Universidad Diego Portales
Criptografia y Seguridad en redes
Primer Semestre 2022

Actividad Recuperativa

Javier Ramos Acevedo javier.ramos1@mail.udp.cl

1. Codigo

Cabe destacar que todo el código que sera explicado a continuación puede ser encontrado en el siguiente enlace: Github: GrahaExarch - Actividad Recuperativa

2. Procedimiento

Para esta experiencia se hizo un algoritmo simple basado solo en un XOR, por lo tanto el código consta de solo 3 funciones, las cuales se listan a continuación:

1. **toBits**.

```python
def toBits(bits: str) -> str:
    """transforma un str ascii en bits

    Parameters
    ----------
    bits : str
        string ascii a transformar a bits
    Returns
    -------
    str: string de bits que representan al string original
    :Authors:
        - Javier Ramos
    """
    bin_bits = bin(int.from_bytes(bits.encode(), 'big'))
    return bin_bits
```

Figura 1: Código y comentario toBits

2. **toAscii**.

```python
def toAscii(bits: str) -> str:
    """transforma un str de bits en ascii

    Parameters
    ----------
    bits : str
        str en bits a transformar en ascii
    Returns
    -------
    str: string en ascii que corresponde al str original
    :Authors:
        - Javier Ramos
    """
    ascii_bits = int(bits, 2)
    ascii_text = ascii_bits.to_bytes(
        (ascii_bits.bit_length() + 7) // 8, 'big'
    ).decode()
    return ascii_text
```

Figura 2: Código y comentario toAscii

3. **xor**.

```python
def xor(base: str, key: str) -> str:
    """calcula el XOR entre 2 streams de bits

    Parameters
    ----------
    base : str
        steam de bit al cual se le aplica el XOR
    key : str
        llave para aplicar el XOR
    Returns
    -------
    str: Retorna un string del XOR resultante
    :Authors:
        - Javier Ramos
    """
    xorList = [(ord(a) ^ ord(b)) for a, b in zip(base, key)]
    return "".join(map(str, xorList))
```

Figura 3: Código y comentario xor

Mientras que el main del código se ve así:

```python
def main():
    while True:
        text = input("Ingrese el texto a cifrar: ")
        key = input("Ingrese el texto que sera la llave: ")
        bits_text = "".join(toBits(text))
        bits_text = bits_text.split('b')[1]
        bits_key = "".join(toBits(key))
        bits_key = bits_key.split('b')[1]
        hash = xor(bits_text, bits_key)
        ascii_hashed = toAscii(hash)
        print(
            f"El resultado en ascii es: {ascii_hashed} mientras que en bits"
            f" es: {hash}"
        )
        choice = input("Deseas descifrar el texto?:(Y/N) ")
        if choice == 'Y':
            dehash = xor(hash, bits_key)
            ascii_dehashed = toAscii(dehash)
            print(f"El texto original era: {ascii_dehashed}")
            break
        else:
            continue
```

Figura 4: Main que hace que el codigo fluya

Donde primer se piden los 2 textos con los que se trabajara, para luego transformarlos ambos a bits, realizar el XOR y luego devolverlo a ascii, en esta ultima parte no siempre se puede mostrar el mensaje, debido a que no se hace ninguna verificación sobre la validez de los caracteres resultantes en ascii. También en caso de querer agregar alguna operación se podría hacer antes o después del xor, de esta forma podría recuperarse mas fácilmente el mensaje original haciendo un shift opuesto.
Por otro lado el proceso de descifrado se ve en la sección del código con la variable “choice”, aquí lo que se hace es realizar un xor nuevamente sobre el hash, manteniendo la misma clave, se podría haber hecho también con el texto base, dado que en un XOR al tener el resultado y uno de los 2 strings originales, se puede recuperar el que falta.

3. Asimétrico

Para implementar alguna forma de hash asimétrico lo que se debería hacer primero es cambiar el funcionamiento para que use 2 llaves, una llave publica que sera usada para encriptar el mensaje y es igual para todos, mientras que cada usuario debería tener una llave privada, la cual es única para cada uno y se utiliza para descifrar. Lo que se debería resolver a continuación es el método con el cual se eligen/crean estas llaves privadas y publicas, dado que debe existir algún método el cual permita cierta coordinación entre ellas, por ejemplo podría utilizarse el protocolo X3DH para el intercambio de las llaves, mientras que las publicas podrían crearse a partir de un cifrado de mochila aplicado a la llave publica con alguna operación y ronda, de forma que genere entropía en las operaciones.

4. Material de Apoyo

Se puede observar a continuación como se imprimen caracteres de ascii-extendido o espacios en blancos por la no validez en el ascii "simple" de los bits que produce el xor.

```
Ingrese el texto a cifrar: Textoprueba
Ingrese el texto que sera la llave: clavesecreta
El resultado en ascii es: 7     ↓☻
♥↨▬↨§ mientras que en bits es: 01101110000100100011001000000100000101000000011000101110001011000010111000001110001010 1
Deseas descifrar el texto?:(Y/N) Y
El texto original era: Textoprueba
```

Figura 5: Ejemplo de la ejecución por consola

El bloque de shifts, indica el lugar ideal en donde se podrían considerar la implementación de esas transformaciones para obtener mas resistencia a los ataques.

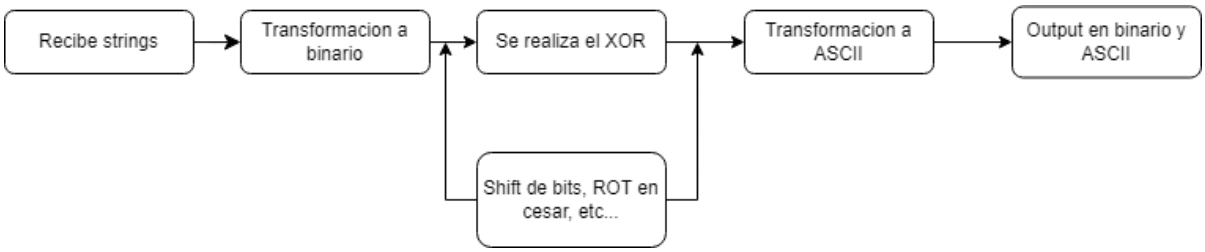

Figura 6: Diagrama del funcionamiento del ofuscado

Aquí se muestra el proceso de descifrado.

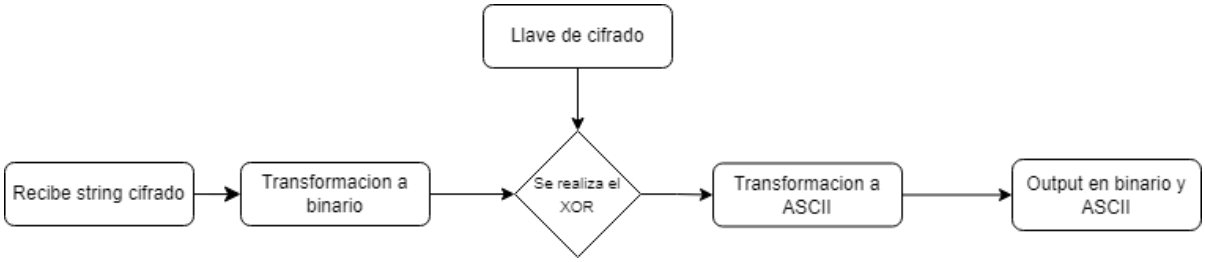

Figura 7: Diagrama del proceso de descifrado

Aquí podemos observar un diagrama de flujo entre Alice y Bob de como debería ser el flujo del proceso de cifrado, mas que del algoritmo mismo, el algoritmo estaría en el rombo del XOR, cabe destacar también que para la coordinación de las llaves se pueden usar distintos protocolos.

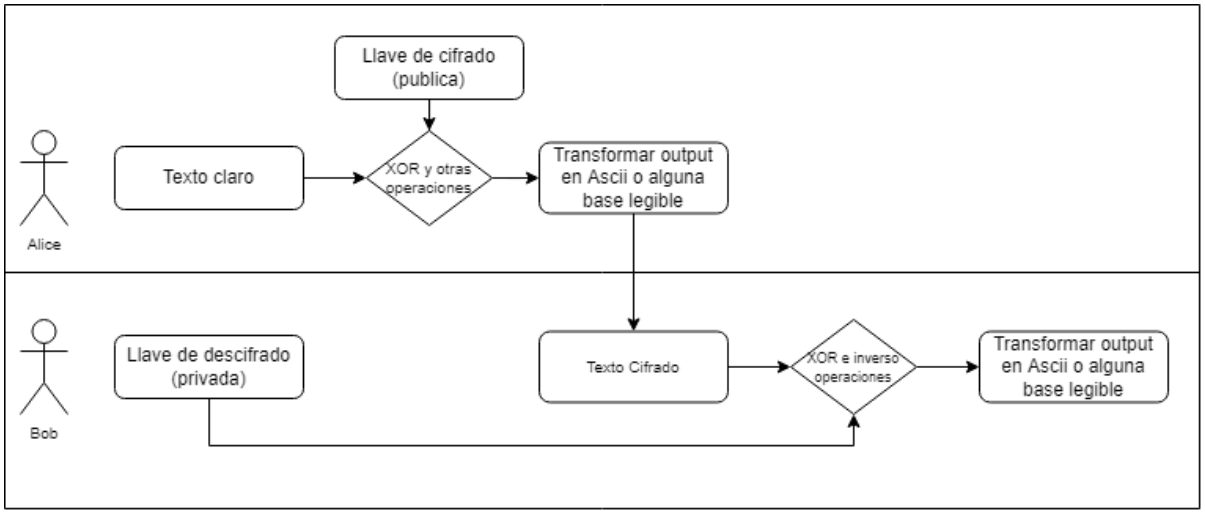

Figura 8: Diagrama de la versión Asimétrica