ARL-TR-8937 • Apr 2020

Hands-on Cybersecurity Studies: Ransomware Key Recovery

by Jaime C Acosta, Adrian J Belmontes, and Salamah Salamah

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

ARL-TR-8937 ● APR 2020

Hands-on Cybersecurity Studies: Ransomware Key Recovery

Jaime C Acosta
Computational and Information Sciences Directorate, CCDC Army Research Laboratory

Adrian J Belmontes and Salamah Salamah
University of Texas at El Paso

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
April 2020	Technical Report	August 2019 – March 2020

4. TITLE AND SUBTITLE
Hands-on Cybersecurity Studies: Ransomware Key Recovery

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Jaime C Acosta, Adrian J Belmontes, and Salamah Salamah

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
CCDC Army Research Laboratory
ATTN: FCDD-RLC-ND
White Sands Missile Range, NM 88002

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TR-8937

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
Ransomware is a type of malicious software that denies access to a computer system or files until a ransom is paid, usually in the form of cryptocurrency. It is typically spread through phishing emails or through websites where the software is downloaded without the user knowing; it can also spread by taking advantage of vulnerabilities in software running on the victim's devices. This report presents a hands-on exercise that demonstrates the effects of ransomware on vulnerable machines and guides participants through a set of steps that will regenerate the key required to decrypt the ransomed data.

15. SUBJECT TERMS
ransomware, WannaCry, key recovery, encryption, hands-on cybersecurity, CyberRIG

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Jaime C Acosta
Unclassified	Unclassified	Unclassified	UU	26	19b. TELEPHONE NUMBER (Include area code) (575) 993-2375

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures

List of Tables

1. Introduction

Ransomware is a type of malicious software that denies access to a computer system or files until a ransom is paid, usually in the form of cryptocurrency. It is typically spread through phishing emails or through websites where the software is downloaded without the user knowing; it can also spread by taking advantage of vulnerabilities in software running on the victim's devices. This report presents a hands-on exercise that demonstrates the effects of ransomware on vulnerable machines and guides participants through the steps that will regenerate the key required to decrypt the ransomed data.

When the WannaCry ransomware crypto worm spread worldwide in 2017, consumers and large corporations felt the impact.[1] Critical data were locked away and made inaccessible to owners until a ransom was paid in the form of anonymous currency known as bitcoin. Over 200,000 victims were affected by this attack, including the National Health Service hospitals in England and Scotland. Even though a kill switch was found and implemented, variants of this malware continue to arise.

The variant that we reference in this report uses the Rivest, Shamir, and Adleman (RSA) encryption method[2] to lock (ransom) files. This method uses a pair of keys, known as public and private keys.[3] The former is used to encrypt while the latter is used to decrypt data. The malware removes the private key from the victim's machine and provides it only after a ransom is paid.

This report demonstrates how a machine becomes infected, allowing participants to experience the effects of the malware. Participants are then guided through the process of identifying and recovering data that is inadvertently left behind by the malware after encrypting files. The data are used to regenerate the private key, and decrypt and regain access to the ransomed files. This hands-on exercise is related to a previous exercise that we developed, showing how to analyze and implement a kill switch for the WannaCry ransomware.[4] It is based on the program logic from the wanakiwi[5] software, developed by Benjamin Delpy and made available on GitHub.

2. Setup and Configuration

The hands-on exercise consists of two virtual machines: one is used to infect a machine and create a memory dump, and the other is used to analyze the memory dump and public key to look for prime numbers and construct a private key. The setup configuration consists of the following software elements:

- VirtualBox (Version 6.0)
- Two Windows 7 Home Basic 32-bit virtual machines
- LibreOffice 6.4.a
- IDA Pro Free (Version 5.0)
- HxD Hex Editor (Version 2.3.0.0)
- WannaCry malware variant with MD5 hash: ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa
- RSA Key Generator[6]
- Little/Big Endian Converter

The first machine, named Ransomed, was set up with IDA Pro Free, HxD Hex Editor, and the WannaCry variant. The second machine, named MemoryAnalysis, was set up with LibreOffice, HxD Hex Editor, and the RSA Key Generator. In addition, the MemoryAnalysis machine was updated to include the WannaCry patch to prevent the malware from infecting the machine. Finally, a shared folder was set up to allow file transfers between the two machines.

The entire exercise runs on the US Army Combat Capabilities Development Command (CCDC) Army Research Laboratory (ARL) South Cyber Rapid Innovation Group (CyberRIG) Collaborative Innovation Testbed (CIT), which provides an isolated environment, ensuring that all the environmental artifacts are segregated from any real systems.

3. Learning Objectives

This exercise will teach participants the following points:

- Participants will have a better understating of how ransomware works, specifically the WannaCry crypto worm. The effects of the ransomware on the sandbox environment should emphasize the importance of having files and other data backed up.
- Participants will gain experience in creating memory dumps and extracting necessary data by examining memory using the IDA Pro software. Working in a timely manner to avoid losing data in memory is important.
- Participants will gain a better understand of RSA encryption. They will learn how a private key is constructed as they calculate all of the necessary values and constructs using the HxD Hex Editor software and RSA Key Generator.

4. Methodology

In creating this exercise, we started by downloading Benjamin Delpy's wanakiwi software onto a virtual machine with the Windows 7 32-bit Operating System. According to the software documentation[5], this version of Windows is the most recent platform on which wanakiwi was written to work correctly. We found that the software works on both Windows 7 32-bit Pro and Home versions. We later realized that the wanakiwi software does not work if additional updates are installed (including those required to recompile the wanakiwi software). Therefore, we created two separate virtual machines: MemoryAnalysis and Ransomed.

To fully understand the inner workings of the software, we traced through the source code. We installed the latest software updates and patches on the MemoryAnalysis machine as well as the Visual Studio integrated development environment. The Ransomed machine was left untouched, except for the inclusion of the wanakiwi executable binary.

We modified the source code to skip some of the longer procedures and to narrow down where, in memory, the prime numbers are placed when the ransomware infects a machine. The Visual Studio debugger was extremely helpful in tracing through the statements that traversed memory.

To support this interactive exercise, we chose IDA Pro because it is an ideal tool to help participants replicate the logical steps in wanakiwi. Using the prime numbers found with wanakiwi, IDA Pro was able to attach to the ransomware process and show their location in the virtual address space.

We encountered slight differences when comparing the wanakiwi logic and the standard RSA algorithm: the key formatting within the file that stores the key values was not the same. The ransomware used an altered form, with parameters in a different order and an additional value—a *magic* number that seemed to indicate a signature for the malware key that existed as a precursor to the standard values. To calculate the key values, we used an RSA Key Generator website. Values in memory are represented in little-endian format, which is expected by the ransomware. However, the RSA Key Generator expected values in big-endian format, so we developed a small application to handle these conversions.

The final step was to understand how the prime numbers could be found in memory without relying on the wanakiwi application. Wanakiwi uses a brute force approach—it iterates through memory and attempts to compute sequences of bytes and then determines if the values are prime. This was computationally expensive and time consuming, so we looked for another way. We first attempted to find the entire private key in memory using IDA Pro, but this did not work. We noticed that

after infection, a public key is placed on the user's desktop. Looking deeper into key structure, we searched memory for the individual values. Using the modulus, which is a product of the two prime numbers, yielded success. Consequently, the prime numbers were always only a few bytes away from the modulus.

To address the issue of timing and overwriting key in-memory values, participants must create a nonvolatile image immediately after infection. We task participants with creating a memory dump of the ransomware process. To demonstrate the importance of this task, participants must then reboot their machines; they will find that the values required for recreating the keys are no longer resident in memory or anywhere on their machine.

5. Exercise

This exercise is separated into five main activities:

1) Infect one machine and take a memory dump of the malware process.
2) Analyze the memory dump to recover the two prime numbers.
3) Calculate the remaining variables needed to create a private key.
4) Combine all variables to create a private key.
5) Decrypt the infected machine using the generated key.

The exercise requires about 1.5–2 h to complete.

5.1 Activity 1: Dynamic Analysis of the Process Memory

Run the WannaCry malware in your sandbox and observe its behavior on the machine.

1) Start the Ransomed computer by clicking the corresponding link in your browser. You should see a Windows desktop. If prompted, log in using the following credentials:

   Username: **Reversing**

   Password: **malware**

2) Open **WannaCry** by double-clicking this file 

on the desktop. Click **No** on any prompts that may come up.

3) The program will execute and finish once you receive the message shown in Fig. 1.

Fig. 1 WannaCry ransomware indicator

Be sure to create a memory dump of the malware process to analyze later. This has to be done as soon as possible to ensure the ransomware and the operating system do not remove important artifacts that are essential for undoing the chaos.

4) Open **Task Manager** by clicking the Windows start key and searching for Task Manager. Select **View running processes with Task Manager**.

5) Click the **Processes** tab and then **Show processes from all users.** Find the WannaCry process (it starts with **ed01ebfbc…**). See Fig. 2.

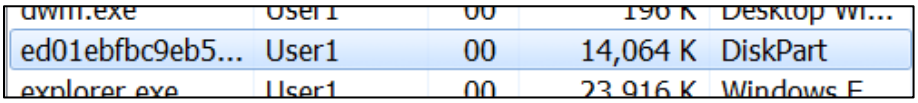


Fig. 2 Processes window

6) Right-click and select **Create Dump File**. You should see the screen shown in Fig. 3.

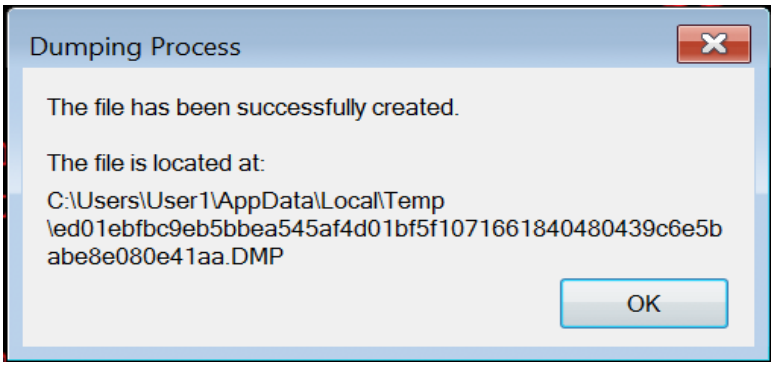


Fig. 3 Dumping Process window

7) Open the File Explorer icon  and enter the path in the path bar as shown in Fig. 4.

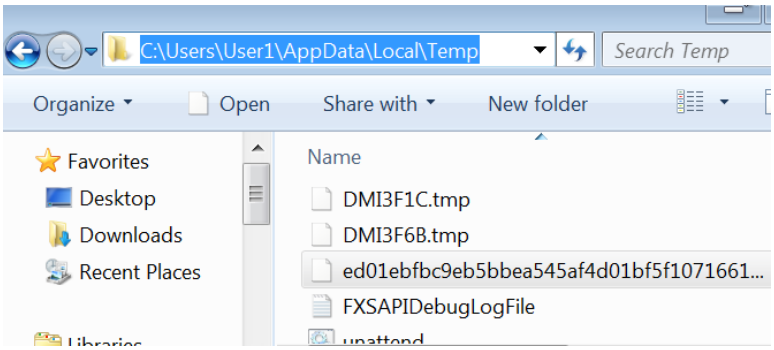


Fig. 4 Path to dump file

8) Drag and drop the memory dump file to the desktop.

9) Copy a few files to a shared folder in order to analyze them on a separate machine. Double-click the folder on the desktop named **Shared Folder.**

10) Locate the public key file named **00000000.pky** on the desktop. Copy both this **public key file** and the **memory dump file** to the shared folder.

11) Open **HxD Hex Editor** by double-clicking the  icon on the desktop. Then click **File->Open** and select the **public key** located on the desktop.

The next step is to determine the attacker's private key, but in order to do so, we need to look at a few values from the public key. The public key is shown in Fig. 5 and consists of a **special header** (red), the public **exponent** (yellow), and the **modulus** (blue). Figure 5 is colored to identify components easier. When using HxD, the components will not be colored.

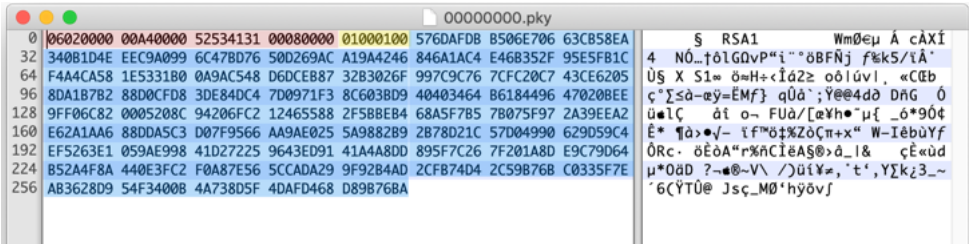


Fig. 5 Public key (color-coded) consists of header (in red), exponent (in yellow), and modulus (in blue)

Note an issue with the values: they are in reverse order. For example, if we encounter the eight digits **12 34 56 78**, we would need to convert them to the following: **78 56 34 12**. This is a known issue in computing called little versus big endianness. Consider the public exponent bytes in little endian: 01 00 01 00. The actual value of these bytes would be 00 01 00 01. The bytes are reversed. Follow the steps below to notate the correct order of the bytes.

12) Look at your hex editor and identify your public exponent. Write down your public exponent **as displayed in the hex editor**.

    _____

13) **Reverse** the bytes from number 12 as described previously and write them here. (Hint: it should match the previous example.)

    _____

14) Look at your hex editor and identify your modulus. What are the **first eight characters (or four bytes)** of the modulus, **as displayed in the hex editor**?

    _____

15) Look at your hex editor and identify your modulus. What are the **last eight characters (or four bytes)** of the modulus, **as displayed in the hex editor**?

    _____

16) **Reverse** the bytes from number 15 as described previously and write them here. _____

Use IDA Pro to search for a few more numbers that were present in the memory dump. You will then simulate the effects of restarting the machine in the hopes that everything will go back to normal.

17) Open up **IDA Pro Free** by double-clicking the 

icon on the desktop. Then click **GO.**

18) On the toolbar, click **Debugger -> Attach -> Local Windows Debugger**. Then look for the WannaCry process (it starts with **ed01ebfbc…**) and click **OK** (Fig. 6).

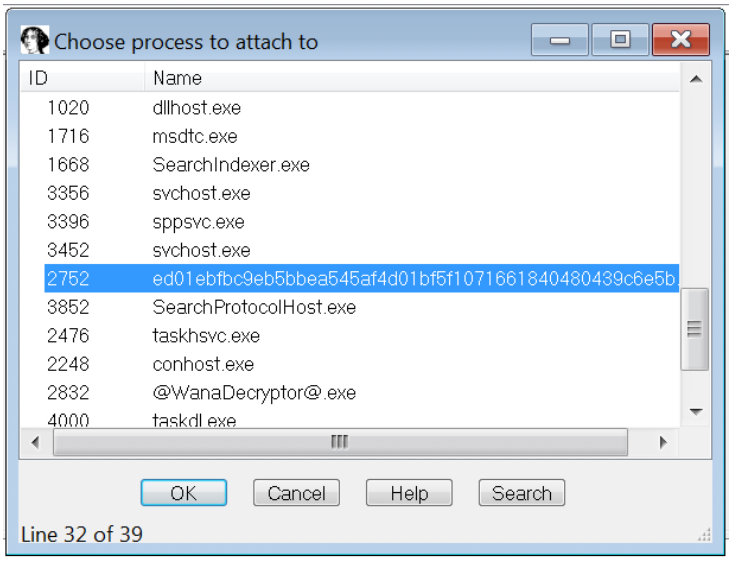


Fig. 6 Debugger attach to process

19) When the main IDA Pro window opens, click the **Play** icon just below the toolbar and then click the **Pause** icon.

20) Now open the search window by clicking **Search -> sequence of bytes**. The search window will take a few seconds to open.

21) In the String box, type in your answer for number 16. Make sure **Find all occurrences** is selected. How many occurrences did it find? _____

22) Click the **Windows** button and select **Restart** (Fig. 7). Force quit any processes that are active.

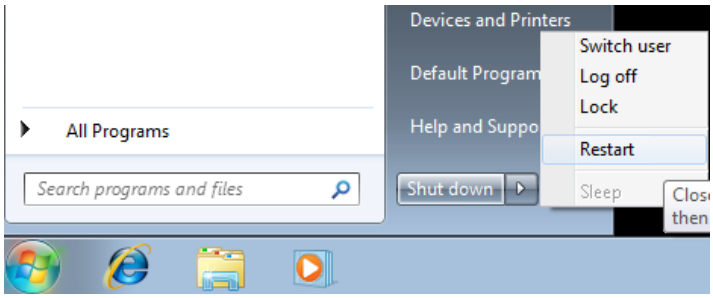


Fig. 7 Windows Restart screenshot

23) After restarting, run the **WannaCry icon** on the desktop again.

24) Repeat steps 17–21.

25) How many occurrences did it find this time? _____

26) Did the number of occurrences change? Write down a few reasons why you think this is the case.

_____
_____
_____
_____

5.2 Activity 2: Extracting Encryption Parameters from the Memory Dump

Recall that in the previous activity, you froze a copy of the process data before you restarted. Now you can look in that image to find all of the values needed to recreate the decryption key.

Before building the private key, we will take a closer look at the full structure of the RSA keys. The RSA structure contains several items in addition to the ones you captured previously; you will need many of these to recreate the private key. The memory dump will provide us with the two prime numbers, and the rest can be obtained with a calculator. Table 1 shows the values that make up a private key structure, and Fig. 8 shows a sample public key with values.

Table 1 Key structure

Name	Size	Variable name in RSA equation	Color-code
Heading	16 bytes (32 characters)	N/A	Red
Public Exponent	4 bytes (8 characters)	E	Yellow
Modulus	256 bytes (512 characters)	N	Blue
Prime Number 1	128 bytes (256 characters)	P	Green
Prime Number 2	128 bytes (256 characters)	Q	Orange
CRT Exponent 1	128 bytes (256 characters)	dP	Purple
CRT Exponent 2	128 bytes (256 characters)	dQ	Light Blue
CRT Exponent Coefficient	128 bytes (256 characters)	qInv	Black
Private Exponent	256 bytes (512 characters)	D	Pink

Fig. 8 Format with color-coded values

1) Exit the current Windows machine by clicking the Back button in your browser. Start the MemoryAnalysis by clicking on the corresponding link.

2) Open **HxD Hex Editor** by double-clicking the  icon on the desktop. Then open the **public key** that you stored in the shared folder (00000000.pky).

When WannaCry creates the public key, it uses two prime numbers to calculate a key value. The primes are stored in memory until they are overwritten or cleared. Creating the dump file in a timely manner decreases the chance of losing that data. (Note: newer machines clear the memory right away.)

3) Click **File -> Open** and select the **DMP file** on the shared folder (it starts with **ed01ebfbc…**).

4) On the toolbar click **Search -> Find.** Then click the **Hex-values** tab and select **All** for the search directions.

5) In the search bar, enter your **answer from Step 1, number 15** and click **Search all**.

6) Double-click through the occurrences and find the one that contains the following characters right after the value you searched: **11000110**.

Now we need to extract a few numbers from the rest of this memory dump. They are mixed with other numbers, so we will have to skip a few numbers and paste only what we need into a temporary spreadsheet file named **Calculations Template**.

7) Open **Calculations Template** by double-clicking the 

icon on the desktop. In the next steps, you will fill in the green and blue cells.

8) Enter you answer from **Step 1, number 12** under **Public Exponent (little endian).**

9) Go back to the HxD application (Fig. 9; your values). From the **end of modulus** (the number you just found), look **eight rows up** to find the value from **Step 1, number 15.** This is the start of the modulus.

10) Highlight the bytes corresponding with the entire modulus (the length of bytes you select can be seen at the bottom of the application labeled **"Length(d):"** make sure it is **256 bytes**). Copy and paste the modulus into the **Calculations Template** spreadsheet under **Modulus (little endian).** See Fig. 9.

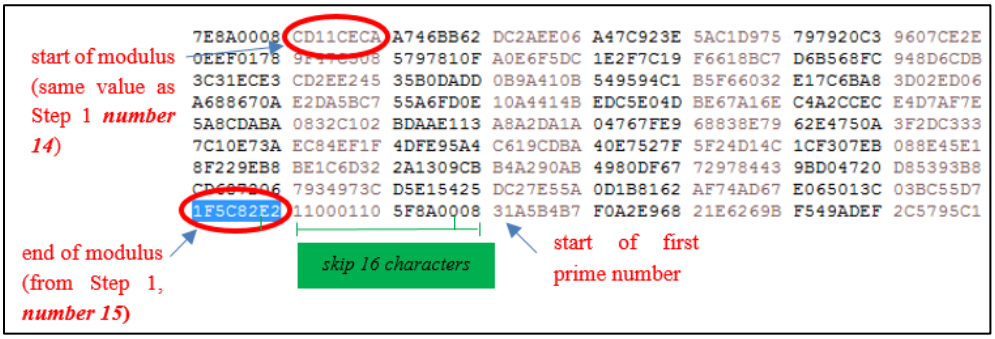


Fig. 9 Encryption numbers in DMP file

11) After the end of the modulus, skip 16 characters (**8 bytes**); the next 256 characters **(128 bytes)** make up the **first prime number** (Fig. 9). Highlight the bytes corresponding with the first prime number (the length of bytes you select can be seen at the bottom of the application—make sure it is **128 bytes**) and then copy and paste them into the spreadsheet under **Prime 1 (little endian).**

12) After the first prime number, skip another 16 characters (**8 bytes**); the next 256 characters (**128 bytes**) make up the **second prime number**. Highlight and copy the second prime number and place it under **Prime 2 (little endian)**. The length of bytes you select can be seen at the bottom of the HxD application—make sure it is **128**.

We have found all of the values we need from our memory dump. The next step is to get the reverse of some of the numbers (or the **big endian** version) for the modulus and primes.

13) Open the **Endian Converter** by double-clicking the 

icon on the desktop. From the **Calculations Template** copy over the values under **Modulus (little endian)**, click **Convert** and then paste the result back into the **Calculations Template** under **Modulus (big endian).**

If the converter prompts you that your number is not even, make sure there is no space at the end. If there is no space, add a 0 (zero) to the start of your number and try again.

14) Repeat the conversion process from **number 13** for all of the little-endian values in your spreadsheet (**primes, public exponent, special header**). (Hint: Public Exponent [big endian] should match your answer from Step 1, number 13.)

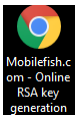
5.3 Activity 3: Calculating the Missing Encryption Values

Table 2 shows the mapping between values and their variable name in RSA equations.

Table 2 Key structure values found up to this point

Name	Variable name in RSA equation	Status
Heading	N/A	Found
Public Exponent	**e**	Found
Modulus	**n**	Found
Prime Number 1	**p**	Found
Prime Number 2	**q**	Found
CRT Exponent 1	**dP**	Need to Calc
CRT Exponent 2	**dQ**	Need to Calc
CRT Exponent Coefficient	**qInv**	Need to Calc
Private Exponent	**d**	Need to Calc

You will need to calculate a few more numbers using the script located on your desktop.

1) Open the **RSA key generation** script by double-clicking the 

icon on the desktop. Scroll down until you see the **Input online RSA key generation form**.

2) Copy and paste the **PRIME 1 (big endian)** and **PRIME 2 (big endian)** from **Calculations Template** into the boxes (order does not matter; see Fig. 10).

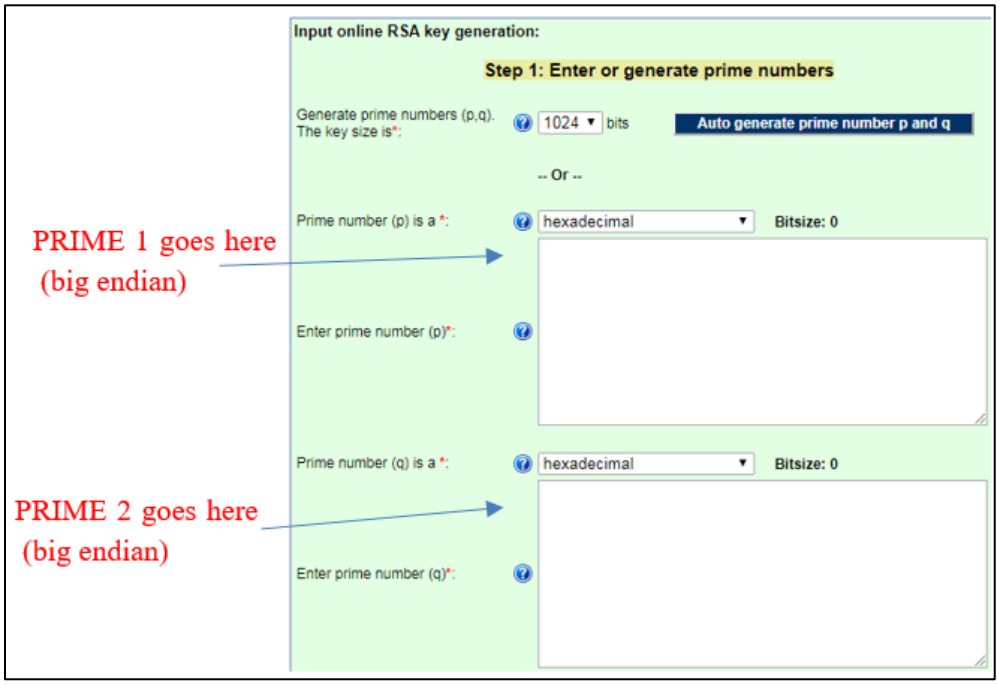


Fig. 10 Prime 1 and prime 2 placement

3) Scroll down further until you see **Step 2: Enter public exponent.** Enter your value for the **public exponent (big endian)** from the Calculations Template, and then click on the **Generate Keys** button (Fig. 11).

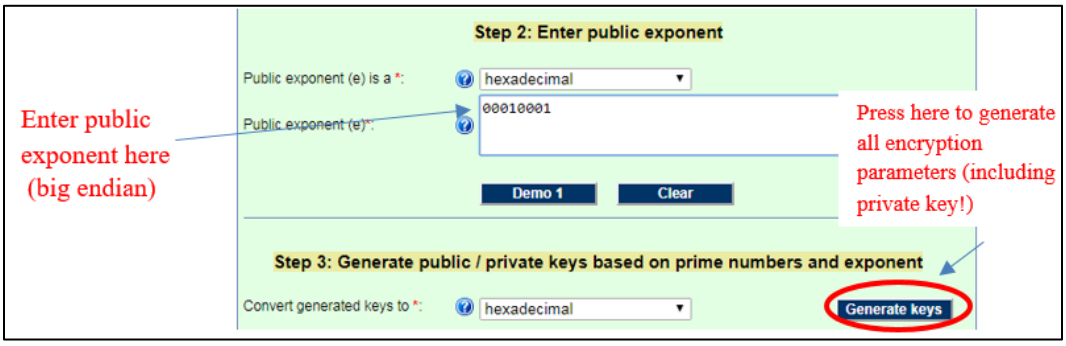


Fig. 11 Public exponent placement

4) Scroll down and verify that the **modulus** matches what you have in your spreadsheet.

5) Look through the form and copy the private exponent (d) into the **private exponent (big endian)** on your spreadsheet. Continue through and also copy into the spreadsheet the **big endian** values for CRT exponent 1 (**dP**), CRT exponent 2 (**dQ**), and CRT coefficient (**qInv**).

6) You should now have all of the values in your spreadsheet under the **big endian** column filled. Use the Endian Converter  to calculate and then fill in the missing values in the little-endian column.

Once you are done, close all other windows except for the **Calculations Template** and the **HxD application**.

The values you just added are part of the creation of RSA keys; however, the formatting and layout may differ.

5.4 Activity 4: Regenerate the Private Key

You should now have all of the values you need to regenerate the private key. Now you need to put them in the correct order (required by the ransomware) using the hex editor.

1) Create a new file by clicking **File->New** to begin creating the private key. Click on **File->Save As…** and name the file **00000000.dky** (eight zeroes) on the desktop. You can switch between the two files using the tabs at the top.

The key that WannaCry reads uses the **little-endian** format; you will need to create the key using the values obtained in the previous steps.

2) First, create the **special header** (Fig. 12). Make sure that your cursor is at the beginning of the file. Copy the value from your spreadsheet under the **Special Header, Little Endian** column. If a warning appears, select the **don't ask again** box and continue.

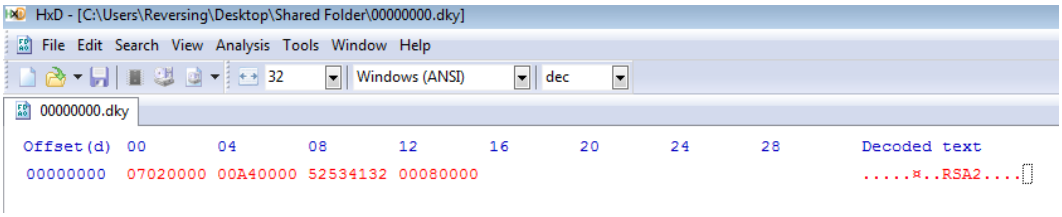


Fig. 12 Special header entry

3) Next, copy over the **Public Exponent, Little Endian Column** and the **Modulus, Little Endian Column**. At this point, your file should have values through **eight full rows** and **five columns** in the **ninth row**. See Fig. 13.

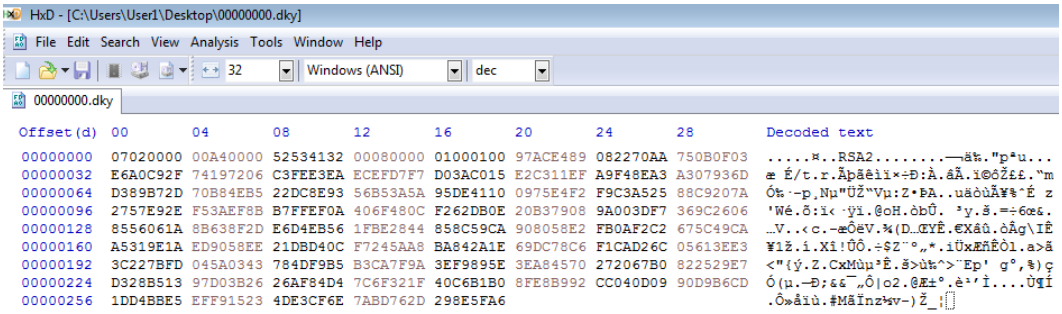


Fig. 13 Key entry

4) Continue by filling in the rest of the values starting with Prime Number 1 (**use little endian from here on**) to create the key structure as shown in Table 3 and Fig. 14.

Table 3 Key generation status

Name	Size	Variable name in RSA equation	Color-code
Heading	16 bytes (32 digits)	N/A	Red
Public Exponent	4 bytes (8 digits)	**e**	Yellow
Modulus	256 bytes (512 digits)	**n**	Blue
Prime Number 1	128 bytes (256 digits)	**p**	Green
Prime Number 2	128 bytes (256 digits)	**q**	Orange
CRT Exponent 1	128 bytes (256 digits)	**dP**	Purple
CRT Exponent 2	128 bytes (256 digits)	**dQ**	Light Blue
CRT Exponent Coefficient	128 bytes (256 digits)	**qInv**	Black
Private Exponent	256 bytes (512 digits)	**d**	Pink

Fig. 14 Key structure

5) Once done, save the key to your desktop.

6) Move the key you just created (00000000.dky) to the shared folder.

5.5 Activity 5: Decrypting Files Using the Regenerated Private Key

You created your own private key. Now it is time to test it on the infected machine.

1) Exit the current Windows machine by clicking the Back button in your browser. Click again on the link corresponding with the Ransomed machine.

2) **Move the private key** (00000000.dky) from the shared folder to the desktop.

3) Click **Decrypt** on the message prompting you to pay (Fig. 15).

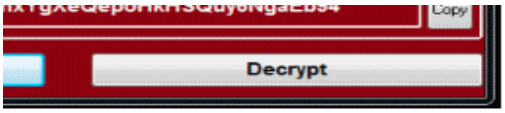


Fig. 15 Decrypt button

4) Click **Start**.

5) As a test to make sure it worked, open **Important.txt** to make sure you can read the data.

Great job! You have successfully analyzed a binary and decrypted your ransomed files.

6. Conclusion

After completing this exercise, participants should have a better understanding of how ransomware works, how RSA encryption works, and the steps needed to develop a new decryption key. This exercise will be shared with collaborators and partners (including professionals, faculty, and students) to help establish a common ground for studying ransomware that is similar to WannaCry, analysis tools, and research in binary analysis, both to secure systems and to develop ways to recover after compromise.

More specifically, we envision this report being the first of many studies to uncover the inner workings of ransomware. We hope the information found herein will enlighten researchers and practitioners in the cybersecurity field to understand ransomware and develop detection and quarantine mechanisms rather than using simple signature-based techniques. In addition, this report demonstrates a way to recover data from devices that have fallen victim to WannaCry. While the wanakiwi code is currently not capable of decrypting files on recent systems, this report details the steps associated with analyzing memory to recover the private key. Still left to future work is whether these or similar techniques can be used to recover keys on other systems to include Linux, Mac, and newer versions of Windows.

7. References

1. Trautman LJ, Ormerod PC. WannaCry, ransomware, and the emerging threat to corporations. Tenn L Rev. 2018;86:503.

2. Rivest RL, Shamir A, Adleman LM, inventors. Cryptographic communications system and method. United States patent US 4,405,829. 1983.

3. Jonsson J, Kaliski, B. Public-key cryptography standards (PKCS) #1: RSA cryptography specification version 2.1. 2003 Feb [accessed 2020 Apr 6]. https://tools.ietf.org/html/rfc3447#appendix-A.1.2.

4. Acosta JC, Escobar de la Torre A, Salamah S. Hands-on cybersecurity studies: multi-perspective analysis of the WannaCry ransomware. Aberdeen Proving Ground (MD); Army Research Laboratory (US); 2019 Jan. Report No.: ARL-TR-8627.

5. Delpy B. Wanakiwi [code]. 2017 May [accessed 2020 Apr 6]. https://github.com/gentilkiwi/wanakiwi/releases

6. Mobilefish. Online RSA key generation. 2019 Dec [accessed 2020 Apr 6]. https://www.mobilefish.com/services/rsa_key_generation/rsa_key_generation.php

List of Symbols, Abbreviations, and Acronyms

ARL	Army Research Laboratory
CCDC	US Army Combat Capabilities Development Command
CIT	Collaborative Innovation Testbed
CyberRIG	Cyber Rapid Innovation Group
RSA	Rivest, Shamir, and Adleman

1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
1 (PDF)	CCDC ARL FCDD RLD CL TECH LIB
2 (PDF)	CCDC ARL RDRL CIN D J CLARKE J ACOSTA