



ITESO, Universidad
Jesuita de Guadalajara

PRÁCTICA 1

Comunicación TCP/IP, Validación CRC32 y encriptado
AES128

Desarrollo de software de comunicaciones en ambientes embebidos

Instituto Tecnológico y de Estudios Superiores de Occidente
Especialidad en Sistemas Embebidos
22-Febrero-2022

MERCADO, IXCHEL DAYANARA
lxchel.mercado@iteso.mx

Índice

Protocolo TCP/IP	2
Modelo de referencia OSI	2
Modelo de arquitectura del protocolo TCP/IP	3
Protocolo IP	3
Protocolo TCP.....	3
CRC.....	4
AES	4
Implementación de módulo de validación	5
Requerimientos	5
Interface.....	5
Código implementado	5
Scripts de Python	8
Resultados de la implementación	8
Problemas enfrentados durante la implementación.....	10
Conclusiones.....	10

Protocolo TCP/IP

"TCP/IP" es el acrónimo que se utiliza comúnmente para el conjunto de protocolos de red que componen el **conjunto de protocolos de Internet**. Para interconectar la red TCP/IP con otras redes, debe obtener una dirección IP única para la red. En el momento en que se redacta esta guía, esta dirección se obtiene a través de un proveedor de servicios de Internet (ISP).

Modelo de referencia OSI

La mayoría de los conjuntos de protocolos de red se estructuran en capas. La Organización Internacional para la Estandarización (ISO) ha diseñado el modelo de referencia de Interconexión de Sistemas Abiertos (OSI) que utiliza capas estructuradas. El modelo OSI describe una estructura con siete capas para las actividades de red. Cada capa tiene asociados uno o más protocolos. Las capas representan las operaciones de transferencia de datos comunes a todos los tipos de transferencias de datos entre las redes de cooperación.

Nº de capa	Nombre de capa	Descripción
7	Aplicación	Se compone de los servicios y aplicaciones de comunicación estándar que puede utilizar todo el mundo.
6	Presentación	Se asegura de que la información se transfiera al sistema receptor de un modo comprensible para el sistema.
5	Sesión	Administra las conexiones y terminaciones entre los sistemas que cooperan.
4	Transporte	Administra la transferencia de datos. Asimismo, garantiza que los datos recibidos sean idénticos a los transmitidos.
3	Red	Administra las direcciones de datos y la transferencia entre redes.
2	Vínculo de datos	Administra la transferencia de datos en el medio de red.
1	Física	Define las características del hardware de red.

El modelo de referencia OSI define las operaciones conceptuales que no son exclusivas de un conjunto de protocolos de red particular. Por ejemplo, el conjunto de protocolos de red OSI implementa las siete capas del modelo OSI. TCP/IP utiliza algunas de las capas del modelo OSI. TCP/IP también combina otras capas. Otros protocolos de red, como SNA, agregan una octava capa.

Modelo de arquitectura del protocolo TCP/IP

Ref. OSI Nº de capa	Equivalente de capa OSI	Capa TCP/IP	Ejemplos de protocolos TCP/IP
5,6,7	Aplicación, sesión, presentación	Aplicación	NFS, NIS, DNS, LDAP, telnet, ftp, rlogin, rsh, rcp, RIP, RDISC, SNMP y otros.
4	Transporte	Transporte	TCP, UDP, SCTP
3	Red	Internet	IPv4, IPv6, ARP, ICMP
2	Vínculo de datos	Vínculo de datos	PPP, IEEE 802.2
1	Física	Red física	Ethernet (IEEE 802.3), Token Ring, RS-232, FDDI y otros.

Protocolo IP

El protocolo IP y sus protocolos de enrutamiento asociados son posiblemente la parte más significativa del conjunto TCP/IP. El protocolo IP se encarga de:

- **Direcciones IP:** Las convenciones de direcciones IP forman parte del protocolo IP. Cómo diseñar un esquema de direcciones IPv4 introduce las direcciones IPv4 y Descripción general de las direcciones IPv6 las direcciones IPv6.
- **Comunicaciones de host a host:** El protocolo IP determina la ruta que debe utilizar un paquete, basándose en la dirección IP del sistema receptor.
- **Formato de paquetes:** el protocolo IP agrupa paquetes en unidades conocidas como datagramas. Puede ver una descripción completa de los datagramas en Capa de Internet: preparación de los paquetes para la entrega.
- **Fragmentación:** Si un paquete es demasiado grande para su transmisión a través del medio de red, el protocolo IP del sistema de envío divide el paquete en fragmentos de menor tamaño. A continuación, el protocolo IP del sistema receptor reconstruye los fragmentos y crea el paquete original.

Protocolo TCP

TCP permite a las aplicaciones comunicarse entre sí como si estuvieran conectadas físicamente. TCP envía los datos en un formato que se transmite carácter por carácter, en lugar de transmitirse por paquetes discretos. Esta transmisión consiste en lo siguiente:

- Punto de partida, que abre la conexión.
- Transmisión completa en orden de bytes.

- Punto de fin, que cierra la conexión.

TCP conecta un encabezado a los datos transmitidos. Este encabezado contiene múltiples parámetros que ayudan a los procesos del sistema transmisor a conectarse a sus procesos correspondientes en el sistema receptor.

TCP confirma que un paquete ha alcanzado su destino estableciendo una conexión de punto a punto entre los hosts de envío y recepción. Por tanto, el protocolo TCP se considera un protocolo fiable orientado a la conexión.

CRC

El CRC o Verificación de Redundancia Cíclica o Comprobación de redundancia cíclica es una técnica utilizada para detectar errores en datos digitales. El CRC es una función hash que detecta cambios accidentales en los datos de computadora sin procesar que se utilizan comúnmente en las redes de telecomunicaciones digitales y en los dispositivos de almacenamiento. Se basa en la división binaria y también se denomina Suma de Comprobación de Código Polinomial.

En la CRC una cantidad fija de bits de verificación (suma de verificación), se anexa al mensaje que necesita ser transmitido. Los receptores de datos reciben los datos e inspeccionan los bits de verificación en busca de errores. Matemáticamente, los receptores de datos verifican el valor de verificación adjunto al encontrar el resto de la división polinomial de los contenidos transmitidos. Si ha ocurrido un error significa que el mensaje ha sido alterado.

Esta técnica fue inventada por el matemático y científico computacional estadounidense William Wesley Peterson en 1961 y desarrollada por el CCITT (Comité Consultatif International Telegraphique et Telephonique). El polinomio de 32 bits usado en funciones CRC de Ethernet (y otros estándares) fue publicado en 1975.

Polinomios más usados:

CRC-12: $x^{12} + x^{11} + x^3 + x^2 + x + 1$. Usado para transmitir flujos de 6 bits, junto a otros 12 de redundancia.

CRC-16: $x^{16} + x^{15} + x^2 + 1$. Para flujos de 8 bits, con 16 de redundancia. Usado en USA, principalmente.

CRC-CCITT: $x^{16} + x^{12} + x^5 + 1$. Para flujos de 8 bits, con 16 de redundancia. Usado en Europa, principalmente.

CRC-32: $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$. Da una protección extra sobre la que dan los CRC de 16 bits, que suelen dar la suficiente. Se emplea por el comité de estándares de redes locales (IEEE-802) y en algunas aplicaciones del Departamento de Defensa de USA.

CRC ARPA: $X^{24} + X^{23} + X^{17} + X^{16} + X^{15} + X^{13} + X^{11} + X^{10} + X^9 + X^8 + X^5 + X^3 + 1$

AES

AES (Advanced Encryption Standard) es una especificación para la encriptación de datos electrónicos establecida por el U.S National Institute of Standards and Technology (NIST) en 2001. El algoritmo se basa en varias sustituciones, permutaciones y transformaciones

lineales, cada una ejecutada en bloques de datos de 16 bytes - por lo tanto, el término blockcipher. Esas operaciones se repiten varias veces, llamadas "rondas". Durante cada ronda, una clave circular única se calcula a partir de la clave de cifrado y se incorpora en los cálculos. Basado en la estructura de bloques de AES, el cambio de un solo bit, ya sea en la clave, o en el bloque de texto sin cifrado, da como resultado un bloque de texto cifrado completamente diferente - una ventaja clara sobre los cifrados de flujo tradicionales. La diferencia entre AES-128, AES-192 y AES-256 finalmente es la longitud de la clave: 128, 192 o 256 bits - todas las mejoras drásticas en comparación con la clave de 56 bits de DES. A modo de ilustración: El agrietamiento de una clave AES de 128 bits con un superordenador de última generación tomaría más tiempo que la presunta edad del universo. Hasta el día de hoy, no existe un ataque factible contra AES. Por lo tanto, AES sigue siendo el estándar de cifrado preferido para los gobiernos, bancos y sistemas de alta seguridad en todo el mundo.

Implementación de módulo de validación

Requerimientos

Los requerimientos de esta práctica eran los siguientes:

- La nueva capa de protocolo debe proporcionar los siguientes servicios:
 - Validación de la integridad del mensaje con CRC32.
 - Cifrado del mensaje con AES128.
- Esta capa debe implementarse como una librería en lenguaje C y probarse comunicando dos tarjetas K64 utilizando TCP/IP sobre Ethernet.
- La comunicación debe ser bidireccional, por lo que cada nodo debe de ser capaz de transmitir y recibir mensajes a través de la nueva capa.
- Para AES pueden utilizar la siguiente librería, o cualquier otra de su preferencia: <https://github.com/kokke/tiny-AES-c>
- Para CRC pueden utilizar el ejemplo de CRC que viene con el SDK.

Interface

La interface creada para la comunicación de este módulo con la capa superior es la siguiente:

```
uint8_t validation_module(void *dataptr, uint16_t len);
```

Esta interface es llamada por la capa de tcpchecho, la cuál le envía al módulo de validación el pointer a la data del buffer y su longitud, y el módulo le devuelve un uint8_t que indicará si la validación exitosamente o no.

Código implementado

Para poder usar la inteface "validation_module" se implementó la librería "module_craes_ixchel", la cuál dentro de sí incluye las librerías de aes, fsl_crc y string, para cumplir su trabajo. Esta práctica fue hecha tomado al microcontrolador frdmk64 como server.

```

1  /*
2   * module_crcaes_ixchel.c
3   *
4   * Created on: 19 feb 2022
5   * Author: Mercado
6   */
7  #include "aes.h"
8  #include "fsl_crc.h"
9  #include "module_crcaes_ixchel.h"
10 #include "fsl_debug_console.h"
11 #include <string.h>

```

Las funciones implementadas en esta librería son las siguientes:

- *InitCrc32*: función tomada de ejemplo dado por el profesor.

```

static void InitCrc32(CRC_Type *base, uint32_t seed)
{
    crc_config_t config;

    config.polynomial      = 0x04C11DB7U;
    config.seed            = seed;
    config.reflectIn       = true;
    config.reflectOut      = true;
    config.complementChecksum = true;
    config.crcBits         = kCrcBits32;
    config.crcResult       = kCrcFinalChecksum;

    CRC_Init(base, &config);
}

```

- *Validation_module*: Esta función hace toda la validación a partir la información proporcionada por la capa superior. La manera en la que funciona consiste en los siguientes pasos:
 - La primera parte del módulo es validar el CRC32, para lograr esto lo primero que se hizo fue tomar el pointer a la data, y recortar el mensaje original del crc32 que viene agregado en la data, para hacer esto, le quitamos los últimos 4 bytes del buffer con la función de *memcpy*, la cual está declarada en la librería de *string.h*. Después de esto, volvemos a usar la función *memcpy* para obtener sólo la parte del buffer que contiene los 4 bytes del sumcheck del crc32.

```

}
//el primero memcpy es para tener sólo el mensaje, y hacerle el sumcheck
memcpy(&message[0], (uint8_t*)dataptr, message_len);
PRINTF("\nafter memcpy cutting first part of the message\r\n");
for(int i= 0; i<message_len;i++)
{
    PRINTF("\nData:%c",message[i]);
}

//el segundo memcpy va a ser para sacar el sumcheck enviado
memcpy(&messages_checksum[0], ptr_initchecksum, (size_t)checksumlen);
PRINTF("\nafter memcpy cutting second part of the message\r\n");
received_checksum = messages_checksum[0] | (messages_checksum[1] << 8) | (messages_checksum[2] << 16)
PRINTF("\nProvided checksum: %d",received_checksum);

```

Lo siguiente es iniciar el CRC32, con la función *InitCrc32*, y calcular el checksum del mensaje original que cortamos anteriormente con la función

CRC_Get32bitResult, contenida en la librería *fsl_crc.h*; una vez obtenido este checksum, lo comparamos con el cortado del buffer, y si son iguales, significa que no hubo errores en la transmisión.

```
InitCrc32(base, 0xFFFFFFFFU);
PRINTF("\nCRC32 initiated...\r\n");
CRC_WriteData(base, (uint8_t *)&message[0], (size_t)message_len); ///
calculated_checksum = CRC_Get32bitResult(base);
PRINTF("\nProvided checksum: %d",received_checksum);

PRINTF("\n");
PRINTF("\nCalculated checksum: %d\r\n",calculated_checksum);
if(received_checksum==calculated_checksum)
{
    crcok=1;
    PRINTF("\nChecksum is the same, there was no error on the transmission");
}
else
{
    crcok=0;
    PRINTF("\nChecksum is not the same, there was an error on the transmission");
    return crcok;
}
```

- La segunda parte, va a depender de si hubo o no problemas en la transmisión, si hubo errores, le contestamos a la capa superior que hubo error y que haga lo que le corresponda de acuerdo a eso, de caso contrario, seguiríamos con la validación. Esta parte consiste en descryptar el mensaje, el mensaje ya lo tenemos separado, así que sólo queda iniciar el AES, con la misma llave e IV usados para encriptarlo, después, comenzamos a descryptar con la función *AES_CBC_decrypt_buffer* declarada en la librería *aes.h*, una vez concluida esta función, tenemos el mensaje descryptado y le contestamos a la capa de arriba que la validación terminó correctamente y puede hacer lo que le corresponda según esta información.

```
/////////////////////////////////AES/////////////////////////////////

if(crcok==1)
{
    PRINTF("\nInitializing AES...\r\n");
    AES_init_ctx_iv(&ctx, key, iv);
    PRINTF("\nDecrypting...\r\n");
    AES_CBC_decrypt_buffer(&ctx,&message[0], message_len);
    PRINTF("\nDecrypted message:\r\n");
    for(int i= 0; i<message_len;i++)
    {
        PRINTF("%c",message[i]);
    }

    return crcok;
}
```


Scripts de Python

Para poder usar el script de Python como cliente para el server de la tarjeta, se tuvo que hacer una ligera modificación, tanto en el programa de la tarjeta como en el del script de Python.

Del lado de la tarjeta, se modificó en el archivo `tcpecho.c`, el puerto en el que el netconn se creará, cambiándolo a 10000.

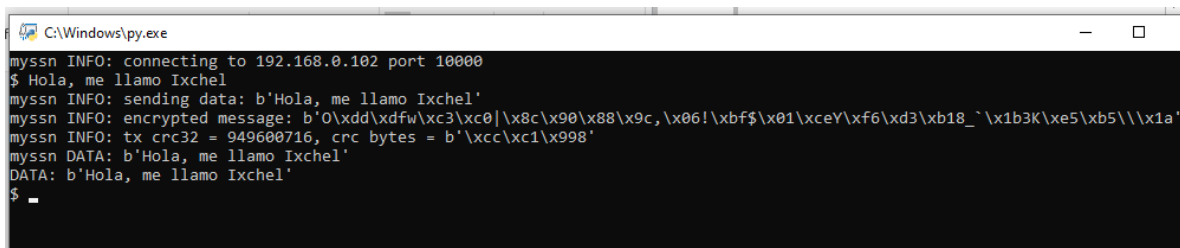
```
/* Bind connection to well known port number
#ifdef LWIP_IPV6
    conn = netconn_new(NETCONN_TCP_IPV6);
    netconn_bind(conn, IP6_ADDR_ANY, 10000);
#else /* LWIP_IPV6 */
    conn = netconn_new(NETCONN_TCP);
    netconn_bind(conn, IP_ADDR_ANY, 10000);
#endif /* LWIP_IPV6 */
```

En el script de Python, modificamos el server address, a la dirección del ethernet al que está conectado nuestra tarjeta.

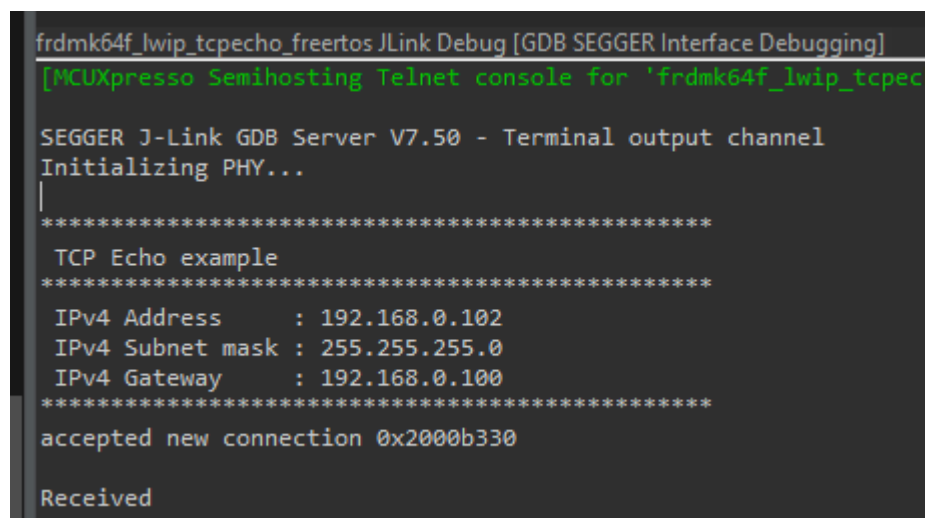
```
SERVER_ADDRESS = '192.168.0.102'
```

Resultados de la implementación

A continuación, dejaré unas imágenes del output del programa ya funcionando junto con lo que se envía del cliente de Python.



```
C:\Windows\py.exe
myssn INFO: connecting to 192.168.0.102 port 10000
$ Hola, me llamo Ixchel
myssn INFO: sending data: b'Hola, me llamo Ixchel'
myssn INFO: encrypted message: b'0\xdd\xdfw\xc3\xc0|\x8c\x90\x88\x9c,\x06!\xbf$\x01\xceV\xfe6\xd3\xb18_`x1b3K\xe5\xb5\\\x1a'
myssn INFO: tx crc32 = 949600716, crc bytes = b'\xcc\xc1\x998'
myssn DATA: b'Hola, me llamo Ixchel'
DATA: b'Hola, me llamo Ixchel'
$ _
```



```
frdmk64f_lwip_tcpecho_freertos JLink Debug [GDB SEGGER Interface Debugging]
[MCUXpresso Semihosting Telnet console for 'frdmk64f_lwip_tcpecho']

SEGGER J-Link GDB Server V7.50 - Terminal output channel
Initializing PHY...

*****
TCP Echo example
*****

IPv4 Address      : 192.168.0.102
IPv4 Subnet mask  : 255.255.255.0
IPv4 Gateway      : 192.168.0.100
*****
accepted new connection 0x2000b330

Received
```

```
Pointer:Æ
Data:0
Data:Ÿ
Data:ß
Data:w
Data:Ã
Data:À
Data:|
Data:Œ
Data:Ⓢ
Data:ˆ
Data:œ
Data:,
Data:Ⓢ
Data:!
Data:¿
Data:$
Data:Ⓢ
Data:Î
Data:Y
Data:ö
Data:Ó
Data:±
Data:8
Data:_
Data:˘
Data:Ⓢ
Data:3
Data:K
Data:å
Data:µ
Data:\
Data:Ⓢ
Data:İ
Data:Á
Validating message...
Testing CRC32
```

```
Provided checksum: 949600716
CRC32 initiated...

Provided checksum: 949600716

Calculated checksum: 949600716

Checksum is the same, there was no error on the transmission
Initializing AES...

Decrypting...

Decrypted message:
Hola, me llamo Ixchel
Validation done, message received correctly...
```

<

Problemas enfrentados durante la implementación

Los problemas enfrentados durante la implementación del módulo fueron los siguientes:

- El primero fue lograr obtener el buffer con los datos del mensaje, al ser pocas las veces que he trabajado con punteros, no tengo mucha experiencia con sus llamadas y sus casteos, este problema fue resuelto gracias a la ayuda del profesor que me mostró como se hace correctamente un casteo de un puntero void.
- El segundo, fue encontrar una función o manera de recortar la información obtenida del buffer para poder analizarla por separado, después de un buen rato de implementación encontré la función *memcpy* que recorta los vectores desde una dirección de memoria hasta el tamaño que le das.
- El tercer problema notorio, y el cuál no pude resolver, fue separar el módulo en un carpeta nueva en el proyecto, intenté solucionarlo de varias maneras pero cuando mandaba los archivos a la nueva carpeta me daba un error de estar redefiniendo mi función, lo cuál no estaba haciendo, y no comprendí porque pasaba esto; intenté solucionar el problema agregando el path de la carpeta nueva al proyecto pero esto no fue de ayuda, intenté varios métodos encontrados en internet pero tampoco lo solucionaron, por lo que decidí dejar los archivos del módulo en la carpeta de Source para entregar un programa que compilara y funcionara correctamente.
- El último problema que noté algo tarde, era el requerimiento de enviar 8 paquetes, no terminé de entender a que se refería esto y por falta de tiempo y conocimiento, eso no está incluido en esta práctica.

Conclusiones

Lo que me llevé de esta práctica es que en primer lugar, comprendí de mejor manera el funcionamiento de los protocolos TCP/IP, CRC y el encriptado AES. He encontrado que son muy útiles. También gracias a esta práctica he podido aprender a usar mejor los punteros y tener un acercamiento a lo que es crear una capa en un proyecto, lo cuál creo que no logré de manera ideal, pero puedo decir cuáles detalles me faltaron pulir y darles más tiempo, debí haber seccionado más el código del módulo, y separar dentro de la librería las validaciones de CRC y de AES y no hacerlo todo dentro de la misma función, también creo que aún me falta comprender más el concepto de transmitir y recibir más de un paquete de información y qué significa. Dejando de lado eso, ahora estoy más segura de los conceptos de los protocolos y los pasos que hay que seguir para implementarlos.