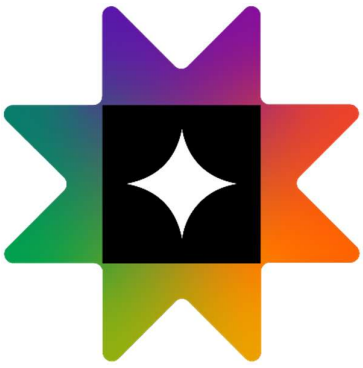
PROJECT ASSIGNMENT

Hex Mysteries

Techniques to Transform & Decode Encrypted Strings

LOKESH PATRA



SYSTEMATICALTRUISM

1. Importing Necessary Libraries:

```
import json
import base64
import urllib.parse
```

2. Loading .json file:

```
def load_json(file_path):
    with open(file_path, "r") as file:
        return json.load(file)

def inspect_dataset(data, key):
    sample = data[key]
    print(f"Unknown: {sample['unknown']}")

print('_____________________________________________________________________________________________')
    print(f"Hex: {sample['hex']}")

print('_____________________________________________________________________________________________')
    print(f"ASCII Text: {sample['ascii_text']}")
    return sample

file_path = r"C:\Users\Lokesh Patra\Documents\Projects\Junior Cryptographic
Analyst\assignment cryptography.json"
data = load_json(file_path)

# Ensure sample is defined by calling inspect_dataset
sample = inspect_dataset(data, "0")
```

```
Unknown: d8ab19d5c7a0f27c10fa57540506ac683bdf3c86c31f402beaf7e441d7eb789364cce03fac2f7ba75b0c4696dc30e8f20bca09ed71c
3e5fc70980ac1581c059c9c17cc118f6187c4a20c1f8b0e83b410e3f6fa6d50d48e006be371843453f675e7233888b4150c5f1c26bf049aad1b6
c0fa97a225baf1fec3cf31356897ef37be650aacdd25ff78dfc304e73d779768e6b19efe5af9477e074e3de07fcaec66014dea40fe186af33fa5
19d816ead854c6da927eb7dd8c349d14bbf68f4a97623ed755cfdc3cf09ab925af7f248f7d946a80ef46ae91094bf675dde11e5a8ba75d3cf98a
63e9acea0e1dadf78c6e839b0ca4c44bfd13c80155d82d14613c68b9843074f02d74710a485a0ae2914746de6583b9cae685aa66239d3f2cefb6
57033269d55aa3c4dc6a6c715cc7e4aa5ed6431d486bd6a3b12fe9dcac4635f98232f5c31e16e41daceccb6e584a62b9dee756e29c2adf82def9
650fad0b0aaf9bebf278dd97861d278e621043cb4d6858912f2ebb697ead4f9956100a5887f1feb84
_____________________________________________________________________________________________
Hex: 7b2275726c223a2268747470733a2f2f6173736574732d676c6f62616c2e776562736974652d66696c65732e636f6d2f363338616563353
0623961323363656331376238633464312f3633663966613663623962623130333538396630383530615f7a7137683373355316a6166636e6c647
4646677632e6a7067222c226861736822 3a22303737376666663738333831303130303830336333633030666566656630666422 2c22736561726
36848617368223a223234313233306a6871797767736f656534686a67656462306461633038222c226c6566745f657965223a5b3235352c31343
25d2c2272696768745f657965223a5b3331372c3134375d2c226e6f7365223a5b3238302c3138325d2c226c6566745f6d6f757468223a5b32343
82c3230325d2c2272696768745f6d6f757468223a5b3331392c3230395d2c22657870697265546 96d65223a313733353633373234303835312c2
2616254657374223a6e756c6c2c22706f726e223a66616c73657d
_____________________________________________________________________________________________
ASCII Text: {'url': 'https://assets-global.website-files.com/638aec50b9a23cec17b8c4d1/63f9fa6cb9bb103589f0850a_zq7h3
s51jafcnldtdfwc.jpg', 'hash': '0777fff783810100803c3c00fefef0fd', 'searchHash': '241230jhqywgsoee4hjgedb0dac08', 'le
ft_eye': [255, 142], 'right_eye': [317, 147], 'nose': [280, 182], 'left_mouth': [248, 202], 'right_mouth': [319, 20
9], 'expireTime': 1735637240851, 'abTest': None, 'porn': False}
```

Figure 1 All the texts extracted separately in HEX, ASCII & Unknown state format.

3. Hex to ASCII Conversion:

```
def hex_to_ascii(hex_str):
    try:
        ascii_text = bytes.fromhex(hex_str).decode('utf-8')
        return ascii_text
    except ValueError as e:
        print(f"Error decoding hex: {e}")
        return None

# Now we can use sample["hex"] without any issues
ascii_result = hex_to_ascii(sample["hex"])

if ascii_result:
    print(f"Decoded ASCII: {ascii_result}")
else:
    print("Failed to decode ASCII.")
```

Decoded ASCII: {"url":"https://assets-global.website-files.com/638aec50b9a23cec17b8c4d1/63f9fa6cb9bb103589f0850a_zq7h3s51jafcnldtdfwc.jpg","hash":"0777fff783810100803c3c00fefef0fd","searchHash":"241230jhqywgsoee4hjgedb0dac08","left_eye":[255,142],"right_eye":[317,147],"nose":[280,182],"left_mouth":[248,202],"right_mouth":[319,209],"expireTime":1735637240851,"abTest":null,"porn":false}

Figure 2 Decoded ASCII

4. ASCII to Hex Conversion

```
def ascii_to_hex(ascii_text):
    hex_str = ascii_text.encode('utf-8').hex()
    return hex_str

# Test the conversion
if ascii_result:
    re_encoded_hex = ascii_to_hex(ascii_result)
    print(f"Re-encoded Hex: {re_encoded_hex}")
else:
    print("No ASCII text to re-encode.")
```

Re-encoded Hex: 7b2275726c223a2268747470733a2f2f6173736574732d676c6f62616c2e776562736974652d66696c65732e636f6d2f3633386165633530623961323363656331376238633464312f3633663966613663623962623130333538396630383530615f7a713768337335316a6166636e6c6474646677632e6a7067222c2268617368223a223037373766666663738333831303130303830336333633030666566656630666422c227365617263684861736822a223234313233306a6871797767736f656534686a67656462306461633038222c226c6566745f657965223a5b3235352c3134325d2c2272696768745f657965223a5b3331372c3134375d2c226e6f7365223a5b3238302c3138325d2c226c6566745f6d6f757468223a5b3234382c3230325d2c2272696768745f6d6f757468223a5b3331392c3230395d2c22657870697265546964652223a313733353633373234303835312c22616254657374223a6e756c6c2c22706f726e223a66616c73657d

Figure 3 Reencoded Hex

5. Base64 Decoding Attempt

```
def decode_base64(encoded_str):
    try:
        return base64.b64decode(encoded_str).decode('utf-8')
    except Exception as e:
        print(f"Decoding failed: {e}")
        return None

# Test Base64 decoding
decoded_attempt = decode_base64(sample["unknown"])

if decoded_attempt:
    print(f"Base64 Decoding Result: {decoded_attempt}")
else:
    print("Failed to decode Base64.")
```

Decoding failed: 'utf-8' codec can't decode byte 0xd7 in position 3: invalid continuation byte
Failed to decode Base64.

FAILED

6. XOR Decryption

```
def xor_decrypt(data, key):
    return "".join(chr(ord(c) ^ key) for c in data)

# Testing with a range of possible keys
for key in range(256):
    decrypted_attempt = xor_decrypt(sample["unknown"], key)

    if "{" in decrypted_attempt and "}" in decrypted_attempt:  # JSON format
indicator
        print(f"Possible XOR Key: {key}")
        print(decrypted_attempt)
        break
```

```
Possible XOR Key: 24
| yz)!|-{/y(~*/{)(~y-/-,(-(.y{. +z|~+{ .{+)~,(*z}y~/},,)|/}z/ !+.,{{}(+~y{*~/zy/-z({,.!.|{+() ~*(z{y(!}|/){+}-~{/(!
(y{)- ){(-!{!{)/{{}) ~.) /{,y*({)~ z() +z,)(}+~.~y.|-(|, }((.z}+/) ,+,-+~./-}/*++   z,)-({-~){*.z~(,!yy|)z.{(~y!/y**
-zy~)~}{+{~+)+-. !/}~+/z}.-(yy{||*-~~/ |~{+(,}/+|//!/. }.z)!}~}-y~!,//}(/,}+|}(/~{y}{..(),|}y,(~}) .y~++~y-)!| ).}y|
-,{.|y!*/}z/|| {+,!|),zz~. ~,y!/.*+}|/--{~|{+{~(!yz!*-y~/~*, ~/|!,.y (}~,.y}!)(!,z~./-||}))}-y zy/-|+{~! y.+}!y{}y
(})|y|~/ {.} +!z({y,{,,z~|)+{ ()--| *|),.)+{. z! ,+(/,~(*|/,/)(y, -y(y}*!),/,.|}.- +z!{y}. -yy..*+!|+~*{}~z.-/(++*.!
|--yy+{,|{.y.{/)-{{/},yy-}|.,+)|, .z|.y+z)*~}!|{y{,.+-~! *+*~-{+})).},)|y{}{{z.}- ,y.*z!|}}/-.}*!{*y|~ *|}~!.-(~y|(z
(yy~!z}z~*/ ||!/ .)|*/ }.*)(,+{z,|. - !)*~*}zz.!/}y|,~!!-.)((y-  /~)~}z ,
```

Figure 4 XOR Keys [24]

7. Hex to Base64Encoding

```python
def encode_base64(hex_str):
    try:
        return base64.b64encode(bytes.fromhex(hex_str)).decode('utf-8')
    except Exception as e:
        print(f"Encoding failed: {e}")
        return None

# Test the encoding
encoded_attempt = encode_base64(sample["hex"])

if encoded_attempt:
    print(f"Encoded Unknown: {encoded_attempt}")
else:
    print("Failed to encode Base64.")
```

```
Encoded Unknown: eyJ1cmwiOiJodHRwczovL2Fzc2V0cy1nbG9iYWwud2Vic2l0ZS1maWxlcy5jb20vNjM4YWVjNTBiOWEyM2NlYzE3YjhjNGQxLzY
zZjlmYTZjYjliYjEwMzU4OWYwODUwYV96cTdoM3M1MWphZmNubGR0ZGZ3Yy5qcGciLCJoYXNoIjoiMDc3N2ZmZjc4MzgxMDEwMDgwM2MzYzAwZmVmZWY
wZmQiLCJzZWFyY2hIYXNoIjoiMjQxMjMwamhxeXdnc29lZTRoamdlZGIwZGFjMDgiLCJsZWZ0X2V5ZSI6WzI1NSwxNDJdLCJyaWdodF9leWUiOlszMTc
sMTQ3XSwibm9zZSI6WzI4MCwxODJdLCJsZWZ0X21vdXRoIjpbMjQ4LDIwMl0sInJpZ2h0X21vdXRoIjpbMzE5LDIwOV0sImV4cGlyZVRpbWUiOjE3MzU
2MzcyNDA4NTEsImFiVGVzdCI6bnVsbCwicG9ybiI6ZmFsc2V9
```

Figure 5 Unknown Encoding

8. Direct Hex to ASCII Decoding

```python
# Function to decode hex to ASCII
def hex_to_ascii(hex_str):
    try:
        return bytes.fromhex(hex_str).decode('utf-8')
    except UnicodeDecodeError:
        return "Decoding failed (possibly binary data)"

# Apply on the dataset
decoded_ascii = hex_to_ascii(sample["hex"])
print("Hex to ASCII Decoding Result:", decoded_ascii)
```

```
Hex to ASCII Decoding Result: {"url":"https://assets-global.website-files.com/638aec50b9a23cec17b8c4d1/63f9fa6cb9bb1
03589f0850a_zq7h3s51jafcnldtdfwc.jpg","hash":"0777fff783810100803c3c00fefef0fd","searchHash":"241230jhqywgsoee4hjged
b0dac08","left_eye":[255,142],"right_eye":[317,147],"nose":[280,182],"left_mouth":[248,202],"right_mouth":[319,20
9],"expireTime":1735637240851,"abTest":null,"porn":false}
```

i.e. Hex > ASCII = Actual Result [ASCII] from 1st Cell: MATCHED ✨

9. URL Encoding

```python
decoded_url = urllib.parse.unquote(sample["unknown"])
print("URL Decoded Unknown:", decoded_url)
```

```
URL Decoded Unknown: d8ab19d5c7a0f27c10fa57540506ac683bdf3c86c31f402beaf7e441d7eb789364cce03fac2f7ba75b0c4696dc30e8f
20bca09ed71c3e5fc70980ac1581c059c9c17cc118f6187c4a20c1f8b0e83b410e3f6fa6d50d48e006be371843453f675e7233888b4150c5f1c2
6bf049aad1b6c0fa97a225baf1fec3cf31356897ef37be650aacdd25ff78dfc304e73d779768e6b19efe5af9477e074e3de07fcaec66014dea40
fe186af33fa519d816ead854c6da927eb7dd8c349d14bbf68f4a97623ed755cfdc3cf09ab925af7f248f7d946a80ef46ae91094bf675dde11e5a
8ba75d3cf98a63e9acea0e1dadf78c6e839b0ca4c44bfd13c80155d82d14613c68b9843074f02d74710a485a0ae2914746de6583b9cae685aa66
239d3f2cefb657033269d55aa3c4dc6a6c715cc7e4aa5ed6431d486bd6a3b12fe9dcac4635f98232f5c31e16e41daceccb6e584a62b9dee756e2
9c2adf82def9650fad0b0aaf9bebf278dd97861d278e621043cb4d6858912f2ebb697ead4f9956100a5887f1feb84
```

i.e. URL-SafeEncode = Actual Result [Unknown] from 1st Cell: MATCHED ✨

10. Bitwise Teansformation

```
hex_length = len(sample["hex"])
unknown_length = len(sample["unknown"])

print(f"Hex Length: {hex_length}")
print(f"Unknown Length: {unknown_length}")
```

Hex Length: 744
Unknown Length: 768

i.e. Bitwise Transformation (maybe (+ extra bytes)) > Actual Result [Hex + (x)% = Unknown] from 1st Cell: MATCHED ✨

THANK YOU!

lightxlk.github.io