Fase Final Proyecto: Chat privado/seguro*

Diego Andrés Barrientos Linares, 201902929,[1] Sthephanie Lorena Bonilla Rodriguez, 201900300,[1] Nicole Alejandra López Calderón, 201800683,[1] and Damacio Barillas Quiñonez, 201806912[1]

[1] *Facultad de Ingeniería, Escuela de Mecánica Eléctrica, Ingeniería Electrónica.*

El presente proyecto es un prototipo de un chat privado y seguro. Se trata de una conexión alámbrica y bidireccional entre dos computadoras. El chat puede enviar mensajes cortos y largos, ya que es privado, el chat ofrece 3 tipos de cifrado para el envío de mensajes (césar, hill y un cifrado propio) que es posteriormente descifrado al llegar a la otra computadora. El programa cuenta con un proceso de detección de errores por medio de Hamming, corrigiendo un bit en cada trama del mensaje enviado. El programa está desarrollado en lenguaje Python.

I. OBJETIVOS

A. Generales

* Realizar un chat privado y seguro por medio de trasmisión alambica entre computadoras y utilizando tres tipos de cifrado

B. Específicos

* Aplicar los conocimientos teóricos adquiridos a lo largo de los cursos prerrequisitos.
* Poner en practica los conocimientos adquiridos a lo largo del laboratorio de comunicaciones 2, así como en la clase magistral y cursos previos.
* Diseñar y configurar un sistema funcional de comunicación.

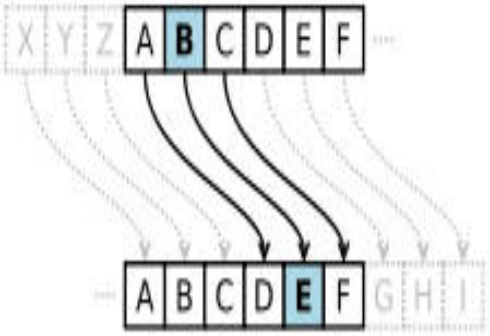
II. MARCO TEÓRICO

A. Cifrado

1. *Cifrado César*

Es una de las técnicas de cifrado más simples y conocidas, es uno de los tipos de cifrados más antiguos y se basa en el cifrado monoalfabético más simple. Se considera un método débil de criptografía, ya que es fácil decodificar el mensaje debido a sus técnicas de seguridad mínimas. En criptografía, un cifrado César se clasifica como un cifrado por sustitución. El cifrado César mueve cada letra un determinado número de espacios en el alfabeto. En este ejemplo se usa un desplazamiento de tres espacios, así que una B en el texto original se convierte en una E en el texto codificado.

Figura 1: Cifrado por sustitución



Fuente: ugr.es

2. *Cifrado Hill*

Este sistema esta basado en el álgebra lineal y ha sido importante en la historia de la criptografía. Este sistema es polialfabético pues puede darse que un mismo caracter en un mensaje a enviar se encripte en dos caracteres distintos en el mensaje encriptado.

Suponiendo que trabajamos con un alfabeto de 26 caracteres. Las letras se numeran en orden alfabético de forma tal que A=0, B=1, ... ,Z=25

Se elije un entero d que determina bloques de d elementos que son tratados como un vector de d dimensiones. Se elije de forma aleatoria una matriz de d × d elementos los cuales seran la clave a utilizar. Los elementos de la matriz de d × d serán enteros entre 0 y 25, además la matriz M debe ser inversible en z. Para la encriptación, el texto es dividido en bloques de d elementos los cuales se multiplican por la matriz d × d.Todas las operaciones aritméticas se realizan en la forma modulo 26, es decir que 26=0, 27=1, 28=2 etc.

Dado un mensaje a encriptar debemos tomar bloques del mensaje de "dçaracteres y aplicar: M×Pi=C, donde C es el código cifrado para el mensaje Pi.

* Laboratorio Comunicaciones 2

B. Raspberry Pi

Raspberry Pi es de las mejores pataformas para desarrollar el proyecto, debido a su definición:
" Es un pequeño computador que correo un sistema operativo linux capaz de permitirle a las personas de todas las edades explorar la computación y aprender a programar lenguajes como Scratch y Python.
Debido a esta definición esta plataforma cumple dos requisitos del proyecto, la comunicación entre ordenadores y que el código del proyecto se pueda realizar en el lenguaje de programación Python. Esta plataforma posee tambien puertos de conexión GPIO que pueden ser programados para distintos protocolos de comunicación alámbrica.

C. Conexiones Alámbricas

1. *Protocolo UART*

Para utilizar este protocolo necesitaremos dos cables de comunicación, los cuales serian Tx y Rx respectivamente. Este tipo de protocolo se ve limitado a 2 dispositivos, ya que la manera de conexión se deben cruzar los cables, esto se refiere que el terminal transmisor (Tx) de un dispositivo se debe conectar al terminal receptor (Rx) del otro dispositivo.

Para iniciar la transmisión de datos primero debemos enviar un Para iniciar la comunicación debemos enviar un bit de start, que siempre será un 0 y lo mantendremos durante un tiempo que llamamos tiempo de bit.
Pasado el tiempo de bit empezaremos a enviar los datos. Un 1, luego cuatro 0, dos 1 y por último un 0.Para finalizar la transmisión de datos se debe enviar un bit de stop, que consiste en enviar un bit 1 en el tiempo de bit.

El tiempo de bit es calculado usando el baudrate que es la cantidad de bits que se pueden transmitir por segundo:

$$T_b = \frac{1}{baudrate} \tag{1}$$

D. Código Hamming

En los sistemas digitales, los datos transmitidos para comunicación puede dañarse debido al ruido externo y cualquier otro fallo físico. Si los datos transmitidos no coinciden con los datos de entrada dados, se denomina 'error'. Los errores de datos pueden eliminar datos vitales en sistemas digitales. La transferencia de datos se realizará en forma de bits (0 y 1) en sistemas digitales. Si se cambia alguno de los bits, el rendimiento de todo el sistema puede verse afectado. Si el bit '1' se cambia al bit '0' o viceversa, entonces se denomina error de bit. Hay diferentes tipos de errores como errores de un solo bit, errores múltiples y errores de ráfaga.

La detección de errores se define como el método utilizado para detectar los errores transmitidos desde el transmisor / emisor al receptor en sistemas digitales. Los códigos de redundancia se agregan a los datos durante la transmisión para encontrar los errores. Estos se denominan códigos de detección de errores.

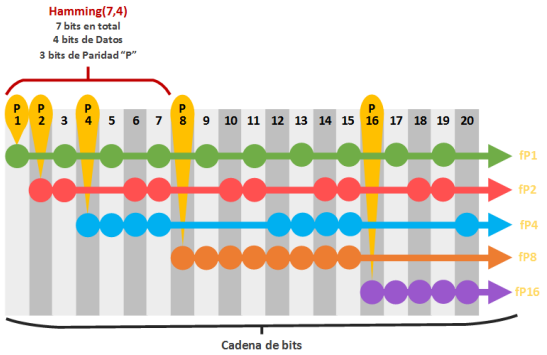
El código Hamming se define como un código lineal que se utiliza en el proceso de detección de errores hasta 2 errores intermedios. También es capaz de detectar errores de un solo bit. En este método, el remitente agrega los bits redundantes a los datos / mensajes para codificar los datos. Para realizar la detección y corrección de errores, estos bits redundantes se agregan en ciertas posiciones para el proceso de corrección de errores.

El proceso de codificación de un mensaje usando un código de Hamming por parte del remitente incluye 3 pasos:

- El primer paso es calcular el número de bits redundantes en un mensaje
- Coloque los bits redundantes en la posición correcta.
- Calcular los valores de bits redundantes

Aquí, los bits de paridad se utilizan para calcular los valores de los bits redundantes. Los bits de paridad pueden hacer que el número de 1 en un mensaje sea par o impar. Si el número total de 1 en un mensaje es par, se usa la paridad par, si el número total de unos en un mensaje es impar, se utiliza la paridad impar.

Figura 2: Funcionamiento código hamming



Fuente: jarroba.com

III. MATERIALES Y SOFTWARE

* Raspberry Pi

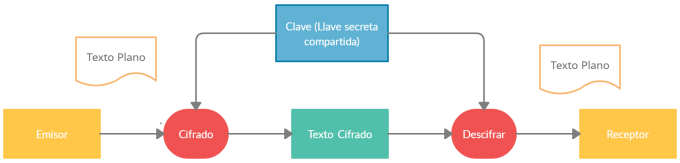
* Arduino

* Cable de conexión

* Python

IV. DIAGRAMAS DE BLOQUES

Figura 3: Funcionamiento del Proyecto



Fuente: Elaboración propia 2022.

V. PROPUESTA DE CIFRADO PROPIO

Figura 4: Código desarrollado para nuestro cifrado

```
def cifradopropio (txt):
    tipcifra = random.randint(0,2)
    print("")
    if tipcifra  == 0:
        txt= txt.replace("m","0")
        txt= txt.replace("u","1")
        txt= txt.replace("r","2")
        txt= txt.replace("c","3")
        txt= txt.replace("i","4")
        txt= txt.replace("e","5")
        txt= txt.replace("l","6")
        txt= txt.replace("a","7")
        txt= txt.replace("g","8")
        txt= txt.replace("o","9")
    if tipcifra  == 1:
        txt= txt.replace("n","0")
        txt= txt.replace("e","1")
        txt= txt.replace("u","2")
        txt= txt.replace("m","3")
        txt= txt.replace("a","4")
        txt= txt.replace("t","5")
        txt= txt.replace("i","6")
        txt= txt.replace("c","7")
        txt= txt.replace("o","8")
        txt= txt.replace("s","9")
```

Fuente: Elaboración propia 2022

```
    if tipcifra  == 2:
        txt= txt.replace("e","0")
        txt= txt.replace("u","1")
        txt= txt.replace("c","2")
        txt= txt.replace("a","3")
        txt= txt.replace("l","4")
        txt= txt.replace("i","5")
        txt= txt.replace("p","6")
        txt= txt.replace("t","7")
        txt= txt.replace("o","8")
        txt= txt.replace("s","9")
    msg = txt+str(tipcifra)
    return (msg)
```

Fuente: Elaboración propia 2022

VI. CÓDIGO UTILIZADO

A. Cifrado y Transmisión de mensajes

Figura 5: Cifrado César

```
def cifradocesar (txt):

    alfabeto = "abcdefghijklmnopqrstuvwxyz"
    alfabeto_may = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    longitud_alfabeto = len(alfabeto)
    codificado_cesar = ""
    for letra in txt:
        if not letra.isalpha() or letra.lower() == 'ñ':
            codificado_cesar += letra
            continue
        valor_letra = ord(letra)
        alfabeto_a_usar = alfabeto
        limite = 97
        if letra.isupper():
            limite = 65
            alfabeto_a_usar = alfabeto_may

        posicion = (valor_letra - limite + 3) % longitud_alfabeto

        codificado_cesar += alfabeto_a_usar[posicion]  #regresa los valores a letras y las concatena
    return codificado_cesar
```

Fuente: Elaboración propia 2022

Figura 6: Cifrado Hill

```
def cifradohill(message, key):

    ciphertext = ''
    matrix_mensaje = []
    list_temp = []
    cifrado_final = ''
    ciphertext_temp = ''
    cont = 0

    # Convertir el mensaje a mayusculas

    message = message.upper()

    # Si el tamaño del mensaje es menor o igual al tamaño de la clave

    if len(message) <= len(key):

        # Convertir el tamaño del mensaje al tamaño de la clave, si

        while len(message) < len(key):
            message = message + 'X'

        # Crear la matriz para el mensaje

        for i in range(0, len(message)):
            matrix_mensaje.append(diccionario_encryt[message[i]])

        # Se crea la matriz

        matrix_mensaje = np.array(matrix_mensaje)
```

Fuente: Elaboración propia 2022

Figura 7: Código Hamming

```
def hamming(mensaje_a_codificar):
    arreglo = list(mensaje_a_codificar)
    letra_a_binario = [ascii_a_binario(num) for num in arreglo]
    for index, value in enumerate(letra_a_binario):
        l = str(value)
        letra = l[1:]
        letra_a_binario[index] = letra
        d1 = int(letra[0])
        d2 = int(letra[1])
        d3 = int(letra[2])
        d4 = int(letra[3])
        d5 = int(letra[4])
        d6 = int(letra[5])
        d7 = int(letra[6])
        p1 = xor_5(d1,d2,d4,d5,d7)
        p2 = xor_5(d1,d3,d4,d6,d7)
        p3 = xor_3(d2,d3,d4)
        p4 = xor_3(d5,d6,d7)
        letraH = str(p1)+str(p2)+str(d1)+str(p3)+str(d2)+str(d3)+str(d4)+str(p4)+str(d5)+str(d6)+str(d7)
        letra_a_binario[index] = letraH

    return (letra_a_binario)
```

Fuente: Elaboración propia 2022

```
    posicion = binario_a_decimal(error)
    deco = [p1,p2,d1,p3,d2,d3,d4,p4,d5,d6,d7]
    poserror = int(letra[posicion -1])
    if (posicion != 0):
        if (poserror == 1):
            deco[posicion-1] = 0
        else:
            deco[posicion-1] = 1

    dec =str(deco[10])+str(deco[9])+str(deco[8])+str(deco[6])+str(deco[5])+str(deco[4])+str(deco[2])+"0"

    valor = 0

    for num in range(len(dec)):
        if(int(dec[num]) == 1):
            valor = valor + (2**num)

    letraenbinario = decimal_a_binario(valor)

    letradeco = binario_a_ascii(letraenbinario)



    salida = salida + letradeco

return salida
```

Fuente: Elaboración propia 2022

B. Recepción y Descifrado de mensajes

Figura 8: Menú de selección de cifrado utilizado

```
print("""
    1)Cifrado Cesar
    2)Cifrado Hill
    3)Cifrado propio
    """)



while (True):
    tipo_cifrado = int(input("Ingrese el tipo de cifrado que quiere: "))
    if(tipo_cifrado <= 3):
        txt = input("Ingrese mensaje recibido: ") #------Mensaje que queremos enviar
        if (tipo_cifrado == 1):
            mensaje_cifrado_y_codificado = decodificacion(txt)
            cifrado = descifrado_cesar(mensaje_cifrado_y_codificado)
            print ("El mensaje decodificado: "," ".join(mensaje_cifrado_y_codificado))
        elif(tipo_cifrado == 2):
            mensaje_cifrado_y_codificado = decodificacion(txt)
            key =  [[13, 17], [10, 16]]
            cifrado = descifrado_hill(mensaje_cifrado_y_codificado,key)
            print ("El mensaje decodificado: "," ".join(mensaje_cifrado_y_codificado))

        elif(tipo_cifrado == 3):
            mensaje_cifrado_y_codificado = decodificacion(txt)
            cifrado = descifrado_propio(mensaje_cifrado_y_codificado)
            print ("El mensaje decodificado: "," ".join(mensaje_cifrado_y_codificado))
    else:
        tipo_cifrado = int(input("Elegir un cifrado valido: "))

    print("Mensaje recibido: ",cifrado)
```

Fuente: Elaboración propia 2022

Figura 9: Descifrado César

```
def descifrado_cesar(mensaje):

    alfabeto = "abcdefghijklmnopqrstuvwxyz"
    alfabeto_may = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    longitud_alfabeto = len(alfabeto)
    decod_cesar = ""
    for letra in mensaje:
        if not letra.isalpha() or letra.lower() == 'ñ':
            decod_cesar += letra
            continue
        valor_letra = ord(letra)
        alfabeto_a_usar = alfabeto
        limite = 97
        if letra.isupper():
            limite = 65
            alfabeto_a_usar = alfabeto_may

        posicion = (valor_letra - limite - 3) % longitud_alfabeto


        decod_cesar += alfabeto_a_usar[posicion]  #regresa los valores a letras y las concatena
    return (decod_cesar)
```

Fuente: Elaboración propia 2022

Figura 10: Descifrado Hill

```
def descifrado_hill(message, key):


    plaintext = ''

    matrix_mensaje = []
    plaintext_temp = ''
    list_temp = []
    matrix_inversa = []
    matrix_mensaje = [message[i:i + len(key)] for i in range(0,
                      len(message), len(key))]

    # Se calcula la matriz inversa aplicando el modulo 41

    matrix_inversa = Matrix(key).inv_mod(41)

    # Se transforma en una matriz

    matrix_inversa = np.array(matrix_inversa)

    # Se pasan los elementos a float

    matrix_inversa = matrix_inversa.astype(float)

    # Para cada bloque

    for bloque in matrix_mensaje:

        # Se encripta el mensaje encriptado
```

Fuente: Elaboración propia 2022

Figura 11: Descifrado Propio

```
def descifrado_propio(txt):
    ultcar = txt[-1]
    if ultcar == '0':
        txt= txt.replace("0","m")
        txt= txt.replace("1","u")
        txt= txt.replace("2","r")
        txt= txt.replace("3","c")
        txt= txt.replace("4","i")
        txt= txt.replace("5","e")
        txt= txt.replace("6","l")
        txt= txt.replace("7","a")
        txt= txt.replace("8","g")
        txt= txt.replace("9","o")
    if ultcar == '1':
        txt= txt.replace("0","n")
        txt= txt.replace("1","e")
        txt= txt.replace("2","u")
        txt= txt.replace("3","m")
        txt= txt.replace("4","a")
        txt= txt.replace("5","t")
        txt= txt.replace("6","i")
        txt= txt.replace("7","c")
        txt= txt.replace("8","o")
        txt= txt.replace("9","s")
    if ultcar == '2':
        txt= txt.replace("0","e")
        txt= txt.replace("1","u")
        txt= txt.replace("2","c")
        txt= txt.replace("3","a")
        txt= txt.replace("4","l")
```

Fuente: Elaboración propia 2022

Figura 12: Código Hamming Inverso

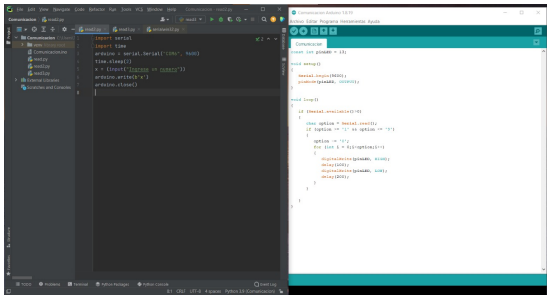
```
def decodificacion(mensaje_a_decodificar):
    arreglo = mensaje_a_decodificar.split()
    salida = ""
    for index, value in enumerate(arreglo):
        letra = arreglo[index]
        p1 = int(letra[0])
        p2 = int(letra[1])
        d1 = int(letra[2])
        p3 = int(letra[3])
        d2 = int(letra[4])
        d3 = int(letra[5])
        d4 = int(letra[6])
        p4 = int(letra[7])
        d5 = int(letra[8])
        d6 = int(letra[9])
        d7 = int(letra[10])
        prueba1= xor_5(d1,d2,d4,d5,d7)
        prueba2= xor_5(d1,d3,d4,d6,d7)
        prueba3= xor_3(d2,d3,d4)
        prueba4= xor_3(d5,d6,d7)
        if (p4 == prueba4):
            error = "0"
        else:
            error = "1"

        if (p3 == prueba3):
            error = error+"0"
        else:
```

Fuente: Elaboración propia 2022

C. Encriptación y desencriptación de Paquetes en Python

Figura 13: Conexión entre Arduino y Python



Fuente: Elaboración propia 2022

Figura 14: Conexión entre dos arduinos

```
#include <SoftwareSerial.h>
SoftwareSerial otroArduino(2, 3);

void setup() {
  Serial.begin(9600);
  while (!Serial) {
    ;
  }
  otroArduino.begin(9600);
}

void loop() {
  if(Serial.available()){
    String miPC=Serial.readString();
    otroArduino.println(miPC);
  }
  if(otroArduino.available()){
    String msjArd=otroArduino.readString();
    Serial.println(msjArd);
```

Fuente: Elaboración propia 2022

VII. REPOSITORIO DEL PROYECTO

En este link redirige al repositorio de Github donde se encuentra todo el código utilizado al proyecto:`https://github.com/nicolexlc/Chat-Privado-Seguro.git`

VIII. PROBLEMAS ENCONTRADOS DURANTE LA REALIZACIÓN DEL PROYECTO

Durante la realización del proyecto nos enfrentamos a una diversidad de problemas, el desarrollo del código del receptor para el Hamming fue uno de ellos pero se logró resolver.

También se presentaron problemas con la conexión alámbrica entre las dos computadoras, por lo que se recurrió al uso de arduinos y la conexión con el protocolo UART.

IX. CONCLUSIONES

- El uso de cifrados permite mantener privacidad y seguridad en el chat de los usuarios en el caso de este proyecto, pero puede servir para infinidad de sistemas de comunicación.

- Al momento de utilizar sistemas de comunicación es posible que se generen errores en la comunicación por ruidos u otros fenómenos. Para ello se utilizan métodos correctores de errores como el código Hamming.

- La Raspberry Pi es una computadora que puede usarse tanto como host o como cliente, por lo que su uso en el proyecto es ideal. El protocolo UART realiza la comunicación bidireccional entre computadoras y es muy fácil de implementarse por medio de Arduino.

[1] RaspberryPi *¿Que es Raspberry Pi?*[En linea][]. Disponible en:
https://raspberrypi.cl/que-es-raspberry/
[2] alfreedom. 2017*Protocolo de comunicación SPI*[En linea][]. Disponible en:
https://vidaembebida.wordpress.com/2017/02/08/protocolo-de-comunicacion-spi/
[3] Oscar Mauricio Fernández Alazte. 2015-2022*i2c*[En linea][]. Disponible en:
http://codigoelectronica.com/blog/i2c
[4] Shutterstock.*i2c*[En linea][]. Disponible en:
https://programarfacil.com/blog/arduino-blog/comunicacion-i2c-con-arduino/
[5] Enrique Gómez 2017.*i2c*[En linea][]. Disponible en:
https://www.rinconingenieril.es/funciona-puerto-serie-la-uart/
[6] Pythones.*¿Qué es Python? – Introducción al lenguaje*[En linea][]. Disponible en:
https://pythones.net/que-es-python-y-sus-caracteristicas/
[7] Jorge Calvo,European Valley*Encriptar mensajes con Python*[En linea][]. Disponible en:
www.europeanvalley.es/noticias/encriptar-mensajes-con-python/