Brandenburg University of Technology
Cottbus-Senftenberg
Chair of IT Security

Introduction to Cyber Security
Practical Exercise Class
Winter Term 2024/2025

Introduction to Cyber Security
– Secret Key Cryptography –

Deadline: 14th November, 2024

Introduction

The main goal of cryptography is confidentiality of some plaintext secret, generally achieved through encryption. However, even if the resulting ciphertext may appear random at first, a weak algorithm often leaves many avenues to analyze and recover its plaintext and/or the encryption keys. This first practical exercise is a study of historical and modern cryptographic algorithms and methods to analyze encrypted text. There is a wide variety of literature on basic cryptographic methods which may be of interest to you [4, 5, 8].

As an introduction into the world of cryptography, we will begin by working with the open source e-learning tool *CrypTool*. The tool serves as a demonstration for many cryptographic algorithms. Once a basic understanding of the historical and modern cryptographic algorithms has been obtained, it is time to analyze some encrypted files on your own. To this end, we will consider different kinds of scenarios typically covered in the field of cryptography. This includes ciphertext-only attacks as well as (partially-)known plaintext attacks. In the setting of ciphertext-only attacks, we will implement a brute-force search to break an AES encryption with a weak password. Later, we will improve the naive brute-force by utilizing a method called hill climbing to accurately reverse a mono-alphabetic substitution. On the side of (partially-)known plaintext attacks, we will analyze an encrypted zip container. Here, the predictable header information of the file format specification serves as a partially known plaintext.

Notes

Please note that these practical tasks assume basic knowledge to have been learned in previous studies. For example, this includes mathematical knowledge, such as matrix calculus or basic statistical methods. It also holds true for topics that will be covered in the lecture or exercise classes of Introduction to Cyber Security at some point, but have not yet been held. If you find yourself missing the knowledge required to solve this task sheet, you must attain it on your own through the process of self study.

Keep in mind that the practical tasks are mandatory to get approval for the final exam.

1 Preparation

To get familiar with the world of cryptography, we shall begin by examining a selection of historic and modern ciphers, using *CrypTool* as a guide. *JCrypTool* provides many historical and modern cryptographic algorithms as well as adequate analysis methods to study and break weakly encrypted messages. The tool also comes with visualization capabilities, which makes *CrypTool* a handy tool to get a better understanding of the complex world of cryptography. It is recommended that you run through the steps discussed here before starting the actual tasks listed in section 2.

There are several versions of *CrypTool*. In this lab we will focus on the Java implementation called *JCrypTool*, which can be downloaded from the official website [3]. For simple demonstrations, you may also take a look at the online version *CTO*. This version runs in your web browser and can be reached at `https://www.cryptool.org/en/cto/`. However, do note that its capabilities are more limited compared to the Java version.

In the following, several scenarios will be discussed. Try to *understand the analysis/attack for every scenario* presented below and think about the reasons *why these attacks were possible* in the first place. While walking through the different scenarios, the overall goal is to gain an adequate understanding of how the cryptographic algorithm works, as well as how the cryptanalysis is mounted. Simply “clicking” through the examples will probably not be enough to gain this understanding.

Please note that while no submissions have to be made for this first part of the task sheet, you are still expected to be familiar with these introductory questions once you submit your solutions. **You may be asked questions about these scenarios during the lab defense, which may influence the points you attain.**

1.1 Caesar Cryptography

Inform yourself about the classical Caesar algorithm and possible methods of analysis. Consider both known-plaintext and ciphertext-only attacks. Which fundamental fact is used to break the Caesar cipher?

Hints:

1. To de- and encrypt with the Caesar cipher using *JCrypTool*, choose the menu *Algorithms* $\rightarrow$ *Classic* $\rightarrow$ *Caesar*. In the online version of *CrypTool* you can find the Caesar cipher in the "Ciphers" menu.
2. To find additional resources you can use either the built in help function of *JCrypTool* or the tutorial provided by the online tool. Keep in mind that external resources are allowed as well.
3. To calculate and visualize frequency use *Analysis* $\rightarrow$ *Entropy Analysis*.

1.2 Mono-alphabetic Substitution

The monoalphabetic substitution is a generalization of the Caesar algorithm. It substitutes letters from the plaintext based on an arbitrary, but fixed permutation / "reordering" of the alphabet. However, there is still the possibility for an easy cryptanalysis. Inform yourself about the monoalphabetic substitution, analysis techniques that are available and why these techniques work.

Hints:

1. To analyze the cipher, you have to choose the menu entry *Analysis* $\rightarrow$ *Substitution Analysis*.
2. If you know the language of the text to analyze, it is sensible to utilize this knowledge in the analysis dialog.

1.3 Vigenère Cryptography

The main vulnerability in Caesar cipher is the monoalphabetic substitution. Thus, Vigenère extended the idea of the Caesar algorithm by using different shifts of the same alphabet during the encryption. Inform yourself about how the polyalphabetic substitution is being realized. How can it still be analyzed? To this end, inform yourself about the Kappa test (also known as the Friedman test) as well as the Kasiski test. Which one is used in the current implementation of *JCrypTool*?

Hints:

1. To de- and encrypt the Vigenère cipher with *JCrypTool* choose the menu *Algorithms* $\rightarrow$ *Classic* $\rightarrow$ *Vigenère*. In the online version of *CrypTool* you can find the Vigenère cipher in the "Ciphers" menu.
2. To find additional resources you can use either the built in help function of *JCrypTool* or the tutorial provided by online tool. Keep in mind that external resources are allowed as well.
3. Note that there is a built-in Vigenère-breaker in *JCryptTool*, which may be helpful to understand the methodology behind the cryptanalysis of Vigenère ciphers.

1.4 XOR Cryptography

While working with binary data, the XOR encryption is a common and performant algorithm[1]. In *CrypTool*, we can apply an XOR encryption, e.g., to compressed data or a hex file. Think about the knowledge required to analyze an XOR cipher, and bring at least one example of such knowledge.

this is just XOR the key with the input, to change its format.
super simple.

1.5 Elliptic Curve Cryptography

Elliptic curve cryptography (ECC) is a method used to date, but the theory of elliptic curves dates way back. Already at the end of the 19th century, all mathematical foundations used in contemporary ECC were known. Here we will look at the properties of elliptic curves over the field of the real numbers, as well as finite fields. To get familiar with elliptic curves, you can start using the visualization provided by *JCrypTool*,

```
Visuals -> Elliptic Curve Calculations
```

Try to understand the basic properties of elliptic curves over the real numbers and how the point addition can be realized geometrically. Why is this not possible over a finite number field? Compare the graphs of the curve in the finite (discrete) field and the continuous case.

[1] Note: The XOR cipher can be seen as Vigenère cipher over the alphabet $\{0, 1\}$.

2 Tasks

Task 1: Ciphertext-only Analysis

Cryptanalysis based on knowledge of the ciphertext only, i.e. without knowing any part of the plaintext, is the most difficult type of analysis. A method that can always be used is exhaustive key search, where every key is tried out. For this lab you were provided a file called `Subst-Rijndael.crypt`, which contains the binary data of a text that has been encrypted twice, using monoalphabetic substitution first and then the AES algorithm in CBC mode. The AES encryption used a weak key, where from the 128 bit key only the first 16 bits were chosen and the rest of the key was padded with zeros. Here, the initialization vector (IV) is the first block of the ciphertext.

Task 1.1: Brute-force Weak AES

Thanks to the small key space, a brute-force attack on `Subst-Rijndael.crypt` becomes a reasonable technique to break the outer AES encryption layer. Write a short prototype script to automate the attack. To this end, you must find a criterion that allows you to automatically distinguish the right key from wrong ones. The decryption shall be computed and stored in `Subst.txt` (in plaintext format) for further processing in the next step. Furthermore, you should save the discovered AES key in `aes.key` as a hexadecimal text string.

Hints:

1. Notice that there are already many implementations (libraries) available to apply AES encryption and decryption, e.g., *PyCryptodome* to interface with Python.
2. Take a look at Shannon entropy. How can it help to find out the correct key among all possible combinations?

Task 1.2: The Hill-Climbing Method to Break Monoalphabetic Substitution

To break the inner monoalphabetic substitution, a naive brute-force attack would require to test up to $26! \approx 4 \cdot 10^{26}$ possible permutations of the alphabet (assuming the alphabet `A-Z` only). Assuming each probe takes just $1\,\mu s$, approximately 10^{13} years would be required and you would likely miss the deadline for this task. Although frequency analysis can help us guess the scheme of the monoalphabetic substitution quickly, there is no guarantee that the key gained from this is 100% correct. Therefore, an even more sophisticated method is required.

The monoalphabetic substitution corresponds to a permutation of the alphabet, which can be represented by a substitution defined in a lookup table like the example depicted in table 1.

reference alphabet	A	B	C	D	E	F	G	H	I	J	K	$\cdots$
key alphabet	Z	H	Q	P	L	A	G	I	Y	X	M	$\cdots$

Table 1: Exemplary substitution key for monoalphabetic substitution

From combinatorics, it is known that starting from an arbitrary permutation of a finite alphabet, we can define any key permutation by applying a finite number of transpositions (essentially, swaps of two characters in the alphabet). So assuming we start with the initial permutation I and the substitution permutation (i.e., the key) used for the monoalphabetic encryption is K, then there exist transpositions $T_1, T_2, \ldots, T_n$, such that:

$$K = T_n \circ T_{n-1} \circ \cdots \circ T_1 \circ I.$$

In other words, the encryption key can be found by finding the correct transpositions T_i, starting from an initial permutation I. That way the cryptanalysis becomes an iterative approach, where we successively test substitutions I, $T_1 \circ I$, $T_2 \circ T_1 \circ I$, etc. and evaluate their impact until we find K. It can be assumed that a used transposition T_i is correct if it increases the scored goodness of the decryption. To formalize this idea into an algorithm called hill climbing, three critical questions have to be answered:

1. How to choose the *initialization key* I, i.e., the permutation to start with?
2. How to derive from one key, e.g., I, the following keys $T_1 \circ I$, $T_2 \circ T_1 \circ I$, …?
3. How to measure the quality of each permutation, i.e., did we improve from $T_i \circ \cdots \circ T_1 \circ I$ to $T_{i+1} \circ T_i \circ \cdots \circ T_1 \circ I$?

Your task is to answer these questions and to implement, as well as to mount, the hill climbing approach to break the monoalphabetic cipher applied on `Subst.txt`. For a successful completion of this subtask, you have to implement the hill-climbing approach with a sensible key initialization routine and rules to derive a suitable next key. Submit your prototype implementation as `break_monoalphabetic.{py,cc,...}`, or, if it consists of several source code files, submit it in a folder called `break_monoalphabetic`. Additionally, you shall submit the plaintext obtained as `Plain.txt`, as well as the substitution key found by your attack as `subst.key` (as a string, for example: `VFTWIJNUCQAGDKLRZMHBSOEPXY`). To guide you through this cryptanalysis, the following paragraphs will provide help regarding the questions mentioned above.

Initialization key. The actual performance of the hill climbing attack will be heavily effected by the initial key from where the algorithm starts. To this end, it makes sense to not just select a random permutation. Instead, we should select a permutation which likely is close to the actual permutation, such that less transpositions are required. For the sake of this lab your task is to initialize the key by frequency analysis of the ciphertext. This process of initialization has to be implemented in an automated way. You may want to implement a function `init_key(cipher)` serving this purpose.

Key derivation algorithm. In the context of the monoalphabetic substitution, the key derivation function is rather simple. As has already been observed, the final key can be represented as a composition of single transpositions of the alphabet. To this end, it makes sense to derive the next key to probe from the given one by applying one single transposition, i.e., we randomly swap two letters in our lookup table. For example, if `A` was initially mapped to `T` and `C` was mapped to `F`, then after the transposition, `A` should be mapped to `F` and `C` to `T`, respectively.

Measuring the quality of the update. The crucial part of hill climbing is the measurement function according to which we decide whether a key derived makes an improvement or not. For the sake of this lab, this part will be given. We provide a scoring function based on so called *n-gram* analysis. Implementations will be provided in C/C++ and Python (`ngram_score.{cpp,h}` / `ngram_score.py`). If you prefer a different language, you will have to translate the given source code on your own. Although this part is already implemented for you, it is expected that you understand how and why this proposed/implemented technique works. Questions about it may be asked during the consultation.

Done

Task 2: Find the Right Algorithm

You are given the ciphertext `cipher.crypt`, and the two opposing sentences contained in `plaintext1.txt` and `plaintext2.txt`. Which of these two messages could actually be hidden here? It turns out that it might be both!

Your task is to find an encryption scheme (Gen, Enc, Dec) and two keys k_1 and k_2 which fulfill the requirements:

- $\text{Dec}_{k_1}(c) = p_1$ and
- $\text{Dec}_{k_2}(c) = p_2$,

i.e., the decryption of `cipher.crypt` results in the plaintext of `plaintext1.txt` if k_1 is being used and in `plaintext2.txt` if k_2 is being used. Submit both keys k_1 and k_2 in the form of `k1.key` and `k2.key` files containing the binary data. Also submit a small writeup (`Writeup.md`) including the algorithm you used and how you obtained the desired results. If you develop a script to aid your purposes, please submit it as well as `cipher_keys.{py | cc | ...}`. For this task, it does not matter if you develop your own script or use a publicly available tool. However, in both cases make sure to guarantee access to the algorithm used, or else the task can not be assessed and no points can be awarded.

XOR encryption algo

Task 3: Partial Known Plain-Text Analysis

done :)

Along with this task sheet you have received the file `XOR.zip.crypt`. This zip file was encrypted with a 96-Bit-long XOR key. The compression was done with the default options using the linux `zip` command-line tool (version 3.0). The zip file contains *no comments*. Your task is to decrypt the given file. Submit the key used for the encryption (encoded as a hexadecimal string) as `XOR.key`, as well as a brief walkthrough of how you have approached the cryptanalysis. You are allowed and encouraged to support your solution process by self-written scripts, which you must submit as well.

Hints:

1. While the fact that the zip file was created using the `zip` utility might be interesting to you, the task is solvable without this information
2. Consider the encoding used in the `XOR`-algorithm if you think about using JCrypTool.
3. Sometimes it is better to reimplement a basic algorithm like `XOR` instead of converting a given input to satisfy existing programs requirements.
4. While analyzing/breaking the encryption, consider a known-plaintext attack. Which parts of the file are known for such an attack?
5. The task may not be solved in one iteration. Assuming a part of the key is known only, you can use zeroes for the missing bytes of the key. So if, for example, you are sure the last two bytes are `0xFF 0xFF`, then the key `0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0xFF 0xFF` may be used in a first iteration. Think about why this fact is important for your analysis.

Bonus Task: Mathematics of the Hill-Cipher

The Hill cipher is vulnerable to a known plain-text attack. However, sometimes it is even enough to know only parts of the plain-text. Consider the cipher text `VBIDUXANLFPYPUSFGPIDTYUHDY`.[2] Assume that the key-length used was four and the language of plain-text is English. Your task is to analyze the cipher by determining the used key as well as to decrypt the complete cipher. Here-fore, you may want to consider the following steps:

1. Analyze the cipher text by checking the letter frequencies. Since the key-length is assumed to be four, 2-grams are of special interest. Think about why this is the case?
2. Sometimes, the consideration of n-grams is not enough to decrypt a ciphertext. In this case, additional knowledge about the plain text is required if we do not want to test out all possible substitutions or keys. Let's assume we already know that "the fox" is part of the plain-text. I.e., one of the listed cases holds true:

Cipher	V	B	I	D	U	X	A	N	L	...
Plain 1	T	H	E	F	O	X	...			
Plain 2	...	T	H	E	F	O	X	...		
Plain 3	...	...	T	H	E	F	O	X	...	

Table 2: Possible plain-text to cipher mappings assuming "the fox" is part of the plain-text.

3. To find a suitable analysis method, let's limit our scope once more: Assume that *Plain* 1 corresponds to the correct mapping. I.e., when encrypting, it holds $(T|H) \mapsto (V|B)$; $(E|F) \mapsto (I|D)$; $(O|X) \mapsto (U|X)$. How does this help to break the cipher?
4. Now, using these assumptions, calculate the key and the plain-text by mathematical means! Why can this method be used to efficiently break the ciphertext, even if the exact position of the partial plaintext were unknown? Submit they key as `key.txt` and the plaintext as `plain.txt`, along with any scripts / writeups you wrote related to this task.

Hints:

1. An overview of letter frequencies and $n-$ grams can be found at, e.g., [6].
2. You may want to revisit mathematical fundamentals relating to linear algebra, especially matrix calculus, before solving this task.
3. You are allowed to use computer algebra tools for all calculations necessary, or you can do them by hand.

Please Note: Points for the bonus task can only be awarded if you have already gained at least 5 points for the rest of this task sheet. If too many questions arise during the lab defense and we run out of time, the bonus task can not be graded.

[2]The plain-text is upper-case and white spaces were omitted.

Submission and Lab Defense

Submit **a single** compressed zip file named *firstname_lastname*.zip, where you replace *firstname* / *lastname* with *your own* first / last name. The zip file contains the solutions for the tasks and its contents must be named and structured as follows. Of course, you are allowed to submit additional files and scripts.

```
submission.zip
├─ task_01
│  ├─ aes.key
│  ├─ subst.key
│  ├─ bruteforce_aes.{ py | cpp | * }
│  └─ break_monoalphabetic.{ py | cpp |* }
├─ task_02
│  ├─ k1.key
│  ├─ k2.key
│  ├─ Writeup.md
│  └─ cipher_keys.{ py | cpp | *}
├─ task_03
│  ├─ Writeup.md
│  ├─ XOR.key
│  └─ XOR_decrypter.{ py | cpp | * }
└─ bonus
   ├─ Writeup.md
   ├─ key.txt
   └─ plain.txt
```

All files should contain the requested key/plaintext only. The encoding is assumed to be UTF-8 for text files. Store all files in the requested format (e.g. hex-encoding, or pure binary). The monoalphabetic key (text file) should be in the form RXLEDNOCISFQ, corresponding to the letters ABCDEF.... The Writeup.md from task 2 should contain the name of the algorithm used, e.g., *AES*, *MD5* and a short explanation on how to apply it to obtain the requested result. Additionally, source code can be submitted to support the explanations in the writeup.

Prepare yourself for a lab defense of up to 30 minutes. In the lab defense, we will go through your solutions and discuss the way in which you solved the tasks. Ensure that you are familiar with all of the concepts that play a role within this lab and are able to defend why you took each step you took.

References

[1] C. Christensen. *Cryptanalysis of the Vigenère Cipher: The Friedman Test*. Northern Kentucky University. 2015. URL: https://www.nku.edu/~christensen/1402%20Friedman%20test%202.pdf (visited on 09/13/2024).

[2] C. Christensen. *Cryptanalysis of the Vigenère Cipher: The Kasiski Test*. Northern Kentucky University. 2015. URL: https://www.nku.edu/~christensen/1402%20vigenere%20cryptanalysis.pdf (visited on 09/13/2024).

[3] CrypTool. *JCrypTool: Cryptography for everybody.* 2020. URL: https://www.cryptool.org/en/jct/downloads/ (visited on 10/24/2020) (cit. on p. 2).

[4] P. B. Esslinger and C. Team. *Learning and Experiencing Cryptography with CrypTool and SageMath*. 2018. URL: https://www.cryptool.org/assets/ctp/documents/CT-Book-en.pdf (visited on 10/04/2022) (cit. on p. 1).

[5] J. Katz and Y. Lindell. *Introduction to Modern Cryptography*. 2nd. Chapman & Hall/CRC Cryptography and Network Security Series. CRC Press, 2020. ISBN: 9781351133012. URL: https://books.google.tm/books?id=RsoOEAAAQBAJ (cit. on p. 1).

[6] *Letter Frequencies in the English Language*. URL: https://www3.nd.edu/~busiforc/handouts/cryptography/Letter%20Frequencies.html (visited on 11/01/2020) (cit. on p. 10).

[7] C. Online. *CrypTool: Cryptography for everybody.* 2020. URL: https://www.cryptool.org/en/cto/ (visited on 10/24/2020).

[8] W. Stallings. *Cryptography And Network Security : Principles And Practice*. 7th. Pearson Education, 2016. ISBN: 9788178089027 (cit. on p. 1).

[9] F. Stridsberg. *The ZIP File Format*. Medium. URL: https://medium.com/@felixstridsberg/the-zip-file-format-6c8a160d1c34 (visited on 09/16/2024).