October 2022

D

ECSC 2021 – Prague

**Challenge author:** CybExer

Writeup for challenge Cracking One-Time Pad

ABOUT ECSC

The growing need for IT security professionals is widely acknowledged worldwide. To help mitigate this shortage of skills, many countries launched national cybersecurity competitions targeting towards students, university graduates or even non-ICT professionals with a clear aim to find new and young cyber talents and encourage young people to pursue a career in cyber security. The European Cyber Security Challenge (ECSC) leverages on these competitions by adding a pan-European layer.

The European Cyber Security Challenge is an initiative by the European Union Agency for Cybersecurity (ENISA) and aims at enhancing cybersecurity talent across Europe and connecting high potentials with industry leading organizations.

CONTACT

For contacting the authors, please use [ecsc@enisa.europa.eu](mailto:ecsc@enisa.europa.eu)  
For media enquiries about this paper, please use [press@enisa.europa.eu](mailto:press@enisa.europa.eu).

Authors/acknowledgements

John Smith, European Union Agency for Cybersecurity

legal notice

Notice must be taken that this publication represents the views and interpretations of ENISA, unless stated otherwise. This publication should not be construed to be a legal action of ENISA or the ENISA bodies unless adopted pursuant to the Regulation (EU) No 2019/881.  
This publication does not necessarily represent state-of the-art and ENISA may update it from time to time.  
  
Third-party sources are quoted as appropriate. ENISA is not responsible for the content of the external sources including external websites referenced in this publication.

This publication is intended for information purposes only. It must be accessible free of charge. Neither ENISA nor any person acting on its behalf is responsible for the use that might be made of the information contained in this publication.

copyright notice

© European Union Agency for Cybersecurity (ENISA), 2022  
Reproduction is authorised provided the source is acknowledged.

Table of contents

[1. Information regarding the challenge 3](#_Toc117437504)

[1.1 Description of the challenge 3](#_Toc117437505)

[1.2 Challenge specification 3](#_Toc117437506)

[1.3 Technical Specification 3](#_Toc117437507)

[1.3.1 Required infrastructure 3](#_Toc117437508)

[1.3.2 Provided files 3](#_Toc117437509)

[1.4 Questions and Answers 3](#_Toc117437510)

[1.4.1 Challenge-specific questions 3](#_Toc117437511)

[1.4.2 Generic questions 4](#_Toc117437512)

[2. Attack Scenario 5](#_Toc117437513)

[2.1 Description of the scenario 5](#_Toc117437514)

[2.2 Installation Instructions 5](#_Toc117437515)

[2.3 Tools needed for solving the challenge 6](#_Toc117437516)

[2.4 Walkthrough (Writeup) 6](#_Toc117437517)

1. Information regarding the challenge

## Description of the challenge

You have managed to acquire two encrypted messages and a plain text message, that corresponds to one of the encrypted messages.

## Challenge specification

* Challenge Category: Crypto
* Difficulty: Easy
* Expected time to solve: 30m

## Technical Specification

The challenge is made up of three files: a plaintext, corresponding ciphertext and another ciphertext

that contains the flag.

The challenge is static: **the files are provided to contesters for analysis**. A Python script is included

in the delivery package that can be used to generate different sets of inputs to use different flag for

every participant.

**Required skills:** knowledge of One-Time Pad algorithm and any scripting or programming language

to automate solving process.

### Required infrastructure

No infrastructure is required to prepare the challenge. Solving can be done offline. You just need to let the included script generate all necessary files for contestants. More in *2.2 Installation Instructions*.

### Provided files

Figure 1: List of files

|  |  |  |  |
| --- | --- | --- | --- |
| File name | Format | Comment | Checksum (SHA256) |
| gen\_task.py | Python script | Generator of challenge | fcbd3818f4602bc2229ec821190720cbdad81e31b200a32f064646632996117c |
| message.txt | Plaintext file | Message to secure | 6b051e65303fede9b28ef0a682f20c8831f7d959d5f106b3ec1982bcade990e0 |
| test.py | Python script | Author’s solver script | 580989517f64bee873a6f794363609385efd7fc83725d7f2f7e3b36d8b4a436e |

## Questions and Answers

### Challenge-specific questions

* What encryption method was used?
  + One-Time Pad
* Is it possible to solve the task without having access to or hints about content of any plaintexts?
  + Not really.

### Generic questions

* What operation is basis of One-time Pad?
  + Exclusive OR
* How long is the encryption key?
  + There is no encryption key, but the pad table must be at least as long as the message
* How to ensure confidentiality of encrypted messages?
  + The pad table must be kept secret, and used only once
* What are the weaknesses of One-time Pad?
  + Pad table is very long and must be distributed secretly, no integrity protection or verification mechanisms

1. Attack Scenario

## Description of the scenario

Participants must recover plain text message from an encrypted file by recovering the pad table from given plaintext and corresponding ciphertext. Little fuzzing is necessary to recover the flag.

## Installation Instructions

The delivery package includes Python script for generating set of files and random flag.

The script takes two or three command line arguments: name of plaintext file and output directory (mandatory) and number of datasets to generate (optional, default 1). If more than one dataset is requested, they will be put into separate subdirectories under the output directory.

Example:

|  |
| --- |
| $ usage: gen\_task.py [-h] [-n TASKS] -m FILE DIR  positional arguments:  DIR Output directory  optional arguments:  -h, --help show this help message and exit  -n TASKS Number of tasks to create  -m FILE File with plaintext  $ python3 gen\_task.py -n 3 message.txt out  flag for participant 1 : ECSC{2a24ed0587c4ad72f75c1cbc2ade151e1e06f0ac}  flag for participant 2 : ECSC{7977ba7c90e79806b7f0755a963bf9eb63f7cbf0}  flag for participant 3 : ECSC{5676ed723ba1c01419fc29d3f2eb25bc172618a1}  $ ls -1R out out: 1 2 3  out/1: mainfile.enc message.txt message.txt.enc  out/2: mainfile.enc message.txt message.txt.enc  out/3: mainfile.enc message.txt message.txt.enc |

## Tools needed for solving the challenge

Needed tools are:

* General Linux tools
* Scripting language

## Walkthrough (Writeup)

The task is built to demonstrate principles of One-Time Pad. The contester is provided with two encrypted files and a plain text that corresponds to one of the encrypted messages. The basis of One-time Pad algorithm is applying a reversible function on each byte of the message and corresponding byte in a fixed table. This task is using the original version of algorithm as described by Frank Miller, where modular addition is used with modulus 256.

Since a plaintext and corresponding ciphertext are available, the table can be recovered easily. Then the main message can be decoded against the same table.

Unfortunately, the message is corrupted:

|  |
| --- |
| $ python3 test.py -c message.txt.enc -p message.txt mainfile.enc | head  Insert required amount of spaces \*here\* to decrypt the message. The flag is:  T(òi«ËhÂÏøýÇJbæ¶1»ßoaæ¶1»ßoa |

Note the message: it asks to insert correct amount of padding into the text to recover the flag. Since One-time Pad does not offer any integrity protection, there is no option to assure, that the message is not correct other than looking at it.

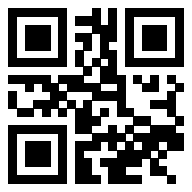
The only way to solve the task is to start adding data to the indicated place, one byte at a time, until something meaningful is displayed as the flag, or the pad table runs out.

An example solution is provided in the *test.py* script:

|  |
| --- |
| with open(args.fname[0], 'rb') as f:  msg = f.read()  dec = [chr((a-b)%256) for a,b in zip(msg, padtable[:len(msg)])]  print (''.join(dec))  m = re.findall(r'(.\*\\*)(here)(.\*)', ''.join(dec), re.S)  l = len(m[0][0])  for i in range(1,len(padtable)-l):  dec = [chr((a-b)%256) for a,b in zip(msg[l:],padtable[l+i:l+i+len(msg)])]  if dec[0:4] == ['h','e','r','e']:  print("\n\nfound solution at offset",i,”:”)  print(m[0][0]+''.join(dec)) |

This script brute-forces the message at word “here” until another place in the pad table is found where decryption result starts with “here”:

|  |
| --- |
| $ python3 test.py -c message.txt.enc -p message.txt mainfile.enc  Insert required amount of spaces \*here\* to decrypt the message. The flag is:  T(òi«ËhÂÏøýÇJbæ¶1»ßoaμ0´¾`ë!PÓ¤²)K4  found solution at offset 1333 :  Insert required amount of spaces \*here\* to decrypt the message. The flag is:  ECSC{2a24ed0587c4ad72f75c1cbc2ade151e1e06f0ac} |



ENISA  
European Union Agency for Cybersecurity

Athens Office  
1 Vasilissis Sofias Str.  
151 24 Marousi, Attiki, Greece

Heraklion Office  
95 Nikolaou Plastira  
700 13 Vassilika Vouton, Heraklion, Greece