

INFSCI 1600 Security and Privacy

Fall Semester 2021

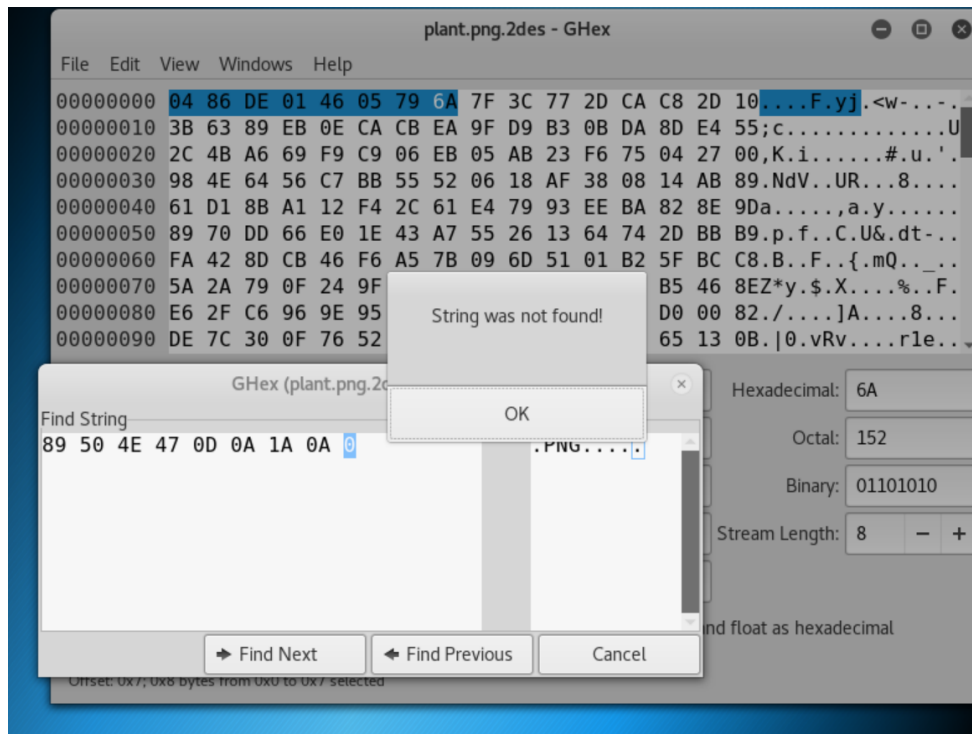
Jonathan Azcona, jka27@pitt.edu

3 November 2021

Project 2 - 2DES Meet-in-the-Middle Attack

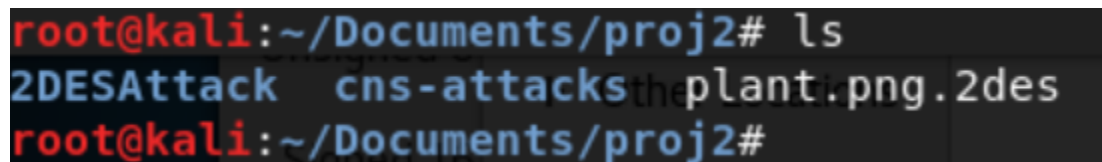
The first step that I did was look at my notes on what I have on project 2. The next step was to download the 2DES png file using file.io to transfer the png file from my host machine to my VM. I then looked up the PNG file signature which turned out to be “89 50 4E 47 0D 0A 1A 0A” and I tried to look for that same hex onto the encrypted PNG file using GHex in figure 1.

(Figure 1: Ghex)



I knew that I would not find it because it was encrypted using 2DES, so the next step was to find out what more information I needed. I wanted to learn more about 2DES and meet-in-the-middle attacks so I googled file signatures, looked at class notes, and websites on what it is I'm doing. I installed a variety of code and then deleted it because I had no idea what it needed and what it wanted me to input. For example, I downloaded a cns-attack on github as shown in figure 2 and 3. The source code here was in another language so I decided to translate it and continue with that code.

(Figure 2)



```
root@kali:~/Documents/proj2# ls
2DESAttack  cns-attacks  the plant.png.2des
root@kali:~/Documents/proj2#
```

A terminal window screenshot showing a file listing command. The prompt is 'root@kali:~/Documents/proj2#'. The command 'ls' has been executed, and the output shows three files: '2DESAttack', 'cns-attacks', and 'the plant.png.2des'. The prompt is repeated on the next line.

(Figure 3)

```
def main():

    # Exercițiu pentru test des2_enc
    key1 = 'Smerenie'
    key2 = 'Dragoste'
    m1_given = 'Fericitii cei saraci cu duhul, ca'
    c1 = 'cda98e4b247612e5b088a803b4277710f106beccf3d020ffcc577ddd889e2f32'
    # ++> done, TODO: implement des2_enc and des2_dec
    m1 = des2_dec(key1, key2, c1.decode('hex'))

    print 'ciphertext: ' + c1
    print 'plaintext: ' + m1
    print 'plaintext in hexa: ' + m1.encode('hex')
    print 'ciphertext: ' + des2_enc(key1, key2, m1).encode('hex')

    # TODO: run meet-in-the-middle attack for the following plaintext/ciphertext
    m1 = 'Pocainta'
    c1 = '9f98dbd6fe5f785d' # in hex string
    m2 = 'Iertarea'
    c2 = '6e266642ef3069c2'
```

I began to think that this did not work because of the necessary inputs that it required so I went ahead and found another repository that I could understand. I stumbled on a github repository as shown in figure 4 and downloaded it onto my VM.

(Figure 4: Github repository)

The screenshot shows a terminal window titled 'root@kali: ~/Documents/proj2'. The user has navigated to the 'Documents' directory and then to 'proj2'. They have cloned a repository from GitHub: 'https://github.com/bishwahang/2DESAttack'. The terminal output shows the cloning process, including enumerating objects, total and reused objects, and unpacking objects. A hex editor overlay is visible, displaying various data types for a selected range of bytes (0x7 to 0x7). The data types include Signed 8 bit, Unsigned 8 bit, Signed 16 bit, Unsigned 16 bit, Float 32 bit, Signed 32 bit, Unsigned 32 bit, Signed 64 bit, Unsigned 64 bit, Float 64 bit, Hexadecimal, Octal, and Binary. The 'Show little endian decoding' checkbox is checked, and the 'Show unsigned and float as hexadecimal' checkbox is unchecked. The offset is set to 0x7, and 0x8 bytes from 0x0 to 0x7 are selected.

```

root@kali:~# cd Documents
root@kali:~/Documents# ls
proj2
root@kali:~/Documents# cd proj2
root@kali:~/Documents/proj2# git clone https://github.com/bishwahang/2DESAttack
git 00000040 61 D1 8B A1 12 F4 2C 61 E4 79 93 EE BA 82 8E 9Da.....,a.y...
Cloning into '2DESAttack'...
remote: Enumerating objects: 11, done.
remote: Total 11 (delta 0), reused 0 (delta 0), pack-reused 11
Unpacking objects: 100% (11/11), done.
root@kali:~/Documents/proj2#

```

Signed 8 bit:	106	Signed 32 bit:	2000453482	Hexadecimal:	6A
Unsigned 8 bit:	106	Unsigned 32 bit:	2000453482	Octal:	152
Signed 16 bit:	32618	Signed 64 bit:	3299109023671680874	Binary:	0110101
Unsigned 16 bit:	32618	Unsigned 64 bit:	3299109023671680874	Stream Length:	8
Float 32 bit:	3.823188e+33	Float 64 bit:	3.894247e-88		

☒ Show little endian decoding ☐ Show unsigned and float as hexadecimal

Offset: 0x7; 0x8 bytes from 0x0 to 0x7 selected

<https://github.com/bishwahang/2DESAttack>

At this point I am stuck because when I tried to run the program using the hex signature of both the PNG file and the encrypted file with a key size of 2, it gives me a wrong output. It is shown in figure 5. The reason that I put 2 for the key size length is because it was given to us on one of the lecture slides.

(Figure 5)

The screenshot shows a terminal window titled 'root@kali: ~/Documents/proj2/2DESAttack'. The user has run the command 'python meetinthemiddle.py 89504E470D0A1A0A 0486DE014605796A 2'. The output shows 'Key1:' and 'Key2:' followed by a blank line.

```

root@kali:~/Documents/proj2/2DESAttack# python meetinthemiddle.py 89504E470D0A1A0A 0486DE014605796A 2
Key1:
Key2:
root@kali:~/Documents/proj2/2DESAttack#

```

After that attempt, I tried to convert those hex signatures into plaintext and then tried it again. I then noticed that if you were to put it into plaintext, the ciphertext would come up encrypted so

that did not make sense. From here, I decided to look at the source code and determine where that display is. I found out that the keys were left blank intentionally and that the code was supposed to find those keys and display it. I had googled what an IV is during a 2DES encryption and to see what it was all about. I found out that it is needed for encryption using a CBC or a cipher block chaining. I then looked at the video that was given to us about project 2 and noted that there are many solutions and that these solutions are created for general purposes. It was also said that we should look into how the code works. This is where I began my research into the source code. All of my input from before was in CBC mode, so I decided to switch it to ECB mode in figure 6. It also came up with nothin for the value of keys 1 and 2. I then went onto Piazza to ask a question about it and followed the instructions to include “.decode(‘hex’)” within the python code as shown in figure 7.

(Figure 6: Change to ECB)

```
#the function handed to us to get the decryption done
def twodes(plain, keyOne, keyTwo):
    cipherOne = des(binascii.unhexlify(keyOne), ECB, "5edcc504", pad=None)
    cipherTwo = des(binascii.unhexlify(keyTwo), ECB, "5edcc504", pad=None)
    return cipherTwo.encrypt(cipherOne.encrypt(plain))
```

(Figure 7: Added “.decode(‘hex’)”)

```
# The following is your plaintext/ciphertext pair. We read it from
# stdin:
plaintext = sys.argv[1].decode("hex")
ciphertext = sys.argv[2]
```

It then gave me an output of both keys in figure 8. Key 1 is 0000000000000000ab and Key 2 is 000000000000000012.

(Figure 8: Key 1 and Key 2)

```

root@kali:~/Documents/proj2/2DESAttack# python meetinthemiddle.py 89504e470d0a1a
0a 0486de014605796a 2
Key1:000000000000000ab
Key2:00000000000000012
root@kali:~/Documents/proj2/2DESAttack#

```

Next step was to figure out what to do with the keys. Now that I got the two keys, next was to decrypt the encrypted png file. I googled a website to find a free decryption tool to use those keys and stumbled upon des.online-domain-tools.com. In the boxes, I inputted the encrypted file and along with the first key because of how 2DES works (figure 9). At first, I inputted the keys as a plaintext and it gave me an error that the plaintext length should be between 1-8 bits long and then I decided to change it to hex because I had used hex code to get those keys.

(Figure 9: [des-online-domain-tools.com](https://des.online-domain-tools.com))

DES – Symmetric Ciphers Online

Input type: File

File: C:\fakepath\plant.png.2des Browse

Function: DES

Mode: ECB (electronic codebook)

Key: 000000000000000ab

(hex)

☐ Plaintext ☒ Hex

> Encrypt! > Decrypt! ▶ 🔗

After the first key was used, next up is to decrypt the most recent file that I had gotten from the website. We save that binary file output, in this case I saved it under the name of “EncryptedwithK1” shown in figure 10, and apply key 2 to that file.

(Figure 10: Decrypt with Key 2)

DES – Symmetric Ciphers Online

Input type: File

File: Browse

Function: DES

Mode: ECB (electronic codebook)

Key:
(hex)

☐ Plaintext ☒ Hex

> Encrypt! > Decrypt! ▶ 🔗

100%
File was uploaded.

Decrypted text:



Displayed output of your task is restricted to the first 16 kB of data. You can get the full result using the [Download as a binary file link](#).

00000000	b7 91 53 70 c5 5c 0f c2 26 db 76 b2 36 d5 25 60	· 0 S p Å \ . Å & Û v ² 6 Ö % `
00000010	38 bc d1 55 be c2 5c 53 b1 a8 71 f1 c2 b8 c0 23	8 % Ñ U % Å \ S ± ~ q ñ Å , À #
00000020	d1 6a 46 59 e2 3c b0 49 1c c6 aa 46 40 a3 74 7b	Ñ j F Y â < ° I . Æ ð F @ £ t {
00000030	48 1b 30 86 f7 2b 41 70 43 0c e3 e6 53 7e 55 2c	H . 0 . ÷ + A p C . ã æ S ~ U ,
00000040	b6 b6 74 2e db 36 3f e2 45 06 26 34 38 cb 97 f4	¶ ¶ t . Û 6 ? â E . & 4 8 È . ò
00000050	c6 a4 ea 38 fe 15 68 d2 6e f1 eb 4a 05 3c c8 b4	Æ x ê 8 p . h Ò n ñ ë J . < È ´
00000060	a9 ac 93 63 0c a4 01 5d ba 8d e7 1c 40 d8 1c 2e	@ ~ . c . x .] º 0 ç . @ 0 . .
00000070	cd c6 5d 9f e7 cd 65 0f bb 2c 6e fb 32 28 ae 50	Í Æ J . ç Í e . » , n û 2 (° P
00000080	88 ab a0 35 75 6f 3c f3 87 39 2c 0a 73 f3 ba 91	0 « 5 u o < ó . 9 , . s ó º 0
00000090	ed d6 71 c5 43 75 2b 84 58 4e 09 a9 39 b4 97 c5	í Ö q Å C u + . X N . 0 9 ´ . Å

[Download as a binary file] [Show more] [Show all] [?]

Inactive

This again is where I got stuck. I spent some time decrypting the same file over and over again. I tried to find out where my problem was occurring. First, I tried to mess around with my input in the virtual machine. I replaced the lowercase letters in the code to uppercase and that output gave me nothing for the keys. I then went ahead and uninstalled and installed the same

program again to make sure that I did not corrupt anything. When I downloaded the program again, I changed it again from CBC to ECB and added the decode(“hex”) function as well. I ran the program again and it outputted the same keys again. I then went to research why it was still encrypted. Then it all clicked while I was thinking. If the goal is to decrypt the encrypted file, why would I follow the steps to encrypt the file in the first place. The whole point is to go backwards with the given ciphertext and plaintext. So next, I went to the same online decryption tool but this time used the second key first instead of Key 1 because we are going backwards. Shown in figure 11, is what I inputted in for the boxes.

(Figure 11)

DES – Symmetric Ciphers Online

Input type: File

File: Browse

Function: DES

Mode: ECB (electronic codebook)

Key: (hex)

☐ Plaintext ☒ Hex

> Encrypt! > Decrypt! ▶ 🔗

100%
File was uploaded.

Decrypted text:

Displayed output of your task is restricted to the first 16 kB of data. You can get the full result using the [Download as a binary file link](#).

00000000	20 f5 5a 0f b7 7b 48 c7 27 db d7 94 1c f8 3f 4d	õ Z . . { H Ç ' Û × . . ø ? M
00000010	fe 7f 09 82 f4 42 71 cc ee c3 14 26 97 b0 f0 f8	p . . õ B q ï ï Æ . & . ° ð ø
00000020	91 e2 79 bb 91 fc 32 ad 8a 4f 00 99 ee 8c e5 fc	ð â y » ð ü 2 . . O . . î . ã ü
00000030	38 34 ae 0e 61 09 87 1c 92 e0 3d e6 89 b8 ff 04	8 4 ° . a . . . à = æ . . ý .
00000040	ea de ff ce 71 22 57 63 cd 97 0a 0a b4 ae ae 9c	è p ý î q " W c í . . . ' ° ø ð
00000050	ea 2d c3 50 0b 60 59 7a 53 ec 95 4c 6b c3 2d 64	è - Æ P . ` Y z S ì ð L k Æ - d
00000060	e8 e8 05 67 32 2d 6f 9c c6 7b 6c 1d 0e 14 ac 6c	è è . g 2 - o ð Æ { 1 . . . ÿ 1
00000070	61 e5 07 d9 d1 dc e4 87 67 42 fd a9 6b 89 8d 39	a ã . Û Ñ Û ä . g B ý ø k . ð 9
00000080	40 76 e2 16 8a ee fa 3b cd a8 c2 45 42 ad b0 67	@ v ã . . î ú ; í " Æ E B . ° g
00000090	2d 81 d1 c1 2c 73 d5 77 20 a5 e6 18 23 92 7a 0e	- . Ñ Á , s Õ w ¥ æ . # . z .

[\[Download as a binary file\]](#) [\[Show more\]](#) [\[Show all\]](#) [\[?\]](#)

Inactive

Next, I went ahead and downloaded the binary file of that output with Key 2 and took that binary file and decrypted it with Key 1 as shown in Figure 12.

(Figure 12)

DES – Symmetric Ciphers Online

Input type: File

File: Browse

Function: DES

Mode: ECB (electronic codebook)


Key:
(hex)

☐ Plaintext ☒ Hex

> Encrypt! > Decrypt! ▶ 🔗

100%
File was uploaded.

Decrypted text:

 Displayed output of your task is restricted to the first 16 kB of data. You can get the full result using the *Download as a binary file link*.

00000000	89 50 4e 47 0d 0a 1a 0a 00 00 00 0d 49 48 44 52	. P N G I H D R
00000010	00 00 03 74 00 00 03 ba 08 06 00 00 00 d7 72 15	. . . t . . . e x r .
00000020	7f 00 00 00 01 73 52 47 42 00 ae ce 1c e9 00 00 s R G B . @ i . é . .
00000030	04 96 69 54 58 74 58 4d 4c 3a 63 6f 6d 2e 61 64	. @ i T X t X M L : c o m . a d
00000040	6f 62 65 2e 78 6d 70 00 00 00 00 00 3c 78 3a 78	o b e . x m p < x : x
00000050	6d 70 6d 65 74 61 20 78 6d 6c 6e 73 3a 78 3d 22	m p m e t a x m l n s : x = "
00000060	61 64 6f 62 65 3a 6e 73 3a 6d 65 74 61 2f 22 20	a d o b e : n s : m e t a / "
00000070	78 3a 78 6d 70 74 6b 3d 22 58 4d 50 20 43 6f 72	x : x m p t k = " X M P C o r
00000080	65 20 35 2e 34 2e 30 22 3e 0a 20 20 20 3c 72 64	e 5 . 4 . 0 " > . < r d
00000090	66 3a 52 44 46 20 78 6d 6c 6e 73 3a 72 64 66 3d	f : R D F x m l n s : r d f =

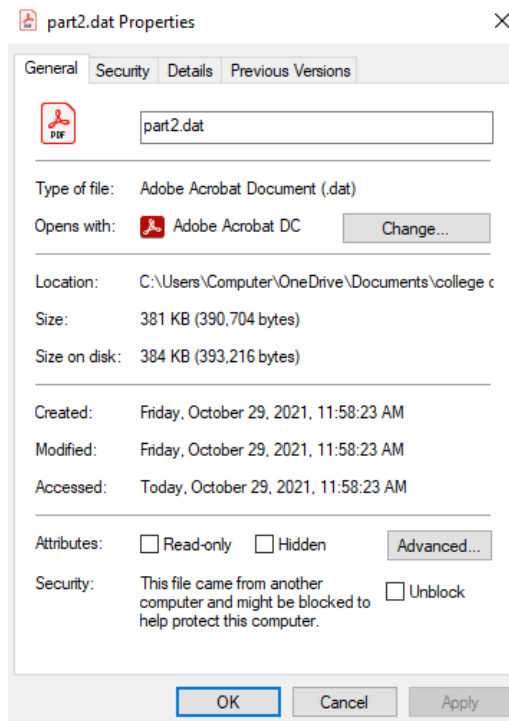
[\[Download as a binary file\]](#) [\[Show more\]](#) [\[Show all\]](#) [\[?\]](#)

Inactive

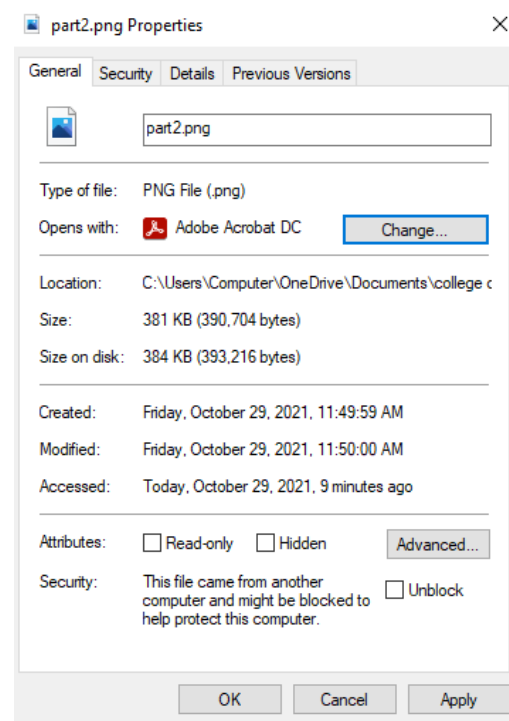
The output here is the decrypted file with both keys and I knew it was normal because of the hex signature of the png file. The PNG file signature matches that of the standard png file. Next step,

I downloaded that binary file and was trying to figure out how to make that png file open as a png file instead of a binary file. The properties on the left, figure 13, are the properties of the decrypted file before I changed it to be a png file. Figure 14 on the right is how I changed it to be a png file. To do that, I changed the file extension to “.png” instead of a “.dat” file and also changed the way the file opens to my default png viewer.

(Figure 13: Pre-changed file)

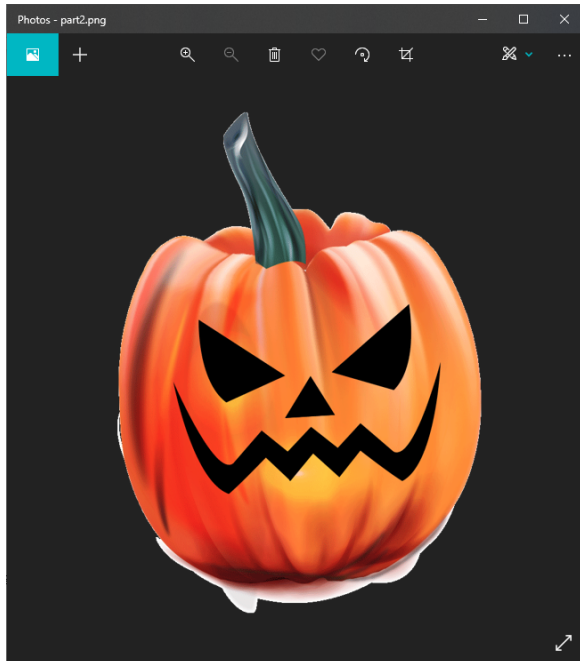


(Figure 14: PNG file)



The last and final step was to open the png file and determine what plant was it and as shown in figure 15, that the plant is a pumpkin.

(Figure 15: Decrypted PNG file)



The overall project took me a great amount of time and thinking. It took me a couple of days to understand what to do and how to do it. I posted a question on Piazza when I did not know what to do next. The process of encrypting and decrypting is what stumped me the most I believe. It took me some time to figure out what I needed to do instead of decrypting it with the first key and then decrypting it with the second key. What also took me some time was looking at the source code and determining what I should change. The source code really stumped me because I had to look at the difference between CBC and ECB, I then had to decide what I should input within the code to make it run. This project has to be one of the hardest projects that I have ever worked on. It required a lot of time to look and understand each part of the process but I'm glad that it finally worked.

Works Cited

<https://www.youtube.com/watch?v=U3dXrJawhg>

<https://github.com/bishwahang/2DESAttack/blob/ec0063733a6156bf32b45db1a3b287c4ee4d6fc/c/meetinthemiddle.py#L102>

https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

<https://github.com/yzchen/cns-attacks/blob/master/03-2des-meet-in-the-middle-attack/2des-meet-in-the-middle-attack.py>

<https://github.com/thomwiggers/des-meet-in-the-middle>