphertext symbols, which we will refer to as unigraphs, bigraphs, trigraphs and 4+graphs. What counts as unigraphs are usually digits, isolated letters or graphic signs. Digraphs, trigraphs and 4+grams are usually clusters of 2, 3, or 4+ symbols of any of the aforementioned kinds, respectively.

In the cases where digits are included in the ciphertext representation for a key entry, the script will calculate how many of the total ciphertext items are digits. With respect to Figure 5, the output for this section would look as follows:

unigraphs: 6

out of which digits: 4

Moreover, the encountered ciphertext symbols are mapped to an existing database of symbols so that the user can also get information on how many items have been matched against those already recorded, and how many are new. If any new symbols are found, they will be printed on screen, along with their count.

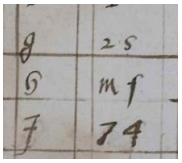

Figure 5: Example of plaintext alphabet encoded by both letters and numbers, extracted from a key from 1569 (TNA-SP106, 2020a). Plaintext units on the left, codes on right.

2.2.3 Plaintext Analysis

We then move on to investigate plaintext units. Similarly to ciphertext, these are separated in unigrams, bigrams, trigrams, and 4+grams. We do add, however, a 5th category of plaintext, which is that of nulls.

For the most part, the type of plaintext unigrams that we find in keys are either letters or digits, even punctuation in some cases. Bigrams and trigrams are commonly either non-lexical units (e.g. double letters that occur frequently in the language of encryption, such as "ll" or "ee" in English, syllables, morphemes etc.), or short function words ("at", "for", "to", "and" etc.). Under 4+grams we include those units that consist of 4 or more elements, such as longer function words or nomenclature entries, which can consist of names, places, common words. Nomenclatures can also include words that are specific to the lingo used in the topic the key was designed for - army terms in military correspondence, for instance.

Nulls are important to keep track of even though they do not always carry any lexical significance. Nulls can be vital in the process of decipherment, as they might be markers for whitespace or word delimitation, as well as fillers in null ciphers (Kahn, 1996), where only every n^{th} element carries significance.

In cases where only the code is written without any attached plaintext (<EMPTY>), the program indicates the number of occurrences of such empty placeholders.

2.2.4 Code Distribution

Once we described the code and plaintext structure, we can analyze the distribution of ciphertext symbols to plaintext elements from several different perspectives.

Cipher type

By analyzing the ratio of plaintext elements to ciphertext units, the script can differentiate between three different types of encoding, namely simple substitution, homophonic substitution or polyphonic substitution. If the encryption method is consistent throughout the key, we print one of these three possible outputs. There are cases, however, where more than one method is used within the same key. A common case is that the alphabet will be encoded by means of homophonic substitution, while the nomenclature will only use simple substitution. For these instances, we will print all types that are used in the key.

Code type

Here we look into the ciphertext symbols only, and determine whether the same type of ngraph was used throughout the whole key or not. Assuming that we are dealing exclusively with bigraphs we can say that the code distribution is fixed. Else, if the key mixes bigraphs and trigraphs, for example, we say that the length is variable.

Establishing the difference between fixed and variable lengths of code is meaningful because, while ciphers with fixed distributions are easier to crack, those who use different ngraph levels can make it more difficult to isolate each individual code from the body of the cipher, and therefore the decryption process becomes more challenging.

Codes encoding plaintext

Next, we look into the number of codes that are used in order to encode a certain level of plaintext. If we have *"unigrams: 30"* as an example output, this would mean that the key uses 30 codes to encipher plaintext unigrams. A potential way to interpret this, assuming that the only unigrams we are dealing with are alphabet letters, would be that the majority of the alphabet entries map to only one code, but that there are some letters that map to several ciphertext symbols. Oftentimes, these letters would be the most frequent ones, which in turn makes the frequency distribution more uniform.

Moreover, the information about the number of unique codes encoding plaintext entities also indicates the complexity of the cipher — the more codes, the harder the decryption of the cipher would be.

Ciphertext:plaintext distribution

For the last feature type, we look at the ratio of ciphertext to plaintext units, for four levels of plaintext, namely alphabet, nomenclature, and empty

plaintext elements, being it nulls or placeholders. For each section, we display 4 different possible distributions, which we illustrate below:

- *Alphabet*
  1:1 16
  2:1 5
  3:1 0
  4+:1 3

Keeping in mind that the pairs represent the "ciphertext:plaintext" distribution, in this exact order, this example output tells us that there are 16 instances where one ciphertext symbol maps to one plaintext unigram, 5 instances where a plaintext unit has 2 ciphertext representations, and 3 instances where one plaintext element maps to 4 or more ciphertext symbols.

For polyphonic ciphers, the distribution scheme could be reversed (i.e. 1:1, 1:2, 1:3, 1:4+), to show exactly how many plaintext elements map to one single ciphertext unit.

If the distribution is uniformly 1:1 for a certain category, be it alphabet, nulls, or nomenclature, the script will simply print a message stating this fact, since printing the whole distribution scheme for such cases would be superfluous.

2.3 Error Analysis

In order to ensure that the key structure analysis is as accurate as possible, we took the additional step of implementing an error analysis part in our code, that aims to check that the input text follows the same format that we describe in Section 2.1. This way, not only do we make sure that the analysis is accurate, but also that the transcriptions we analyse are all uniform so that we can reliably compare keys among each other.

We differentiate between three major types of user errors that the script can identify, locate, and provide suggestions on how to address them. The most common types of errors that we can encounter are either related to metadata that is not mark correctly or to the fact that the ciphertext and the plaintext are not separated properly. The other type of formatting error that we can detect, and which can be difficult for the user to identify on their own, is the accidental use of tab spacing instead of regular spacing. Even though this might not happen as often, it would negatively impact the way the key statistics are calculated.

3 Conclusion & Future Work

In this paper, we presented a tool for the automatic analysis of original historical keys including a common transcription scheme applied to various types of keys. The extracted features include a description of the symbol system used to encode various types of plaintext entities, the code structure, the type of encoded plaintext entities, and the mapping and relation between the code and plaintext entities.

In the future, we are considering expanding to another output format in addition to the already existing plain text one, presented in the Appendix. A viable alternative would be to create a corresponding JSON file, which would in turn allow for easier data manipulation and processing. We plan to add automatic language identification of the plaintext to be included in the automatic structure extraction of keys. Lastly, we would also like to test the tool on a large number of keys of various types, and expand it to allow comparisons across several keys simultaneously.

Acknowledgments

This work has been supported by the Swedish Research Council, grant 2018-06074: DECRYPT - Decryption of historical manuscripts.

References

ASV-ARM-XLIV-7. 2016. Segr.di.Stato-ASV-ARM-XLIV-7-1 Archivio Apostolico Vaticano. DECODE link: https://cl.lingfil.uu.se/decode/database/record/205.

John D. Byrum. 1999. Iso 639-1 and iso 639-2: International standards for language codes. iso 15924: International standard for names of scripts.

David Kahn. 1996. *The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet*. Scribner.

Benedek Láng. 2018. *Real Life Cryptology*. Amsterdam University Press.

Beáta Megyesi, Nils Blomqvist, and Eva Pettersson. 2019. The DECODE Database: Collection of Ciphers and Keys. In *Proceedings of the 2nd International Conference on Historical Cryptology, HistoCrypt19*, Mons, Belgium.

Beáta Megyesi. 2020. Transcription of historical ciphers and keys. In *Proceedings of the 3rd International Conference on Historical Cryptology, HistoCrypt20*.

TNA-SP106. 2020a. Reproduced image based on TNA-SP106/1-ElizabethI-f53(0056) National Archives in Kew, UK. DECODE link: https://cl.lingfil.uu.se/decode/database/record/331.

TNA-SP106. 2020b. Reproduced image based on TNA-SP106/2-ElizabethI-f58(0069) National Archives in Kew, UK. DECODE link: https://cl.lingfil.uu.se/decode/database/record/345.

TNA-SP106. 2020c. Reproduced image based on TNA-SP106/6-CharlesII-(0030-0031) National Archives in Kew, UK. DECODE link: https://cl.lingfil.uu.se/decode/database/record/428.

Unicode. 2019. The unicode® standard version 12.0 – core specification.

ÖstA HHStA Staatzkanzlei Interiora, Kt. 13. Fasc. 20. f. 22. 2020. Reproduced image from Österreichisches Staatsarchiv, Haus-, Hof- und Staatsarchiv, Staatskanzlei Interiora, Chiffrenschlüssel, Kt. 13. Fasc. 20. 22. DECODE link: https://cl.lingfil.uu.se/decode/database/record/1199. The picture has been reproduced by the permission of the Österreichisches Staatsarchiv, Haus-, Hof- und Staatsarchiv, all rights reserved.

A Appendix - Example Output

An example output is illustrated for a reproduced key from the Österreichisches Staatsarchiv, Haus-, Hof- und Staatsarchiv (ÖstA HHStA Staatzkanzlei Interiora, Kt. 13. Fasc. 20. f. 22., 2020), shown in Figure 6.

```
Metadata
#CATALOG NAME: ÖStA_HHStA_Stk_Int_
Chiffrenschlüssel_fasc_20_22
#LANGUAGE: IT
#STATUS:complete
#TRANSCRIPTION TIME: 4h 10min

Cipher symbols:digits

Total number of unique ciphertext
symbols:337
     unigraphs:9
          out of which digits:   9
     digraphs:90
          out of which digits:  90
     trigraphs:238
          out of which digits: 238
     4+graphs:                   0

No new ciphertext symbols were found.

Total number of unique plaintext units: 249
     out of which unigrams: 20
     out of which bigrams:   2
     out of which trigrams:  3
     out of which 4+grams: 223
     out of which nulls:     1
     out of which empty:     1

Code distribution

Cipher type:mixed
(homophonic substitution, simple substitution)

Code type:variable length

Number of codes encoding plaintext
     unigrams:44
     bigrams:  2
     trigrams: 3
     4+grams:223
     nulls:   24
     empty:   41

Distribution according to plaintext type
(ciphertext:plaintext)
     1. Alphabet
        1:1     1
        2:1    14
        3:1     5
        4+:1    0
     2. Nomenclature
     The nomenclature has a uniform 1:1
     distribution.
     3. Nulls
        4+:1     1
     4. Empty
        4+:1     1


The transcription file contains comments
from the transcriber and/or transcriptions
of cleartext from the original document
which are not included in the statistics
above. Please check the transcription file
for more details.

Total number of ciphertext symbols matched: 337
```

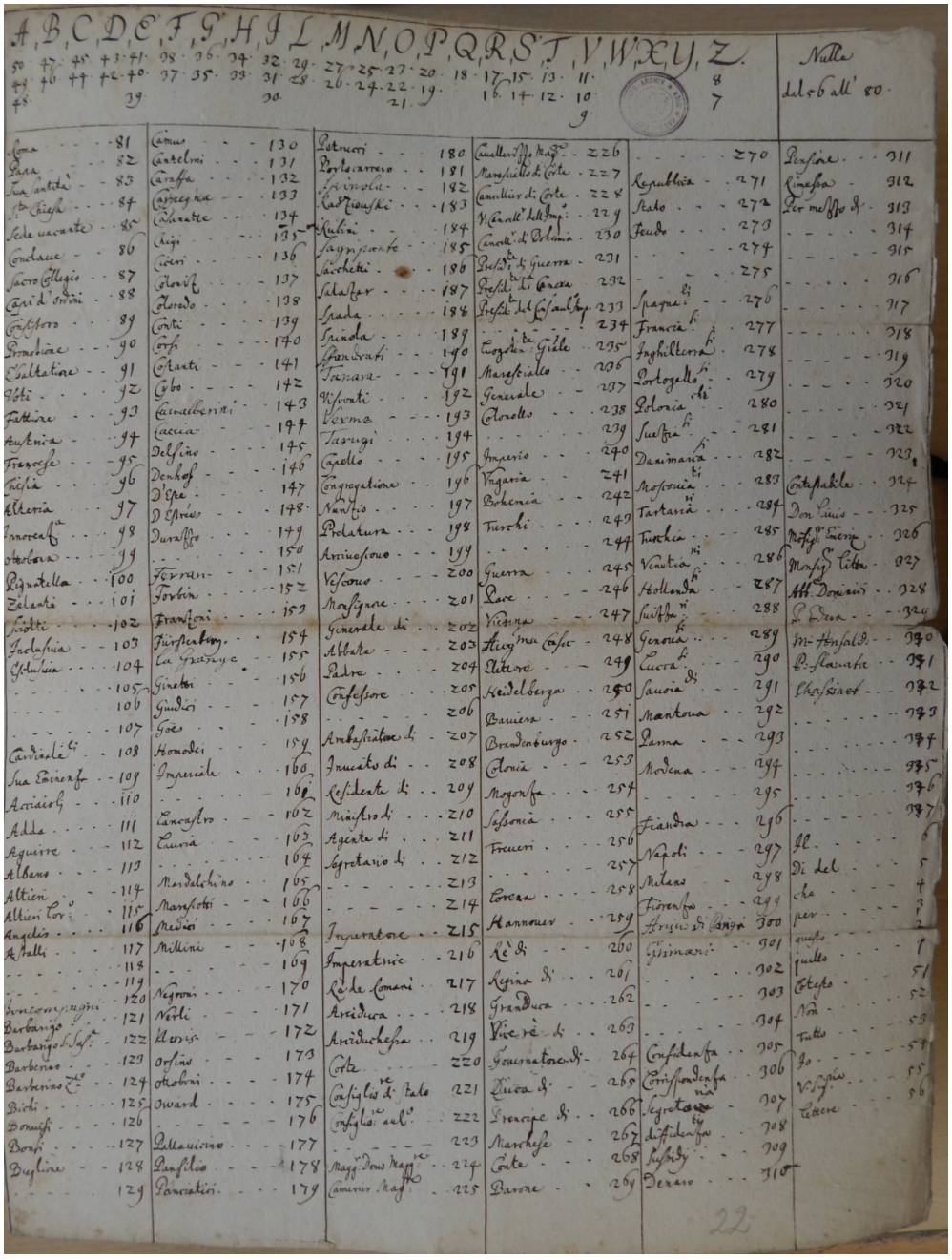

Figure 6: Original document of a key (ÖstA HHStA Staatzkanzlei Interiora, Kt. 13. Fasc. 20. f. 22., 2020).