October 2022

D

ECSC 2021 – Prague

**Challenge author:** CybExer

Writeup for challenge Week of the RSA

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 an encrypted message together with a set of RSA public keys that belong to employees of the company. The keys are already extracted from certificates and saved to files with random names, thus anonymous. It is known that the message is encrypted with one of those keys.

There is a hint that some of the keys may be vulnerable to attacks, and this could allow the decryption of the secret message.

## Challenge specification

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

By choosing the number of keys, focus of the task can be shifted:

* Having only few keys might tempt contesters to break them one by one, which is not the right path, because the individual keys are not weak.
* Having a large number (hundreds) of keys can be a hint that one of them must somehow stand out from the stack.
* Having a very large number (thousands or millions) of keys requires a well optimized algorithm to avoid computing for days.

## Technical Specification

The challenge is made up of a binary file containing encrypted flag and set of RSA public keys, one of which is used to create the encrypted file.

The challenge is static: **encrypted flag and set of keys are provided to contesters for analysis**. A Perl script is delivered that can be used to generate set of keys and corresponding flag. This allows to use different flag for every participant. An example dataset with 300 keys is provided in the delivery

package.

**Required skills:** knowledge of RSA 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) |
| check.pl | Perl script | Result verifier | f500f3fc15eec092bdd441786945250f3e88b7e5e44713c9e235adf30d1453b3 |
| generate-keys-and-flag.pl | Perl script | Generator of challenge | 13226da6bcb3b8cff218f264fb971f0e4351bdf6b4160d406fb15ad0ef863100 |
| example-keys.tar.gz | TAR archive | Challenge files | 23d125d0fb8f6c2c9324ab9ce9e53a82fc2d3205816c5c44e4349d1ce27d6b6c |

## Questions and Answers

### Challenge-specific questions

* What is the size of keys in the challenge?
  + 2048 bits
* What makes some of the keys in the given set vulnerable?
  + Non-unique primes
* Is it possible to solve the task with access to only the key that was used to encrypt the flag?
  + No

### Generic questions

* What mathematical problem is basis of security of RSA algorithm?
  + Factoring problem
* How to calculate modulus of RSA public key?
  + Multiply the two random primes p and q
* Why p and q must be large prime numbers?
  + The effort to factor public key depends on its smallest prime component
* How long RSA keys are considered secure in 2021?
  + 2048 bits for 1-2 years, 3072 bits if security is required for longer period
* Why it is important to choose p and q randomly?
  + Having access to public keys that share primes allows to factor them quickly by finding greatest common divisor of the keys
* How can an attacker get hold of large amounts of public keys?
  + Initiating TLS or SSH sessions to every public server in Internet, searching for digitally signed documents or e-mails, gaining access to LDAP or Active Directory servers

1. Attack Scenario

## Description of the scenario

Participants must recover plain text message from an encrypted file by recovering a private key from given set of public keys, where one key is specially crafted to share a prime with another one.

## Installation Instructions

The delivery package includes Perl script for generating the private keys and encrypted flag. The scripts use Convert::ASN1 module, which is not part of default Perl installation, but available as a separate package in most popular Linux distributions (e.g. libconvert-asn1-perl in Ubuntu) or CPAN.

Installation steps:

* Create an empty directory for artefacts.
* Run generate-keys-and-flag.pl script to generate set of public keys and encrypted flag.
* Optionally run check.pl script to verify the result.
* Pack the output directory for distribution to participant(s).

One iteration creates one flag and one set of public keys. The keys are stored in files with random names, weakened one is also chosen randomly from last 10% of the keys after having sorted the file names alphabetically.

To have different flag for every participant, repeat the procedure for required number of times.

Note that the algorithm implemented in check.pl is quite inefficient. It is suggested to use it with a few

selected keys to verify that the flag can indeed be decrypted. The execution time of check.pl with larger

key sets is matter of luck, because the files are checked in undetermined order.

Example:

|  |
| --- |
| $ mkdir output  $ perl generate-keys-and-flag.pl output 30 Generating 30 RSA keypairs Creating weak keys output/BaggN1yu1z and output/Ebespqke3Q Flag saved to output/encrypted\_flag Removing private keys The flag is: ECSC{95a088a236b8c0b2798cbe2f44ee239d3a775c0c}  $ perl check.pl output/encrypted\_flag output/\*.pub Reading public keys Computing gcd-s .... Prime 147912091710608458553804075781411231799801044682282835667601269348810907633403886349179949506811301799167581207323271889004155922696093689974470666249065031424322856597499374210342793408989248212893334605754416562089050387934156288996690404973830880928898332574797860977040966638661654438334412124782926460177 found in output/BaggN1yu1z.pub and output/Ebespqke3Q.pub  The flag is: ECSC{95a088a236b8c0b2798cbe2f44ee239d3a775c0c} |

## Tools needed for solving the challenge

Needed tools are:

* General Linux tools
* Scripting language that supports calculations with big numbers and parsing of base64 encoding and ASN.1 structure
* HW requirements (CPU, RAM) depend heavily on the amount of public keys included in the challenge
* Setup scripts use Perl module *Convert::ASN1* and *OpenSSL* command line utility

## Walkthrough (Writeup)

The task is built to demonstrate importance of randomness in cryptography.

A set of public keys is given to the contester, that are all presumably secure, when approached individually (2048-bit keys, generated with up-to-date OpenSSL). This can be verified using some RSA cracking tool, e.g., *yafu*.

The fact that a large set of public keys are provided should be an indicator that the goal is not to factor them one at a time, but there should be something that makes one or few keys much weaker than others. It is not visible immediately but knowing important properties of the RSA algorithm gives an opportunity to test them. When all the primes that were used to generate the set of keys are unique, greatest common divisor of any pair of moduli is 1 (one). But in case of non-unique primes, the operation will reveal the prime.

Once one prime is known, the other one can be computed for both keys by dividing modulus to the known prime.

This is indeed the case - one of the supposedly random primes appears in two keys. This can be attacked to easily recover both private keys. One of the recovered keys decrypts the flag.

Should there be a very large number of keys, the repeated prime can be found by computing *gcd* of a modulus and product of all other moduli. With this algorithm, every key needs to be checked only once at the expense of using more RAM.

**Flag in example-keys.tar.gz: ECSC{88a24f9a623519885adbbfc5f096ad1d65338d95}**

A little knowledge about programming and relevant file formats is also required to extract the moduli from PEM-encoded public key files that were generated by *OpenSSL*. The *check.pl* script in the delivery package is essentially implementation of one possible solution for the challenge.



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