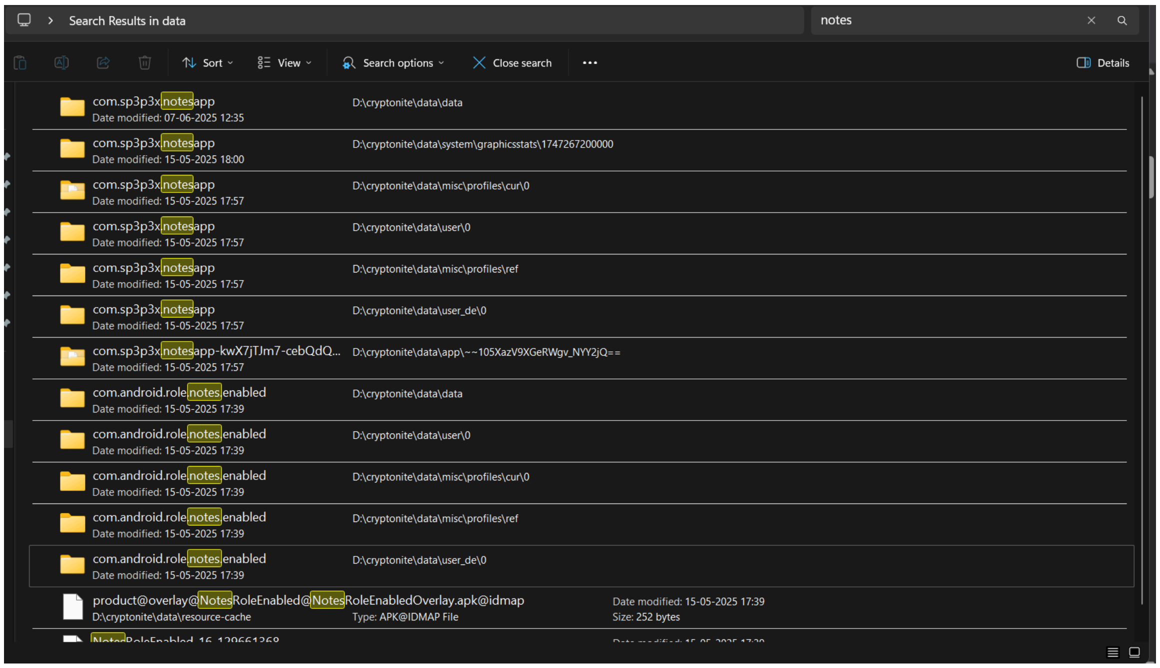
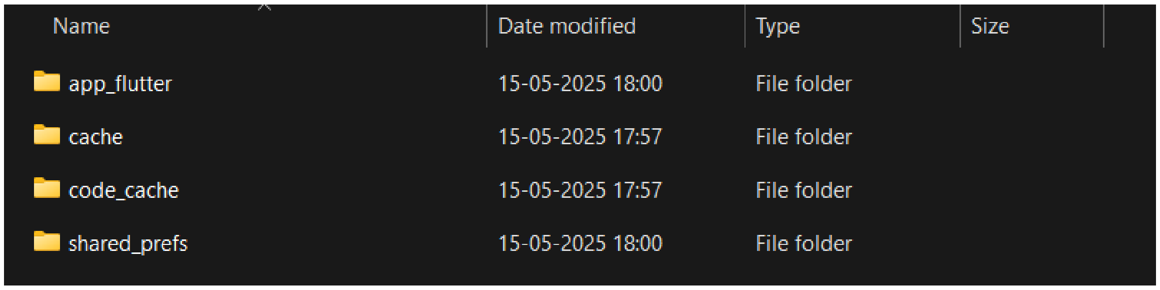
Initial Enumeration

After opening the handout, it appeared to be an Android filesystem dump. I started by searching for anything related to notes, which led me to the application directory com.sp3p3x.notesapp.



```
com.sp3p3x.notesapp                          D:\cryptonite\data\data
Date modified: 07-06-2025 12:35
com.sp3p3x.notesapp                          D:\cryptonite\data\system\graphicsstats\1747267200000
Date modified: 15-05-2025 18:00
com.sp3p3x.notesapp                          D:\cryptonite\data\misc\profiles\cur\0
Date modified: 15-05-2025 17:57
com.sp3p3x.notesapp                          D:\cryptonite\data\user\0
Date modified: 15-05-2025 17:57
com.sp3p3x.notesapp                          D:\cryptonite\data\misc\profiles\ref
Date modified: 15-05-2025 17:57
com.sp3p3x.notesapp                          D:\cryptonite\data\user_de\0
Date modified: 15-05-2025 17:57
com.sp3p3x.notesapp-kwX7jTJm7-cebQdQ...      D:\cryptonite\data\app\~~105XazV9XGeRWgv_NYY2jQ==
Date modified: 15-05-2025 17:57
com.android.role.notes.enabled               D:\cryptonite\data\data
Date modified: 15-05-2025 17:39
com.android.role.notes.enabled               D:\cryptonite\data\user\0
Date modified: 15-05-2025 17:39
com.android.role.notes.enabled               D:\cryptonite\data\misc\profiles\cur\0
Date modified: 15-05-2025 17:39
com.android.role.notes.enabled               D:\cryptonite\data\misc\profiles\ref
Date modified: 15-05-2025 17:39
com.android.role.notes.enabled               D:\cryptonite\data\user_de\0
Date modified: 15-05-2025 17:39
product@overlay@NotesRoleEnabled@NotesRoleEnabledOverlay.apk@idmap    Date modified: 15-05-2025 17:39
D:\cryptonite\data\resource-cache            Type: APK@IDMAP File     Size: 252 bytes
```



Name	Date modified	Type	Size
app_flutter	15-05-2025 18:00	File folder	
cache	15-05-2025 17:57	File folder	
code_cache	15-05-2025 17:57	File folder	
shared_prefs	15-05-2025 18:00	File folder	

Notes Application Analysis

Inside the application directory, a shared preferences XML file was found containing multiple key and value pairs. These appeared to reference saved notes along with their corresponding encryption keys.

```xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string
name="flutter.15052025175732777833">&quot;1OmIyq5YT50YlWB0&quot;</string>
    <string
name="flutter.15052025175747121993">&quot;oIdeaSz9iySlAmKJ&quot;</string>
    <string
name="flutter.15052025175936114230">&quot;YKnQqnrzfTIM9HLu&quot;</string>
    <string
name="flutter.15052025180002742685">&quot;RhjZrO2JGKQLamST&quot;</string>
    <string
name="flutter.15052025175724299736">&quot;SkZFksurgEq3Tdhe&quot;</string>
    <string
name="flutter.15052025175950593733">&quot;M52JUgdj9r6kkVg4&quot;</string>
    <string
name="flutter.15052025175944264611">&quot;lunMORQQjKhX9u5H&quot;</string>
</map>
```

Since, flutter uses the flet framework, i searched for `flet` and got a folder in `data\user\0\com.sp3p3x.notesapp\files\flet\app`, this contained `main.py`

```python
import flet as ft
import datetime, random, os, string
def key_scheduling(key):
    sched = [i for i in range(0, 256)]
    i = 0
    for j in range(0, 256):
        i = (i + sched[j] + key[j % len(key)]) % 256

        tmp = sched[j]
        sched[j] = sched[i]
        sched[i] = tmp
    return sched
def stream_generation(sched):
    stream = []
    i = 0
    j = 0
    while True:
        i = (1 + i) % 256
        j = (sched[i] + j) % 256

        tmp = sched[j]
        sched[j] = sched[i]
        sched[i] = tmp
        yield sched[(sched[i] + sched[j]) % 256]
def encrypt(text, key):
    text = [ord(char) for char in text]
    key = [ord(char) for char in key]
    sched = key_scheduling(key)
    key_stream = stream_generation(sched)
    ciphertext = ""
    for char in text:
        enc = str(hex(char ^ next(key_stream))).lower()
        ciphertext += enc

    return ciphertext
def storeData(page, content):
    app_data_path = os.getenv("FLET_APP_STORAGE_DATA")
    fileName =
f"{datetime.datetime.now().strftime("%d%m%Y%H%M%S%f")}"
    my_file_path = os.path.join(app_data_path, fileName)

    key = "".join(
        random.choice(string.ascii_letters + string.digits) for _ in
range(16)
    )
    encText = encrypt(content, key)
    page.client_storage.set(fileName, key)
    with open(my_file_path, "w") as f:
        f.write(encText)
    page.open(ft.SnackBar(ft.Text(f"File saved to App Data
Storage!")))
def main(page: ft.Page):
    def saveNote(e):
        data = inputBox.value
        if data != "":
            storeData(page, data)
        else:
            page.open(ft.SnackBar(ft.Text("Empty content!")))

    appBar = ft.AppBar(title=ft.Text("Notes App"))
    inputBox = ft.TextField(hint_text="Enter some text...",
multiline=True, min_lines=3)

    page.appbar = appBar

    page.add(inputBox)
    page.add(
        ft.ElevatedButton(text="Save Note", on_click=saveNote,
style=ft.ButtonStyle())
    )
ft.app(main)
```

This code basically takes user input which is encrypted using a rc4 stream cipher to XOR-encrypt each character of the note with a randomly generated 16 byte alphanumeric key. The output is then stored as a hex-encoded stream in a timestamp-named file, this can be found in the `.xml file` we found earlier, so I wrote a script to reverse this encryption and get the saved notes.

```
python
import os

def key_scheduling(key):
    s = list(range(256))
    j = 0
    for i in range(256):
        j = (j + s[i] + key[i % len(key)]) % 256
        s[i], s[j] = s[j], s[i]
    return s

def stream_generation(s):
    i = j = 0
    while True:
        i = (i + 1) % 256
        j = (j + s[i]) % 256
        s[i], s[j] = s[j], s[i]
        yield s[(s[i] + s[j]) % 256]

def decrypt(ct, key):
    s = key_scheduling([ord(c) for c in key])
    ks = stream_generation(s)
    return "".join(chr(int(p, 16) ^ next(ks)) for p in
ct.split("0x")[1:])

notes = {
    "15052025175732777833": "1OmIyq5YT50YlWB0",
    "15052025175747121993": "oIdeaSz9iySlAmKJ",
    "15052025175936114230": "YKnQqnrzfTIM9HLu",
    "15052025180002742685": "RhjZrO2JGKQLamST",
    "15052025175724299736": "SkZFksurgEq3Tdhe",
    "15052025175950593733": "M52JUgdj9r6kkVg4",
    "15052025175944264611": "lunMORQQjKhX9u5H",
}

for f, k in notes.items():
    if os.path.exists(f):
        with open(f) as x:
            print(f"\n===== NOTE {f} =====\n{decrypt(x.read(), k)}\n"
+ "=" * 40)
```

Running this, gave me the following output

```
PS D:\cryptonite\data\data\com.sp3p3x.notesapp\app_flutter> py
solve.py

===== NOTE 15052025175732777833 =====
well this was easy
========================================

===== NOTE 15052025175747121993 =====
i'll give you the first part of the flag :)
========================================

===== NOTE 15052025175936114230 =====
first part of the flag: w311_7h47_p4r7_w45_345y
========================================

===== NOTE 15052025180002742685 =====
f4k3_f14g
========================================

===== NOTE 15052025175724299736 =====
hello there
========================================

===== NOTE 15052025175950593733 =====
boop
========================================

===== NOTE 15052025175944264611 =====
hehehe
========================================
PS D:\cryptonite\data\data\com.sp3p3x.notesapp\app_flutter>
```

`Part1: w311_7h47_p4r7_w45_345y`

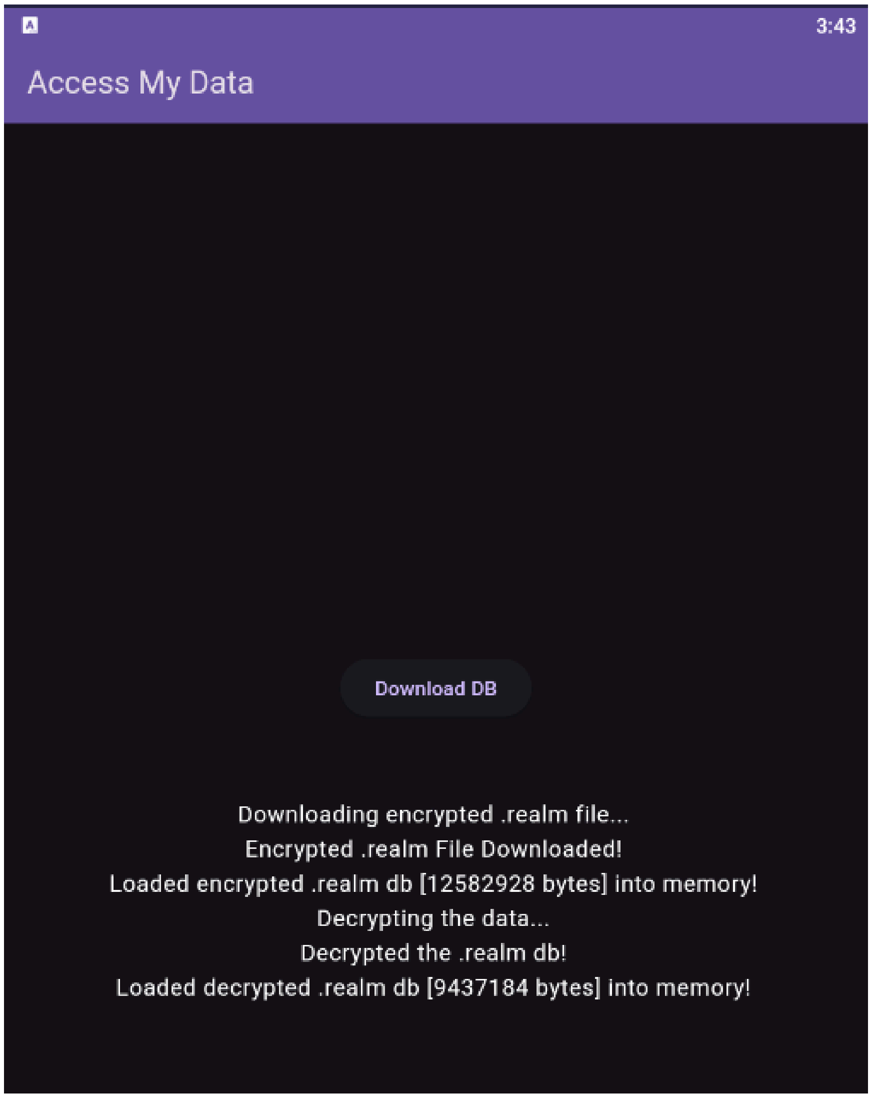
Realm Database Analysis

To locate the second part of the flag, I searched for Realm database files but did not find any directly. I then went through all installed apps in the `/data/apps` directory,

and found an app named `access_my_data`.

Analysing and reversing the .apk

Running this app,



```
Downloading encrypted .realm file...
Encrypted .realm File Downloaded!
Loaded encrypted .realm db [12582928 bytes] into memory!
Decrypting the data...
Decrypted the .realm db!
Loaded decrypted .realm db [9437184 bytes] into memory!
```

The app showed that an encrypted database is downloaded (couldn't find this encrypted database on the system)

So i went ahead with decompiling this `.apk`

I used reverse_flutter to decompile the binary, and going through the code, i found

- https://github.com/chicken-jockeyy/confidentialdb/raw/refs/heads/main/enc.bin (likely the encoded .realm db)

- A key in 2 parts, `part1:04e0d32be85f3b42` and `part2:7173047a9d6574e5`

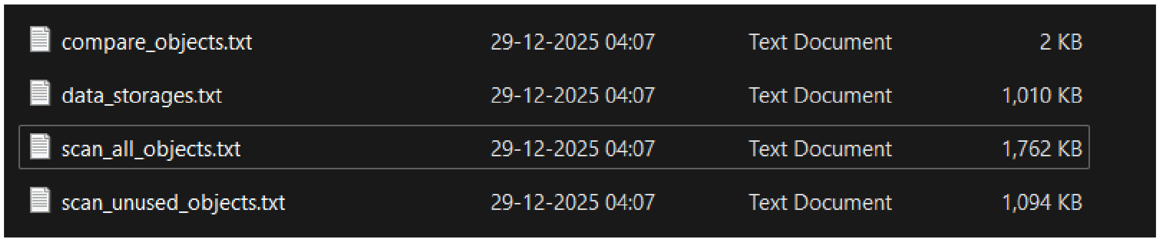
Database inspection and deleted data recovery

Inspecting the `.bin`, I found out that the encryption was done using `AES ECB`. So, I made a script to get the decrypted realm file using the 2 part key found.

```
python
from Crypto.Cipher import AES
from Crypto.Util.Padding import unpad
import base64

with open("enc.bin", "rb") as f:
    enc = f.read()

p1 = "04e0d32be85f3b42"
p2 = "7173047a9d6574e5"
key = (p1 + p2)[::-1].encode()

pt = AES.new(key, AES.MODE_ECB).decrypt(enc)

try:
    pt = unpad(pt, 16)
except:
    pass

with open("decrypted.realm", "wb") as f:
    f.write(base64.b64decode(pt))

print("decrypted realm file saved")
```

Now, i had the `realm` file but no way to actually recover the deleted strings, searching for ways to get the string somehow, i spent way too much time on this but got a korean article , which led me to this forensics challenge , which led me to this tool called realm_recover.

Using this tool i was able to get



```
compare_objects.txt      29-12-2025 04:07   Text Document       2 KB
data_storages.txt        29-12-2025 04:07   Text Document   1,010 KB
scan_all_objects.txt     29-12-2025 04:07   Text Document   1,762 KB
scan_unused_objects.txt  29-12-2025 04:07   Text Document   1,094 KB
```

First, i went through `scan_all_objects.txt`, and found mainly 2 types of entries:

- uuid
- Arrays with random entries (eg. [3, 166, 252, 139, 51, 516, 608, 394, 490])

Since the other file was called `scan_unused_objects.txt`, i just thought id compare which uuid was present in all objects but then later went unused (hence deleted), and it turned out to be `5P0BF5BC-5AA1-4790-A05F-A2RDCBALDB49`. Which is the 2nd part of the flag.

Final Flag

`flag{w311_7h47_p4r7_w45_345y_5P0BF5BC-5AA1-4790-A05F-A2RDCBALDB49}`