Universidad Nacional Autónoma de México

Facultad de Ciencias

Criptografía y Seguridad - 7133

P R A C T I C A 1

Equipo:

****** - 320083527**

Sosa Romo Juan Mario - 320051926

****** - 320017438**

****** - 320258211**

****** - 320340594**

Fecha de entrega:
1 de Septiembre de 2025

Profesora:
M. en C. Anayanzi Delia Martínez Hernández

Ayudantes:
Cecilia del Carmen Villatoro Ramos
José Angel Arévalo Avalos
David Armando Silva de Paz
Jesús Alberto Reyes Gutiérrez

Practica 1

Introducción

Desarrollo - Descifrado de archivos

1.

2.

3. **Descifrar el archivo file3.lol**

   Comencé por inspeccionar un poco el archivo utilizando un poco de código de Python:

```python
def ascii_preview(b, length=512):
  s = b[:length]
  txt = ''.join(chr(x) if 32 <= x < 127 or x in (9,10,13) else '.' for x in s)
  return txt

print("=== ASCII preview")
print(ascii_preview(head, 512))
print("\n=== Hex ===")
print(' '.join(f"{x:02x}" for x in head[:128]))
```

   Como nota, todo el código que voy a mostrar para este procedimiento está incluido en el `p1CriptoEj3.ipynb` dentro del `src`.

```
=== ASCII preview
AAAAIGZ0eXBpc29tAAACAGlzb21pc28yYXZjMW1wNDEAABLebW9vdgAAAGxtdmhkAAAAAAAAAAAAAAAAAAAD

=== Hex ===
41 41 41 41 49 47 5a 30 65 58 42 70 63 32 39 74 41 41 41 43 41 47 6c 7a 62 32 31 70
```

   De aquí, consulte con diversos LLMs para que me resaltaran el hecho de que todos los caracteres que encontré son imprimibles pertenecientes al alfabeto Base64 por lo que hice algunos tests para ver si se trataba de esta codificación.

```python
# 1) bytes del head en base 64
only_b64 = all((c in B64_CHARS_WITH_NL) for c in head)
print("\n1) Bytes de head en b64?", only_b64)

# 2) padding con =
has_eq = b'=' in raw[-16:] or b'=\n' in raw or raw.rstrip().endswith(b'=')
print("2) Padding con =:", has_eq)

# 3) contar caracteres en ASCII
visible_chars = [c for c in head if 32 <= c < 127 or c in (9,10,13)]
if visible_chars:
    b64_count = sum(1 for c in visible_chars if c in B64_STD or c in (10,13))
    frac = b64_count / len(visible_chars)
else:
```

```
    frac = 0.0
print(f"3) Fraccion de contenido que es ASCII: {frac:.3f}")

# 4) Medir porcentaje de lineas longitud 4
lines = ascii_preview(raw, 4096).splitlines()
if lines:
    lengths = [len(l) for l in lines if len(l.strip())>0]
    if lengths:
        multiples_of_4 = sum(1 for L in lengths if L % 4 == 0)
        pct_mult4 = multiples_of_4 / len(lengths)
    else:
        pct_mult4 = 0.0
else:
    pct_mult4 = 0.0
print(f"4) Medir porcentaje de lineas longitud 4: {pct_mult4:.3f}")

# 5) Buscar firmas de formatos binarios
signs = []
if raw.startswith(b"%PDF"):
    signs.append("PDF")
if raw.startswith(b"\x89PNG\r\n\x1a\n"):
    signs.append("PNG")
if raw.startswith(b"\xff\xd8\xff"):
    signs.append("JPEG")
if raw.startswith(b"PK\x03\x04"):
    signs.append("ZIP")
print("5) Firmas de formatos binarios en el inicio:", signs if signs else "False")
```

Dandome el siguiente resultado:

```
1) Bytes de head en b64? True
2) Padding con =: True
3) Fraccion de contenido que es ASCII: 1.000
4) Medir porcentaje de lineas longitud 4: 1.000
5) Firmas de formatos binarios en el inicio: False
```

Por estas métricas, todo indica que es base64. Como breviario cultural, la base 64 es una forma de codificar binario en texto de manera que los datos binarios están representados en secuencias de caracteres imprimibles usando el formato ASCII, específicamente cada 6 bits de los datos binarios de representan con un carácter del conjunto de 64 posibles. Es útil para transmitir datos binarios si solo podemos poner texto ademas, si la cadena no es del tamaño adecuado se utiliza el carácter de relleno =. Vemos que de las métricas recolectadas, todo parece apuntar a esta codificación.

Hice un código que verifica con heurísticas si es base 64, e implemente una decodificación de base64, después busca en la cabeza del nuevo archivo los magic bytes para saber la extensión del archivo y después de saber que es `mp4` puse también código para poder verlo en el mismo notebook. Todo esto se puede ver en la ruta antes mencionada y aquí los resultados:

```
[+] Leído file3.lol: 205704 bytes
[*] Heurística: parece Base64
[+] Decodificación Base64 OK, tamaño 154276 bytes
[+] Guardado: file3_decoded.mp4  (detected: MP4 (ftyp))
Salida: file3_decoded.mp4

Video:
```

Incluyo aquí también el decodificador:

```python
def base64_decode_from_scratch(s):
    """
    Decodificador base64 desde cero.
    Acepta s como str (puede contener espacios/newlines).
    Devuelve bytes decodificados.
    """
    # Mantener solo chars válidos + '='
    s = ''.join(ch for ch in s if ch in _b64chars or ch == '=')
    if not s:
        return b''
    # Si la longitud no es múltiplo de 4, rellenar con '='
    pad_len = (4 - (len(s) % 4)) % 4
    if pad_len:
        s += '=' * pad_len

    out = bytearray()
    for i in range(0, len(s), 4):
        block = s[i:i+4]
        vals = []
        pad = 0
        for ch in block:
            if ch == '=':
                vals.append(0)
                pad += 1
            else:
```

```
            # Asumir 'A' (valor 0) si char inválido
            vals.append(_b64rev.get(ch, 0))
    # recomponer 24 bits
    triple = (vals[0] << 18) | (vals[1] << 12) | (vals[2] << 6) | (vals[3])
    b1 = (triple >> 16) & 0xFF
    b2 = (triple >> 8) & 0xFF
    b3 = triple & 0xFF
    out.append(b1)
    if pad < 2:
        out.append(b2)
    if pad == 0:
        out.append(b3)
return bytes(out)
```

4. Desarrollo - preguntas

1.
2.
3.
4.
5.
6.

Conclusiones

Bibliografía

Referencias