Recover Saved Passwords in Chromium-based Browsers

Introduction

- Browsers play a pivotal role in our online activities.
- They are gateways to the web, handling a wide array of information, including login credentials.
- Browser security is crucial, both for users and forensic investigators.

Problem Statement

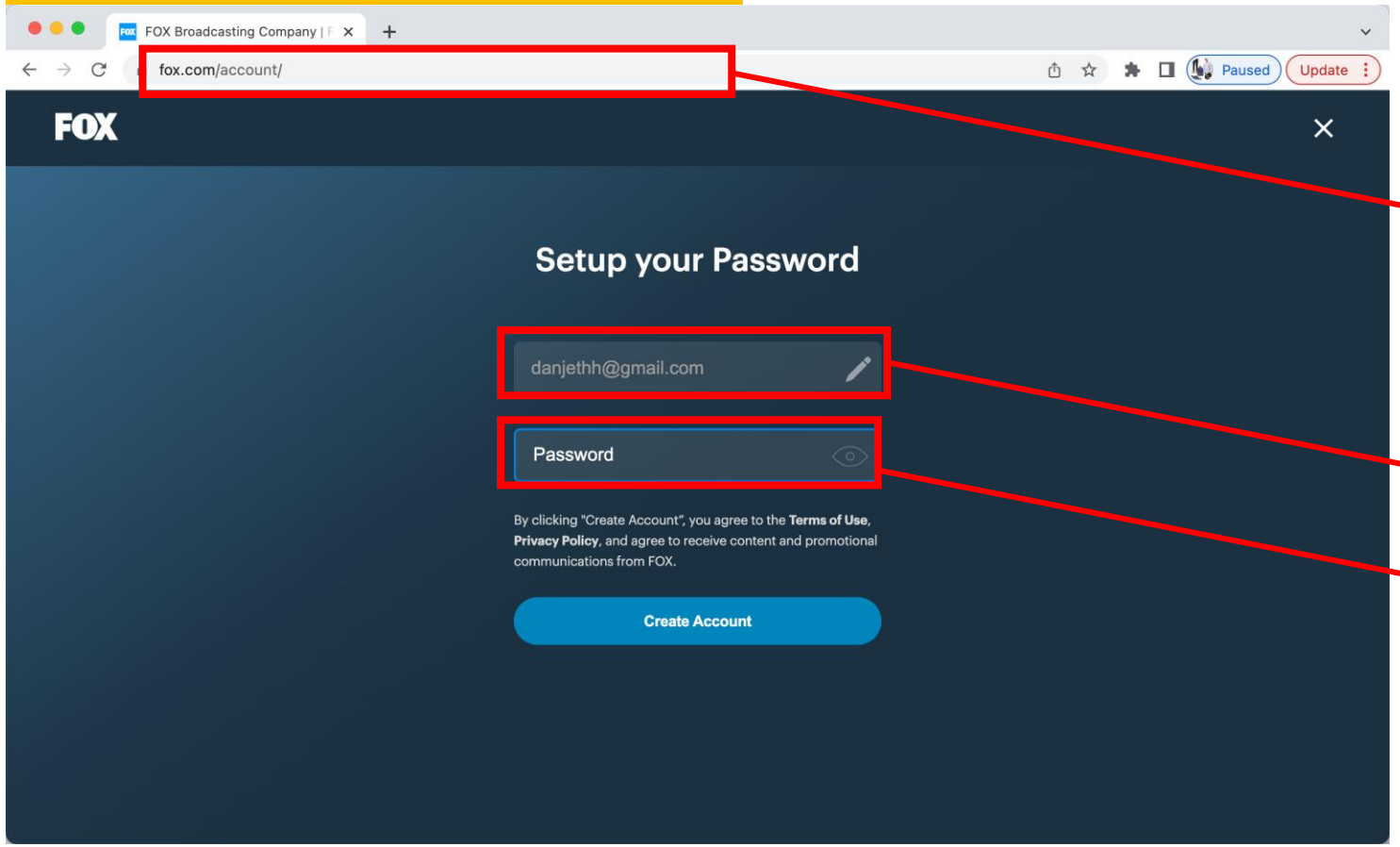
- While browsers offer convenience and access to a world of information, they also handle something more - the browser plays a crucial role in managing your login details.
- Browsers are the keepers of sensitive data, including login credentials, payment information, and browsing history. Understanding how browsers store and secure this information is not just a matter of curiosity; it's a critical aspect of modern digital security.

The Significance of the problem

- Chromium-based browsers are widely used, including Google Chrome and Microsoft Edge.
- They are essential in digital investigations due to their market prevalence.
- Understanding how they store and encrypt login credentials is a key forensic aspect.

Creating an account or Signing into a website every time you visit can be cumbersome

Launch any Chromium Based Browser

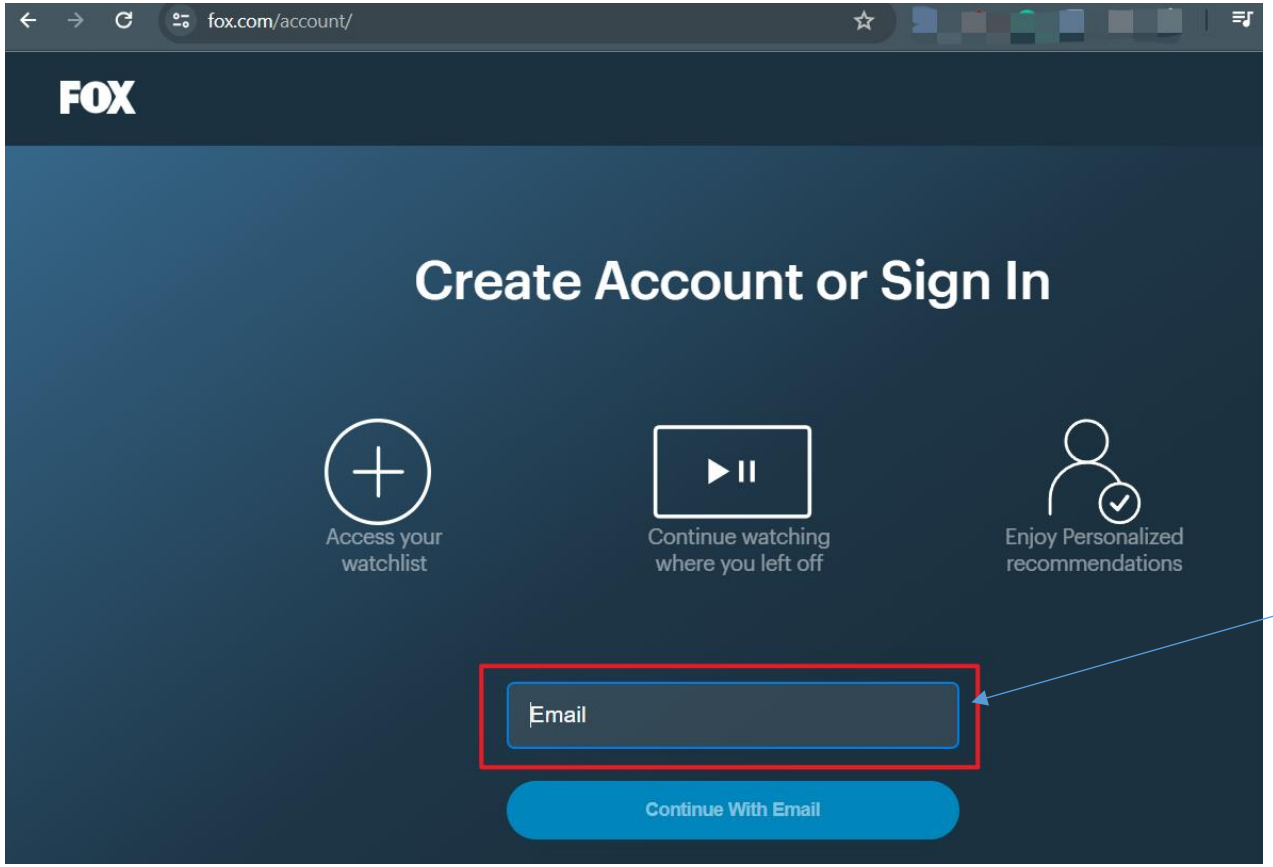


Account Sign-Up URL

Email address

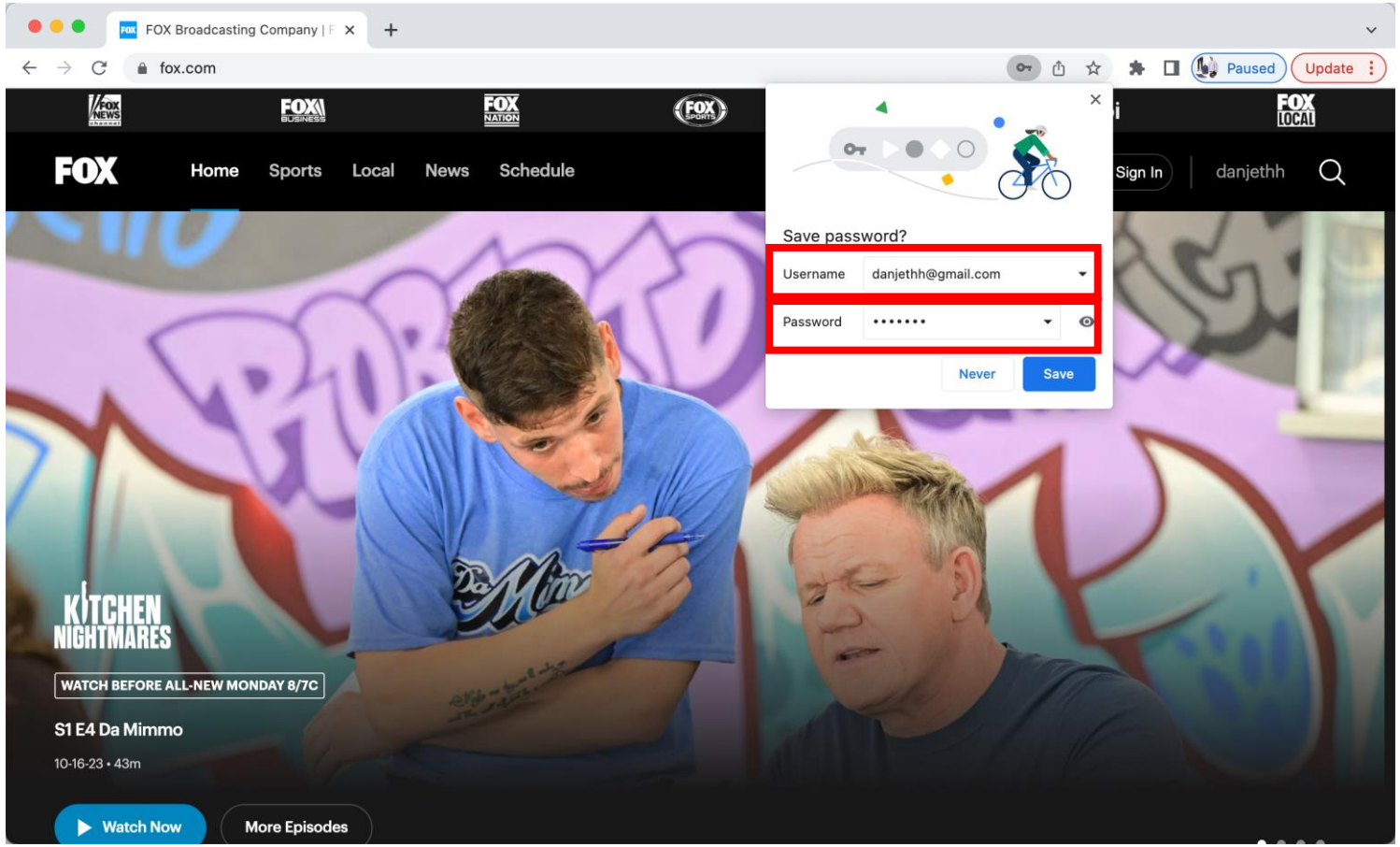
Password

Signing into a website every time your visit can be cumbersome

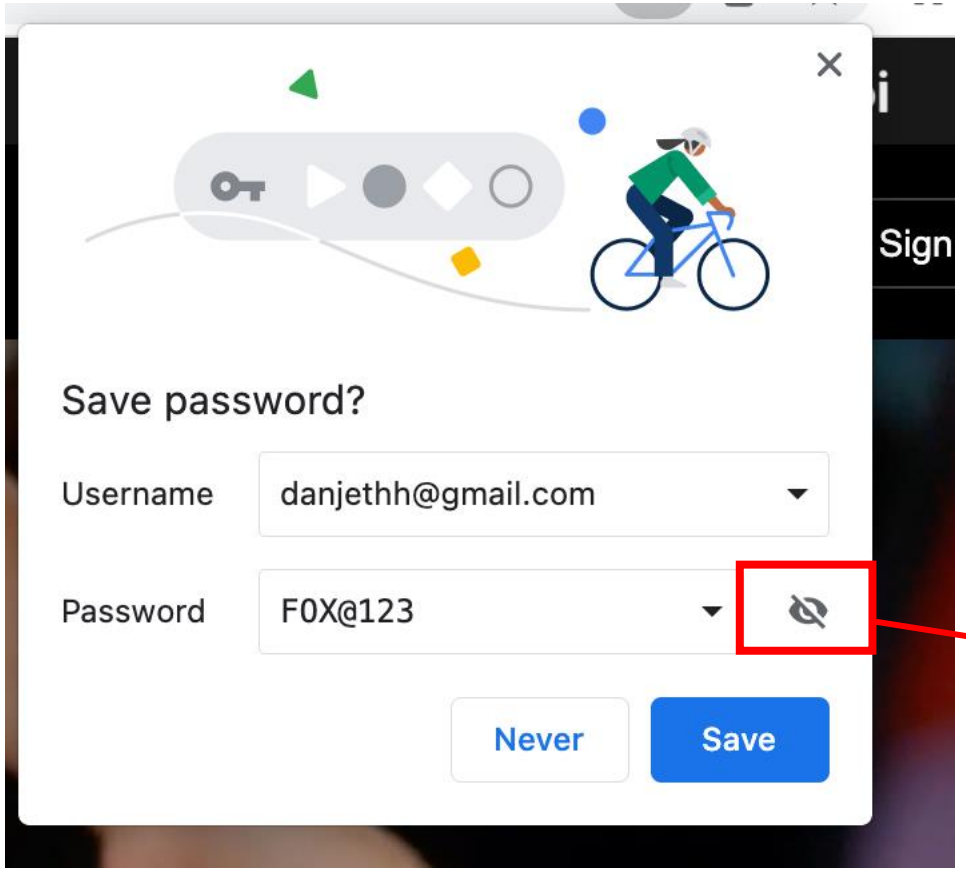


You have to type your username and password each time

Browsers save users' credentials for convenience



Browsers save users' credentials for convenience



Chrome stores the username, Password and its associated URL link in a Database file called "Login Data". The Plaintext website Password is encrypted using the Chrome-generated Secret key.

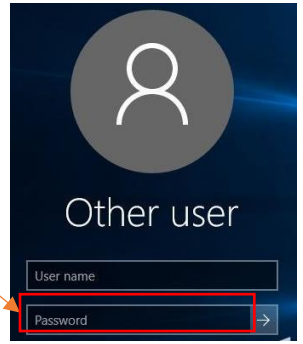
Investigation goal

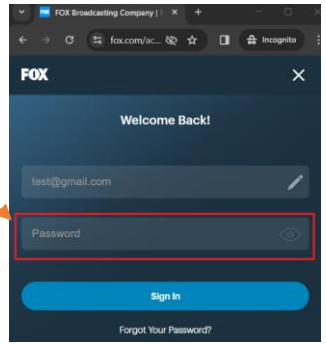
- Recovering saved passwords in Chromium-based browsers for a given website
  - e.g., find password of fox.com
- Note: you only access a disk image

Image generated by Bing

Terms related to password (1)

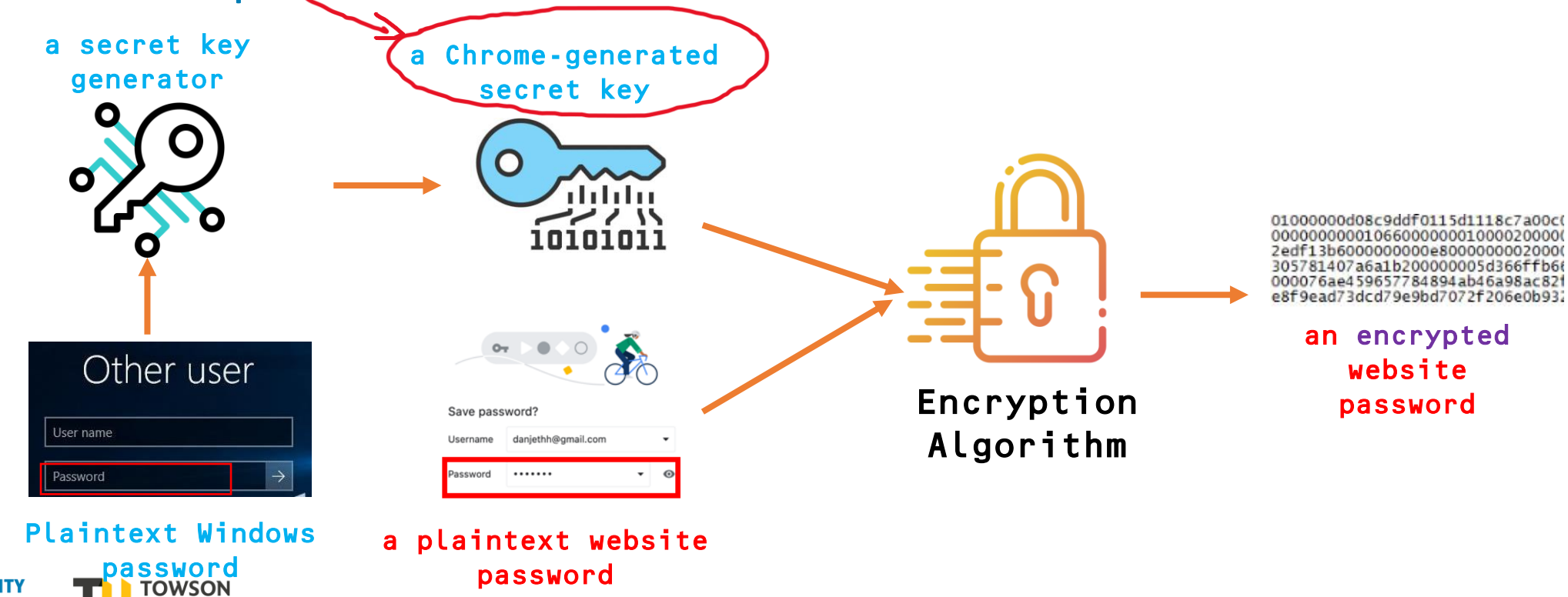
- Plaintext Windows password: a user password in plaintext used to log into a Windows OS
- Plaintext website password: a password in plaintext for a website running on a Chromium-based browser
- Chrome-Generated Secret Key: the key generated by chrome from the plaintext windows password.
- Encrypted website password: a password encrypted with Chrome-generated secret key and stored in a login Database





Terms related to password (2)

- Chrome-generated secret key: A secret key derived from a plaintext Windows password. The secret key is used to encrypt a plaintext website password.



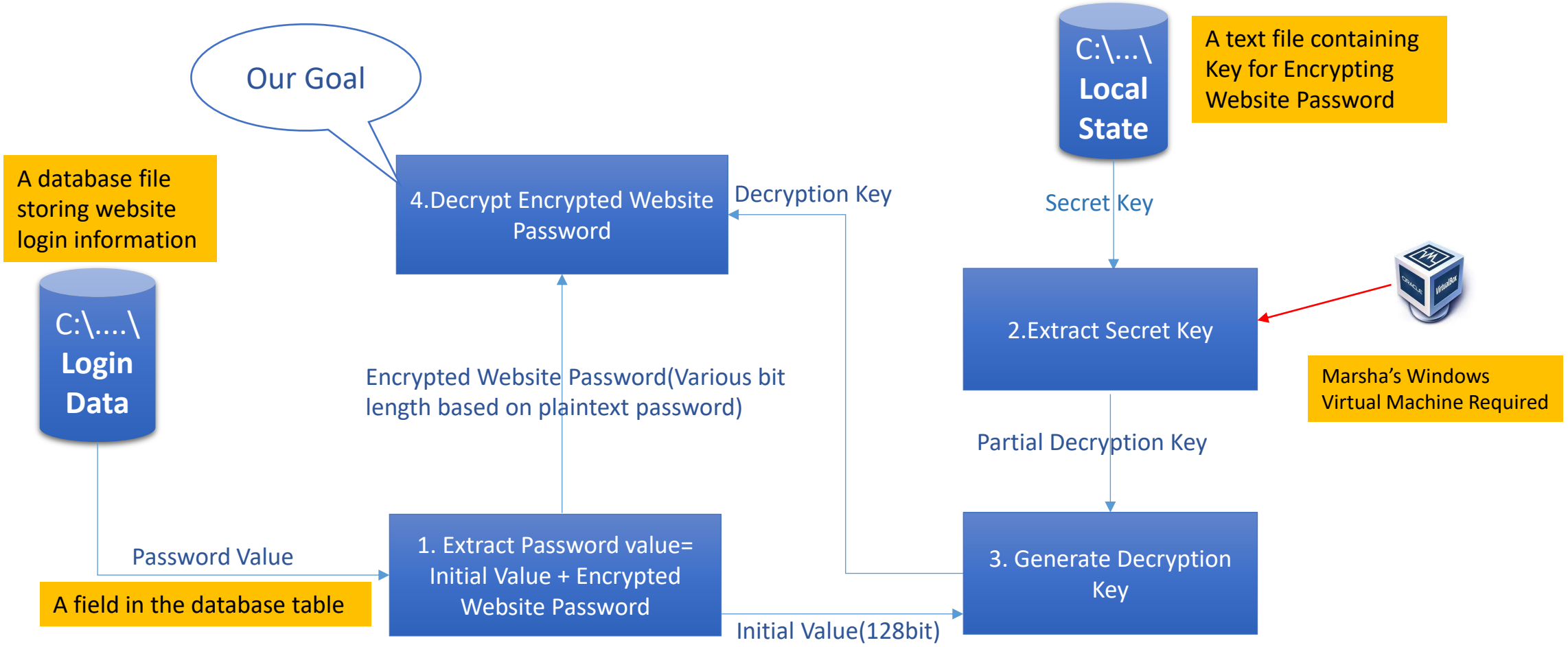
Steps

Prerequisites:

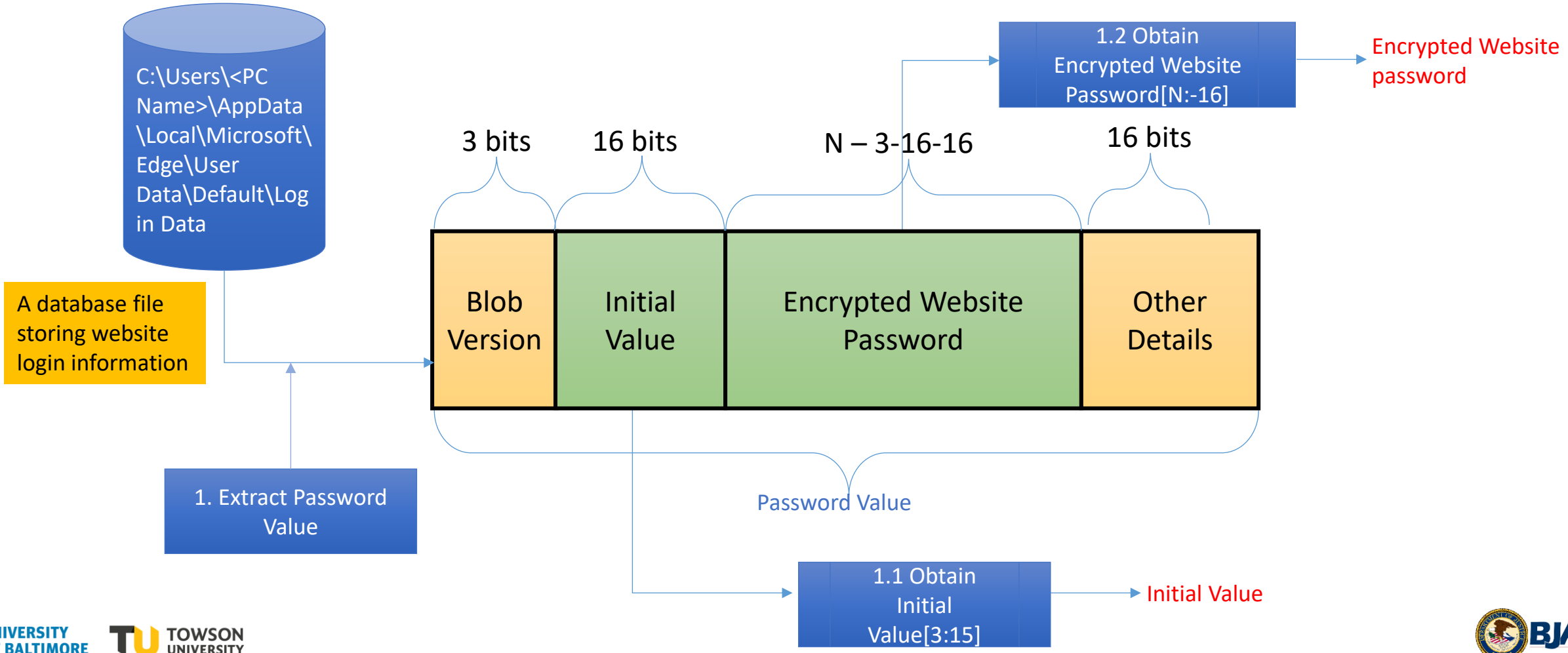
- Follow the PPT to convert the E01 forensic image into a Windows Virtual Machine for this activity: 00_Convert_Forensic Image_to_Windows_Virtual_Machine_Disk.pptx
- Link to how PIN was cracked: 10_WindowsHello_Login_PinCode.pptx
- Run Python script in the bootable Marsha's Windows Virtual Machine

1. Extract Key String= Initial Value + Encrypted Website Password.
2. Find the partial decryption key.
3. Generate decryption key.
4. Decrypt Encrypted Website Password Using AES.

How Info Stealer Malwares are used to Decrypt and Extract Your Saved Browser(Chromium) Passwords



Step 1: How to Extract the Password Value



Step 1.1: Extract Password Value

- The Password Value is stored in an SQLite database which can be found in the following location of your Windows PC
- **C:\Users\Marsh\AppData\Local\Microsoft\Edge\User Data\Default\Login Data**

```
(base) ┌──(root💀kali)-[/media/…/Local/Google/Chrome/User Data]
└─# cat /media/kali/Windows/Users/marsh/AppData/Local/Google/Chrome/User\ Data/Local\ State
/home/kali/Documents/
```

Copy the ”Login Data” database file from the Image to the Documents folder of the Local System.

Step 1.1: Extract Password Value

Show the Password Value stored in the Login Data Database file using SQL

```
(base) ┌──(root💀kali)-[/home/kali/Documents]
└─# file "Login Data"
Login Data: SQLite 3.x database, last written using SQLite version 3035005, page size 2048, fi
le counter 26, database pages 24, cookie 0xe, schema 4, UTF-8, version-valid-for 26

(base) ┌──(root💀kali)-[/home/kali/Documents]
└─# sqlite3 "Login Data"
SQLite version 3.41.2 2023-03-22 11:56:21
Enter ".help" for usage hints.
sqlite>
```

Using SQLite3 tool to access the content of the file

Show the Password Value stored in the ‘Logins’ Table

```
(base) ┌──(root💀kali)-[/home/kali/Documents]
└─# sqlite3 "Login Data"
SQLite version 3.41.2 2023-03-22 11:56:21
Enter ".help" for usage hints.
sqlite> .tables
breached                logins                  stats
field_info              logins_edge_extended    sync_entities_metadata
insecure_credentials    meta                    sync_model_metadata
sqlite>
```

Show the column name storing the URL, username and Password_Value

```
sqlite> .schema logins
CREATE TABLE logins (origin_url VARCHAR NOT NULL, action_url VARCHAR, username_element VARCHAR
, username_value VARCHAR, password_element VARCHAR, password_value BLOB, submit_element VARCHA
R, signon_realm VARCHAR NOT NULL, date_created INTEGER NOT NULL, blacklisted_by_user INTEGER N
OT NULL, scheme INTEGER NOT NULL, password_type INTEGER, times_used INTEGER, form_data BLOB, d
ate_synced INTEGER, display_name VARCHAR, icon_url VARCHAR, federation_url VARCHAR, skip_zero_
click INTEGER, generation_upload_status INTEGER, possible_username_pairs BLOB, id INTEGER PRIM
ARY KEY AUTOINCREMENT, date_last_used INTEGER NOT NULL DEFAULT 0, moving_blocked_for BLOB, UNI
QUE (origin_url, username_element, username_value, password_element, signon_realm));
CREATE INDEX logins_signon ON logins (signon_realm);
sqlite>
```

Combined String is stored in binary format in the Database

Display URL and its associated Usernames and Password Values.

```
sqlite> SELECT action_url, username_value, password_value FROM logins;

https://login.live.com/ppsecure/post.srf|marsha4mellos@gmail.com|v10◆◆◆◆>◆`◆n9◆,K◆◆◆zSZ◆◆◆◆|◆◆
◆蹸◆◆ПQ!◆◆
```

Display the Password_value(In binary Format)

```
sqlite> SELECT password_value FROM logins;

v10◆◆◆◆>◆`◆n9◆,K◆◆◆zSZ◆◆◆◆|◆◆◆蹸◆◆ПQ!◆◆
```

Show the Password Value stored in the Login Data Database file using python script

```python
def get_db_connection(chrome_path_login_data):
    try:
        shutil.copy2(chrome_path_login_data, "Loginvault.db")
        conn = sqlite3.connect("Loginvault.db")
        print("connection successful")
        cursor = conn.cursor()
        cursor.execute("SELECT origin_url, username_value, password_value FROM logins")
        for index, login in enumerate(cursor.fetchall()):
            url = login[0]
            username = login[1]
            ciphertext = login[2]
            print("Sequence: %d" % index)
            print("URL: %s\nUser Name: %s\nPassword: %s\n" % (url, username, ciphertext))
            print("*" * 100)
            cursor.close()
        conn.close()
```

```
Sequence: 0
URL: https://login.live.com/login.srf
User Name: marsha4mellos@gmail.com
Password: b'v10\x99\xf6\xc5\xf2>\xea`\xbdn9\xb5\x11,K\xfe\xac\xf9zSZ\xad\x80\xc6\xea|\xec\xf6\x0e\x8f\xe8\xb9\xb8\xd4\xd4\xd4\xa4Q!\x90\x94'

****************************************************************************************************
```

Checking connection to Login Data database file

Fetching require URL, Username and Password values from Database

Print out the data fetched

Password value(Python displays the string in a Byte format)

Step 1.2: Obtain Initial Value and Encrypted Website Password

Password Value(Ciphertext)

Show the initial value and Encrypted Website password from the Password value using the python script

```
def decrypt_password(ciphertext):
    try:
        #(3-a) Initialisation vector for AES decryption
        initialisation_vector = ciphertext[3:15]
        #(3-b) Get encrypted password by removing suffix bytes (last 16 bits)
        #Encrypted password is 192 bits
        encrypted_password = ciphertext[15:-16]
        #(4) Build the cipher to decrypt the ciphertext
        print("initialisation_vector: %s\nencrypted_password: %s\n" % (initialisation_vector, encrypted_password))
        print("*" * 100)
```

```
****************************************************************************************************
initialisation_vector: b'\x99\xf6\xc5\xf2>\xea`\xbdn9\xb5\x11'
encrypted_password: b',K\xfe\xac\xf9zSZ\xad\x80\xc6\xea'

****************************************************************************************************
```

Show the initial value and Encrypted Website password from the Password Value using the python script

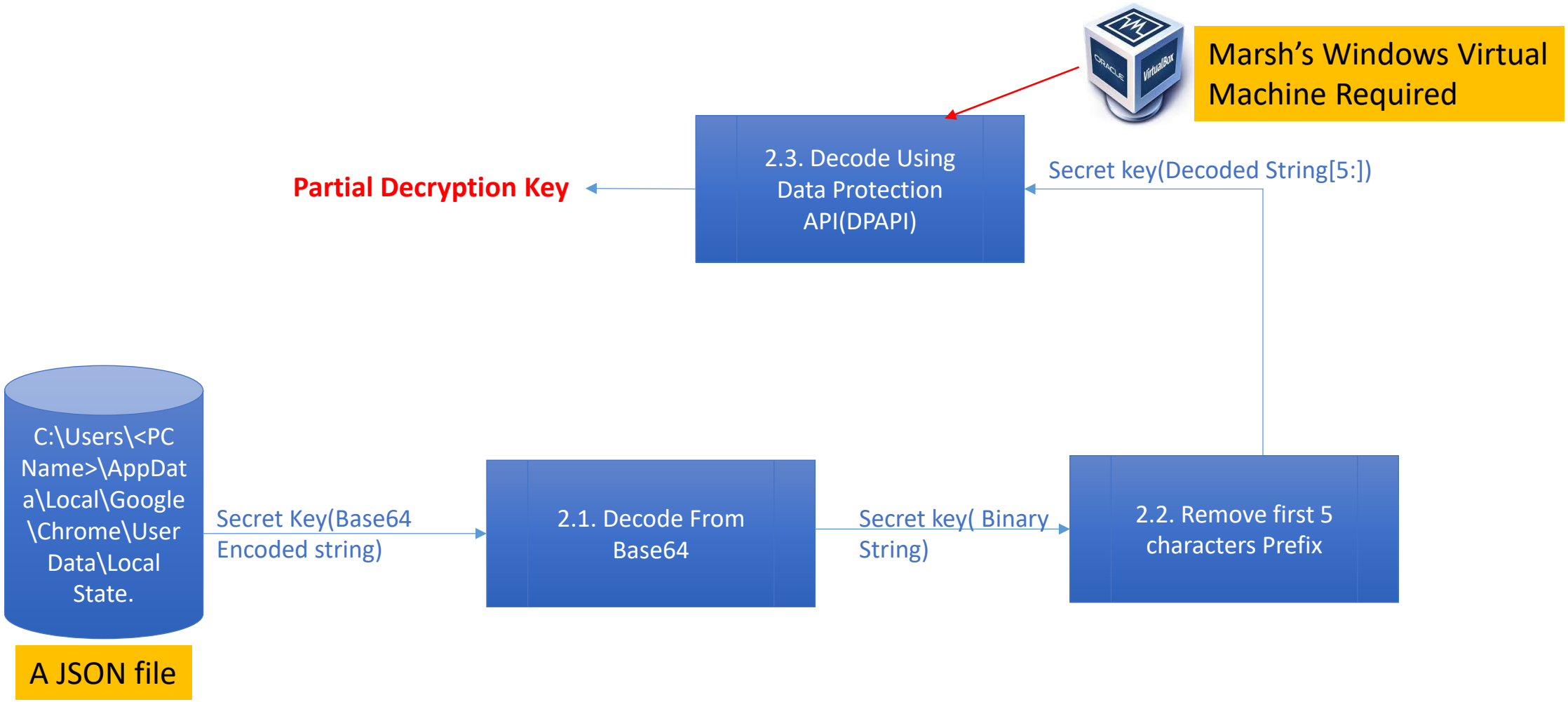
```
connection successful
Sequence: 0
URL: https://login.live.com/login.srf
User Name: marsha4mellos@gmail.com
Password: b'v10\x99\xf6\xc5\xf2>\xea`\xbdn9\xb5\x11,K\xfe\xac\xf9zSZ\xad\x80\xc6\xea|\xec\xf6\x0e\x8f\xe8\xb9\xb8\xd4\xd4\xd4\xa4Q!\x90\x94'

*****************************************************************************************
initialisation_vector: b'\x99\xf6\xc5\xf2>\xea`\xbdn9\xb5\x11'
encrypted_password: b',K\xfe\xac\xf9zSZ\xad\x80\xc6\xea'

*****************************************************************************************
```

Initial Value

Encrypted Website Password

Password Value(Ciphertext)

Step 2: How to Find the partial decryption key



Step 2 How to Find the partial decryption key

- The Chrome-generated Secret key is stored in a JSON file which can be found in the following location on your Windows PC: C:\Users\<PC Name>\AppData\Local\Google\Chrome\User Data**Local State.**

```
(base) ┌──(root💀kali)-[/media/…/Local/Google/Chrome/User Data]
└─# cat /media/kali/Windows/Users/marsh/AppData/Local/Google/Chrome/User\ Data/Local\ State
/home/kali/Documents/
```

Copy the ”Local State” file from the Image to the Documents folder of the Local System.

Step 2.1 Find the Decoded partial decryption key

Show the Encoded Base64 string stored in the “Local State” JSON file

```
(base) ┌──(root💀kali)-[/home/kali/Documents]
└─# file Local\ State
Local State: JSON data

(base) ┌──(root💀kali)-[/home/kali/Documents]
└─# jq ".os_crypt" Local\ State
{
  "encrypted_key": "RFBBUEkBAAAA0Iyd3wEV0RGMegDAT8KX6wEAAAA3TAnCzsDgTJj+U+CaVT01EAAAAAoAAABFAG
QAZwBlAAAAEGYAAAABAAAgAAAAZ+u476qy9iWIHZinVBtV9BRDSNaU7lYQ3mfOOBPUmeIAAAAADoAAAAACAAAgAAAAG7D+
CDT2QfUoZ5yBCielGzge/ZEMD7P2308lAqHRla0wAAAAPVQ1kJXonwrstIZfNw4VvkSdnsabpGDn1OiQd0rhMNx2iYcKc8
QmzYDaKKd6PDAaQAAAABLCUzxf4IEHYIygJt0jKIEexmdQbxWQSRLxPrDwuK/RacqdtYukc2lpPqD3O7VaS+VH1gbGuxF1
hGTA1c0v7/Y="
}
```

Using the ”jq” tool to access the value of the “os_crypt” Key to get the Partial Decryption key

Partial Decryption Key

JSON is a structured data format. It is in a form of a Dictionary Data Structure with a Key and Value Pair(“Key”:”Value”)
“jq” is a lightweight and flexible command-line JSON processor.

Step 2.1 Decode the Partial Decryption Key from Base64

Decoding the Partial Decryption Key from Base64 using Python script

```
def get_secret_key():
    try:
        with open(CHROME_PATH_LOCAL_STATE, "r", encoding='utf-8') as f:
            local_state = json.load(f)
        secret_key = base64.b64decode(local_state["os_crypt"]["encrypted_key"])
        print(secret_key)
```

```
b'DPAPI\x01\x00\x00\x00\xd0\x8c\x9d\xdf\x01\x15\xd1\x11\x8cz\x00\xc0O\xc2\x97\xeb\x01\x00\x00\x007L\t\xc2\
xce\xc0\xe0L\x98\xfeS\xe0\x9aU=5\x10\x00\x00\x00\n\x00\x00\x00E\x00d\x00g\x00e\x00\x00\x00\x10f\x00\x00\x0
0\x01\x00\x00 \x00\x00\x00g\xeb\xb8\xef\xaa\xb2\xf6%\x88\x1d\x98\xa7T\x1bU\xf4\x14CH\xd6\x94\xeeV\x10\xdeg
\xce8\x13\xd4\x99\xe2\x00\x00\x00\x00\x0e\x80\x00\x00\x00\x02\x00\x00 \x00\x00\x00\x1b\xb0\xfe\x084\xf6A\x
f5(g\x9c\x81\n'\xa5\x1b8\x1e\xfd\x91\x0c\x0f\xb3\xf6\xdfO%\x02\xa1\xd1\x95\xad0\x00\x00\x00=T5\x90\x95\xe8
\x9f\n\xec\xb4\x86_7\x0e\x15\xbeD\x9d\x9e\xc6\x9b\xa4`\xe7\xd4\xe8\x90wJ\xe10\xdcv\x89\x87\ns\xc4&\xcd\x80
\xda(\xa7z<0\x1a@\x00\x00\x00\x12\xc2S<_\xe0\x81\x07`\x8c\xa0&\xdd#(\x81\x1e\xc6gPo\x15\x90I\x12\xf1>\xb0\
xf0\xb8\xaf\xd1i\xca\x9d\xb5\x8b\xa4sii>\xa0\xf7;\xb5ZK\xe5G\xd6\x06\xc6\xbb\x11u\x84d\xc0\xd5\xcd/\xef\xf
6"
```

Decoded Partial Decryption key

Step 2.2 Remove first 5 characters Prefix

Remove first 5 characters Prefix from Base64 Decoded Partial Decryption Key

```
def get_secret_key():
    try:
        with open(CHROME_PATH_LOCAL_STATE, "r", encoding='utf-8') as f:
            local_state = json.load(f)
        secret_key = base64.b64decode(local_state["os_crypt"]["encrypted_key"])
        print(secret_key)
        # Remove the DPAPI prefix
        secret_key = secret_key[5:]
        print(secret_key)
```

```
b"\x01\x00\x00\x00\xd0\x8c\x9d\xdf\x01\x15\xd1\x11\x8cz\x00\xc0O\xc2\x97\xeb\x01\x00\x00\x007L\t\xc2\xce\x
c0\xe0L\x98\xfeS\xe0\x9aU=5\x10\x00\x00\x00\n\x00\x00\x00E\x00d\x00g\x00e\x00\x00\x00\x10f\x00\x00\x00\x01
\x00\x00 \x00\x00\x00g\xeb\xb8\xef\xaa\xb2\xf6%\x88\x1d\x98\xa7T\x1bU\xf4\x14CH\xd6\x94\xeeV\x10\xdeg\xce8
\x13\xd4\x99\xe2\x00\x00\x00\x00\x0e\x80\x00\x00\x00\x02\x00\x00 \x00\x00\x00\x1b\xb0\xfe\x084\xf6A\xf5(g\
x9c\x81\n'\xa5\x1b8\x1e\xfd\x91\x0c\x0f\xb3\xf6\xdfO%\x02\xa1\xd1\x95\xad0\x00\x00\x00=T5\x90\x95\xe8\x9f\
n\xec\xb4\x86_7\x0e\x15\xbeD\x9d\x9e\xc6\x9b\xa4`\xe7\xd4\xe8\x90wJ\xe10\xdcv\x89\x87\ns\xc4&\xcd\x80\xda(
\xa7z<0\x1a@\x00\x00\x00\x12\xc2S<_\xe0\x81\x07`\x8c\xa0&\xdd#(\x81\x1e\xc6gPo\x15\x90I\x12\xf1>\xb0\xf0\x
b8\xaf\xd1i\xca\x9d\xb5\x8b\xa4sii>\xa0\xf7;\xb5ZK\xe5G\xd6\x06\xc6\xbb\x11u\x84d\xc0\xd5\xcd/\xef\xf6"
```

Remove the DPAPI Prefix

Decoded Partial Decryption Key

Step 2.3: How to Find the partial decryption key

```
with open(CHROME_PATH_LOCAL_STATE, "r", encoding='utf-8') as f:
    local_state = f.read()
    local_state = json.loads(local_state)
secret_key = base64.b64decode(local_state["os_crypt"]["encrypted_key"])
#Remove suffix DPAPI
secret_key = secret_key[5:]
secret_key = win32crypt.CryptUnprotectData(secret_key, None, None, None, 0)[1]
print(secret_key)
```

Run on Marsha's Windows Virtual Machine

Decoded Partial Decryption Key using DPAPI

```
**************************************************
Secret Key: b"\xd0\xa0\xc9'w\xe5\xf4\x0c\x17\xa2\x0c$\x06\xe47V:\x1b \xb5\xe0\xebM\x14\x1cz\xc8\xae\xe2{\xe9\xe8"
C:\Users\marsh\AppData\Local\Microsoft\Edge\User Data\Default\Login Data
**************************************************
```

Partial Decryption Key

Step 3: Generate decryption key

```python
def decrypt_password(ciphertext, secret_key):
    try:
        #(3-a) Initialisation vector for AES decryption
        initialisation_vector = ciphertext[3:15]
        #(3-b) Get encrypted password by removing suffix bytes (last 16 bits)
        #Encrypted password is 192 bits
        encrypted_password = ciphertext[15:-16]
        #(4) Build the cipher to decrypt the ciphertext
        cipher = generate_cipher(secret_key, initialisation_vector)
        print(cipher)
```

Generate Decryption Key

```
*****************************************************
cipher: <Cryptodome.Cipher._mode_gcm.GcmMode object at 0x0000024E47C71190>
*****************************************************
```

Decryption key Generation successful

Step 4: Decrypt Encrypted Website Password

```
cipher = generate_cipher(secret_key, initialisation_vector)
print(cipher)
decrypted_pass = decrypt_payload(cipher, encrypted_password)
decrypted_pass = decrypted_pass.decode()
print(decrypted_pass)
```

Decrypt Encrypted Website Password

```
**********************************************
decrypted_pass: b'C3ll3br1t3!m'
**********************************************
decrypted_pass: C3ll3br1t3!m
```

Decrypted Website Password

```
*****************************************************************
decrypted_pass: C3ll3br1t3!m
Sequence: 0
URL: https://login.live.com/ppsecure/post.srf
User Name: marsha4mellos@gmail.com
Password: C3ll3br1t3!m
```

Locating Chrome Resources

- **Google Chrome**

Windows: %LocalAppData%\Google\Chrome\User Data\Default

MacOS: ~/Library/Application Support/Google/Chrome

Linux: ~/.config/google-chrome

- **Microsoft Edge (Chromium):**

Windows: %LocalAppData%\Microsoft\Edge\User Data\Default

MacOS: ~/Library/Application Support/Microsoft Edge Dev

- **Brave:**

Windows: %LocalAppData%\BraveSoftware\Brave-Browser\User Data\Default

MacOS: ~/Library/Application Support/BraveSoftware/Brave-Browser

Linux: ~/.config/BraveSoftware/Brave-Browser

- **Opera:**

Windows: %AppData%\Opera Software\Opera Stable

MacOS: ~/Library/Application Support/com.operasoftware.Opera

Linux: ~/.config/opera