



LAB WORKBOOK

IAM Bootcamp

**Identity & Access
Management (IAM)
Cybersecurity**

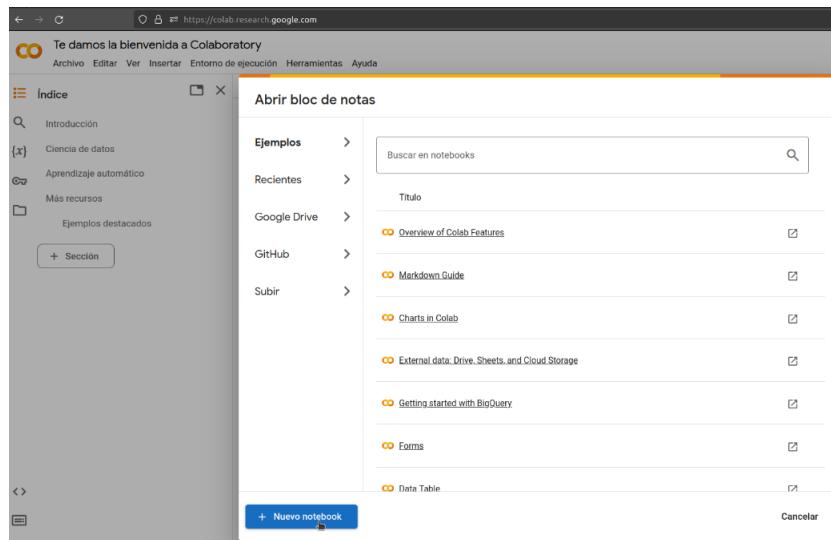


Laboratory 1. Hashing Basics

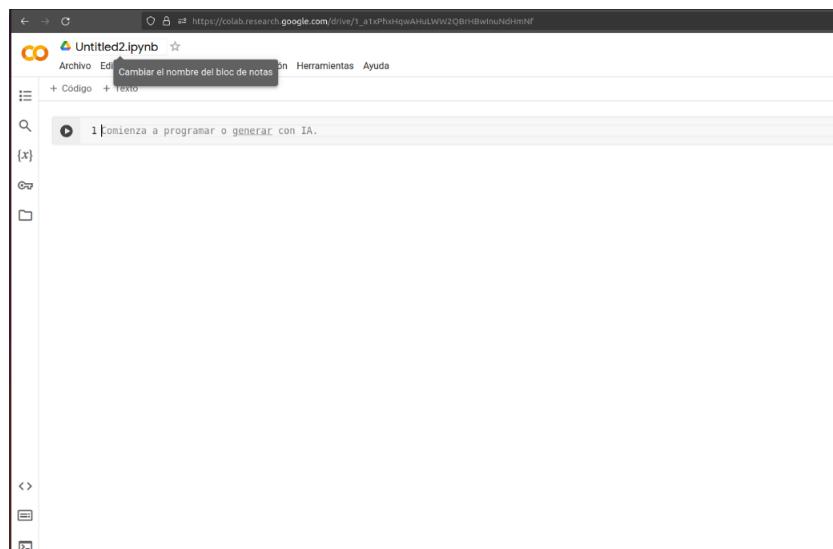
Prepare a Google Account.

TASK 1

Step 1. Go to <https://colab.research.google.com> and click on New Notebook.



Step 2. You can change the notebook's name if you wish and use the cell to navigate on the machine provided by Google and program in Python or R.



Step 3. Enter to the **sample_data** folder and locate the files in that folder.

```
[1] 1 pwd
{x} ↵ /content
[13] 1 cd sample_data/
[14] 1 pwd
↳ ↵ '/content/sample_data'
[15] 1 ls -la
↳ total 5512
drwxr-xr-x 1 root root    4096 May  9 13:24 .
drwxr-xr-x 1 root root    4096 May  9 13:24 ..
-rw-r--r-- 1 root root     1697 Jan  1 2000 anscombe.json*
-rw-r--r-- 1 root root   301141 May  9 13:24 california_housing_test.csv
-rw-r--r-- 1 root root  1706430 May  9 13:24 california_housing_train.csv
-rw-r--r-- 1 root root 18289443 May  9 13:24 mnist_test.csv
-rw-r--r-- 1 root root 36523880 May  9 13:24 mnist_train_small.csv
-rw-r--r-- 1 root root      930 Jan  1 2000 README.md*
```

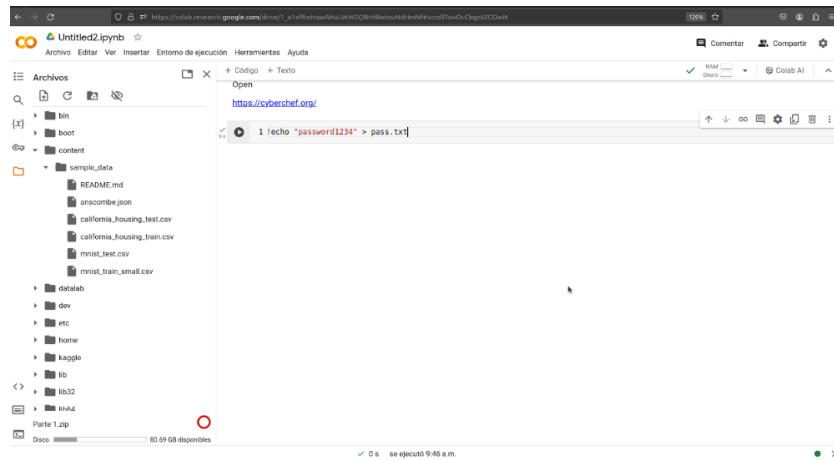
Step 4. Use **sha256sum** to get the hash of the **california_housing_test.csv** file.

```
[1] 1 pwd
{x} ↵ '/content/sample_data'
[1] 1 sha256sum sample_data/california_housing_test.csv
↳ be3f531aac5aca44fe1c04fd888b4aa68a9a69f69c05d0eb2e0473dfe702dd2  sample_data/california_housing_test.csv
```

Step 5. Use **md5sum** to get the hash of the **california_housing_test.csv** file.

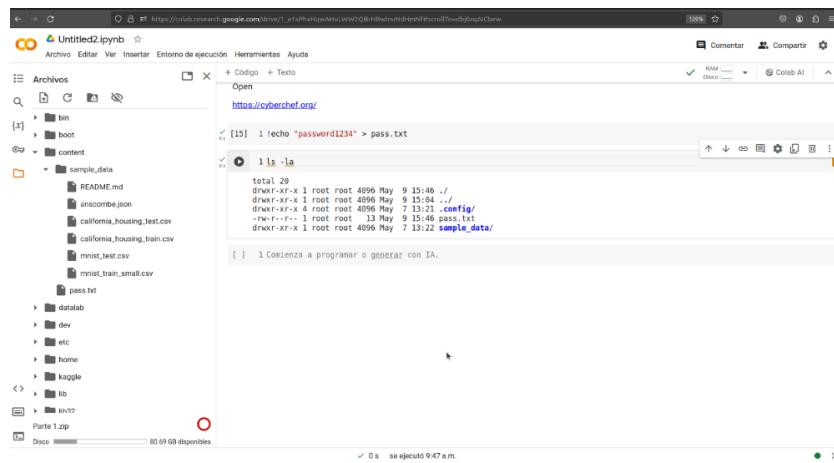
```
[1] 1 !md5sum sample_data/california_housing_test.csv
{x} ↵ d9a4b24d17770fad3209a18a4a5a3750  sample_data/california_housing_test.csv
```

Step 6. Use the command `echo` to create a text file `pass.txt` containing "password1234".



A screenshot of a Jupyter Notebook interface. The left sidebar shows a tree view of files and folders, including 'Parte 1.zip' and several CSV files under 'sample_data'. The main area has two code cells. The top cell contains the command `1 echo "password1234" > pass.txt`. The bottom cell shows the output of the command, which is the creation of a file named 'pass.txt' with a size of 0 bytes. The status bar at the bottom indicates the command was executed at 9:46 a.m.

Step 7. Verify that it was created.



A screenshot of a Jupyter Notebook interface. The left sidebar shows a tree view of files and folders, including 'Parte 1.zip' and several CSV files under 'sample_data'. The main area has two code cells. The top cell contains the command `[15] 1 echo "password1234" > pass.txt`. The bottom cell shows the output of the command, which is the creation of a file named 'pass.txt' with a size of 0 bytes. The status bar at the bottom indicates the command was executed at 9:47 a.m.

Step 8. Use the command `cat` to view the content of the created file.

```
[15] 1 echo "password1234" > pass.txt
[18] 1 ls -la
total 20
drwxr-xr-x 1 root root 4096 May  9 13:46 .
drwxr-xr-x 1 root root 4096 May  9 13:46 ..
drwxr-xr-x 4 root root 4096 May  9 13:21 .config/
-rw-r--r-- 1 root root   13 May  9 13:46 pass.txt
-rw-r--r-- 1 root root   13 May  9 13:46 renamed-pass.txt
drwxr-xr-x 1 root root 4096 May  9 13:24 sample_data/
1 cat pass.txt
password1234
```

Step 9. Get the sha256 from the pass.txt file.

```
[15] 1 echo "password1234" > pass.txt
[18] 1 ls -la
total 20
drwxr-xr-x 1 root root 4096 May  9 13:46 .
drwxr-xr-x 1 root root 4096 May  9 13:46 ..
drwxr-xr-x 4 root root 4096 May  7 13:21 .config/
-rw-r--r-- 1 root root   13 May  9 13:46 pass.txt
-rw-r--r-- 1 root root   13 May  9 13:46 renamed-pass.txt
drwxr-xr-x 1 root root 4096 May  9 13:24 sample_data/
1 cat pass.txt
password1234
1 sha256sum pass.txt
427bce94ca938c94b4a8cc3cacfc7b8bf294eaa9eb7bc2e777c693df1688aa1 pass.txt
```

Step 10. Copy the file pass.txt and rename it like renamed-pass.txt. Use cp to perform this activity.

```
+ Código + Texto
[5] 1 cp pass.txt renamed-pass.txt
[6] 1 ls -la
total 24
drwxr-xr-x 1 root root 4096 May 14 04:26 .
drwxr-xr-x 1 root root 4096 May 14 04:24 ..
drwxr-xr-x 4 root root 4096 May  9 13:24 .config/
-rw-r--r-- 1 root root   13 May 14 04:25 pass.txt
-rw-r--r-- 1 root root   13 May 14 04:26 renamed-pass.txt
drwxr-xr-x 1 root root 4096 May  9 13:24 sample_data/
```

Step 11. Get the sha256 from the file **renamed-pass.txt**.

The screenshot shows a Jupyter Notebook interface with the title "IAM-Lab1.ipynb". The code cell contains the command `!sha256sum renamed-pass.txt`. The output shows the hash value `427bce94ca938c94b4a6cc3cacfc7b8bf294eaa0eb7bc2e777c693df1688aa1` followed by the filename `renamed-pass.txt`.

Step 12. Please verify that the file, despite changing its name, still has the same content since the hash is the same.

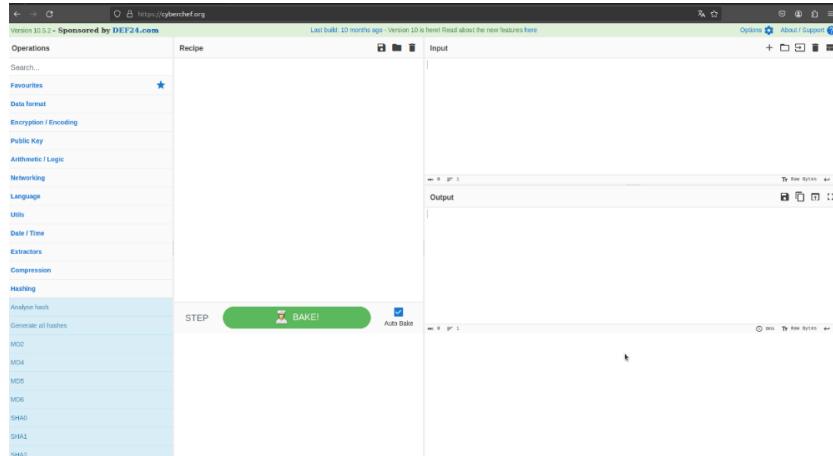
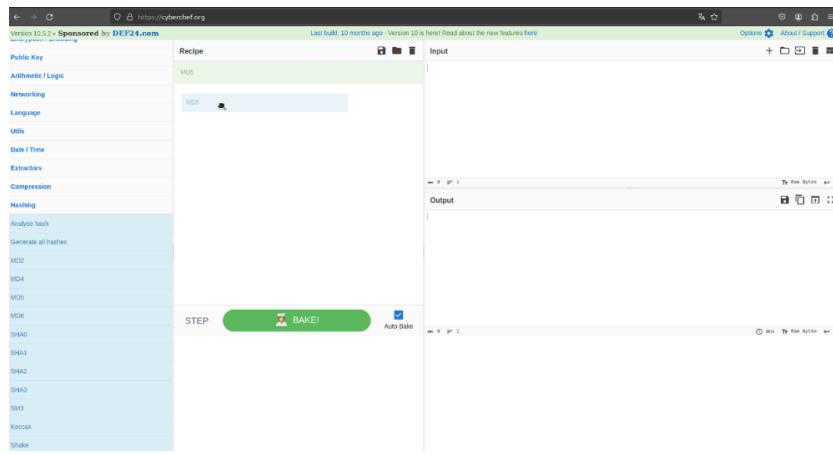
The screenshot shows a Jupyter Notebook interface with the title "IAM-Lab1.ipynb". It displays two code cells. The first cell contains `[7] !sha256sum renamed-pass.txt` with output `427bce94ca938c94b4a6cc3cacfc7b8bf294eaa0eb7bc2e777c693df1688aa1 renamed-pass.txt`. The second cell contains `[8] !sha256sum pass.txt` with output `427bce94ca938c94b4a6cc3cacfc7b8bf294eaa0eb7bc2e777c693df1688aa1 pass.txt`.

Step 13. Now, use sed to change the lowercase letter “p” to the uppercase letter “P” in the contents of the **renamed-pass.txt** file. Verify the change using cat.

The screenshot shows a Jupyter Notebook interface with the title "IAM-Lab1.ipynb". It displays three code cells. The first cell contains `[9] cat renamed-pass.txt` with output `password1234`. The second cell contains `[10] !sed -i 's/p/P/g' renamed-pass.txt`. The third cell contains `[11] cat renamed-pass.txt` with output `Password1234`, indicating the replacement was successful.

Step 14. Get the hash of the renamed-pass.txt file.

```
+ Código + Texto  
[13] 1 !sha256sum renamed-pass.txt  
32a2a9cc6a263bd1db4403643809c37b6ccdb43210c4a20737596ae04193ea19  renamed-pass.txt  
[14] 1 !sha256sum pass.txt  
427bce94ca938c94b4a6cc3cacfc7b8bf294eaa0eb7bc2e777c693df1688aa1  pass.txt
```

Task 2.**Step 15.** Open <https://cyberchef.org/>.**Step 16.** Use the Operations section on the left to drag the MD5 option to the center, in Recipe section.

Step 17. In the upper right part, place the string “pass2024” in the input area and identify the output in the lower right part.

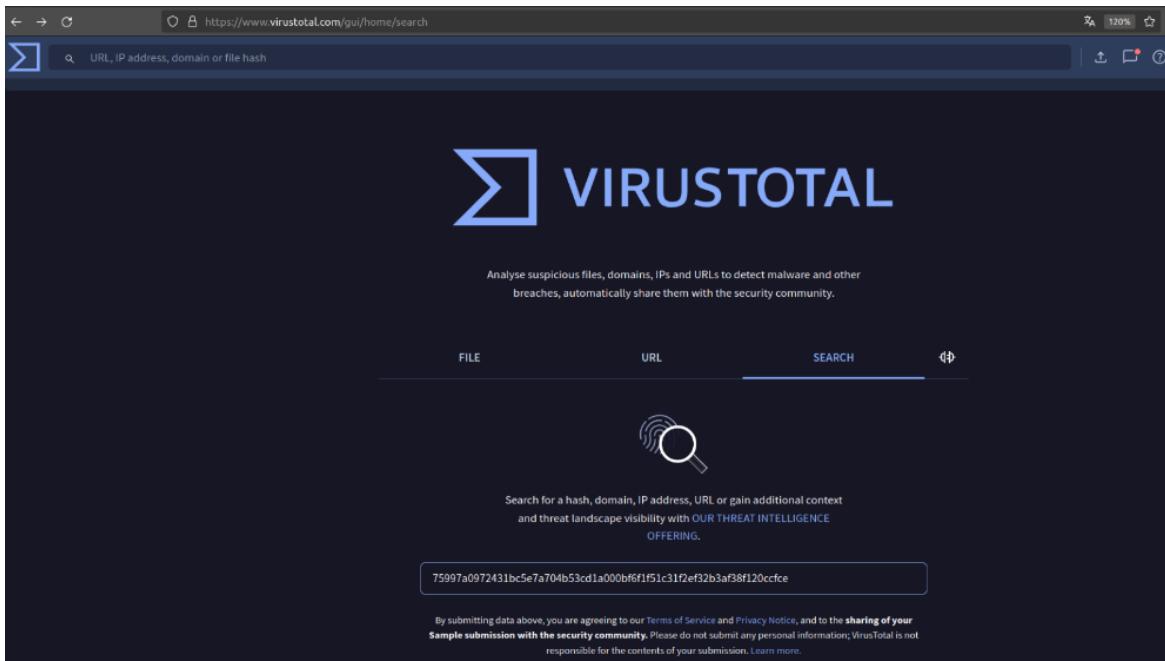
The screenshot shows the CyberChef interface with the MD5 recipe selected. In the 'Input' field, the string 'pass2024' is entered. The 'Output' field displays the resulting hash: 5e7ba5ab843cf9d5324c56a10a99d17e.

The screenshot shows the CyberChef interface with the MD5 recipe selected. In the 'Input' field, the string 'pass20241' is entered. The 'Output' field displays the resulting hash: bfb24346441e3bafac4dcdbca1a6c5.

Step 18. Identify what happens if you change/add/remove a character from the text string.

The screenshot shows the CyberChef interface with the MD5 recipe selected. In the 'Input' field, the string '3.1415926538898' is entered. The 'Output' field displays the resulting hash: 2836d67c023e875da51938a2b2b4cc0f.

Step 19. Open <https://www.virustotal.com/gui/home/upload>, and in the **Search section**, enter the value 75997A0972431BC5E7A704B53CD1A000BF6F1F51C31F2EF32B3AF38F120CCFCE and press **Enter**.



Step 20. Observe the information provided by the page.

A screenshot of the VirusTotal analysis page for the file hash 75997A0972431bc5e7a704b53cd1a000bf6f1f51c31f2ef32b3af38f120ccfce. The top navigation bar shows the URL https://www.virustotal.com/gui/file/75997a0972431bc5e7a704b53cd1a000bf6f1f51c31f2ef32b3af38f120ccfce. A message at the top states: 'We have changed our Privacy Notice and Terms of Use, effective July 18, 2024. You can view the updated [Privacy Notice](#) and [Terms of Use](#)'. The main content area includes a circular 'Community Score' meter showing 53/73, a summary box stating '53/73 security vendors and 4 sandboxes flagged this file as malicious', and a detailed file info section with fields like 'Name' (75997a0972431bc5e7a704b53cd1a000bf6f1f51c31f2ef32b3af38f120ccfce), 'Type' (EXE), 'Size' (527.50 KB), and 'Last Modification Date' (8 hours ago). Below this are tabs for 'DETECTION', 'DETAILS', 'RELATIONS', 'BEHAVIOR', and 'COMMUNITY' (with 10 items). A green bar encourages joining the community. The 'DETECTION' tab displays a table of 'Security vendors' analysis' results, showing various vendor names, threat labels, and associated threat categories like 'trojan.dadic/deepscan', 'trojan', 'dadic', 'stealer', etc. The table also includes columns for 'Suspicious' status and vendor-specific threat IDs.

Step 21. Go to the Details section and identify the information in Basic Properties.

The screenshot shows the 'Basic properties' section of a VirusTotal analysis report. It includes fields such as MD5, SHA-1, SHA-256, Vhash, Authentihash, ImpHash, Rich PT Header hash, SSDeep, TLSH, File type, Magic, TrID, DetectItEasy, and File size. The file is identified as a Win32 EXE (detectable) with various compiler and linker details.

	Value
MD5	144e9b197d28bf00018a06681636fb
SHA-1	82fb8d1799ad9d1cdcb0b1365c0aa90475
SHA-256	75997a0972431bc5e7a704b53cd1a000bf6f1f51c31f2ef32b3af38f120ccfce
Vhash	053064655d79e8a2
Authentihash	526aa404b701e30be309a03dec006d15a1d99c5edf2e329fe030ab1423
ImpHash	b173d164aa5d93a1a0964202116e39
Rich PT Header hash	17705575d6b0c575164e771a7a3
SSDeep	12238rbrJ0+9dg.freySCUTPAKRxRclEdsTmDeymOpWCMx40Xg.v./MnfgdRwesTTPnfcDMv
TLSH	TUJ984FL1B5C0C07204E2267059407885F70B700466CFC63985ATDF112C1A621F98
File type	Win32 EXE (detectable) (win32, win7, pe, generic)
Magic	PE32 executable (console) Intel 80386, for MS Windows
TrID	Win32 Executable MS Visual C++ (generic) (47.3%) Win64 Executable (generic) (15.0%) Win32 Dynamic Link Library (generic) (9.0%) Win16 NE executable (generic) (1.0%)
DetectItEasy	PE32 Compiler: EP:Microsoft Visual C/C++ (2017 v15.10) [LVE32] Compiler: Microsoft Visual C/C++ (19.30.33023) [LVC/C++] Linker: Microsoft Linker (14.30.33021..)
File size	527.50 KB (540160 bytes)

Step 22. The names associated with the consulted hash are observed in the Names section.

The screenshot shows the 'Names' section of a VirusTotal analysis report. It lists several names associated with the file, including 'unknown', '75997a0972431bc5e7a704b53cd1a000bf6f1f51c31f2ef32b3af38f120ccfce.exe', '75997a0972431bc5e7a704b53cd1a000bf6f1f51c31f2ef32b3af38f120ccfce', and 'y05cv2qcu3jkf8ultk6fsszn.exe'.

	Value
TrID	Win32 Executable MS Visual C++ (generic) (47.3%)
DetectItEasy	PE32 Compiler: EP:Microsoft Visual C/C++ (2017)
File size	527.50 KB (540160 bytes)



Laboratory 2. Password dumps

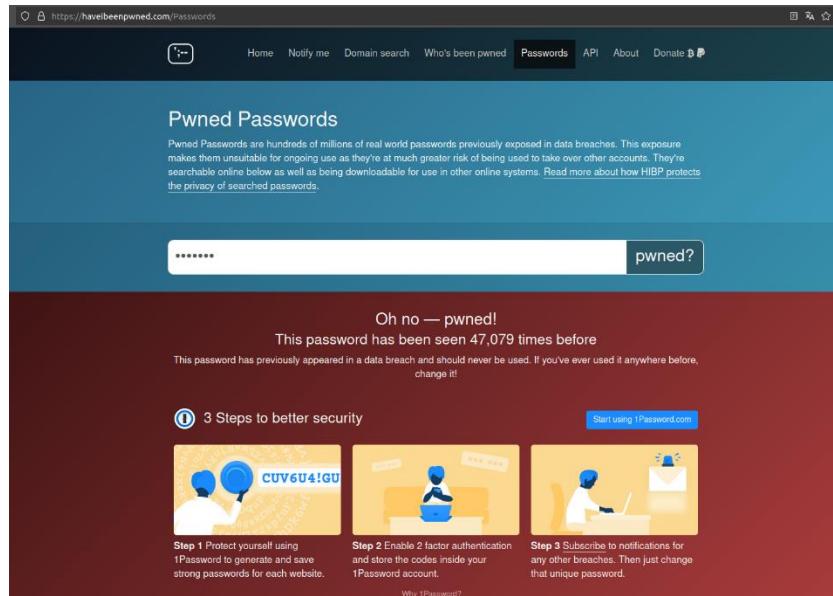
Task 1. Check your credentials

Step 1. Go to <https://haveibeenpwned.com/> and navigate across the site.

The screenshot shows the homepage of haveibeenpwned.com. At the top, there's a navigation bar with links for Home, Notify me, Domain search, Who's been pwned, Passwords, API, About, and Donate. Below the navigation is a large blue header with the text "';--have i been pwned?'". Underneath it, a sub-header says "Check if your email address is in a data breach". There are two input fields: "email address" and "pwned?". Below these fields is a small note: "Using Have I Been Pwned is subject to the terms of use". A button labeled "Generate secure, unique passwords for every account" is present, along with a link to "Learn more at 1Password.com" and a "Why 1Password?" link. The main content area displays four statistics: "773 pwned websites", "13,107,146,673 pwned accounts", "115,769 pastes", and "228,884,627 paste accounts". Below these stats are sections titled "Largest breaches" and "Recently added breaches", each listing several breached data sources with their respective counts.

Step 2. Put your email address and password in the Home and Password options and check if you have been pwned.

The screenshot shows the same homepage as before, but now with an email address entered into the "email address" field: "gadoors@hotmail.com". The "pwned?" button is highlighted. Below the input fields, a message reads "Oh no — pwned!" followed by "Pwned in 5 data breaches and found no pastes (subscribe to search sensitive breaches)". A "Start using 1Password.com" button is visible. The main content area now features three steps for better security: "Step 1 Protect yourself using 1Password to generate and save strong passwords for each website.", "Step 2 Enable 2 factor authentication and store the codes inside your 1Password account.", and "Step 3 Subscribe to notifications for any other breaches. Then just change that unique password.". Each step has an associated illustration.



Step 3. Check the information on breaches involved.

The screenshot shows a list of breached services with their logos and brief descriptions:

- Adobe:** In October 2013, 153 million Adobe accounts were breached with each containing an internal ID, username, email, encrypted password and a password hint in plain text. The password cryptography was poorly done and many were quickly resolved back to plain text. The unencrypted hints also disclosed much about the passwords adding further to the risk that hundreds of millions of Adobe customers already faced.
Compromised data: Email addresses, Password hints, Passwords, Usernames
- Data Enrichment Exposure From PDL Customer:** In October 2019, security researchers Vinny Troia and Bob Diachenko identified an unprotected Elasticsearch server holding 1.2 billion records of personal data. The exposed data included an index indicating it was sourced from data enrichment company People Data Labs (PDL) and contained 622 million unique email addresses. The server was not owned by PDL and it's believed a customer failed to properly secure the database. Exposed information included email addresses, phone numbers, social media profiles and job history data.
Compromised data: Email addresses, Employers, Geographic locations, Job titles, Names, Phone numbers, Social media profiles
- Mathway:** In January 2020, the math solving website Mathway suffered a data breach that exposed over 25M records. The data was subsequently sold on a dark web marketplace and included names, Google and Facebook IDs, email addresses and salted password hashes.
Compromised data: Device information, Email addresses, Names, Passwords, Social media profiles
- TARINGA:** In September 2017, news broke that Taringa had suffered a data breach exposing 28 million records. Known as "The Latin American Reddit", Taringa's breach disclosure notice indicated the incident dated back to August of that year. The exposed data included usernames, email addresses and weak MD5 hashes of passwords.
Compromised data: Email addresses, Passwords, Usernames
- Twitter (200M):** In early 2023, over 200M records scraped from Twitter appeared on a popular hacking forum. The data was obtained sometime in 2021 by abusing an API that enabled email addresses to be resolved to Twitter profiles. The subsequent results were then composed into a corpus of data containing email addresses alongside public Twitter profile information including names, usernames and follower counts.
Compromised data: Email addresses, Names, Social media profiles, Usernames

But Is it a safe search? Read this article <https://www.troyhunt.com/ive-just-launched-pwned-passwords-version-2/#cloudflareprivacyandkanonymity> or Go to API → Overview → Pwned Passwords → Overview and read about security search of passwords in the website of HIBP.

Task 2. Find passwords from password dumps

Scenery

You are a hacker and have found a repository of password dumps from the company Seven Kingdoms Corp. Try to obtain the passwords from that repository.

(For this exercise, we assume that the dumped passwords are in the HIBP repository).

What do you think about doing the activity?



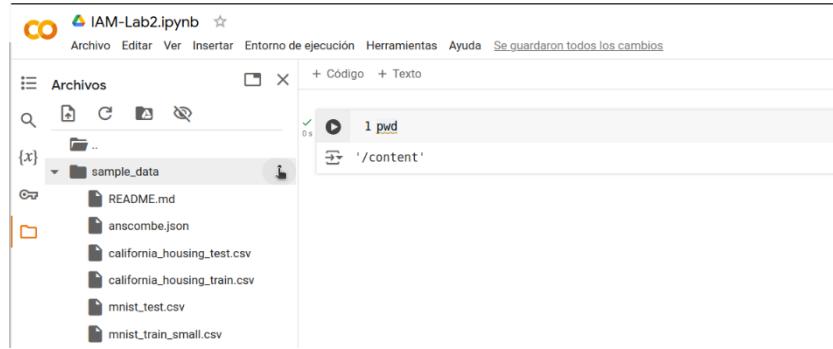
Task 2.1 Generate Passwords

Step 4. You can create a dictionary by putting all the passwords you can think of in a Txt file, or you can generate a dictionary with Python. In this exercise, we will create a dictionary using a set of keywords.

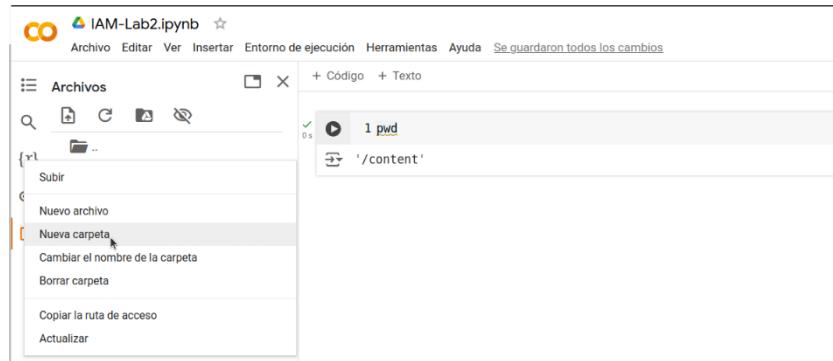
On the far right side, click on the folder icon.



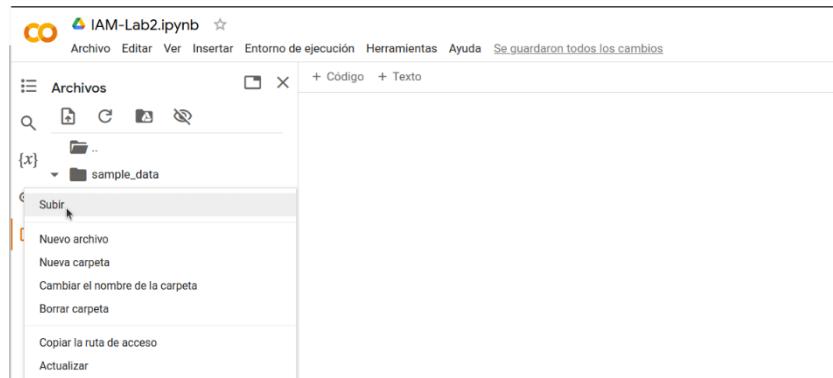
Step 5. Then click on the three dots in the **sample_data** folder.



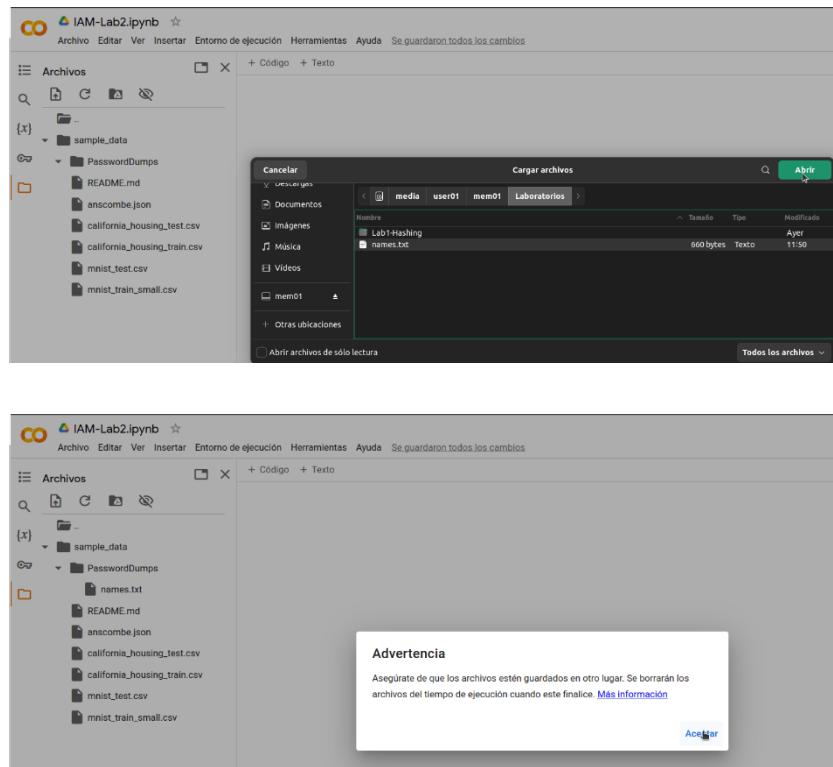
Step 6. Create a new folder called **PasswordDumps**.



Step 7. Upload the text file with the names of the possible passwords to the created folder.



Step 8. Click **OK**, and then in the Warning box, click **OK**.



Step 9. Enter the PasswordDumps folder and verify that the Txt file is found.

```
[4] 1 cd sample_data/PasswordDumps/
[5] 1 pwd
[6] 1 ls -la
```

The image shows a Jupyter Notebook terminal window. The left pane displays a file tree with a 'sample_data' directory containing a 'PasswordDumps' folder. The right pane shows a terminal session with the following commands and output:

- [4] 1 cd sample_data/PasswordDumps/
- [5] 1 pwd
- [6] 1 ls -la

The terminal output shows the contents of the '>PasswordDumps' folder, including 'names.txt'.

Step 10. Start creating your password list. First, create two lists: 1 with the passwords in the **names.txt** file (**original_list**) and another with the same passwords in lowercase (**lower_list**). Verify that the lists are correct.



```

 0s [32] 1 original_list=[]
{xs} 0s [33] 1 lower_list = []
0s [20] 1 path='/content/sample_data/PasswordDumps/names.txt'
[35] 1 with open(path,'r+') as f:
2   for line in f:
3     lines = line.replace('\n','')
4     original_list.append(lines)
5     lower_list.append(lines.lower())
0s [ ] 1 print(original_list)
2 print(lower_list)

[>] ['EddardStark', 'NedStark', 'Stark', 'RobertBaratheon', 'JaimeLannister', 'Lannister', 'CatelynStark', 'CerseiLannister',
['eddardstark', 'nedstark', 'stark', 'robertbaratheon', 'jaimeLannister', 'lannister', 'catelynStark', 'cerseiLannister'],

```

Step 11. Join the two lists into one (**simple_list**) for greater ease of use of the data.



```

 0s [163] 1 simple_list=original_list+lower_list
{xs} 0s [166] 1 print('Total simple list: ', len(simple_list), '|', 'Original: ', len(original_list), '|', 'Lower: ', len(lower_list))
0s [ ] 166 Total simple list: 112 | Original: 56 Lower: 56
[167] 1 print(simple_list)
[>] ['EddardStark', 'NedStark', 'Stark', 'RobertBaratheon', 'JaimeLannister', 'Lannister', 'CatelynStark', 'CerseiLannister',

```

Step 12. Many passwords have this structure: 'password2023', 'qwerty2022', etc. Create a list (**list_range**) with the years from 2015 to 2023 (This is just an example; you can later try other combinations of numbers such as 1234, 9876, 1111, etc.).



```

 0s [168] 1 list_rango=list(range(2015,2025,1))
{xs} 0s [169] 1 print(list_rango)
0s [ ] 169 [2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023, 2024]

```

Step 13. Create a list (**years_list**) where the years are added to the end of the **simple_list**.



The screenshot shows a Jupyter Notebook interface with the title "IAM-Lab2.ipynb". The code cell [174] contains the assignment `years_list = []`. The code cell [178] contains a for loop that iterates over `simple_list`, concatenates each item with a year (e.g., "Stark" + "2015"), and appends the result to `years_list`. The code cell [179] prints the contents of `years_list`, which is a list of strings: ['EddardStark2015', 'EddardStark2016', 'EddardStark2017', 'EddardStark2018', 'EddardStark2019', 'EddardStark2020'].

```
+ Código + Texto
[174] 1 years_list = []
[178] 1 for element in simple_list:
         2     for item in list_rango:
         3         np = element + str(item)
         4         years_list.append(np)
[179] 1 print(years_list)
['EddardStark2015', 'EddardStark2016', 'EddardStark2017', 'EddardStark2018', 'EddardStark2019', 'EddardStark2020']
```

Step 14. Create a new list (**total_list**) that includes **year_list** (list of uppercase and lowercase passwords + years) and **simple_list** (list of uppercase + lowercase passwords).



The screenshot shows a Jupyter Notebook interface with the title "IAM-Lab2.ipynb". The code cell [180] concatenates `simple_list` and `years_list` into `total_list`. The code cell [183] prints the contents of `total_list`, which is a list of strings: ['EddardStark', 'NedStark', 'Stark', 'RobertBaratheon', 'JaimeLannister', 'Lannister', 'CatelynStark', 'EddardStark2015', 'EddardStark2016', 'EddardStark2017', 'EddardStark2018', 'EddardStark2019', 'EddardStark2020']. The code cell [184] calculates the length of `total_list`, which is 2352.

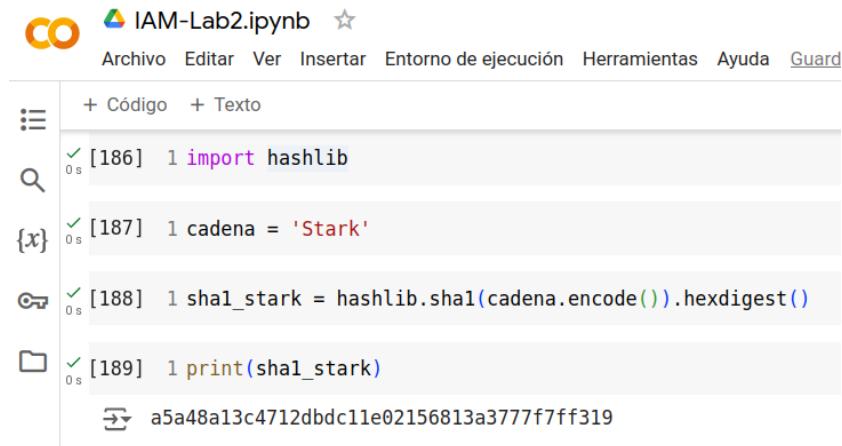
```
+ Código + Texto
[180] 1 total_list=simple_list+years_list
[183] 1 print(total_list)
['EddardStark', 'NedStark', 'Stark', 'RobertBaratheon', 'JaimeLannister', 'Lannister', 'CatelynStark', 'EddardStark2015', 'EddardStark2016', 'EddardStark2017', 'EddardStark2018', 'EddardStark2019', 'EddardStark2020']
[184] 1 len(total_list)
2352
```

The **total_list** is the passwords generated by the company Seven Kingdoms Corp.

Task 2.2 Calculate Hash

To carry out this sub-task, you can visit <https://www.geeksforgeeks.org/sha-in-python/>

Step 15. Try generating the **SHA1** for a string.



```
+ Código + Texto
[186] 1 import hashlib
[187] 1 cadena = 'Stark'
[188] 1 shal_stark = hashlib.sha1(cadena.encode()).hexdigest()
[189] 1 print(shal_stark)
a5a48a13c4712dbdc11e02156813a3777f7ff319
```

Step 16. Replicate the activity for the entire list (**total_list**).



```
+ Código + Texto
[190] 1 hash_list = []
[191] 1 for i in total_list:
[191] 2   sha_string = hashlib.sha1(i.encode()).hexdigest()
[191] 3   hash_list.append(sha_string)
[192] 1 print(hash_list)
['0712cb28205044922dfada70c48f09a936503594', '1e0faaae26c8c2b05a5dfa5f1b05dec8bc8ab79d',
```

Step 17. Create a Data Frame to associate the hashes with the passwords.

Reference: <https://www.geeksforgeeks.org/different-ways-to-create-pandas-dataframe/>

```
[195]: 1 import pandas as pd
        2
        3 data = {'Password': total_list, 'SHA1':hash_list}
        4 df = pd.DataFrame(data)
        5
        6 print(df)

          Password           SHA1
0     EddardStark  0712cb28205044922dfada70c48f09a936503594
1       NedStark  1e0faaae26c8c2b05a5dfa5f1b05dec8bc8ab79d
2        Stark  a5a48a13c4712dbdc11e02156813a3777f7ff319
3  RobertBaratheon  20256560ba901d3eefe76156c1c60090f3edf9a3
4   JaimeLannister  202623e0868be26d39d5f6e82a182cd5dad0495
...
2347    qwerty2020  2c1e9a77c005e132a0d055a2fad1bac407c20a38
2348    qwerty2021  e919564d6d140ab8340ac004f8e8848803c4685a
2349    qwerty2022  32f3b58fb0d372b7c750f0d14f0c6f74b8043404
2350    qwerty2023  3d4bbabd52a749d7decef874055b802d68549fa0
2351    qwerty2024  d0219b87cc88f83402a9a028cbe234e2c377a591

[2352 rows x 2 columns]
```

Task 2.3 Search SHA1 in Password Dump

Step 18. Import the following libraries:

```
[197]: 1 import requests
        2 import time
        3 import pandas
```

Step 19. Declare the HIBP URL variable where to query SHA1 hashes.

For more information, see:

<https://haveibeenpwned.com/API/v3#PwnedPasswords>

```
[201]: 1 url_part1 = 'https://api.pwnedpasswords.com/range/'
```

Step 20. Start writing Python code to search for SHA1s in the HIBP Password Dumps database.

TIP: Use `time.sleep(2)` to avoid crashes from automated queries.

Where are the hashes and passwords for each hash in the data frame?



```
1 for item in df['SHA1']: # for every hash (item) in column SHA1 of df
2 time.sleep(2)
```

Step 21. Declare two variables to get the first five characters of the hash and the last five characters.

Remember that to search in HIBP, you must provide the first five characters and compare the last ones with the results obtained.



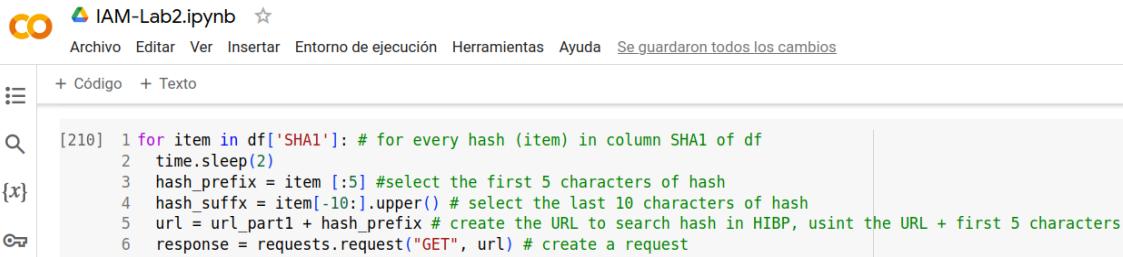
```
[210] 1 for item in df['SHA1']: # for every hash (item) in column SHA1 of df
2 time.sleep(2)
3 hash_prefix = item[:5] #select the first 5 characters of hash
4 hash_suffix = item[-10:].upper() # select the last 10 characters of hash
```

Step 22. Create the full URL to look up the hash.



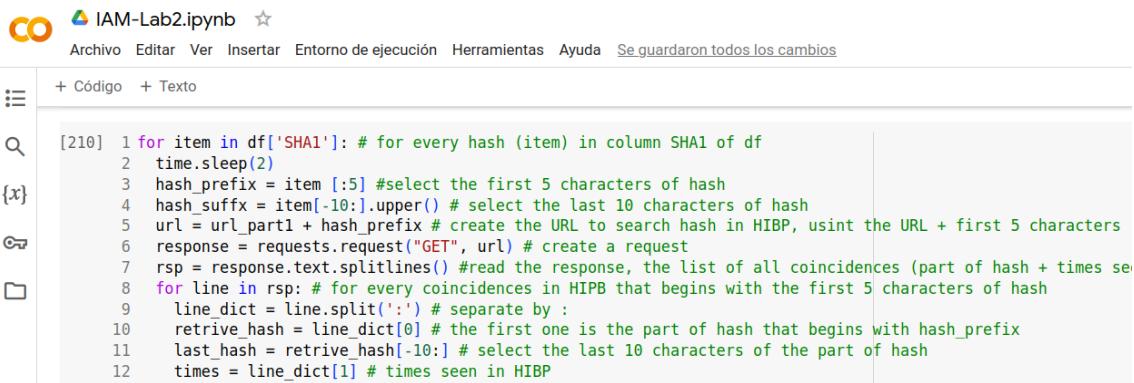
```
1 for item in df['SHA1']: # for every hash (item) in column SHA1 of df
2 time.sleep(2)
3 hash_prefix = item[:5] #select the first 5 characters of hash
4 hash_suffix = item[-10:].upper() # select the last 10 characters of hash
5 url = url_part1 + hash_prefix # create the URL to search hash in HIBP, usint the URL + first 5 characters
```

Step 23. Request the HIBP page and read the response provided.



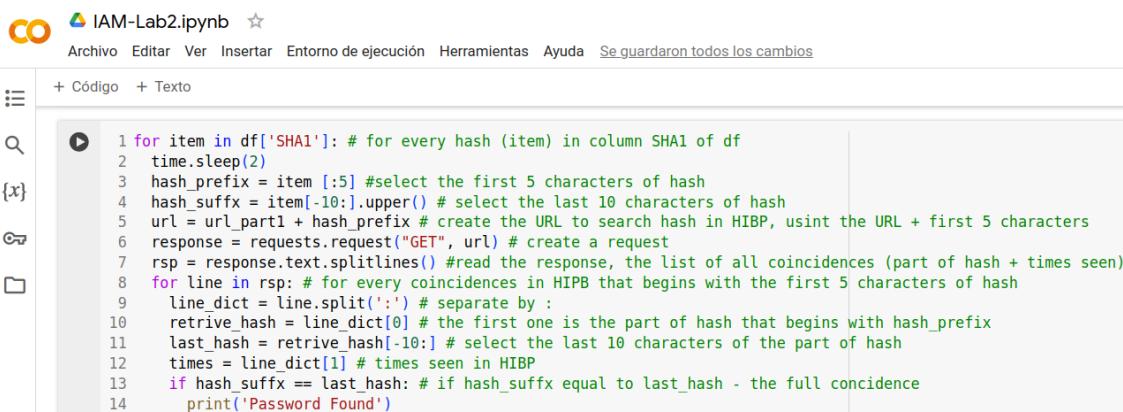
```
[210]: 1 for item in df['SHA1']: # for every hash (item) in column SHA1 of df
2     time.sleep(2)
3     hash_prefix = item [:5] #select the first 5 characters of hash
4     hash_suffix = item[-10:].upper() # select the last 10 characters of hash
5     url = url_part1 + hash_prefix # create the URL to search hash in HIBP, usint the URL + first 5 characters
6     response = requests.request("GET", url) # create a request
```

Step 24. Read the information provided by HIBP and prepare the variables that will help you find the matches.



```
[210]: 1 for item in df['SHA1']: # for every hash (item) in column SHA1 of df
2     time.sleep(2)
3     hash_prefix = item [:5] #select the first 5 characters of hash
4     hash_suffix = item[-10:].upper() # select the last 10 characters of hash
5     url = url_part1 + hash_prefix # create the URL to search hash in HIBP, usint the URL + first 5 characters
6     response = requests.request("GET", url) # create a request
7     rsp = response.text.splitlines() #read the response, the list of all coincidences (part of hash + times seen)
8     for line in rsp: # for every coincidences in HIBP that begins with the first 5 characters of hash
9         line_dict = line.split(':') # separate by :
10        retrieve_hash = line_dict[0] # the first one is the part of hash that begins with hash_prefix
11        last_hash = retrieve_hash[-10:] # select the last 10 characters of the part of hash
12        times = line_dict[1] # times seen in HIBP
```

Step 25. Look for hash matches in the results obtained from HIBP.



```
[210]: 1 for item in df['SHA1']: # for every hash (item) in column SHA1 of df
2     time.sleep(2)
3     hash_prefix = item [:5] #select the first 5 characters of hash
4     hash_suffix = item[-10:].upper() # select the last 10 characters of hash
5     url = url_part1 + hash_prefix # create the URL to search hash in HIBP, usint the URL + first 5 characters
6     response = requests.request("GET", url) # create a request
7     rsp = response.text.splitlines() #read the response, the list of all coincidences (part of hash + times seen)
8     for line in rsp: # for every coincidences in HIBP that begins with the first 5 characters of hash
9         line_dict = line.split(':') # separate by :
10        retrieve_hash = line_dict[0] # the first one is the part of hash that begins with hash_prefix
11        last_hash = retrieve_hash[-10:] # select the last 10 characters of the part of hash
12        times = line_dict[1] # times seen in HIBP
13        if hash_suffix == last_hash: # if hash_SUFFIX equal to last_hash - the full coincidence
14            print('Password Found')
```

Task 2.4 Compare the results.

Step 26. Create a DataFrame (df_results) to store the password data found in the HIBP database.

```
[200]: 1 df_results = pd.DataFrame(columns = ['Password', 'SHA1(Orig)', 'SHA1(partHIBP)', 'NoTimes'])
2 print(df_results)
```

{x} Empty DataFrame
Columns: [Password, SHA1(Orig), SHA1(partHIBP), NoTimes]
Index: []

Step 27. Identify which password each hash corresponds to and save it in the results data frame (df_results)

```
1 for item in df['SHA1']: # for every hash (item) in column SHA1 of df
2     time.sleep(2)
3     hash_prefix = item[:5] #select the first 5 characters of hash
4     hash_sufix = item[10:].upper() # select the last 10 characters of hash
5     url = url_part1 + hash_prefix # create the URL to search hash in HIBP, usint the URL + first 5 characters
6     response = requests.request("GET", url) # create a request
7     rsp = response.text.splitlines() # read the response, the list of all coincidences (part of hash + times seen)
8     for line in rsp: # for every coincidences in HIBP that begins with the first 5 characters of hash
9         line_dict = line.split(':') # separate by :
10        retrieve_hash = line_dict[0] # the first one is the part of hash that begins with hash_prefix
11        last_hash = retrieve_hash[-10:] # select the last 10 characters of the part of hash
12        times = line_dict[1] # times seen in HIBP
13        if hash_sufix == last_hash: # if hash_sufix equal to last_hash - the full coincidence
14            print('Password Found')
15            pswd = df.loc[df['SHA1'] == item, "Password"].values.item() # Bring password value of the hash
16            array = [pswd, item, line, times] # create a list to save the results
17            df_results.loc[len(df_results)] = array # save the list in data frame of results
18            print(pswd,item,times)
19        else:
20            pass
21 print(df_results) # Print the results of all coincidences
```

Step 28. Verify passwords found in HIBP.

```
EddardStark 0712cb28205044922dfada70c48f09a936503594 15
Password Found
...
NedStark 1e0faaae26c8c2b05a5dfa5f1b05dec8bc8ab79d 12
Password Found
Stark a5a48a13c4712dbdc11e02156813a3777f7ff319 148
Password Found
JaimeLannister 202623e0868be26d39d5f6e82a182cdd5dad0495 5
Password Found
Lannister c744756c6f32cb17c250a74ce55211a95414318d 151
Password Found
DaenerysTargaryen 152e17a964f9e54e91aeae9adb6a22e2addb6baa 4
Password Found
Targaryen d84176542068c5982f78fbfdb25df3ba8273046 156
Password Found
JorahMormont 5be6691b9e909acc5b3b7d58324cc96075543ec 1
Password Found
Mormont 958af94bfa435e781d80515d6f452ca67a7f4847 17
Password Found
JonSnow a24836c226a625a4f2f7bbfeaeab283e4f227095e 109
Password Found
RobbStark 57872514949aabaa4857459109ee50e42659fc96 15
Password Found
SansaStark 55789218e7d64c8eff4991023e56e63ed6cf7d61 9
Password Found
AryaStark b0589d95b051d7221d2b029abf6ca7a2110d7dd9 16
Password Found
BranStark 2eb212665f4f285d9488cd56a6f375aec3844d1b 1
Password Found
SandorClegane 884e85f83ecc9c0415973bb417f7dd7d82be4ef2 31
Password Found
TheHound 7fc3ad58b8438f855b0784b0150f0b1818be79f7 9
Password Found
TyrionLannister 0ee52eec5f8fa2747f4a6f6acff120b65d9372a9 14
```



Laboratory 3. Cryptography

Task 1. Symmetric Encryption

Step 1. Create a txt file (`clear_message.txt`) with the text “**Este es un mensaje en texto claro.**”

```
[ ] 1 !echo "Este es un mensaje en texto claro" > clear_message.txt ←  
  
[ ] 1 ls -la  
  
→ total 28  
drwxr-xr-x 1 root root 4096 May 23 01:08 ./  
drwxr-xr-x 1 root root 4096 May 23 00:37 ../  
-rw-r--r-- 1 root root 34 May 23 01:02 clear_message.txt ←
```

Step 2. Use `cat`, to verify its content.

```
▶ 1 !cat clear_message.txt  
  
→ Este es un mensaje en texto claro
```

Step 3. Use `openssl` to create an `encrypted_message` file from the `clear_message.txt` file with:

```
!openssl aes-256-cbc -e -in clear_message.txt -out encrypted_message
```

When required, enter an encryption key. As shown in the following image.

```
[ ] 1 !openssl aes-256-cbc -e -in clear_message.txt -out encrypted_message  
  
→ enter AES-256-CBC encryption password: ←  
Verifying - enter AES-256-CBC encryption password:  
*** WARNING : deprecated key derivation used.  
Using -iter or -pbkdf2 would be better.
```

Step 4. Verify the contents of the `encrypted_message` file.

```
▶ 1 !cat encrypted_message  
  
→ Salted__$msBO!Rt~|vWizcqryD~5
```

Step 5. Decrypt the `encrypted_message` message to the `encrypted_message.txt` file, using:

```
!openssl aes-256-cbc -d -in encrypted_message -out decrypt_message.txt
```

Enter the key from the previous step.

```
[ ] 1 !openssl aes-256-cbc -d -in encrypted_message -out decrypt_message.txt  
→ enter AES-256-CBC decryption password: ←  
*** WARNING : deprecated key derivation used.  
Using -iter or -pbkdf2 would be better.
```

Step 6. Verify that the message was decrypted using `cat`.

```
→ 1 !cat decrypt_message.txt  
→ Este es un mensaje en texto claro
```

Step 7. To strengthen the encryption use `-iter` y `-pbkdf2`. Create a new encrypted file `2_encrypted_message` from `clear_message.txt`.

```
!openssl aes-256-cbc -pbkdf2 -iter 1000 -e -in clear_message.txt -  
out 2_encrypted_message
```

Step 8. Enter the password to encrypt the file `2_encrypted_message`.

```
[ ] 1 !openssl aes-256-cbc -pbkdf2 -iter 1000 -e -in clear_message.txt -out 2_encrypted_message  
→ enter AES-256-CBC encryption password: ←  
Verifying - enter AES-256-CBC encryption password: ←
```

Step 9. Verify that the `2_encrypted_message` file was created.

```
[ ] 1 ls -la  
→ total 36  
drwxr-xr-x 1 root root 4096 May 23 01:27 ./  
drwxr-xr-x 1 root root 4096 May 23 00:37 ../  
-rw-r--r-- 1 root root 34 May 23 01:27 2_encrypted_message  
-rw-r--r-- 1 root root 64 May 23 01:25 clear_message.txt  
drwxr-xr-x 4 root root 4096 May 21 13:23 .config/  
-rw-r--r-- 1 root root 34 May 23 01:08 decrypt_message.txt  
-rw-r--r-- 1 root root 64 May 23 01:06 encrypted_message  
drwxr-xr-x 1 root root 4096 May 21 13:23 sample_data/
```

Step 10. Use `cat` to read the contents of the `2_encrypted_message` file.

```
▶ 1 !cat 2_encrypted_message  
→ Salted_&^73&p{xbb: N:t;0;48xGú3=
```

Step 11. Decrypt the `2_encrypted_message` file in the `decrypt_message` file with the key used for encryption.

```
[ ] 1 !openssl aes-256-cbc -pbkdf2 -iter 1000 -d -in 2_encrypted_message -out 2_decrypt_message  
→ enter AES-256-CBC decryption password: ↵
```

Step 12. Using `cat` verifies that the file was decrypted.

```
[ ] 1 !cat 2_decrypt_message  
→ Este es un mensaje en texto claro
```

Task 2. Asymmetric Encryption

Step 13. Use `openssl` to generate a private key `private-key.pem` of size 2048 bits.

```
openssl genrsa -out private-key.pem 2048
```

Step 14. Verify that the private key was created.

```
[ ] 1 ls -la  
→ total 44  
drwxr-xr-x 1 root root 4096 May 23 02:41 ./  
drwxr-xr-x 1 root root 4096 May 23 00:37 ../  
-rw-r--r-- 1 root root 34 May 23 01:27 2_decrypt_message  
-rw-r--r-- 1 root root 64 May 23 01:25 2_encrypted_message  
-rw-r--r-- 1 root root 34 May 23 01:02 clear_message.txt  
drwxr-xr-x 4 root root 4096 May 21 13:23 .config/  
-rw-r--r-- 1 root root 34 May 23 01:08 decrypt_message.txt  
-rw-r--r-- 1 root root 64 May 23 01:06 encrypted_message  
-rw----- 1 root root 1704 May 23 02:40 private-key.pem →
```

Step 15. Get the public key **public-key.pem** from the private key **private-key.pem**.

```
[ ] 1 !openssl rsa -in private-key.pem -pubout -out public-key.pem
```

```
→ writing RSA key
```

Step 16. Reads the contents of the public key **public-key.pem**.

```
[ ] 1 !cat public-key.pem
```

```
→ -----BEGIN PUBLIC KEY-----  
MIIBIjANBhkqkiG9w0BAQEFAOCAQ8AMIIIBCgKCAQEArTDgPRV99y3VJY4J3+x4  
44b3bdU2cAEjAactge25hmX482tPMWJVNDcOGm79hbyjQrjiKIcAIfhQLi+hXP2U  
CbvmNM7kjVyPSujvaJTS1mgvb98pzOzRVzHYZadt3NZJRI7MUSPXR0+ouShC+3RI  
+MXAEf0S3XJTsnNb2aNZuQ+kUH60q04vQmvgQ1Bijqlt55UdJKcDP+VFT/ive06  
i0bPbDpNNk/z59TPCCZ+sEERVdldGXWy/JfzTPDXUgo8cY3Q8xb1QaM7DpoECGGY  
kyOZ4qP/Rdwye9VpTNOHf5tV/4NGw76RuHYOUuHFEKZ1F+l+MdpBBomArcY6VuV  
1QIDAQAB  
-----END PUBLIC KEY-----
```

Step 17. Reads the contents of the private key **private-key.pem**.

```
[ ] 1 !openssl rsa -in private-key.pem -text -noout  
→ Private-Key: (2048 bit, 2 primes)  
modulus:  
00:ad:30:e0:3d:15:7d:f7:2d:d5:25:8e:09:df:ec:  
78:e3:86:f7:6d:d5:36:70:01:23:01:a7:2d:81:ed:  
b9:86:65:f8:f3:6b:4f:31:62:55:34:37:0e:1a:6e:  
fd:85:bc:a3:42:b8:e2:28:87:00:21:f8:50:2e:2f:  
a1:5c:fd:94:09:bb:e6:34:ce:e4:8d:5c:8f:4a:e8:  
ef:68:94:d2:d6:68:2f:6f:df:29:cc:ec:d1:57:31:  
d8:65:a7:6d:dc:d6:49:44:8e:cc:51:23:d7:47:4f:  
a8:b9:28:42:fb:74:48:f8:c5:c0:11:fd:12:dd:72:  
53:b0:d3:5b:d9:a6:4d:66:e4:3e:91:41:fa:d2:ad:  
38:bd:09:af:81:0d:41:8a:3a:8b:b7:9e:54:74:92:  
9c:0c:ff:95:15:3f:e2:bd:ed:3a:88:e6:cf:6c:3a:  
4d:34:af:f3:e7:d4:cf:08:26:7e:b0:41:11:55:d9:  
5d:19:75:b2:fc:97:f3:4c:f0:d7:52:0a:3c:71:8d:  
d0:f3:16:e5:41:a3:3b:0e:9a:04:08:61:98:93:23:  
99:e2:a3:ff:45:dc:32:89:ef:55:a5:33:4e:1d:fe:  
6d:57:fe:0d:1b:0e:fa:46:e1:d8:39:4b:87:14:42:  
99:d4:5f:a5:f8:c7:69:04:1a:26:02:b7:18:e9:5b:  
95:d5  
publicExponent: 65537 (0x10001)  
privateExponent:  
05:e2:7d:6e:bb:3e:62:a4:8d:33:af:a3:58:97:4d:  
8b:96:f2:ec:fe:f6:7b:66:21:27:e0:63:d4:a0:87:  
67:70:80:93:2e:28:17:35:50:50:2c:0c:ba:76:34:  
a9:68:d2:f7:f9:eb:3e:aa:f3:a4:b3:d8:c4:46:51:  
b5:24:ad:4a:8a:b6:be:ea:f1:6b:77:8e:80:e2:45:  
a6:e7:e3:a5:6d:b8:80:3f:1f:74:50:99:a9:07:a4:  
62:de:86:5b:9f:71:06:e2:29:41:6f:e1:29:3c:4f:  
84:81:c0:8b:90:69:c8:56:2e:79:63:55:7d:99:1c:  
61:e0:e9:f9:37:c4:35:2f:94:3c:a7:d7:ef:20:4d:  
7f:cd:5c:f2:51:31:20:83:b0:4f:23:5f:b7:26:ae:  
45:ad:f0:2d:08:12:ef:65:cc:ad:33:f3:08:d7:dc:  
5b:fc:9b:9f:8e:96:35:bb:be:db:7f:b5:47:b2:06:
```

Step 18. Encrypts the file **clear_message.txt** (from Task 1) into **rsa_ciphertext**, using the public key created earlier **public-key.pem**.

```
[ ] 1 !openssl pkeyutl -encrypt -in clear_message.txt -out rsa_ciphertext -inkey public-key.pem -pubin
```

Step 19. Verify that the **rsa_ciphertext** file was created.

```
[ ] 1 ls -la  
total 48  
drwxr-xr-x 1 root root 4096 May 23 02:49 ./  
drwxr-xr-x 1 root root 4096 May 23 00:37 ../  
-rw-r--r-- 1 root root    34 May 23 01:27 2_decrypt_message  
-rw-r--r-- 1 root root   64 May 23 01:25 2_encrypted_message  
-rw-r--r-- 1 root root    34 May 23 01:02 clear_message.txt  
drwxr-xr-x 4 root root 4096 May 21 13:23 .config/  
-rw-r--r-- 1 root root    34 May 23 01:08 decrypt_message.txt  
-rw-r--r-- 1 root root   64 May 23 01:06 encrypted_message  
-rw----- 1 root root 1704 May 23 02:40 private-key.pem  
-rw-r--r-- 1 root root   451 May 23 02:41 public-key.pem  
-rw-r--r-- 1 root root   256 May 23 02:49 rsa_ciphertext  
drwxr-xr-x 1 root root 4096 May 21 13:23 sample_data/
```

Step 20. Read the contents of the `rsa_ciphertext` file.

Step 21. Decrypt the `rsa_ciphertext` file with the private key `private-key.pem`, and obtain the `rsa_decrypt.txt` file.

```
!openssl pkeyutl -decrypt -in rsa_ciphertext -inkey private-key.pem  
-out rsa_decrypt.txt
```

Step 22. Verify that the `rsa_decrypt.txt` file was created.

```
[ ] 1 ls -la  
total 52  
drwxr-xr-x 1 root root 4096 May 23 02:51 ./  
drwxr-xr-x 1 root root 4096 May 23 00:37 ../  
-rw-r--r-- 1 root root 34 May 23 01:27 2_decrypt_message  
-rw-r--r-- 1 root root 64 May 23 01:25 2_encrypted_message  
-rw-r--r-- 1 root root 34 May 23 01:02 clear_message.txt  
drwxr-xr-x 4 root root 4096 May 21 13:23 .config/  
-rw-r--r-- 1 root root 34 May 23 01:08 decrypt_message.txt  
-rw-r--r-- 1 root root 64 May 23 01:06 encrypted_message  
-rw----- 1 root root 1704 May 23 02:40 private-key.pem  
-rw-r--r-- 1 root root 451 May 23 02:41 public-key.pem  
-rw-r--r-- 1 root root 256 May 23 02:49 rsa_ciphertext  
-rw-r--r-- 1 root root 34 May 23 02:51 rsa_decrypt.txt ←  
drwxr-xr-x 1 root root 4096 May 21 13:23 sample_data/
```

Step 23. Use `cat` to read the contents of the `rsa_decrypt.txt` file.



```
1 !cat rsa_decrypt.txt
```



Este es un mensaje en texto claro

Task 3. Certificates

Task 3.1

Step 24. Use `openssl` to generate a certificate signing request using:

```
!openssl req -new -nodes -newkey rsa:4096 -keyout key.pem -out cert.csr
```

Step 25. Write the solicitation options.

You are about to be asked to enter information that will be incorporated into your certificate request.

What you are about to enter is what is called a Distinguished Name or a DN
There are quite a few fields but you can leave some blank
For some fields there will be a default value,

If you enter '.', the field will be left blank.

Country Name (2 letter code) [AU]:MX ↵

State or Province Name (full name) [Some-State]:CDMX ↵

Locality Name (eg, city) []:BENITO JUAREZ ↵

Organization Name (eg, company) [Internet Widgits Pty Ltd]:TRINITY CORP ↵

Organizational Unit Name (eg, section) []:HACKING TEAM ↵

Common Name (e.g. server FQDN or YOUR name) []:TRINITY ↵

Email Address []:HACK@TRINITYCORP.HACK ↵

Step 26. Verify that files “`cert.csr`” and “`key.pem`” were created.



```
1 ls -la
```



total 32

drwxr-xr-x 1 root root 4096 Jun 5 21:59 ./

drwxr-xr-x 1 root root 4096 Jun 5 20:32 ../

-rw-r--r-- 1 root root 1769 Jun 5 21:59 cert.csr

drwxr-xr-x 4 root root 4096 Jun 4 13:31 .config/

-rw----- 1 root root 3272 Jun 5 21:59 key.pem

Go to <https://certstream.calidog.io/> and visualise in real time the new certificates created and signed.

Task 3.2 Self Signed Certificate

Step 27. Use `openssl` to create a self-signed certificate (`domain.crt`) without an existing private key and CSR.

```
!openssl x509 -signkey key.pem -in cert.csr -req -days 365 -out domain.crt
```

Step 28. Verify if the certificate request self-signed signature was successful.

```
1 !openssl x509 -signkey key.pem -in cert.csr -req -days 365 -out domain.crt
2 Certificate request self-signature ok
subject=C = MX, ST = CDMX, L = BENITO JUAREZ, O = TRINITY CORP, OU = HACKING TEAM, CN = TRINITY, emailAddress = HACK@TRINITYCORP.HACK
```

Step 29. Use `openssl` to open in plain text the certificate created.

```
!openssl req -text -in cert.csr -noout -verify
```

```
1 !openssl x509 -text -noout -in domain.crt
2 Certificate:
Data:
Version: 1 (0x0)
Serial Number:
55:50:88:d6:84:33:45:97:45:90:ff:b7:4a:8a:8f:57:50:f4:37:a3
Signature Algorithm: sha256WithRSAEncryption
Issuer: C = MX, ST = CDMX, L = BENITO JUAREZ, O = TRINITY CORP, OU = HACKING TEAM, CN = TRINITY, emailAddress = HACK@TRINITYCORP.HACK
Validity
Not Before: Jun 5 22:14:29 2024 GMT
Not After : Jun 5 22:14:29 2025 GMT
Subject: C = MX, ST = CDMX, L = BENITO JUAREZ, O = TRINITY CORP, OU = HACKING TEAM, CN = TRINITY, emailAddress = HACK@TRINITYCORP.HACK
Subject Public Key Info:
Public Key Algorithm: rsaEncryption
Public-Key: (4096 bit)
Modulus:
00:b8:ff:0a:3e:d8:20:06:e5:25:3d:27:45:d4:e4:
bc:29:25:61:a9:2d:1c:d3:6c:5d:cc:e2:b5:69:8b:
86:99:72:fe:22:f7:ca:db:42:99:1a:0b:e2:4c:08:
3f:a4:df:d2:26:fd:7e:0d:2e:70:2a:6f:7f:a5:78:
e5:48:e3:33:6c:c3:ca:ee:83:54:67:b5:3a:53:29:
```