Module 5: Cryptography

Demo 1 – Caesar Cipher

1. Download the caesar_cipher.py file from the LMS
2. Run the caesar_cipher.py file from the terminal

Demo 2 – DES

1. Download the des.py file from the LMS
2. Run the des.py file from the terminal

Demo 3 – AES

1. Download the aes_d.py file from the LMS
2. Run the aes_d.py file from the terminal

Demo 4 – Generating Hash

Problem Statement 1

Generate MD5 hash of a text file

Solution

Syntax:

md5sum <filename> > <hash file name>

Command:

md5sum a.txt > md5.hash

Output:

```
root@kali:~# md5sum a.txt > md5.hash
root@kali:~# cat md5.hash
6f5902ac237024bdd0c176cb93063dc4  a.txt
```

Problem Statement 2

Generate SHA512 hash of a text file

Solution

Syntax:
sha512sum <filename> > <hash file name>

Command:
sha512sum a.txt > sha.hash

Output:

```
root@kali:~# sha512sum a.txt > sha.hash
root@kali:~# cat sha.hash
db3974a97f2407b7cae1ae637c0030687a11913274d578492558e39c16c017de84eacdc8c62fe34e
e4e12b4b1428817f09b6a2760c3f8a664ceae94d2434a593  a.txt
```

Demo 5 – Identifying Hash

Problem Statement
Identify the hashing algorithm from a given hash.

Solution

Step 1: Start the hash-identifier tool

Command:
hash-identifier

Output:

```
root@kali:~# hash-identifier
   #########################################################################
   #     __  __                     __           ______    _____           #
   #    /\ \/\ \                   /\ \         /\__  _\  /\  _ `\         #
   #    \ \ \_\ \     __      ____ \ \ \___     \/_/\ \/  \ \ \/\ \        #
   #     \ \  _  \  /'__`\   / ,__\ \ \  _ `\      \ \ \   \ \ \ \ \       #
   #      \ \ \ \ \/\ \_\ \_/\__, `\ \ \ \ \ \      \_\ \__ \ \ \_\ \      #
   #       \ \_\ \_\ \___ \_\/\____/  \ \_\ \_\     /\_____\ \ \____/      #
   #        \/_/\/_/\/__/\/_/\/___/    \/_/\/_/     \/_____/  \/___/  v1.1 #
   #                                                             By Zion3R #
   #                                                    www.Blackploit.com #
   #                                                   Root@Blackploit.com #
   #########################################################################

   --------------------------------------------------------------------------
 HASH:
```

Step 2: Enter the hash and hit Enter

```
 HASH: 5d41402abc4b2a76b9719d911017c592

Possible Hashs:
[+]  MD5
[+]  Domain Cached Credentials - MD4(MD4(($pass)).(strtolower($username)))

Least Possible Hashs:
[+]  RAdmin v2.x
[+]  NTLM
[+]  MD4
[+]  MD2
[+]  MD5(HMAC)
[+]  MD4(HMAC)
[+]  MD2(HMAC)
[+]  MD5(HMAC(Wordpress))
[+]  Haval-128
[+]  Haval-128(HMAC)
[+]  RipeMD-128
[+]  RipeMD-128(HMAC)
[+]  SNEFRU-128
[+]  SNEFRU-128(HMAC)
[+]  Tiger-128
```

The tool gives shows the list of the most possible hash type and the least possible hash type.

Demo 6 – Signing a File with Digital Signature

Problem Statement

Sign a file with a digital signature.

Solution

Generate the keys that can be used to digitally signing files.

Step 1:

Command:

```
gpg --full-generate-key
```

Step 2: Select the algorithm to be used

```
root@kali:~# gpg --full-generate-key
gpg (GnuPG) 2.2.12; Copyright (C) 2018 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Please select what kind of key you want:
   (1) RSA and RSA (default)
   (2) DSA and Elgamal
   (3) DSA (sign only)
   (4) RSA (sign only)
Your selection? 1
```

Step 3: Enter the keysize and hit Enter

```
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (3072) 4096
```

Step 4: Enter the validity of the key and hit Enter

```
Please specify how long the key should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for? (0) 0
```

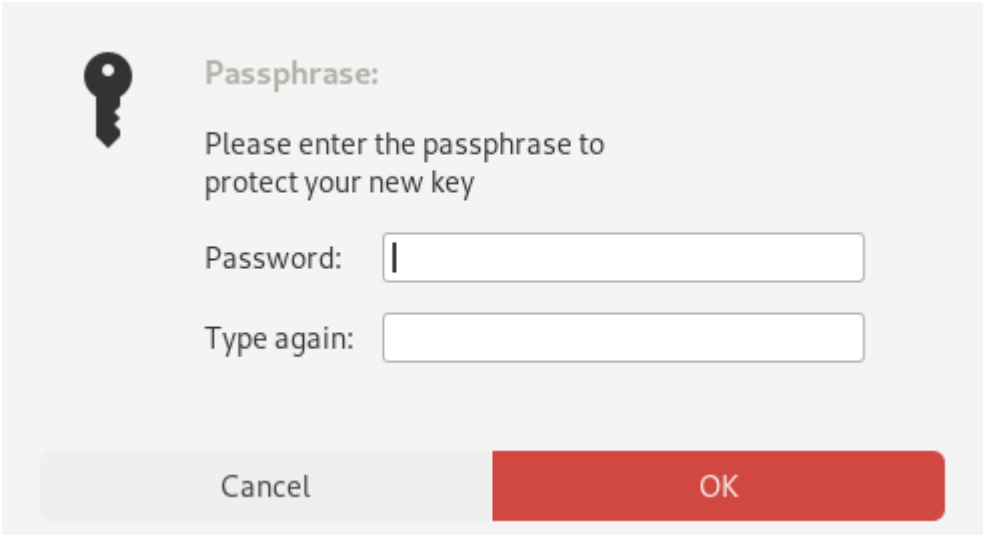
Step 5: Enter required details and confirm

```
GnuPG needs to construct a user ID to identify your key.

Real name: edureka
Email address: abc@gmail.com
Comment: none
You selected this USER-ID:
    "edureka (none) <abc@gmail.com>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? O
```

Step 6: Enter the passphrase



Keys will be generated

```
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
gpg: /root/.gnupg/trustdb.gpg: trustdb created
gpg: key 3CCAD3756091396D marked as ultimately trusted
gpg: directory '/root/.gnupg/openpgp-revocs.d' created
gpg: revocation certificate stored as '/root/.gnupg/openpgp-revocs.d/B93C2A69587
401E927B664253CCAD3756091396D.rev'
public and secret key created and signed.

pub   rsa4096 2020-03-13 [SC]
      B93C2A69587401E927B664253CCAD3756091396D
uid                      edureka (none) <abc@gmail.com>
sub   rsa4096 2020-03-13 [E]
```

Step 7:

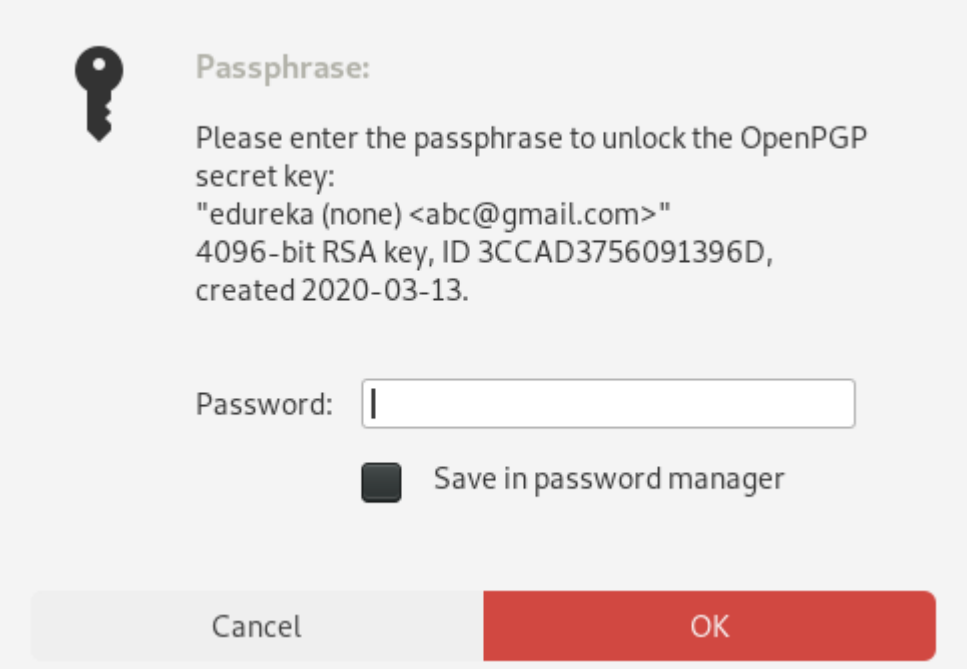
Syntax:

`gpg --sign <filename>`

Command:

gpg --sign a.txt

Step 2: Enter passphrase



This will generate a file with **.gpg** extension as shown in the image below

```
root@kali:~# gpg --sign a.txt
root@kali:~# ls
a.txt      Desktop    Downloads  Pictures  Templates
a.txt.gpg  Documents  Music      Public    Videos
```

Demo 7 – Known-plaintext attack

1. Download the known_plaintext_attack.py file from the LMS
2. Run the known_plaintext_attack.py file from the terminal