BÁO CÁO THỰC HÀNH ĐỒ ÁN MÔN HỌC
Thực hiện Service $27H (Security Access) của chuẩn giao tiếp CAN

Môn học: **CHUYÊN ĐỀ THIẾT KẾ HỆ THỐNG NHÚNG 1**- Mã lớp: **CE437.N11**
Giảng viên hướng dẫn thực hành: Phạm Minh Quân

Thông tin sinh viên	Mã số sinh viên: 19521022 – 19521387 Họ và tên: Nguyễn Văn Tín – Cao Phan Tiến Dũng
Link các tài liệu tham khảo *(nếu có)*	
Đánh giá của giảng viên: *+ Nhận xét* *+ Các lỗi trong chương trình* *+ Gợi ý*	

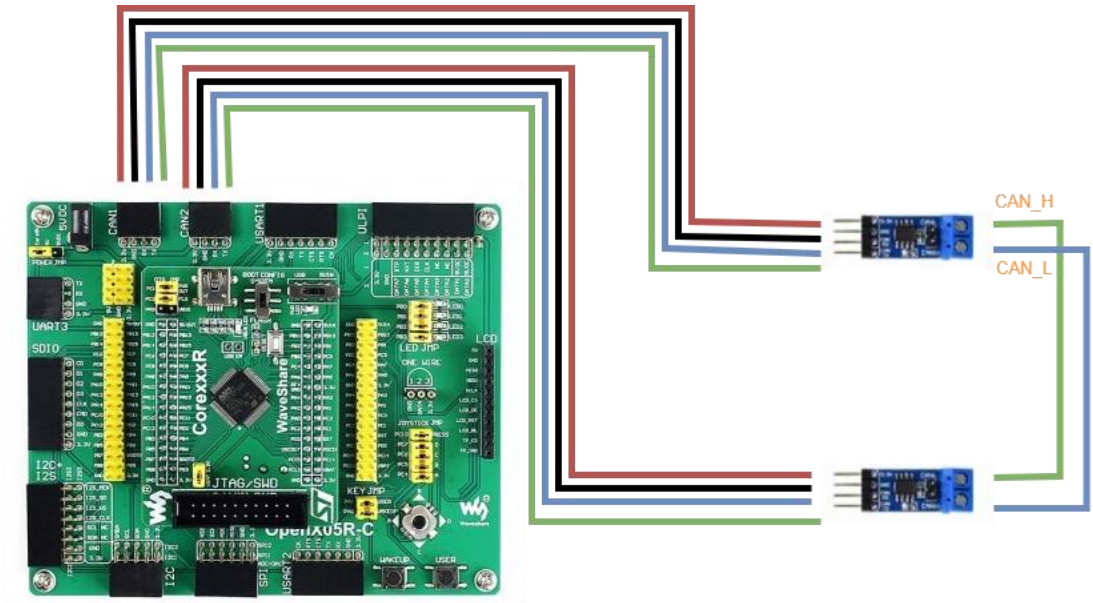
[Báo cáo chi tiết các thao tác, quy trình sinh viên đã thực hiện trong quá trình làm bài thực hành. Chụp lại hình ảnh màn hình hoặc hình ảnh kết quả chạy trên sản phẩm. Mô tả và giải thích chương trình tương ứng để cho ra kết quả như hình ảnh đã trình bày. Sinh viên xuất ra file .pdf và đặt tên theo cấu trúc: MSSV_HoTen_Labx_Report.pdf (Trong đó: MSSV là mã số sinh viên, HoTen là họ và tên, x trong Labx là chỉ số của bài thực hành tương ứng]

Mục lục

1 Tổng quan hệ thống: 3
1.1 Dịch vụ Security Access....3
1.2 Cấu trúc gói tin....4
1.2.1 Request Seed....4
1.2.2 Send Key....5
2 Hiện thực một số hàm ở lớp mạng....5
2.1 Các cấu trúc dữ liệu....5
2.1.1 CanTP_HandleTypedef....5
2.1.2 CanTP_Packet....6
2.1.3 Khác....6
2.2 Các hàm hiện thực....6
2.2.1 Hàm khởi tạo....6
2.2.2 Hàm gửi CanTP_Transmit....7
2.2.3 Hàm nhận CanTP_Receive....9
3 Hiện thực dịch vụ $27H Security Access ở lớp chẩn đoán:....10
3.1 Xử lý trên ECU....10
3.1.1 Khởi tạo....10
3.1.2 Vòng lặp....12
3.2 Xử lý trên Tester....13
3.2.1 Khởi tạo....14
3.2.2 Kịch bản kiểm tra.....15
3.2.3 Chương trình xử lý.....15
4 Kết quả....17
4.1 Trường hợp gửi Tester gửi lại Key sai.....17
4.2 Trường hợp Tester tính và gửi key đúng.....18

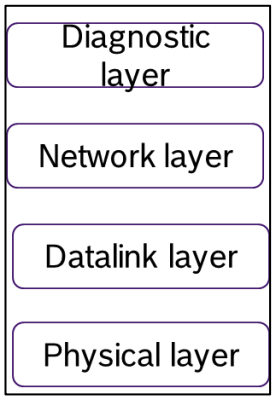
1 Tổng quan hệ thống:

Trên KIT thực hành có 2 module CAN1 và CAN2, ta thực hiện kết nối mạng CAN giữa 2 module CAN1 và CAN 2 như **Error! Reference source not found.**.



Hình 1.1: Sơ đồ kết nối

CAN1 sẽ đóng vai trò là ECU Module và CAN2 sẽ đóng vai trò là Tester Module. Mỗi module sẽ được chạy ở một luồng riêng. Hai module thực hiện giao tiếp chẩn đoán qua giao thức CAN. Hình 2 mô tả bố trí các lớp trong giao thức CAN.



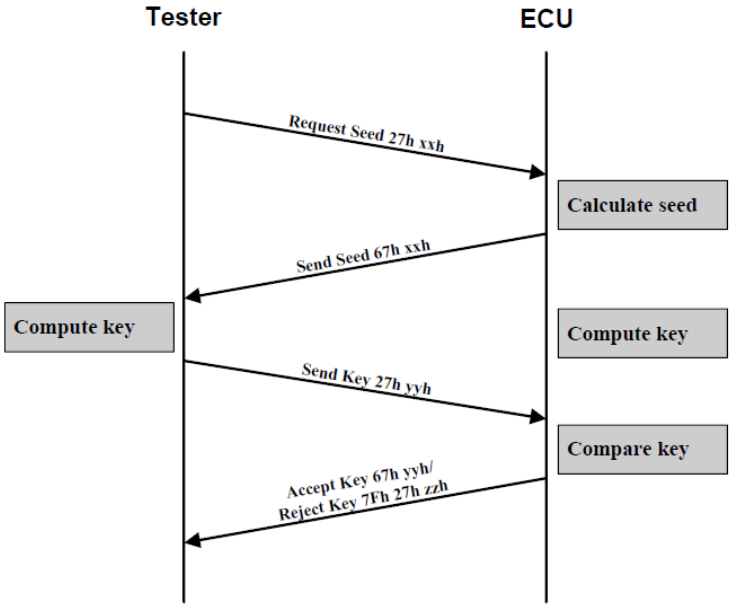
Hình 1.2: Sơ đồ các tầng

Đặc tả của các lớp đã được nêu trong bài thực hành số 4. Đề tài này nhóm sẽ hiện thực service $27H Security Access trong lớp chẩn đoán. Lớp mạng được hiện thực sẽ bao gồm lớp liên kết dữ liệu bên trong. Và thư viện STM32 HAL CAN sẽ được sử dụng để tương tác với lớp vật lý trong module.
Vì ECU và Tester module đều trên cùng một board và ở trên cùng một vii điều khiển nên nhóm sẽ sử dụng thư CMSIS-RTOS v2 để tạo 2 luồng song song một luồng là ECU một luồng là Tester. Hai luồng chỉ có thể giao tiếp với nhau thông qua giao thức CAN như **Error! Reference source not found.**.

1.1 Dịch vụ Security Access

Dịch vụ Security Access được cung cấp như là một phương tiện để truy cập vào dữ liệu và các dịch vụ chẩn đoán mà giới hạn truy cập vì lí do bảo mật, an toàn,…

Dịch vụ Security Access sử dụng cơ chế thuật toán seed và key. Đầu tiên Tester sẽ yêu cầu đến ECU mở khóa qua dịch vụ Security Access – Request Seed. Và sau đó ECU sẽ trả về seed. Từ giá trị seed Tester và ECU sẽ tính toán ra giá trị Key.
Bước tiếp theo, Tester sẽ gửi giá trị Key mà mình tính được về cho ECU thông qua tin nhắn Security Access – Send Key. ECU sẽ so sánh giá trị Key mình nhận được với giá trị đã được tính sẵn trong ECU. Nếu 2 giá trị là trùng nhau, ECU sẽ mở khóa và một số tính năng dịch vụ đặc biệt và thông báo về cho Tester. Còn nếu 2 giá trị không trùng nhau thì sẽ được xem là một lần truy cập lỗi.



Hình 1.3: Sơ đồ giao tiếp

1.2 Cấu trúc gói tin

1.2.1 Request Seed

Request

Byte	Parameter Name	Cvt	Value (hex)
#1	RequestServiceIdentifier	M	27
	Sub-Function = [		

Byte	Parameter Name	Cvt	Value (hex)
#2	RequestSeed]	M	01, 09

Positive Response

Byte	Parameter Name	Cvt	Value (hex)
#1	PositiveResponseServiceIdentifier	M	67
#2	Sub-Function = [RequestSeed]	M	01, 09
	SecuritySeed [ ] = [		
#3	seed#1 (high byte)	M	00-FF
#4	seed#2	M	00-FF
#5	seed#3	M	00- FF
#6	seed#4 (low byte)]	M	00-FF

1.2.2 Send Key

Request

Byte	Parameter Name	Cvt	Value (hex)
#1	RequestServiceIdentifier	M	27

Byte	Parameter Name	Cvt	Value (hex)
#2	Sub-Function = [SendKey]	M	02, 0A
	SecurityKey [ ] = [		
#3	key#1 (high byte)	M	00-FF
#4	key#2	M	00- FF
#5	key#3	M	00-FF
#6	key#4 (low byte)]	M	00-FF

Positive Response

Byte	Parameter Name	Cvt	Value (hex)
#1	PositiveResponseServiceIdentifier	M	67
#2	Sub-Function = [SendKey]	M	02, 0A

2 Hiện thực một số hàm ở lớp mạng

2.1 Các cấu trúc dữ liệu

2.1.1 CanTP_HandleTypedef

```
typedef struct CanTP_HandleTypedef
{
    uint16_t CanTxID;
    uint16_t CanRxID;
    CAN_HandleTypeDef* hcan;
    uint16_t CanRxFifo;
    CAN_FilterTypeDef CanFilter;
    osEventFlagsId_t Status;
    uint8_t CanRxBuffer[8];
    uint8_t CanRxLen;
} CanTP_HandleTypedef;
```

Hình 2.1 Cấu trúc quản lý đối tượng lớp mạng

Trong đó:

- CanTxID: ID gói tin sẽ gửi đến mạng CAN
- CanRxID: Can ID đối tượng sẽ lắng nghe và nhận dự liệu
- Hcan: Đối tượng quản lý việc giao tiếp tầng vật lý của CAN
- CanRxFifo: Fifo nhận dữ liệu gói tin CAN trong phần cứng STM32
- CanFilter: Chứa cấu hình bộ lọc gói tin cho chip STM32
- Status: Cờ báo trạng thái hiện tại của đối tượng.
- CanRxBuffer: Buffer chứa dữ liệu mỗi frame ở tầng vật lý nhận đc. (Vì ở tầng vật lý với gói tin chuẩn giao tiếp CAN chỉ gửi được tối đa 8 Bytes)
- CanRxLen: Số lượng bytes nhận được ở tầng vật lý.

2.1.2 CanTP_Packet

```
typedef struct CanTP_Packet
{
    uint16_t Len;
    uint8_t* Data;
} CanTP_Packet;
```

Hình 2.2 Cấu trúc chứa gói tin nhận và gửi

Trong đó

- Data: Dữ liệu nhận được hoặc sẽ gửi đi.
- Len: Độ dài dữ liệu sẽ gửi đi hoặc được nhận vào.

2.1.3 Khác

```
typedef struct CanTP_SFFrameTypedef
{
    uint8_t SF_DL;
    uint8_t* Data;
}CanTP_SFFrameTypedef;

typedef struct CanTP_FFFrameTypedef
{
    uint16_t FF_DL;
    uint8_t* Data;
}CanTP_FFFrameTypedef;

typedef struct CanTP_CFFrameTypedef
{
    uint8_t SN;
    uint8_t* Data;
    uint8_t DataLen;
}CanTP_CFFrameTypedef;

typedef struct CanTP_FCFrameTypedef
{
    uint8_t FS;
    uint8_t BS;
    uint8_t STmin;
}CanTP_FCFrameTypedef;
```

Hình 2.3: Cấu trúc các khung truyền

Ngoài ra dựa vào đặc tả định dàng từng gói tin nhóm định nghĩa các cấu trúc sau nhằm dễ dàng quản lý dữ liệu.

- CanTP_SFFrameTypedef: Cấu trúc chứa dữ liệu gói tin SingleFrame
- CanTP_FFFrameTypedef: Cấu trúc chứa dữ liệu gói tin FirstFrame
- CanTP_CFFrameTypedef: Cấu trúc chứa dữ liệu gói tin ConsequenceFrame
- CanTP_FCFrameTypedef: Cấu trúc chứa dữ liệu gói tin FlowControl

2.2 Các hàm hiện thực

2.2.1 Hàm khởi tạo

```
HAL_StatusTypeDef CanTP_Init(CanTP_HandleTypedef* hCanTP)
{
    HAL_CheckERR(HAL_CAN_ConfigFilter(hCanTP->hcan, &(hCanTP->CanFilter)));
    if(hCanTP->CanRxFifo == CAN_FilterFIFO0)
    {
        HAL_CheckERR(HAL_CAN_ActivateNotification(hCanTP->hcan, CAN_IT_RX_FIFO0_MSG_PENDING));
    }
    else
    {
        HAL_CheckERR(HAL_CAN_ActivateNotification(hCanTP->hcan, CAN_IT_RX_FIFO1_MSG_PENDING));
    }
    HAL_CAN_ActivateNotification(hCanTP->hcan, CAN_IT_ERROR | CAN_IT_LAST_ERROR_CODE);
    HAL_CheckERR(HAL_CAN_Start(hCanTP->hcan));
    return HAL_OK;
}
```

Đầu tiên ta gọi đến hàm cấu hình bộ lọc nhằm mục đích cho vi xử lý biết ta sẽ nghe và nhận những CAN ID nào. Tiếp đó ta tiến hành bật ngắt nhận thông qua hàm HAL_CAN_ActivateNotification. Khi có dữ liệu đến FIFO sẽ xảy ra ngắt. Kế đến ta khởi động ngoại vi CAN trên vi điều khiển qua hàm HAL_CAN_Start.

2.2.2 Hàm gửi CanTP_Transmit

```
HAL_StatusTypeDef CanTP_Transmit(CanTP_HandleTypedef* hCanTP, CanTP_Packet Packet)
{

    if(Packet.Len <= 7)
    {
        HAL_CheckERR(CanTP_TxSingleFrame(hCanTP, Packet));
    }
    else
    {
        HAL_CheckERR(CanTP_TxMultiFrame(hCanTP, Packet));
    }
    return HAL_OK;
}
```

Hàm gửi ở tầng mạng sẽ bao gồm 2 hàm là hàm TxSingleFrame sẽ gửi đi gói tin SF nếu Data < 7 Bytes. Và hàm TxMultiFrame sẽ cắt gói tin ra thành nhiều gói tin nhỏ và gửi đi theo sơ đồ hình dưới nếu độ dài gói tin lớn hơn 7 Bytes.

Message Type	CAN ID	CAN Frame Data Field					
		Byte 0			Byte 1	Byte 2	Byte 3-7
		Bit 7-4	Bit 3	Bit 2-0			
SingleFrame (SF)	CAN ID	00b	SF_DL		Data		
FirstFrame (FF)	CAN ID	01b	FF_DL			Data	
ConsecutiveFrame (CF)	CAN ID	10b	SN		Data		
FolwControl (FC)	CAN ID	11b	FS		BS	STmin	-

Hình 2.4: Bảng cấu trục các gói tin mạng CAN

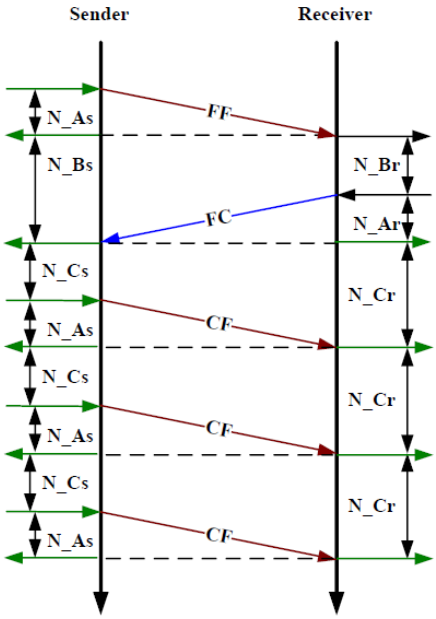
Hàm gửi 1 Frame CanTP_TxSingleFrame:

```
static HAL_StatusTypeDef CanTP_TxSingleFrame(CanTP_HandleTypedef* hCanTP, CanTP_Packet Packet)
{
    CanTP_SFFrameTypedef MsgFrame;
    MsgFrame.SF_DL = Packet.Len;
    MsgFrame.Data = Packet.Data;
    HAL_CheckERR(CanTP_TxSF(hCanTP, MsgFrame));
    return HAL_OK;
}

static HAL_StatusTypeDef CanTP_TxSF(CanTP_HandleTypedef* hCanTP, CanTP_SFFrameTypedef FrameInfo)
{
    uint8_t Payload[8] = {0};
    if(FrameInfo.SF_DL > 7) return HAL_ERROR;
    Payload[0] = CANTP_SF_ID | FrameInfo.SF_DL;
    memcpy(Payload + 1, FrameInfo.Data, FrameInfo.SF_DL);
    memset(Payload + 1 + FrameInfo.SF_DL, 0x55, 8 - FrameInfo.SF_DL - 1);
    return CAN_Transmit(hCanTP->hcan, hCanTP->CanTxID, Payload, 8);
}
```

Hàm CanTP_TxSF làm nhiệm vụ đóng gói lại các dữ liệu theo format của khung truyền loại SF sau đó gọi đến hàm CAN_Trasmit để gửi dữ liệu với ID gói tin là CanTxID. Vì dựa theo đặc tả của lớp liên kết dữ liệu mỗi khung truyền trên mạng CAN sẽ truyền 8 bytes dữ liệu, những byte trống không sử dụng sẽ có giá trị là 0x55.

Vì ở lớp vật lý với mỗi gói tin CAN chỉ gửi được tối đa 8 bytes dữ liệu nên sẽ cần cơ chế cắt ghép các gói tin có kích thước lớn ra thành nhiều gói tin nhỏ để gửi. Quy trình gửi như hình dưới.



Hình 2.5: Sơ đồ giao tiếp mạng CAN ở lớp mạng

Hàm CanTP_TxMultiFrame làm nhiệm vụ chia nhỏ các gói tin ra và gửi đi.

```
static HAL_StatusTypeDef CanTP_TxMultiFrame(CanTP_HandleTypedef* hCanTP, CanTP_Packet Packet)
{
    CanTP_FFFrameTypedef FFFrame = {0};
    CanTP_CFFrameTypedef CFFrame = {0};
    CanTP_FCFrameTypedef FCFrame = {0};

    FFFrame.FF_DL = Packet.Len;
    FFFrame.Data = Packet.Data;
    HAL_CheckERR(CanTP_TxFF(hCanTP, FFFrame));
    HAL_CheckERR(CanTP_WaitFC(hCanTP, &FCFrame, 200));
    uint16_t DataSent = 6;
    while (DataSent < Packet.Len)
    {
        CFFrame.Data = Packet.Data + DataSent;
        CFFrame.DataLen = (Packet.Len - DataSent < 7) ? Packet.Len - DataSent : 7;
        CFFrame.SN++;
        osDelay(5);
//      SyncPrintf("CF Tx Len %d, SN %d \r\n", CFFrame.DataLen, CFFrame.SN);
        HAL_CheckERR(CanTP_TxCF(hCanTP, CFFrame));
        DataSent += CFFrame.DataLen;
    }
    return HAL_OK;
}
```

Trước khi gửi dữ liệu trên mạng CAN ta sẽ có một khoảng Delay nhỏ bởi vì ta đang truyền và nhận trên cùng một board với 2 luồng song song do đó cần khoảng delay để nhường tài nguyên cho luồng nhận đảm bảo luồng nhận đã sẵn sàng để nhận.

- Đầu tiên ta sẽ gửi 6 bytes đầu của dữ liệu thông qua khung truyền FF.
- Sau đó ta đợi bên nhận gửi một số thông tin cấu hình thông qua khung truyền FC.
- Sau khi nhận được khung truyền FC ta tiến hành gửi tiếp những dữ liệu còn lại thông qua khung truyền CF.

```
static HAL_StatusTypeDef CanTP_TxFF(CanTP_HandleTypedef* hCanTP, CanTP_FFFrameTypedef FrameInfo)
{
    uint8_t Payload[8] = {0};
    Payload[0] = CANTP_FF_ID | (FrameInfo.FF_DL>>8);
    Payload[1] = FrameInfo.FF_DL;
    memcpy(Payload + 2, FrameInfo.Data, 6);
    return CAN_Transmit(hCanTP->hcan, hCanTP->CanTxID, Payload, 8);
}

static HAL_StatusTypeDef CanTP_TxCF(CanTP_HandleTypedef* hCanTP, CanTP_CFFrameTypedef FrameInfo)
{
    uint8_t Payload[8] = {0};
    Payload[0] = CANTP_CF_ID | FrameInfo.SN;
    memcpy(Payload + 1, FrameInfo.Data, FrameInfo.DataLen);
    if(FrameInfo.DataLen < 7)
    {
        memset(Payload + 1 + FrameInfo.DataLen, 0x55, 7 - FrameInfo.DataLen);
    }
    return CAN_Transmit(hCanTP->hcan, hCanTP->CanTxID, Payload, 8);
}
static HAL_StatusTypeDef CanTP_WaitFC(CanTP_HandleTypedef* hCanTP, CanTP_FCFrameTypedef* FrameInfo, uint32_t Timeout)
{
    CanTP_WaitData(hCanTP,Timeout);
    uint8_t RcvData[8] = {0};
    memcpy(RcvData, hCanTP->CanRxBuffer, hCanTP->CanRxLen);
    memset(hCanTP->CanRxBuffer, 0, 8);
    hCanTP->CanRxLen = 0;
    if((RcvData[0]& 0xf0) != CANTP_FC_ID) return HAL_ERROR;
    FrameInfo->BS = RcvData[1];
    FrameInfo->STmin = RcvData[2];
    FrameInfo->FS = RcvData[0] &0x0f;
    return HAL_OK;
}
```

2.2.3 Hàm nhận CanTP_Receive

```
HAL_StatusTypeDef CanTP_Receive(CanTP_HandleTypedef* hCanTP, CanTP_Packet* Packet, uint32_t Timeout)
{
    HAL_CheckERR(CanTP_WaitData(hCanTP,Timeout));
    uint8_t FrameType = CanTP_GetFrameType(hCanTP->CanRxBuffer);
    if(FrameType == CANTP_SF_ID)
    {
        uint8_t* RcvData = hCanTP->CanRxBuffer;
        Packet->Len = RcvData[0] & 0x0f;
        Packet->Data = pvPortMalloc(Packet->Len);
        memcpy(Packet->Data, RcvData + 1, Packet->Len);
    }
    else if(FrameType == CANTP_FF_ID)
    {
        CanTP_CFFrameTypedef CFFrame = {0};
        CanTP_FCFrameTypedef FCFrame = {0};

        uint8_t* RcvData = hCanTP->CanRxBuffer;
        uint16_t FF_DL = ((RcvData[0] & 0x0f)<<8) | RcvData[1] ;
        Packet->Len = FF_DL;
        Packet->Data = pvPortMalloc(Packet->Len + 7);
        memcpy(Packet->Data, RcvData + 2, 6);
        FCFrame.BS = 8;
        FCFrame.FS = 0;
        FCFrame.STmin = 25;
        CanTP_TxFC(hCanTP, FCFrame);

        uint16_t DataReceived = 6;
        uint8_t CurrentSN = 1;
        CFFrame.Data = (uint8_t*)pvPortMalloc(8);
        while(DataReceived < FF_DL)
        {
            HAL_CheckERR(CanTP_WaitCF(hCanTP, &CFFrame, Timeout));
            if(CurrentSN != CFFrame.SN ) return HAL_ERROR;
            memcpy(Packet->Data + DataReceived, CFFrame.Data, 7);
            DataReceived += CFFrame.DataLen;
            CurrentSN++;
        }
        vPortFree(CFFrame.Data);
```

```
            vPortFree(CFFrame.Data);
            Packet->Data[Packet->Len] = 0;
            SyncPrintf("Done Rcv \r\n");
        }
        else
        {
            return HAL_ERROR;
        }
        return HAL_OK;

}
```

Nhóm thiết kế hàm nhận theo cơ chế Polling. Trong hàm nhận đầu tiên chương trình sẽ đợi khung dữ liệu đầu tiên được gửi đến.

- Nếu khung dữ liệu đầu thuộc loại SF ta chỉ cần lấy dữ liệu ra và kết thúc hàm nhận
- Nếu khung dữ liệu đầu tiên nhân được thuộc loại FF ta tiến hành gửi tín hiệu FC và tiến vào chu trình nhận dữ liệu đến khi đủ dữ liệu được thông báo trong khung truyền FF.
- Nếu không thuộc hai loại trên hàm sẽ trả về lỗi.

```
static HAL_StatusTypeDef CanTP_WaitData(CanTP_HandleTypedef* hCanTP, uint32_t Timeout)
{
    int Status = osEventFlagsWait(hCanTP->Status, CANTP_RCV_DoneFlag, osFlagsWaitAll, Timeout);
    if (Status == osFlagsErrorTimeout) return HAL_TIMEOUT;
    else if( Status < 0 )return HAL_ERROR;
    return HAL_OK;
}

HAL_StatusTypeDef CanTP_RcvCBHandler(CanTP_HandleTypedef* hCanTP, uint8_t* Data, uint8_t DataLen)
{
    memcpy(hCanTP->CanRxBuffer, Data, DataLen);
    hCanTP->CanRxLen = DataLen;
    osEventFlagsSet(hCanTP->Status, CANTP_RCV_DoneFlag);
    return HAL_OK;
}
```

Để đợi dữ liệu nhận vào nhóm sử dụng cơ chế EventFlagWait của RTOS kết hợp với ngắt nhận của CAN trong STM32.

Khi có dữ liệu đến sẽ gọi đến hàm ngắt RcvCBHandler. Trong này ta sẽ tiến hành đọc các dữ liệu nhận được và phát một cờ tín hiệu để luồng đang nhận dữ liệu được mở khóa và xử lý các tín hiệu đó.

3 Hiện thực dịch vụ $27H Security Access ở lớp chẩn đoán:

3.1 Xử lý trên ECU

```
void ECUTask_Handler(void *argument)
{
    HAL_StatusTypeDef Status;
    Status = ECU_Init();
    SyncPrintf("ECU Init State %ld\r\n", (uint32_t) Status);
    while(1)
    {
        ECU_Loop();
    }
}
```

3.1.1 Khởi tạo

```
#ifndef ECUDEFINE_H_
#define ECUDEFINE_H_

#define ECU_TxID 0x712
#define ECU_RxID 0x7A2

#define ECU_ReadData_ByID_SID   0x22
#define ECU_WriteData_ByID_SID  0x2E
#define ECU_SecurityAccess_SID  0x27

#define ECU_ReadData_ByID_RespSID   0x62
#define ECU_WriteData_ByID_RespSID  0x6E
#define ECU_SecurityAccess_RespSID  0x67


#define ECU_AnalogRead_DID  0xE015
#define ECU_Write_DID       0xF002

#endif /* ECUDEFINE_H_ */
#define ECU_CANRxFifo       CAN_FILTER_FIFO0
#define ECU_CANFilterBank   10
#define ECU_HCAN        hcan1
#define ECU_HADC        hadc1

extern ADC_HandleTypeDef ECU_HADC;
extern CAN_HandleTypeDef ECU_HCAN;
extern osEventFlagsId_t ECUCanTPStatusFlagHandle;
extern RNG_HandleTypeDef hrng;

CanTP_HandleTypedef ECUCanTP ={0};

uint16_t ADCVal[1]={0};

HAL_StatusTypeDef ECU_Init()
{
    ECUCanTP.CanFilter.FilterActivation = CAN_FILTER_ENABLE;
    ECUCanTP.CanFilter.FilterIdHigh = ECU_RxID << 5;
    ECUCanTP.CanFilter.FilterMode = CAN_FILTERMODE_IDLIST;
    ECUCanTP.CanFilter.FilterScale = CAN_FILTERSCALE_16BIT;
    ECUCanTP.CanFilter.FilterBank = ECU_CANFilterBank;
    ECUCanTP.CanFilter.FilterFIFOAssignment = ECU_CANRxFifo;
    ECUCanTP.CanFilter.SlaveStartFilterBank = 15;
//  ECUCanTP.CanRxFifo = ECU_CANRxFifo;
    ECUCanTP.hcan = &ECU_HCAN;
    ECUCanTP.Status = ECUCanTPStatusFlagHandle;
    ECUCanTP.CanRxID = ECU_RxID;
    ECUCanTP.CanTxID = ECU_TxID;
    HAL_CheckERR(CanTP_Init(&ECUCanTP));
    HAL_CheckERR(HAL_ADC_Start_DMA(&ECU_HADC, (uint32_t*)ADCVal, 1));
    return HAL_OK;
}
```

Ta khởi tạo một đối tượng ECUCanTP để quản lý quá trình giao tiếp truyền nhận dữ liệu theo lớp mạng. Với các cấu hình như sau. ECU sẽ sử dụng CAN1 trên vi điều khiển để giao tiếp. Sử dụng Fifo0 trong CAN1 để nhận các gói tin. Các gói tin gửi đi trong mạng CAN sẽ có ID là 0x712 và sẽ nhận và xử lý những gói tin có ID là 0x7A2.

```
void HAL_CAN_RxFifo0MsgPendingCallback(CAN_HandleTypeDef *hcan)
{
    if(hcan->Instance == ECU_HCAN.Instance)
    {
        CAN_RxHeaderTypeDef RcvHeader;
        uint8_t RcvData[8];
        HAL_CAN_GetRxMessage(hcan, ECU_CANRxFifo, &RcvHeader, RcvData);
        if(RcvHeader.StdId == ECUCanTP.CanRxID)
        {
            CanTP_RcvCBHandler(&ECUCanTP, RcvData, RcvHeader.DLC);
        }
    }

}
```

Ngoài ra ta cài đặt hàm ngắt Callback xử lý các gói tin nhận được của ECUCanTP.

3.1.2 Vòng lặp

```
HAL_StatusTypeDef ECU_Loop()
{
    CanTP_Packet DataPacket ={0};
    HAL_StatusTypeDef Status = CanTP_Receive(&ECUCanTP, &DataPacket, osWaitForever);
    SyncPrintf("ECU: RcvData ");
    for(int i = 0; i <DataPacket.Len; ++i)
    {
        SyncPrintf("0x%.2x ", DataPacket.Data[i]);
    }
    SyncPrintf("\r\n");

    if(Status == HAL_OK)
    {
        uint8_t SID = DataPacket.Data[0];
        if(SID == ECU_ReadData_ByID_SID)
        {
            ECU_ReadData_Service(DataPacket.Data + 1, DataPacket.Len - 1); // Remove First Byte SID
        }
        else if(SID == ECU_WriteData_ByID_SID)
        {
            ECU_WriteData_Service(DataPacket.Data + 1, DataPacket.Len - 1);
        }
        else if(SID == ECU_SecurityAccess_SID)
        {
            SyncPrintf("ECU: Security Access \r\n");
            ECU_SecurityAccess_Service(DataPacket.Data + 1, DataPacket.Len - 1);

        }
        vPortFree(DataPacket.Data);
    }

    return HAL_OK;
}
```

Chương trình sẽ tiến hành đợi dữ liệu đến. Khi có dữ liệu đến ta lấy ra Bytes đầu tiên của gói tin để kiểm tra gói tin đó là dịch vụ gì nếu là dịch vụ SecurityAccess $27H sẽ gọi đến hàm ECU_SecurityAccess_Service nhắm vào chu trình thực hiện dịch vụ đó.

```
static HAL_StatusTypeDef ECU_SecurityAccess_Service(uint8_t* Data, uint8_t DataLen)
{
    uint32_t SeedValue;
    uint8_t SubFuncID = Data[0];
    HAL_StatusTypeDef Status;
    if(SubFuncID == 0x01)
    {
        HAL_RNG_GenerateRandomNumber(&hrng, &SeedValue);
        SyncPrintf("ECU: Generate Seed %ld \r\n", SeedValue);
        HAL_CheckERR(ECU_SecurityAccess_RespSeed(SubFuncID, SeedValue));
        uint32_t ExpectedKey = Key_Calculate(SeedValue);
        SyncPrintf("ECU: Expected Key %ld \r\n", ExpectedKey);
        Status = ECU_SecurityAccess_CheckKey(ExpectedKey, 500);
        return Status;
    }
    return HAL_OK;
}
```

Với SubFuncID là 0x01 sẽ là lệnh RequestSeed SecurityAccess Service. Khi đó ta sẽ sinh ra một số để làm seed cho thuật toán mã hóa Seed-Key. Để sinh ra seed nhóm sẽ sử dụng ngoại vi RNG (Random Number Generator) để tạo một số ngẫu nhiên 32bit là seed cho dịch vụ Security Access. Sau khi tính toán seed sẽ phản hồi về Tester.

```
static HAL_StatusTypeDef ECU_SecurityAccess_RespSeed(uint8_t SubFunctionID, uint32_t SecuritySeed)
{
    uint8_t Buffer[6] = {0};
    Buffer[0] = ECU_SecurityAccess_RespSID;
    Buffer[1] = SubFunctionID;
    uint32_t Seed_BE = __htonl(SecuritySeed);
    memcpy(Buffer+2, &Seed_BE, 4);
    CanTP_Packet RespPacket = {
            .Data = Buffer,
            .Len  = sizeof(Buffer)
    };
    osDelay(10);
    HAL_CheckERR(CanTP_Transmit(&ECUCanTP, RespPacket));
    return HAL_OK;
}
```

Từ giá trị seed vừa có được sẽ ECU sẽ tính toán ra Key. Ở đây nhóm sử dụng thuật toán crc32 để sinh Key. Sau khi có Key ECU sẽ tiến hành đợi trong vòng 500ms để đợi gói tin Security Access – SendKey từ Tester.

```
static HAL_StatusTypeDef ECU_SecurityAccess_CheckKey(uint32_t ExpectedKey, uint32_t Timeout)
{
    CanTP_Packet RcvPacket ={0};
    HAL_CheckERR(CanTP_Receive(&ECUCanTP, &RcvPacket, Timeout));
    uint8_t SID = RcvPacket.Data[0];
    uint8_t SubFunctID = RcvPacket.Data[1];
    uint32_t RcvKey_BE, RcvKey;

    memcpy(&RcvKey_BE, RcvPacket.Data + 2, 4);
    vPortFree(RcvPacket.Data);

    if(SID != ECU_SecurityAccess_SID) return HAL_ERROR;
    if(SubFunctID == 0x02)
    {
        RcvKey = __ntohl(RcvKey_BE);
        if(RcvKey == ExpectedKey)
        {
            ECU_SecurityUnlock();
            ECU_SecurityAccess_RespKey(SubFunctID);
        }
        else
        {
            ECU_SecurityLock();
        }
    }
    else
    {
        return HAL_ERROR;
    }
    return HAL_OK;
}
```

Sau khi nhận được gói tin SendKey ECU sẽ tiến hành kiểm tra giá trị Key mình nhận được và giá trị Key mình tính được có trùng nhau hay không. Nếu trùng nhau sẽ tiến hay mở khóa ECU và gửi về gói tin phần hồi thành công đến Tester. Nếu khác sẽ tính là truy cập sai và thực hiện việc khóa ECU

```
static HAL_StatusTypeDef ECU_SecurityUnlock()
{
    SyncPrintf("ECU: Security Unlock \r\n");
    HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, GPIO_PIN_SET);

    return HAL_OK;
}
static HAL_StatusTypeDef ECU_SecurityLock()
{
    SyncPrintf("ECU: Security Lock \r\n");
    HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, GPIO_PIN_RESET);
    return HAL_OK;
}
```

Nhóm sẽ thực hiện việc bật và tắt LED1 trên board thực hành để mô tả trạng thái của board là có khóa hay không. Nếu LED1 được bật tức ECU đang được mở khóa và tắt nếu ECU đang ở trạng thái khóa.

3.2 Xử lý trên Tester

```
void TesterTask_Handler(void *argument)
{
    HAL_StatusTypeDef Status;
    Status = Tester_Init();
    SyncPrintf("Tester Init State %ld\r\n", (uint32_t) Status);
    while(1)
    {
        Tester_Loop();
    }
}
```

3.2.1 Khởi tạo

```
#ifndef TESTERDEFINE_H_
#define TESTERDEFINE_H_

#define Tester_TxID 0x7A2
#define Tester_RxID 0x712

#define Tester_ReadData_ByID_SID    0x22
#define Tester_WriteData_ByID_SID   0x2E
#define Tester_SecurityAccess_SID   0x27

#define Tester_ReadData_ByID_RespSID    0x62
#define Tester_WriteData_ByID_RespSID   0x6E
#define Tester_SecurityAccess_RespSID   0x67

#define Tester_SecurityAccess_ReqSeedID 0x01
#define Tester_SecurityAccess_SendKeyID 0x02

#define Tester_AnalogRead_DID   0xE015
#define Tester_Write_DID        0xF002

#endif /* TESTERDEFINE_H_ */


#define Tester_CANRxFifo        CAN_FILTER_FIFO1
#define Tester_CANFilterBank    5
#define Tester_HCAN     hcan2

extern CAN_HandleTypeDef Tester_HCAN;
extern osEventFlagsId_t TesterCanTPStatusFlagHandle;
CanTP_HandleTypedef TesterCanTP ={0};

uint8_t Joystick_Pressed = 0;
uint8_t BTN0_IsPress = 0;
uint8_t BTN1_IsPress = 0;

HAL_StatusTypeDef Tester_Init()
{
    TesterCanTP.CanFilter.FilterActivation = CAN_FILTER_ENABLE;
    TesterCanTP.CanFilter.FilterIdHigh = Tester_RxID << 5;
    TesterCanTP.CanFilter.FilterMode = CAN_FILTERMODE_IDLIST;
    TesterCanTP.CanFilter.FilterScale = CAN_FILTERSCALE_32BIT;
    TesterCanTP.CanFilter.FilterBank = Tester_CANFilterBank;
    TesterCanTP.CanFilter.FilterFIFOAssignment = Tester_CANRxFifo;
    TesterCanTP.CanFilter.SlaveStartFilterBank = 0;
    TesterCanTP.hcan = &Tester_HCAN;
    TesterCanTP.Status = TesterCanTPStatusFlagHandle;
    TesterCanTP.CanRxID = Tester_RxID;
    TesterCanTP.CanTxID = Tester_TxID;
//  osDelay(10);
    CanTP_Init(&TesterCanTP);
    return HAL_OK;
}
void HAL_CAN_RxFifo1MsgPendingCallback(CAN_HandleTypeDef *hcan)
{

    if(hcan->Instance == Tester_HCAN.Instance)
    {
        CAN_RxHeaderTypeDef RcvHeader;
        uint8_t RcvData[8];
        HAL_CAN_GetRxMessage(&Tester_HCAN, Tester_CANRxFifo, &RcvHeader, RcvData);
        if(RcvHeader.StdId == TesterCanTP.CanRxID)
        {
            CanTP_RcvCBHandler(&TesterCanTP, RcvData, RcvHeader.DLC);
        }
    }

}
```

Tương tự như khi ta khởi tạo ECU. Ta sẽ khởi tạo Tester sẽ sử dụng CAN2 trên vi điều khiển để giao tiếp. Dùng FIFO1 để đọc và nhận gói tin. Các gói tin Tester gửi đi sẽ có ID là 0x7A2 và sẽ nhận vào và xử lý các gói tin có ID 0x712.

3.2.2 Kịch bản kiểm tra.

Sau khi Tester request Seed từ ECU, Tester sẽ tính toán ra giá trị Key.

- Tester gửi Key đúng ECU sẽ được mở khóa và đèn LED1 sẽ sáng.
- Tester gửi Key sai ECU sẽ bị khóa lại đèn LED1 không sáng.

3.2.3 Chương trình xử lý.

```
HAL_StatusTypeDef Tester_Loop()
{
//  Tester_ReadDataByID();
//  osDelay(1000);
//   Tester_WriteDataByID();
    if(BTN0_IsPress == 1)
    {
        SyncPrintf("------------------------------ \r\n");

        BTN0_IsPress = 0;
        SyncPrintf("Security Access Press 0: False Key, Press 1: True Key \r\n");
        while(BTN0_IsPress == 0 && BTN1_IsPress == 0);
        uint8_t SendTrueKey = 0;
        if(BTN0_IsPress)
        {
            SendTrueKey = 0;
            SyncPrintf("Tester: Send False Key \r\n");
        }else if(BTN1_IsPress)
        {
            SendTrueKey = 1;
            SyncPrintf("Tester Send True Key \r\n");
        }
        Tester_SecurityAccess(SendTrueKey);
        BTN0_IsPress = 0;
        BTN1_IsPress = 0;
    }
    return HAL_OK;
}
```

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    if(GPIO_Pin == BTN0_Pin)
    {
        BTN0_IsPress = 1;
    }
    if(GPIO_Pin == BTN1_Pin)
    {
        BTN1_IsPress = 1;
    }
    if(GPIO_Pin == Joystick_A_Pin)
    {
        Joystick_Pressed = 1;
    }
    if(GPIO_Pin == Joystick_B_Pin)
    {
        Joystick_Pressed = 2;
    }
    if(GPIO_Pin == Joystick_Mid_Pin)
    {
        Joystick_Pressed = 3;
    }
}
```

Nhóm sử dụng Interrupt để đọc các nút nhấn. Khi nhấn nút sẽ gọi đến hàm xử lý ngắt. Đối ứng với nút ta nhấn sẽ bật cờ báo đã nhấn nút đó lên.

Bắt đầu chu trình ta sẽ nhấn nút 0 (WakeupBTN trên kit TH) bắt đầu chương trình kiểm tra dịch vụ Security Access.

Nếu ta tiếp tục nhấn nút 0 một lần nữa Tester sẽ cố tình gửi một Key sai ECU.

Nếu ta nhấn nút 1 (USER button trên kit TH) chương trình sẽ tính toán Key dựa trên Seed và gửi đến ECU.

```
static HAL_StatusTypeDef Tester_SecurityAccess(uint8_t IsSendTrueKey)
{
    uint32_t SecuritySeed;
    osDelay(10);
    HAL_CheckERR(Tester_SecurityAccess_RequestSeed(&SecuritySeed));
    SyncPrintf("Tester: Seed Receive %ld\r\n", SecuritySeed);
    uint32_t SecurityKey = 0;
    if(IsSendTrueKey == 1)
    {
        SecurityKey = Key_Calculate(SecuritySeed);
    }
    else
    {
        SecurityKey = 12346789;
    }
    osDelay(10);
    SyncPrintf("Tester Send Key %ld \r\n", SecurityKey);
    Tester_SecurityAccess_ValidKey(Tester_SecurityAccess_SendKeyID, SecurityKey);
    return HAL_OK;
}
```

Để bắt đầu dịch vụ Security Access ta sẽ gửi gói tin SecurityAccess – Request Seed đến cho ECU. Sau đó ta tiến hành đợi gói tin phản hồi từ ECU về trong gói tin sẽ chứa SecuritySeed.

```
static HAL_StatusTypeDef Tester_SecurityAccess_RequestSeed(uint32_t* Seed)
{
    uint8_t Buffer[] = {Tester_SecurityAccess_SID, Tester_SecurityAccess_ReqSeedID};
    CanTP_Packet ReqPacket = {
            .Data = Buffer,
            .Len = 2
    };
//  HAL_StatusTypeDef Status =0;
    osDelay(10);
    HAL_CheckERR(CanTP_Transmit(&TesterCanTP, ReqPacket));
    CanTP_Packet RespPacket ={0};
    HAL_CheckERR(CanTP_Receive(&TesterCanTP, &RespPacket,500));
    uint8_t SID = RespPacket.Data[0];
    uint8_t SubFuncID = RespPacket.Data[1];
    uint32_t Seed_BE;
    memcpy(&Seed_BE, RespPacket.Data + 2, 4);
    vPortFree(RespPacket.Data);
    if(SID == Tester_SecurityAccess_RespSID && SubFuncID == Tester_SecurityAccess_ReqSeedID)
    {
        *Seed = __ntohl(Seed_BE);
        return HAL_OK;
    }
    else
    {
        SyncPrintf("Tester: Request Seed Failed \r\n");
        return HAL_ERROR;
    }

}
```

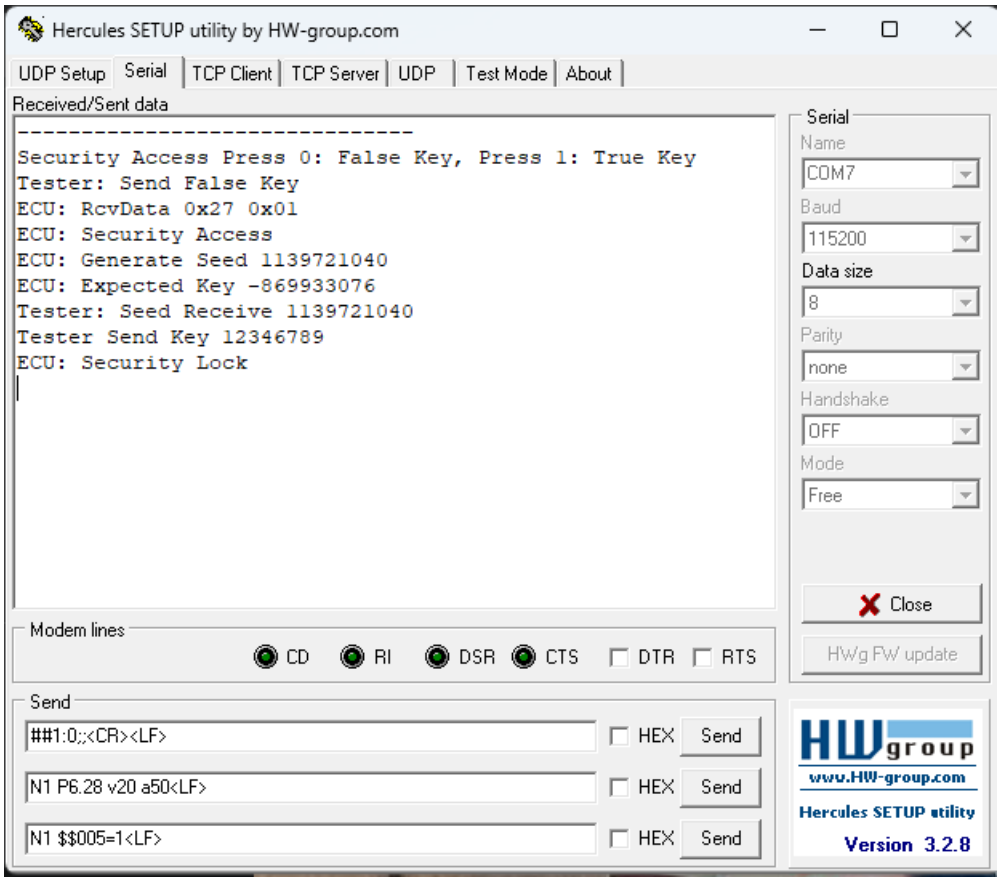
Sau khi có Seed ECU sẽ tiến hành tính toán ra Key cũng theo thuật toán CRC32 giống ECU. Tiếp đến Tester gửi giá trị Key mình tính toán được đến cho ECU kiểm tra thông qua dịch vụ Security Access – SendKey.

```
static HAL_StatusTypeDef Tester_SecurityAccess_ValidKey(uint8_t SubFunctionID, uint32_t Key)
{
    uint8_t Buffer[6] = {0};
    Buffer[0] = Tester_SecurityAccess_SID;
    Buffer[1] = SubFunctionID;
    uint32_t Key_BE = __htonl(Key);
    memmove(Buffer+2, &Key_BE, 4);
    CanTP_Packet SendPacket = {
            .Data = Buffer,
            .Len = sizeof(Buffer)
    };
    osDelay(10);
    HAL_CheckERR(CanTP_Transmit(&TesterCanTP, SendPacket));

    CanTP_Packet RespPacket ={0};
    HAL_CheckERR(CanTP_Receive(&TesterCanTP, &RespPacket, 500));
    uint8_t RespSID = RespPacket.Data[0];
    uint8_t RespSubFuncID = RespPacket.Data[1];
    if(RespSID == Tester_SecurityAccess_RespSID && RespSubFuncID == SubFunctionID)
    {
        SyncPrintf("Tester: Security Access Success \r\n");
        return HAL_OK;
    }
    else
    {
        SyncPrintf("Tester: Security Access Failed \r\n");
        return HAL_ERROR;
    }
}
```

4 Kết quả

4.1 Trường hợp gửi Tester gửi lại Key sai.

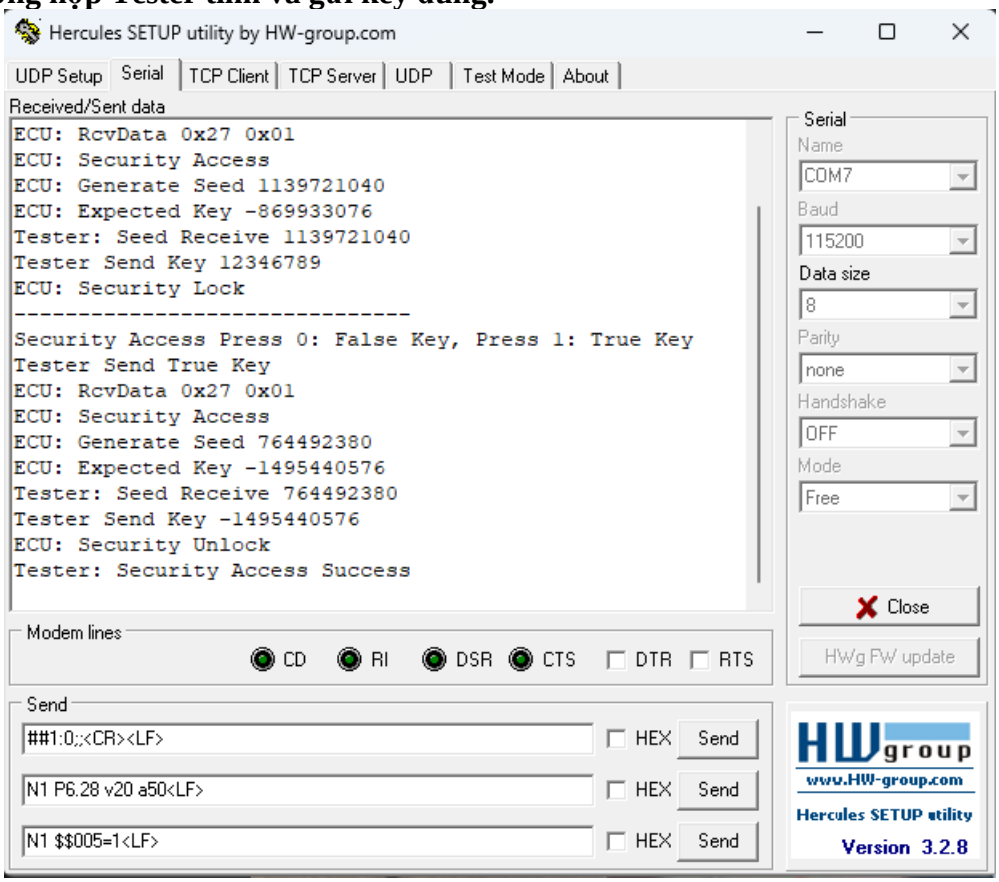


Hình 4.1: Trường hợp Tester gửi Key sai

Tester gửi gói tin 0x27 0x01 tức Security Access – Request Seed đến ECU và ECU báo đã nhận được gói tin và tiến hành vào dịch vụ Security Access. ECU sinh ra seed 1139721040 và tính được Key là -869933076.

- Tester báo là nhận được Seed với giá trị 1139721040 và vì ta đang để Tester gửi Key sai nên sẽ gửi về ECU key là 123456789.
- ECU nhận được Key là 123456789 khác với key ECU tính được nên ECU sẽ bị khóa.

4.2 Trường hợp Tester tính và gửi key đúng.



Hình 4.2: Trường hợp Tester gửi key đúng

Tương tự như trên:

- Tester gửi đến ECU gói tin Security Access – Request Seed.
- ECU sinh ra seed với giá trị 764492380
- ECU tính ra Key với giá trị -1495440576
- Tester nhận được seed với giá trị 764492380
- Tester tính toán và gửi key đến ECU với giá trị -1495440576
- Vì giá trị Key ECU tính toán được và giá trị ECU nhận được giống nhau nên ECU tiến hành mở khóa. Và phản hồi gói tin mở khóa thành công đến Tester.
- Tester nhận được gói tin phản hồi mở khóa thành công.