REDES PARA SISTEMAS EMBEBIDOS

Profesor: Sergio Nicolas Santana Sánchez

Practica 1

Carlos Emilio de Santiago Tovar -734514

Victor Manuel Sandoval Maciel – 734514

P

Contenido

Introducción.

Este proyecto implementa un canal de comunicación seguro sobre el ethernet en el microcontrolador FRDM K66. Busca aplicar una capa de encriptado y desencriptado con AES-CBC.

El proyecto se divide en 3 capas; Una primer y mas baja capa destinada al manejo de la comunicación Ethernet, esto significa que se encarga de manejar el transporte de los paquetes, el armado del paquete, checar banderas cuando se recibe un paquete y configurar e manejar el hardware de la tarjeta de desarrollo (k66). Una segunda capa se encarga de agregar seguridad a los paquetes enviados y limpiar de la seguridad a los paquetes recibidos para dar a conocer el contenido.

Por último, si se cuenta como capa, está la aplicación main, que ejecuta las instrucciones.

Diagrama de flujo

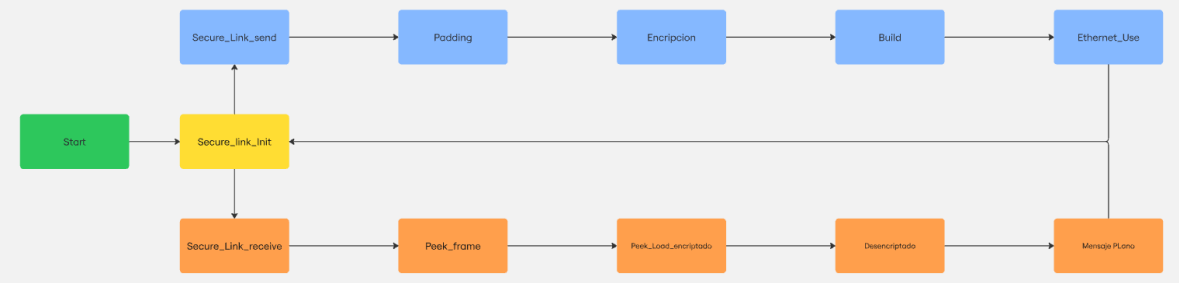

Ilustración 1: Diagrama de flujo

Funciones

El main, o raíz del proyecto solo inicializa y establece un ciclo while sobre el cual se repetirá el código ejecutando un ejemplo que comprueba el funcionamiento y uso de las funciones creadas.

```
29 int main(void)
30 {
31     SecureLink_Init();  /* Funcion pedida como requerimiento SecureLink_Init */
32     rxCb = app_rx_handler;
33
34
35     while (1)
36     {
37         SecureLink_Send((const uint8_t*)mensaje, strlen(mensaje));  /* Funcion pedida como requerimiento: SecureLink_Send Tx*/
38         SecureApp_Process();//Rx
39
40         for (volatile int i=0; i<10000000; i++) asm("nop"); // un pequeño delay
41     }
42 }
```

Ilustración 2: "Main"

SecureLink_Init

función pedida como requerimiento, aquí inicializamos todo lo necesario para el uso de ethernet, para este caso en su mayoría son tomadas del ejemplo del SDK_enet_txrx_transfer.

```
29 int main(void)
30 {
31     SecureLink_Init();  /* Funcion pedida como requerimiento SecureLink_Init */
32     rxCb = app_rx_handler;
33
```

Ilustración 3: Primer función para inicialización de variables y hardware para el uso de Ethernet

```
/*!
 * @brief Inicializa la capa de enlace seguro.
 *
 * Llama a la inicialización de la interfaz Ethernet (HAL).
 */
void SecureLink_Init(void)
{
    ETHERNET_TxRx_Init();
}
```

Ilustración 4:Contenido de la funcion SecureLink_Init

```
void ETHERNET_TxRx_Init(void)
{
    BOARD_InitBootPins();
    BOARD_InitBootClocks();
    BOARD_InitDebugConsole();
    enet_init();
    bool link = false;
    while (!link)
        PHY_GetLinkStatus(&phyHandle, &link);
}
```

Ilustración 5:Contenido de ETHERNET_TxRx_Init

rxCb:

Se crea un apuntador a función para ayudarnos a imprimir los datos próximamente, esta apunta a la función app_rx_handler;

```
    SecureLink_Init();  /* Funcion pedida como requerimiento SecureLink_Init */
    rxCb = app_rx_handler;

```

Ilustración 6: Uso de la función rxCb

```

static SecureApp_RxCb rxCb = NULL;

```

Ilustración 7: Declaración de la función rxCb

```
static void app_rx_handler(const uint8_t *msg, size_t len)
{
    PRINTF(">> Mensaje App recibido (%d bytes): %s\r\n", len, msg);
}
```

Ilustración 8: función a la que apunta, contiene el formato de impresión de lo recibido por ethernet.

Entramos a un while(1)...

Esta es la parte donde el código se mantiene en un ciclo, mandando y recibiendo el mensaje

```
    while (1)
    {
        SecureLink_Send((const uint8_t*)mensaje, strlen(mensaje));  /* Funcion pedida como requerimiento: SecureLink_Send Tx*/
        SecureApp_Process();//Rx

        for (volatile int i=0; i<10000000; i++) asm("nop"); // un pequeño delay
    }
}
```

Ilustración 9:Captura de la funcion while infinito y su contenido

SecureLink_Send((const uint8_t*)mensaje, strlen(mensaje));

Función pedida como requerimiento, a partir de esta función mandamos el mensaje por ethernet. Por lo que dentro armamos la trama del mensaje, primero agregamos el padding, antes de agregar el mensaje el aplicamos el cifrado AES, agregamos la MAC de destino y de origen y el payload cifrado. Por ultimo se entrega la trama completa y envía haciendo uso de la capa inferior de ethernet, sobre la cual se construyó el proyecto.

```
    while (1)
    {
        SecureLink_Send((const uint8_t*)mensaje, strlen(mensaje));  /* Funcion pedida como requerimiento: SecureLink_Send Tx*/
```

Ilustración 10: función en main

```
/*!
 * @brief Cifra un payload y envia una trama Ethernet segura.
 *
 * @param plain Puntero al payload en claro.
 * @param len   Longitud del payload en bytes (debe estar entre 1 y 112).
 * @return      true si el envio fue exitoso; false en caso contrario.
 */
bool SecureLink_Send(const uint8_t *plain, size_t len)
{
    if (len == 0U || len > 112U) { return false; }  // No cumple con el tamaño

    /* Copiar datos y aplicar padding PKCS7 */
    static uint8_t buf[128];
    memcpy(buf, plain, len);
    size_t clen = pad_pkcs7(buf, len);

    /* Inicializar contexto AES-CBC y cifrar */
    struct AES_ctx ctx;
    AES_init_ctx_iv(&ctx, key, iv);
    AES_CBC_encrypt_buffer(&ctx, buf, clen);

    /* Empaquetar trama: MAC destino / MAC origen / longitud / payload */
    extern uint8_t g_frame[];   /*!< Búfer de trama definido en enet_txrx_transfer.h */
    extern uint8_t g_macAddr[]; /*!< Dirección MAC origen */

    /* MAC destino fija */
    g_frame[0] = 0x00; g_frame[1] = 0x90;  g_frame[2] = 0x9E; g_frame[3] = 0x9A;  g_frame[4] = 0x9F ;g_frame[5] = 0x42;
    /* MAC origen */
    memcpy(&g_frame[6], g_macAddr, 6);
    /* Longitud del payload en little-endian */
    g_frame[12] = (uint8_t)(clen & 0xFF);
    g_frame[13] = (uint8_t)((clen >> 8) & 0xFF);
    /* Copiar payload cifrado */
    memcpy(&g_frame[14], buf, clen);

    /* Enviar trama completa */
    return ETHERNET_Tx(g_frame, clen + 14U);
}
```

Ilustración 11: Toma el payload en claro, comprueba su tamaño, aplica padding PKCS#7 y lo cifra con AES-CBC, empaqueta la trama Ethernet (MAC destino, MAC origen y longitud) junto al payload cifrado y la envía con ETHERNET_Tx, devolviendo true si el envío fue exitoso.

SecureApp_Process();

Esta función se prepara para actualizar las variables con los datos del mensaje recibido por la función SecureLink_Receive();

```
        SecureApp_Process();//Rx

        for (volatile int i=0; i<10000000; i++) asm("nop"); // un pequeño delay
```

Ilustración 12:Uso en main de la funcion SecureApp_Process();

SecureLink_Receive();

En esta función se recibe el mensaje o frames y se separa, se desencripta y actualiza dirección con el mensaje y tamaño de este; así como también se filtran mensajes que no encajen con el tamaño y formato del mensaje esperado

```
void SecureApp_Process(void)
{
    uint8_t buf[256];
    size_t len;
    if (SecureLink_Receive(buf, &len) && rxCb)  /* Funcion pedida como requerimiento: SecureLink_Receive */
    {
        rxCb(buf, len);
    }
}
```

Ilustración 13: Funcion donde se usa SecureLink_Receive();

```
/*!
 * @brief Recibe una trama Ethernet segura, la descifra y valida el padding.
 *
 * @param out     Búfer de salida para el payload descifrado.
 * @param out_len Puntero donde se deja la longitud del payload en bytes.
 * @return        true si la trama fue recibida y descifrada correctamente; false en caso contrario.
 */
bool SecureLink_Receive(uint8_t *out, size_t *out_len)
{
    /* Verificar longitud mínima de trama */
    size_t frame_len = ETHERNET_PeekFrameSize();
    if (frame_len < 14U) { return false; }

    /* Leer la trama completa */
    uint8_t frame[1514];
    if (!ETHERNET_Rx(frame, frame_len)) { return false; }

    /* Extraer longitud del payload (bytes 12-13, little-endian) */
    uint16_t plen = (uint16_t)frame[12] | ((uint16_t)frame[13] << 8);
    if (plen == 0U || plen > frame_len - 14U || (plen % AES_BLOCK_SIZE) != 0U) { return false; }
    /* Copiar payload cifrado */
    static uint8_t buf[1500];
    memcpy(buf, &frame[14], plen);

    /* Inicializar contexto AES-CBC y descifrar */
    struct AES_ctx ctx;
    AES_init_ctx_iv(&ctx, key, iv);
    AES_CBC_decrypt_buffer(&ctx, buf, plen);

    /* Validar padding PKCS 7 */
    uint8_t p = buf[plen - 1U];
    if (p == 0U || p > AES_BLOCK_SIZE)
    {
        return false;
    }
    for (uint8_t i = 0U; i < p; i++)
    {
        if (buf[plen - 1U - i] != p)
        {
            return false;
        }
    }
```

Ilustración 14: Recibe una trama Ethernet cifrada, extrae y descifra el payload con AES-CBC, valida el padding PKCS#7 y devuelve true sólo si todas las comprobaciones pasan.

```
Frame received, length = 60
Mensaje desencriptado recibido: ...si aÃºn te queda la luna.
```

Ilustración 115: Salida en la terminal de la K66 desencriptando el paquete que llegó por ethernet e imprimiendo en la terminal.

```
>>> Paquete recibido
Longitud total del paquete recibido: 62
Primeros 20 bytes: 00909e9a9f42d4bed94522603000a29488aa6944
Payload length from header: 48 bytes
Mensaje desencriptado: No digas que el sol se ha puesto...
Respuesta: ...si aún te queda la luna.
Tamaño del mensaje cifrado de respuesta: 32 bytes
Enviando frame con longitud total: 46 bytes
```

Ilustración 116: Terminal en Python, recibe el paquete de la K66 por ethernet, lo desencripta y manda de regreso la respuesta encriptada por etherntet.

```python
while True:
    print("Esperando mensaje de la K66...")

    rx_packet = sniff(lfilter=lambda x: x.src == frdm_eth_mac, count=1, iface=conf.iface)

    print("\n>>> Paquete recibido")
    packet = rx_packet[0]

    if Ether in packet:
        raw_bytes = bytes(packet)

        # Leer longitud del payload cifrado desde bytes 12 y 13 de la trama Ethernet
        padded_len = raw_bytes[12] + (raw_bytes[13] << 8)

        print(f"Longitud total del paquete recibido: {len(raw_bytes)}")
        print(f"Primeros 20 bytes: {raw_bytes[:20].hex()}")

        print(f"Payload length from header: {padded_len} bytes")

        encrypted_payload = raw_bytes[14:14 + padded_len]

        if len(encrypted_payload) < padded_len:
            print(f"Payload incompleto, esperado: {padded_len} bytes, recibido: {len(encrypted_payload)} bytes")
            continue
```

Ilustración 17: La siguiente sección del código en Python se encarga de recibir tramas Ethernet enviadas desde la placa K66. Utiliza la librería Scapy para capturar paquetes crudos a nivel de capa 2 (Ethernet), filtra por la dirección MAC de origen correspondiente a la K66, y posteriormente extrae y desencripta el mensaje recibido utilizando AES-128 en modo CBC.

```python
        # Construir y enviar paquete Ethernet de respuesta
        # [0-5]   MAC destino (K66)
        # [6-11]  MAC origen (PC)
        # [12-13] tamaño del payload (little endian)
        # [14-...] payload cifrado

        def mac_str_to_bytes(mac_str):  2 usages
            return bytes(int(b, 16) for b in mac_str.split(':'))


        dst_mac_bytes = mac_str_to_bytes(frdm_eth_mac)
        src_mac_bytes = mac_str_to_bytes(pc_eth_mac)
        payload_len_bytes = len(encrypted_reply).to_bytes( length: 2,  byteorder: 'little')

        frame = dst_mac_bytes + src_mac_bytes + payload_len_bytes + encrypted_reply

        print(f"Enviando frame con longitud total: {len(frame)} bytes")

        sendp(frame, iface=conf.iface)
```

Ilustración 18: Esta parte del código construye y envía manualmente una trama Ethernet personalizada desde la PC hacia la K66, utilizando el protocolo definido. Se utiliza cifrado AES-128 en modo CBC para proteger el contenido, y el paquete se estructura a bajo nivel.

Retos enfrentados

Dado que tenemos un tiempo utilizar Python, el descargar un IDE y el programa junto a las extensiones o librerías necesarias fue un reto el volver a comprender su funcionamiento y Principal reto, comprender código Python.

Un reto mas interesante fue el comprender como trabaja el ejemplo del SDK, y como se puede construir una capa sobre este para agregar los requerimientos en seguridad. Por otro lado, un problema

Conclusiones

Esta práctica me permitió integrar de manera práctica conocimientos clave de redes y sistemas embebidos. Diseñar y depurar una comunicación bidireccional entre la PC y la K66, utilizando tramas Ethernet personalizadas con cifrado AES, fue una experiencia valiosa para comprender cómo se construyen protocolos seguros a bajo nivel, sin depender de capas superiores como IP o TCP.

Uno de los principales retos fue implementar correctamente el padding PKCS#7 y garantizar que tanto la K66 como la PC interpretaran de forma coherente el formato de los mensajes. Esto me llevó a reflexionar sobre la importancia de alinear no solo los algoritmos criptográficos, sino también la estructura precisa de cada byte transmitido en el protocolo, especialmente en contextos donde no hay tolerancia a errores de interpretación.

Además, este proyecto reforzó la importancia de asegurar que una PC y un microcontrolador compartan un protocolo robusto, claro y seguro requiere atención al detalle, pruebas sistemáticas y una comprensión sólida de todos los componentes involucrados.

Repositorio (Github)