2019 – 2 임베디드 시스템 설계 및 실험 목요일 분반

텀 프로젝트 결과 보고서

지능형 방범 시스템

7조

안도현

이선영

이승준

최준혁

1. 목표

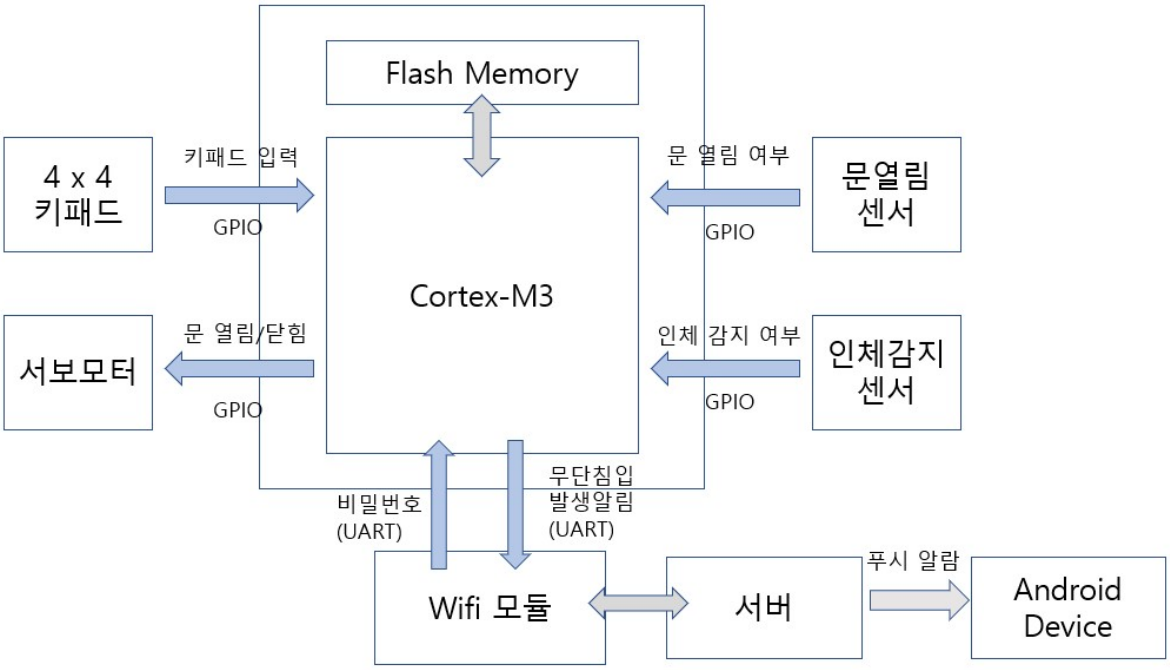
- 비밀번호를 통해 문을 열고 닫을 수 있는 도어락 시스템을 만든다.

- 문 열림 상태일 때 저장된 비밀번호를 변경할 수 있도록 한다.

- 문 잠김 상태(방범 모드)일 때 센서를 이용해 무단 침입을 감지하고 대응조치를 취하도록 한다.

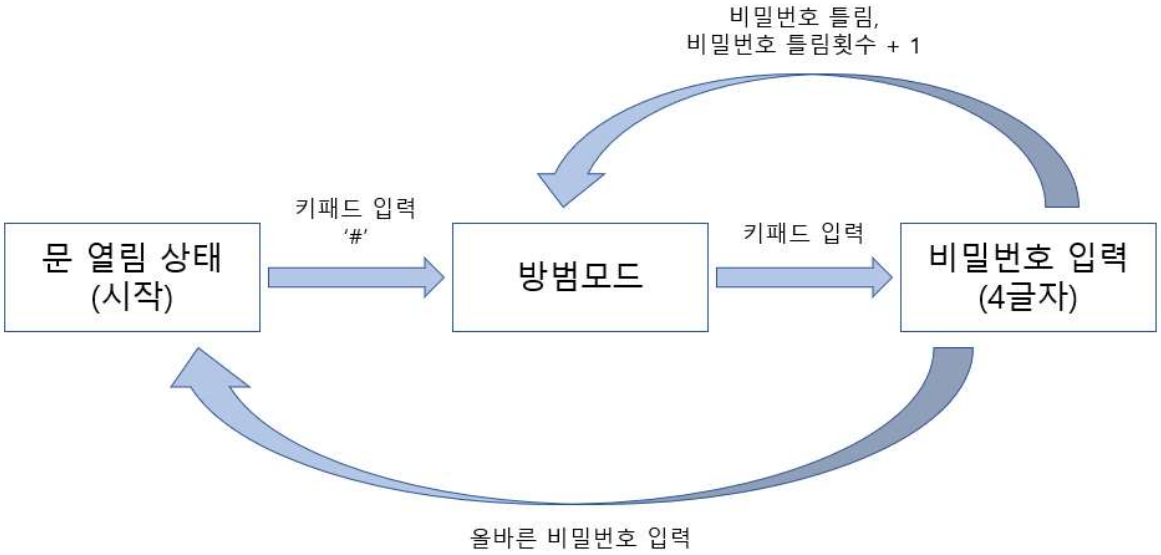
- 임시로 발급되어 문을 열수 있고, 일정 시간이 지나면 만료되는 임시 비밀번호 기능을 만든다.

2. 시스템 구성

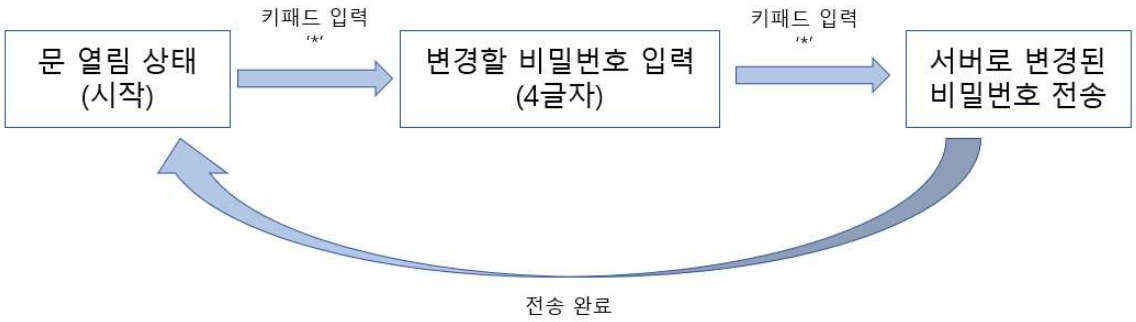


3. 시나리오

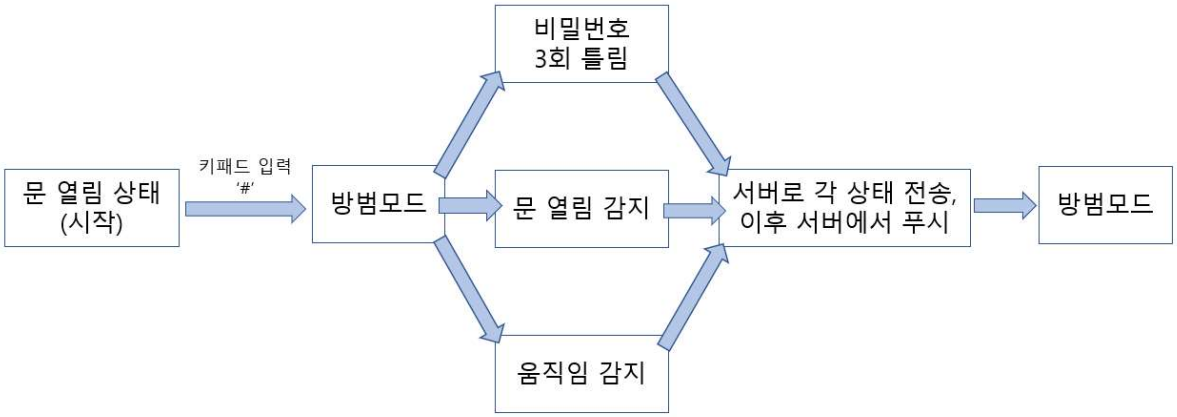
1) 기본 비밀번호를 이용한 문 열기



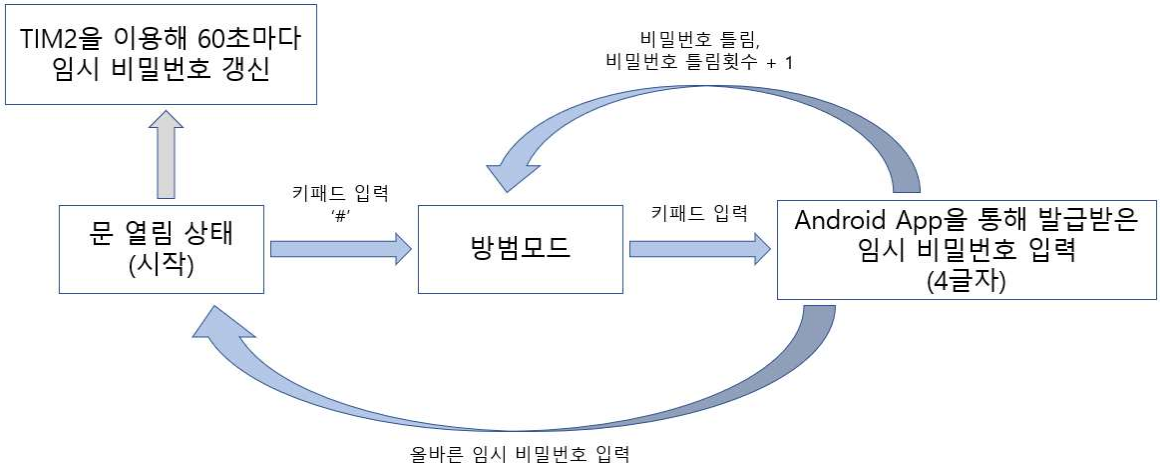
2) 문 열림 상태에서 비밀번호 변경



3) 방범모드에서 무단침입 시도 감지



4) 임시 비밀번호를 이용한 문 열기



4. 사용 부품

1) 서보 모터

Servo Motor는 문의 잠금 장치를 담당한다. 비밀번호의 일치 여부에 따라 동작한다. 프로젝트에서 사용한 서보 모터는 TowerPro MG90S Micro Servo Motor이다.

```
void RCC_Configure(void) {
                                        …
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);
                                        …
}
void GPIO_Configure(void) {
                                        …
    // Servo Motor
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOC, &GPIO_InitStructure);
    GPIO_PinRemapConfig(GPIO_FullRemap_TIM3, ENABLE);
                                        …
}
```

TIM3의 Channel4를 사용하므로 먼저 TIM3에 클럭을 인가해준다. PC9 핀을 Alternative Function Push-pull 모드로 설정한다.

```
void TIM_Init() {
    TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;

    TIM_OC4PreloadConfig(TIM3, TIM_OCPreload_Enable);
    TIM_ARRPreloadConfig(TIM3, ENABLE);
    TIM_Cmd(TIM3, ENABLE);

    TIM_TimeBaseStructure.TIM_Prescaler = (uint16_t)(SystemCoreClock / 1000000)- 1;
```

TIM_Init() 함수에서는 타이머를 설정한다. 사용된 서보 모터는 20ms, 즉 50Hz 주기의 파형을 생성한다. 이를 설정하기 위해서 먼저 72MHz의 System Core Clock에서 Prescaler를 이용하여 1MHz Timer Clock을 생성한다. 따라서 TIM_Prescaler 값은 72MHz / 1MHz – 1 = 71이 된다. 그 다음 50Hz의 주기 파형을 생성하기 위해 주기를 1MHz / 50Hz = 20000으로 설정한다. 따라서 TIM_Period = 20000 – 1이 된다. 이때 Prescaler와 Period는 0부터 count를 하기 때문에 설정하고자 하는 값에 -1을 해주어야 한다.

```
uint16_t Pulse[2] = { 2300, 700 };


void openDoor() {
    TIM_OCInitTypeDef TIM_OCInitStructure;
    TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
    TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
    TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
    TIM_OCInitStructure.TIM_Pulse = Pulse[0];
    TIM_OC4Init(TIM3, &TIM_OCInitStructure);
`
```

문 잠금 장치의 동작은 PWM의 Duty Cycle을 조절하여 설정함으로써 구현된다. 프로젝트에서 사용한 서보 모터는 0.7 ~ 2.3ms High를 유지하는 20ms 주기의 파형으로 -90도 ~ 90도 사이의 동작이 가능하다. 즉 서보 모터를 구동하기 위해 3.5 ~ 11.5%의 Duty Cycle을 설정해야 한다. Period를 통해 계산한 Duty Cycle을 TIM_Pulse에 설정함으로써 각도를 조절한다. 각도에 따른 TIM_Pulse 값은 다음과 같다.

-90도 : 20000 * 3.5 / 100 = 700

0도 : 20000 * 7.5 / 100 = 1500

90도 : 20000 * 11.5 / 100 = 2300

제공된 정보에 따르면 180도 회전이 가능하나 실제 구동했을 때 약 120도 정도 회전하는 것을 확인했다. 문 잠금 장치의 상태를 확실히 표현하기 위해 TIM_Pulse를 각각 2300, 700으로 설정했다.

2) 인체감지센서(PIR Sensor)

사람의 몸에서 방사되는 적외선을 센싱하여 움직임(모션)이 있는지 없는지를 판단하는 센서로써, 방범모드에서 움직임이 감지되면 와이파이모듈을 통해 긴급 신호를 송신한다.

```
void RCC_Configure(void)
{
   RCC_APB2PeriphClockCmd(RCC_APB2ENR_IOPCEN, ENABLE);
}

void GPIO_Configure(void)
{
   GPIO_InitTypeDef GPIO_InitStructure;
   // PIR sensor
   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPD;
   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
   GPIO_Init(GPIOC, &GPIO_InitStructure);

   GPIO_EXTILineConfig(GPIO_PortSourceGPIOC, GPIO_PinSource1);

}
```

PC에 클럭을 인가하고 PIR Sensor의 핀을 PC1 모드는 IPD(input_Pull-Down)로 설정하며 두 핀 모두 입력 및 인터럽트 핀으로써 사용한다.

```
void EXTI_Configuration()
{
   EXTI_InitTypeDef EXTI_InitStructure;

   EXTI_InitStructure.EXTI_Line = EXTI_Line1;
   EXTI_InitStructure.EXTI_LineCmd = ENABLE;
   EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
   EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising_Falling;
   EXTI_Init(&EXTI_InitStructure);
}

void NVIC_Configuration()
{
   NVIC_InitTypeDef NVIC_InitStructure;
   // Priority Table의 PriorityGroup_2를 사용
   NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);
   //EXT11
   NVIC_InitStructure.NVIC_IRQChannel = EXTI1_IRQn;
   NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
   NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
   NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
   NVIC_Init(&NVIC_InitStructure);

}
```

인체감지센서의 핀 PC1을 외부인터럽트로 사용하기 위해 모드를 Interrupt, 방범모드에서 사람이 있는 경우 신호를 보내고 없는 경우는 보내지 않으므로 트리거를 Rising_Falling으로 설정한다. 우선순위는 다른 외부 센서와 동일하게 1로 설정한다.

```
void EXTI1_IRQHandler(void)
{
   /* motion sensing */
   if (EXTI_GetITStatus(EXTI_Line1) != RESET)
   {
      if (GPIO_ReadInputDataBit(GPIOC, GPIO_Pin_1))
      {
         /* Person exists in security mode  */
        if(status == 1){
           GPIO_SetBits(GPIOD, GPIO_Pin_2);
             motion_flag = 1;
        }
      }
      else if (!GPIO_ReadInputDataBit(GPIOC, GPIO_Pin_1))
      {
         /* Person not exists  */
         motion_flag = 0;
         GPIO_ResetBits(GPIOD, GPIO_Pin_2);
      }
      EXTI_ClearITPendingBit(EXTI_Line1);
   }
}
```

EXTI_Line1에서 인터럽트가 발생하면 인터럽트 핸들러에서 PC1의 디지털 값을 읽고 status 변수에 따라 motion_flag의 값을 0 또는 1로 설정했다.

3) 마그네틱 도어 센서

문에 부착하여 문이 열리는 것을 감지하는 센서이다. 방범모드에서 문열림이 감지되면 와이파이 모듈을 통해 긴급 신호를 송신한다..

```
void RCC_Configure(void)
{
   RCC_APB2PeriphClockCmd(RCC_APB2ENR_IOPCEN, ENABLE);
}

void GPIO_Configure(void)
{
   GPIO_InitTypeDef GPIO_InitStructure;
   // Door sensor
   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
   GPIO_Init(GPIOC, &GPIO_InitStructure);

   GPIO_EXTILineConfig(GPIO_PortSourceGPIOC, GPIO_PinSource2);

}
```

PC에 클럭을 인가하고 핀은 PC2 모드는 IPU(input_Pull-UP)로 설정하며 입력 및 인터럽트 핀으로

써 사용한다.

```
void EXTI_Configuration()
{
   EXTI_InitTypeDef EXTI_InitStructure;

   EXTI_InitStructure.EXTI_Line = EXTI_Line2;
   EXTI_InitStructure.EXTI_LineCmd = ENABLE;
   EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
   EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising_Falling;
   EXTI_Init(&EXTI_InitStructure);
}

void NVIC_Configuration()
{
   NVIC_InitTypeDef NVIC_InitStructure;
   // Priority Table의 PriorityGroup_2를 사용
   NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);
   //EXT12
   NVIC_InitStructure.NVIC_IRQChannel = EXTI2_IRQn;
   NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
   NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
   NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
   NVIC_Init(&NVIC_InitStructure);

}
```

도어센서의 핀 PC2를 외부 인터럽트로 사용하기 위해 모드를 Interrupt, 방범모드에서 문열림이 있는 경우 신호를 보내고 없는 경우는 보내지 않으므로 트리거를 Rising_Falling으로 설정한다. 우선순위는 다른 외부 센서와 동일하게 1로 설정한다.

```
void EXTI2_IRQHandler(void)
{
   if (EXTI_GetITStatus(EXTI_Line2) != RESET)
   {
      if (GPIO_ReadInputDataBit(GPIOC, GPIO_Pin_2))
      {
        /* Door opens in security mode */
        if(status == 1){
           GPIO_SetBits(GPIOD, GPIO_Pin_3);
        }
         /* door open */
         door_flag = 1;
      }
      else if (!GPIO_ReadInputDataBit(GPIOC, GPIO_Pin_2))
      {
         /* door close */
         GPIO_ResetBits(GPIOD, GPIO_Pin_3);
         door_flag = 0;
      }
      EXTI_ClearITPendingBit(EXTI_Line2);
   }
}
```

EXTI_Line2에서 인터럽트가 발생하면 인터럽트 핸들러에서 PC2의 디지털 값을 읽고 status 변수

에 따라 door_flag의 값을 0 또는 1로 설정했다.

4) 4x4 멤브레인 키보드

```
void INPUT_PASSWORD()
{
   GPIO_ResetBits(GPIOA, GPIO_Pin_4);
   GPIO_SetBits(GPIOA, GPIO_Pin_5);
   GPIO_SetBits(GPIOA, GPIO_Pin_6);
   GPIO_SetBits(GPIOA, GPIO_Pin_7);
   while (!GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_0)) {
   value = 'D';
   delay(10);
   }
   while (!GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_1)) {
   value = '#';
   delay(10);
   }
   while (!GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_11)) {
   value = 0;
   delay(10);
   }
   while (!GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_12)) {
   value = '*';
   delay(10);
   }
.....
```

4x4 멤브레인 키보드는 4개의 입력과 4개의 출력 선을 가지며, 각각의 선은 4x4 키 배열의 세로줄과 가로줄에 연결되어 있다. 따라서 4개의 출력 선 중 위의 코드와 같이 셋이 Set 상태이고 하나가 Reset 상태이면 입력 선을 읽었을 때 0이 나오는 세로와 가로줄의 버튼이 눌렸음을 판별할 수 있다. 이런 원리로 4x4 키패드는 입력 대기 상태일 때 루프를 돌며 4개의 출력 선에 Set과 Reset을 빠르게 반복하며 버튼 입력을 확인한다. 버튼이 지속적으로 눌려지거나 짧게 떨어졌다가

다시 눌려졌을 경우 입력 값을 보정하기 위해 while(!GPIO_ReadInputDataBit(...)) 구문으로 버튼 입력을 판별한다.

5) Wifi 모듈

본 프로젝트의 Wifi 모듈은 ESP-8266을 사용한다. 해당 모듈은 AT 명령을 이용하여 AP/Station 모드를 선택할 수 있고 특정 서버와 TCP/UDP 통신을 할 수 있다. 해당 모듈과의 통신을 위해 UART를 이용할 수 있고, Baud rate는 115200이 기본이며 추가 명령을 통해 설정할 수 있다.

서버와의 통신은 socket을 이용하며, 해당 서버에서 요청에 대한 응답을 어떻게 처리하는지 6) Server 장에서 확인할 수 있다.

이 프로젝트에서 사용된 AT 명령어는 아래와 같다.

- **AT+CWMODE=mode** : Wifi 모드를 설정하는 역할을 한다. AP/Station 모드를 설정할 수 있다.
- **AT+CWJAP=[ssid],[password]** : ssid라는 이름의 Wifi에 password안의 비밀번호를 이용해 접속하게 하는 명령어이다.
- **AT+CIPSTART=[TCP/UDP]** : TCP 또는 UDP 포트를 설정하고 연결하게 한다. 이 프로젝트에선 TCP로 서버에 연결시키는 역할을 한다.
- **AT+CIPSTATUS** : 현재 접속 정보를 알아내는 명령어이다.
- **AT+CIPSEND=n** : 연결되어 있는 접속에 n만큼의 데이터를 전송하게 한다.

5.1) Initialization

```
/* Configure USART related Clock
    RCC
    GPIOA 2, 3, 9, 10
    NVIC
    USART1, 2 BaudRate 115200 */
void USART_Configure(){
    GPIO_InitTypeDef GPIO_InitStructure;
    NVIC_InitTypeDef NVIC_InitStructure;
    USART_InitTypeDef USART_InitStructure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE);

    RCC_APB2PeriphClockCmd(RCC_APB2ENR_AFIOEN, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2ENR_IOPAEN, ENABLE);


    //USART1,2 Tx
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2 | GPIO_Pin_9;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    //USART1,2 RX
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3 | GPIO_Pin_10;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    //USART1
    NVIC_EnableIRQ(USART1_IRQn);
    NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);

    //USART2
    NVIC_EnableIRQ(USART2_IRQn);
    NVIC_InitStructure.NVIC_IRQChannel = USART2_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);

    //USART1 Init
    USART_Cmd(USART1, ENABLE);

    USART_InitStructure.USART_BaudRate = 115200;
    USART_InitStructure.USART_WordLength = USART_WordLength_8b;
    USART_InitStructure.USART_StopBits = USART_StopBits_1;
    USART_InitStructure.USART_Parity = USART_Parity_No;
    USART_InitStructure.USART_HardwareFlowControl =
USART_HardwareFlowControl_None;
    USART_InitStructure.USART_Mode = (USART_Mode_Rx | USART_Mode_Tx);
```

```
    USART_Init(USART1, &USART_InitStructure);
    USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);

    //USART2 Init
    USART_Cmd(USART2, ENABLE);

    USART_InitStructure.USART_BaudRate = 115200;
    USART_InitStructure.USART_WordLength = USART_WordLength_8b;
    USART_InitStructure.USART_StopBits = USART_StopBits_1;
    USART_InitStructure.USART_Parity = USART_Parity_No;
    USART_InitStructure.USART_HardwareFlowControl =
USART_HardwareFlowControl_None;
    USART_InitStructure.USART_Mode = (USART_Mode_Rx | USART_Mode_Tx);

    USART_Init(USART2, &USART_InitStructure);

    USART_ITConfig(USART2, USART_IT_RXNE, ENABLE);
}
```

USART_Configure에서는 보드에서 사용되는 USART에 관한 사전 설정을 진행한다. 우선 RCC를 통해 GPIO, AFIO, USART1,2에 클럭을 주고, GPIO의 핀들을 활성화 시킨다. 또한 NVIC을 통해 USART 인터럽트에서 사용될 우선순위를 지정하고 USART1, 2에 대한 통신 설정을 진행한다.

```
void ServerTimer_Configure(){
    NVIC_InitTypeDef NVIC_InitStructure;
    TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;

    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);

    NVIC_InitStructure.NVIC_IRQChannel = TIM2_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);

    TIM_TimeBaseStructure.TIM_Period= 1000 -1; // 1ms
    TIM_TimeBaseStructure.TIM_Prescaler=72-1;
    TIM_TimeBaseStructure.TIM_ClockDivision=0;
    TIM_TimeBaseStructure.TIM_CounterMode=TIM_CounterMode_Up;
    TIM_TimeBaseInit(TIM2,&TIM_TimeBaseStructure);

    TIM_Cmd(TIM2,ENABLE);
    TIM_ITConfig(TIM2,TIM_IT_Update,ENABLE);
}
```

또한 본 프로젝트에서는 서버를 이용하기 때문에 지속적인 업데이트에 필요한 타이머 또한 활성화시켜 주어야한다. 서버와 지속적인 통신을 위해 보드의 Timer 2를 이용하며 1ms마다 Clock이 발생하도록 설정해주었다.

5.2) USART Interrupt handler

```
/*  USART1 Handler for debugging.
    Echo USART2 Rx */
void USART1_IRQHandler(){
    uint16_t word;

    if (USART_GetITStatus(USART1, USART_IT_RXNE) != RESET)
    {
        word = USART_ReceiveData(USART1);
        while (USART_GetFlagStatus(USART1, USART_FLAG_TXE) == RESET){}
        USART_SendData(USART1, word);
    }

    USART_ClearITPendingBit(USART1, USART_IT_RXNE);
}

/*  USART2 Handler. Communicate with Wifi Module
    when Rx is not empty, read Rx and fill buffer buf
    buf will be used parsing part */
void USART2_IRQHandler()
{
    uint16_t word;

    if (USART_GetITStatus(USART2, USART_IT_RXNE) != RESET)
    {
        word = USART_ReceiveData(USART2);

        if(cnt > 100) cnt = 0;

        // when come with new line, clear buffer
        if (word == '\n')
        {
            cnt++;
            buf[cnt] = 0;
            cnt = 0;
        }
        else
        {
            buf[cnt] = word;
            cnt++;
        }

        while (USART_GetFlagStatus(USART2, USART_FLAG_TXE) == RESET){}
        USART_SendData(USART1, word);
        while ((USART2->SR & USART_SR_TC) == 0){}
    }

    USART_ClearITPendingBit(USART2, USART_IT_RXNE);
}
```

ESP-8266의 경우 UART를 이용하여 보드와 통신한다. 따라서 Wifi 모듈에서 어떠한 신호를 되돌려주었을 때 그것을 처리할 수 있는 USART Interrupt handler를 설정해주어야 한다. USART1의 경우 디버깅을 위해 사용되며 USART2는 보드와 모듈간의 통신을 위해 사용한다.

우선 USART1_IRQHandler의 경우에는 자신의 Rx에 있는 데이터를 Tx로 그대로 넘기는 echo 형식으로 구성되어 있다. 후술할 USART2로부터 모든 데이터를 전송받기 때문에 모듈에서 어떠한 신호가 오고 가는지 확인할 수 있도록 하였다.

USART2_IRQHandler는 보드와 모듈간의 통신을 처리한다. UART의 특성상 하나의 바이트만을 한 번의 인터럽트에서 처리할 수 있다. 따라서 해당 함수는 모듈에서 보드로 어떠한 요청에 대해 응답하였을 때 해당 응답이 끝날 때까지 버퍼에 한 개의 byte마다 써 주는 작업을 처리한다. 본 프로젝트에서는 특정 응답에 대해서만 처리할 것이기 때문에 Ring buffer를 사용하지 않으며 제일 마지막에 응답한 내용만 기록하도록 구성되었다. 또한 디버깅을 위해서 모든 응답에 대해 USART1으로 다시 써주는 작업 또한 처리한다.

5.3) Timer Interrupt Handler

```
/*  Timer2 Hander for continuously update temp password.
   temp_req_flag and temp_flag will be used in main.c */
void TIM2_IRQHandler(){
        if(TIM_GetITStatus(TIM2, TIM_IT_Update) != RESET){
                temp_req_flag += 1;
                temp_flag += 1;
        }
        TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
}
```

서버와 주기적인 통신을 위해 타이머를 본 프로젝트에서 사용하며, 해당 타이머는 4.1.에서 설명했던 것과 같이 1ms마다 Clock이 발생된다. Clock이 발생될 때 마다 TIM2_IRQHandler가 콜백으로 호출된다. 해당 함수에서는 main에서 처리될 temp_req_flag와 temp_flag의 개수를 하나씩 증가시킨다. 따라서 해당 변수가 각각 300000(5분), 60,000(1분)이 될 때마다 특정 기능을 수행할 수 있도록 설정했다.

5.4) Board – Wifi Module Communication

```
/*  Send char to USART1 */
void sendUSART1(uint16_t data)
{
    // wait Tx is clear
    while (USART_GetFlagStatus(USART1, USART_FLAG_TXE) == RESET){}

    USART_SendData(USART1, data);

    // make Tx clear flag to 0
    while ((USART1->SR & USART_SR_TC) == 0) {}
}

/*  Send char to USART2 */
void sendUSART2(uint16_t data)
{
    // wait Tx is clear
    while (USART_GetFlagStatus(USART2, USART_FLAG_TXE) == RESET) {}

    USART_SendData(USART2, data);

    // make Tx clear flag to 0
    while ((USART2->SR & USART_SR_TC) == 0) {};
}

/*  Send string to USART1 for debug */
void sendTTY(char *str)
{
    int len = strlen(str);

    for (i = 0; i < len; i++){
        sendUSART1((uint16_t)str[i]);
    }
}

/*  Send AT Command to USART1 for set Wifi Module */
void sendWIFI(char *str)
{
    int len = strlen(str);

    for (i = 0; i < len; i++){
        sendUSART2((uint16_t)str[i]);
    }

    // AT Commandd must be end with \r\n
    sendUSART2('\r');
    sendUSART2('\n');
}

/*  To parse OK Command.
    When AT command properly be executed, Wifi Module return String 'OK'.
    If more AT command given while previous command is executed, Wifi
module return String 'busy s...' and drop given command cus it hasn't
buffer.
    So should wait previous command is executed */
```

```
void waitOK(){
        while(1){
                if(buf[0] == 'O' && buf[1] == 'K') {
           cnt = 0;
           memset(buf, 0, 500);
           break;
       }

                delay(100);
        }
}

/*  To parse specific char.
    When give 'AT+CIPSEND' command, Wifi Module return char '>'
    For same reason waitOK() we should wait that char.  */
void wait(uint16_t str){
        while(1){
                if ( buf[0] == str) {
                    cnt = 0;
           memset(buf, 0, 500);
           break;
       }

                delay(100);
        }
}

/*  To parse '+IPD,' string
    When give 'AT+CIPSEND' is executed and server return some string, Wifi
Module return server response with '+IPD,[num of recv bytes]:[Server
response]'
    For same reason for waitOK(), we should wait that char. */
void recvWIFI(char *str){
   if (buf[0] == '+' && buf[1] == 'I' && buf[2] == 'P' && buf[3] == 'D'){
     str[0] = buf[7];
     str[1] = buf[8];
     str[2] = buf[9];
     str[3] = buf[10];
   }
   else
     str = "Fail";

   sendWIFI("AT+CIPSEND=4");
   wait('>');
   delay(100);

   // Close Socket
   sendNoLine("CLOS");
   delay(200);
}

/*  Same functioning recvWIFI(). But, not close socket. */
void recvWIFIWithoutClose(char *str){
   if (buf[0] == '+' && buf[1] == 'I' && buf[2] == 'P' && buf[3] == 'D'){
     str[0] = buf[7];
     str[1] = buf[8];
     str[2] = buf[9];
```

```
      str[3] = buf[10];
      str[4] = '\0';
   }
   else
      str = "Fail";
}

/*  Open socket and send some string to server */
void requestWIFI(char *cmd){
        sendWIFI("AT+CIPSTART=\"TCP\",\"server.gomsoup.com\",4000");
        delay(100);
        waitOK();

        sendWIFI("AT+CIPSEND=4");
        delay(100);
        wait('>');
        delay(100);

        sendNoLine(cmd);
        delay(100);
}

/* Send some string to current socket */
void requestWIFICurrentSocket(char *cmd){
        sendWIFI("AT+CIPSEND=4");
        delay(100);
        wait('>');
        delay(100);

        sendNoLine(cmd);
        delay(100);
}

/*  Initialize Wifi module  */
void WIFI_Init(){
    // Station mode
    sendWIFI("AT+CWMODE=1");
    waitOK();

    // Connect to AP
    sendWIFI("AT+CWJAP=\"ABC\",\"123456789a\"");
    waitOK();

    // Single communication mode
    sendWIFI("AT+CIPMUX=0");
    waitOK();

    // Get Status
    sendWIFI("AT+CIPSTATUS");
    waitOK();
}
```

보드와 Wifi 모듈간의 통신, 디버깅을 위한 함수들이다. 해당 함수들의 기능은 다음과 같다.

- **sendUSART1** : USART1으로 특정 문자를 보낸다. 디버깅을 위해 사용된다.
- **sendUSART2** : USART2로 특정 문자를 보낸다. Wifi 모듈에 특정 문자를 전송하기 위해 사용된다.
- **sendTTY** : USART1으로 특정 문자열을 보낸다. 디버깅을 위해 사용된다.
- **sendWIFI** : USART2(Wifi Module)로 특정 문자열을 보낸다. Wifi 모듈에서는 AT 명령만을 인식하며, 명령의 끝은 항상 '₩r₩n'으로 끝나야 하기 때문에 해당 처리에 대한 루틴을 수행한다.
- **waitOK** : ESP-8266 모듈에서는 전달받은 AT 명령을 수행한 후 특정 명령에 대한 응답으로 'OK'를 전송한다. 만약 어떠한 명령을 수행 중에 다른 명령이 도착한다면 'busy s...'라는 문자열을 응답하며, 새롭게 전달받은 명령은 수행되지 않는다. 따라서 모든 명령은 'OK' 응답을 받은 후 전달되어야 하며, 해당 처리를 위한 루틴을 수행한다.
- **wait** : waitOK()와 같은 이유로, 특정 명령에 대해 특정 문자를 응답할 때 까지 대기하는 루틴을 수행한다.
- **recvWIFI** : 서버와 통신하는 AT 명령을 수행한 후 모듈은 '+IPD,[받은 응답의 바이트 수]:[받은 응답 내용]'과 같은 형식으로 보드에 응답한다. 해당 문자열을 처리하고, 서버의 응답만을 얻기 위한 파싱 기능을 수행한다. 또한 처리 후 소켓을 닫는다.
- **recvWIFIWithoutClose** : recvWIFI()와 기능은 같지만, 소켓을 닫지 않는다. 소켓을 닫지 않고 지속적인 서버와의 통신을 처리해야 할 때 사용된다.
- **requestWIFI** : Wifi 모듈과 서버의 통신을 처리하는 AT 명령을 전송할 때 사용된다. 서버와 소켓을 생성하고, 전달받은 내용을 서버로 전송하는 기능을 수행한다.
- **requestWIFICurrentSocket** : requestWIFI()와 같은 기능을 수행하지만, 소켓을 새로 생성하지 않고 현재 소켓에 대한 전송을 수행한다.
- **WIFI_Init** : Wifi 모듈에 대한 초기화를 진행한다. 제일 먼저 Station 모드로 설정하고, AP에 연결하며 단일 통신에 대한 처리만 지원하도록 한다.

5.5) Wifi Module – Server Communication

```
void getPassword(){
   char str[4];
   requestWIFI("PASS");
   delay(100);
   recvWIFI(str);


   if (strcmp("Fail", str)){
      password[0] = str[0]-48;
      password[1] = str[1]-48;
      password[2] = str[2]-48;
      password[3] = str[3]-48;
   }
}

void getTempPassword(){
   char pass[4];
   requestWIFI("REQT");
   recvWIFI(pass);


   if (strcmp("Fail", pass)){
      temp_password[0] = pass[0]-48;
      temp_password[1] = pass[1]-48;
      temp_password[2] = pass[2]-48;
      temp_password[3] = pass[3]-48;
   }
}

void updatePassword(char* new_passwd){
        char str[4];
        requestWIFI("UDAT");
        recvWIFIWithoutClose(str);
        sendTTY("recv PSWD : ");
        sendTTY(str);
        sendTTY("\r\n");

        if(!strcmp("PSWD", str))
       requestWIFICurrentSocket(new_passwd);
        else
       LCD_ShowString(10,30, "Update Password Failed", BLACK, WHITE);

        recvWIFI(str);
}

void removeTempPassword(){
        char str[4];
        requestWIFI("DELT");
        recvWIFI(str);
}

void urgentPush(){
        char str[4];
        requestWIFI("URGT");
        recvWIFI(str);
```

```
}

void urgentDoorPush(){
        char str[4];
        requestWIFI("URDR");
        recvWIFI(str);
}

void passwordWrongPush(){
        char str[4];
        requestWIFI("OVRP");
        recvWIFI(str);
}
```

Wifi 모듈과 서버와의 통신을 위해 사용되는 함수들이다. 해당 함수들의 기능은 다음과 같다.

- **getPassword** : 서버로부터 도어락의 비밀번호를 전송받는다. 이 때 응답받는 값은 ASCII 타입이므로, uint16_t에 맞게 캐스팅해 준다.
- **getTempPassword** : 서버로부터 도어락의 임시 비밀번호를 전송받는다. 이 때 응답받는 값은 ASCII 타입이므로, uint16_t에 맞게 캐스팅해 준다.
- **updatePassword** : 서버로부터 도어락 비밀번호 갱신을 요청한다.
- **removeTempPassword** : 서버에 임시 비밀번호 삭제를 요청한다.
- **urgentPush** : 서버에 침입 상황 알람을 보낸다. 문이 잠겨있지만, 내부에 동작이 감지되었을 경우 사용된다.
- **urgentDoorPush** : 서버에 침입 상황 알람을 보낸다. 도어락은 잠김 모드로 설정되어 있지만, 문이 열린 경우 사용된다.
- **passwordWrongPush** : 비밀번호가 3회 이상 틀렸을 경우 서버에 알람을 보낸다.

6) Server

본 프로젝트에서는 서버와 통신을 진행하며, 서버의 경우 특정 요청을 소켓으로 받고, 해당 요청에 대한 적절한 기능을 수행한 후 응답한다. 또한 특정 알람 상황에 대해 7) Android Application 에서 후술할 푸시 기능을 처리한다.

6.1) Server code

```
import socket
import threading
import string
import random
from pyfcm import FCMNotification

host = ''
port = 4000
con = 0

def pushFCM(head, body, key):
    push_service = FCMNotification(api_key=key)
    result =
push_service.notify_topic_subscribers(topic_name="com_example_doorlock",
message_title=head, message_body=body)
    print(result)

def sendPassword():
    print('send')

def getInput():
    while(True):
        data = con.recv(4).decode()
        print('client: ', data)
        if data == 'PASS':
            fd = open('./pass')
            password = fd.read(4)
            if not password:
                con.send('1234'.encode('utf-8'))  #default passwd
            else:
                con.send(password.encode('utf-8')) #user password
            fd.close()

        if data == 'TEMP': # requset temp passwd
            open('./temp', 'w').close()
            fd = open('./temp', 'w')

            pool = string.digits
            temp = ""

            for i in range(4):
                temp += random.choice(pool);

            fd.write(temp)
```

```
            con.send(temp.encode('utf-8'))

            print("temp passwd : ", temp)
            fd.close()

        if data == 'REQT':
            fd = open('./temp')
            pw = fd.read(4)

            if not pw:
                pw = "0000"

            con.send(pw.encode('utf-8'))
            fd.close()

        if data == "DELT":
            open('./temp', "w").close()
            fd = open('./temp', "w")

            fd.write("0000")
            fd.close()

            con.send('OKOK'.encode('utf-8'))
            print("delete temp passwd");


        if data == 'URGT': # argent moment
            fd = open('/home/ubuntu/api_key')
            key = fd.readline().splitlines()
            pushFCM("URGENT!", "침입이 감지되었습니다!", key[0])

            print("Urgent alert pushed")
            fd.close()

        if data == 'URDR':
            fd = open('/home/ubuntu/api_key')
            key = fd.readline().splitlines()
            pushFCM("URGENT!", "문이 열렸습니다!", key[0])

            print("Urgent door open alert pushed")
            fd.close()

        if data == 'OVRP':
            fd = open('/home/ubuntu/api_key')
            key = fd.readline().splitlines()
            pushFCM("URGENT!", "비밀번호가 3 회 틀렸습니다!", key[0])

            print("Password wrong !")
            fd.close()

        if data == 'UDAT':
            open('./pass', 'w').close()
            fd = open('./pass', 'w')
            con.send('PSWD'.encode('utf-8'))
            print('request password change ')

            password = con.recv(4).decode()
```

```
            print("update passwd ", password)
            fd.write(password)
            fd.close()

        if data == 'CLOS':
            break
    con.close()


s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
s.bind((host, port))
s.listen(1)
print("doorlock server now listen..")

while True:
    con, addr = s.accept()
    print('Connected by :' , addr)
    threading._start_new_thread(getInput, ())
```

서버 단 프로그램은 python을 이용하고 있다. 4000번 포트에 바인딩 되어 있으며 멀티 스레드를통한 다중 클라이언트 요청에 응답할 수 있도록 구성되어 있다. DB 등을 이용하여 확장할 수 있지만, 본 프로젝트에서는 하나의 디바이스로만 구성되어 있으므로, File IO를 이용하여 간결성을 추구하였다.

소켓 통신의 특성상, delimiter가 지정되지 않으면, 특정 바이트 수만 읽도록 처리해야 한다. 따라서 모든 요청과 응답은 4byte만을 이용하도록 구성되어 있다.

6.1.1) Server-Client Communication

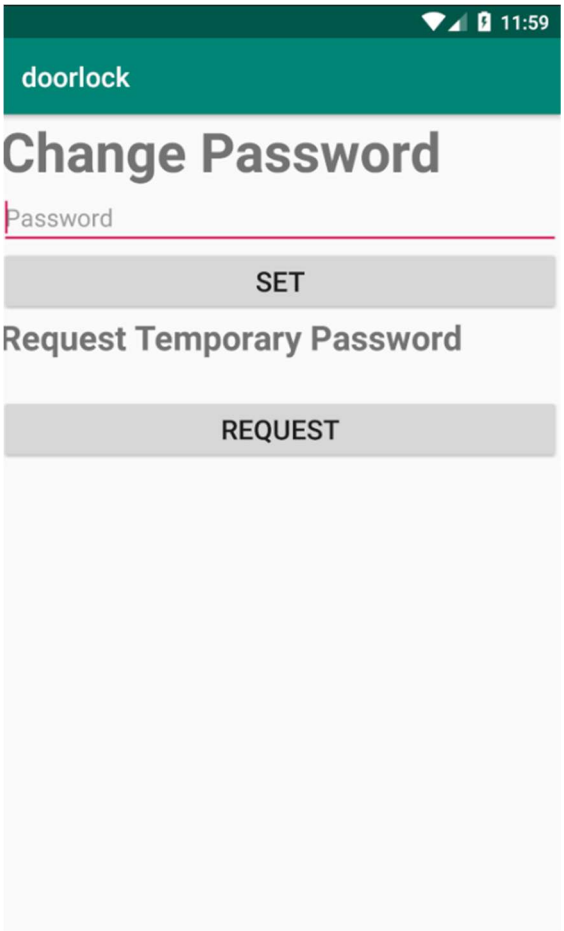
클라이언트는 서버에 4바이트로 구성된 요청을 보내게 된다. 요청에 대한 응답은 다은과 같다.

- **PASS** : 클라이언트에 현재 저장된 비밀번호를 응답한다. 만약 저장된 비밀번호가 없을 경우 기본 패스워드로 1234를 응답한다.
- **TEMP** : 임시비밀번호를 생성한 후 저장하고 클라이언트에 응답한다. 만약 이미 저장된 비밀번호가 있을 경우, 삭제 후 재생성한다.
- **REQT** : 저장된 임시비밀번호를 응답한다. 만약 저장된 비밀번호가 없을 경우 0000을 응답한다.
- **DELT** : 저장된 임시 비밀번호를 삭제한다.
- **UDAT** : 저장된 비밀번호를 업데이트한다. 해당 요청을 수신한 후 클라이언트로 새로운 비밀번호를 재요청하게 되며, 해당 응답을 통해 비밀번호를 재설정 한다.
- **URGT** : 안드로이드 어플리케이션으로 침입 상황 알람을 푸시한다.
- **URDR** : 안드로이드 어플리케이션으로 문 열림 긴급 상황을 푸시한다.
- **OVRP** : 안드로이드 어플리케이션으로 비밀번호 3회 틀림 상황을 푸시한다.

6.1.2. FCM Push Request

서버에서는 FCM(Firebase Cloud Messaging)을 통해 안드로이드 앱으로 푸시한다. pushFCM 함수에서 살펴볼 수 있듯, 사전 발급된 API 키를 이용하여 특정 상황에 맞는 알람을 앱으로 보낼 수 있다. 푸시 결과는 7) Android application에서 후술된다.

7) Android Application

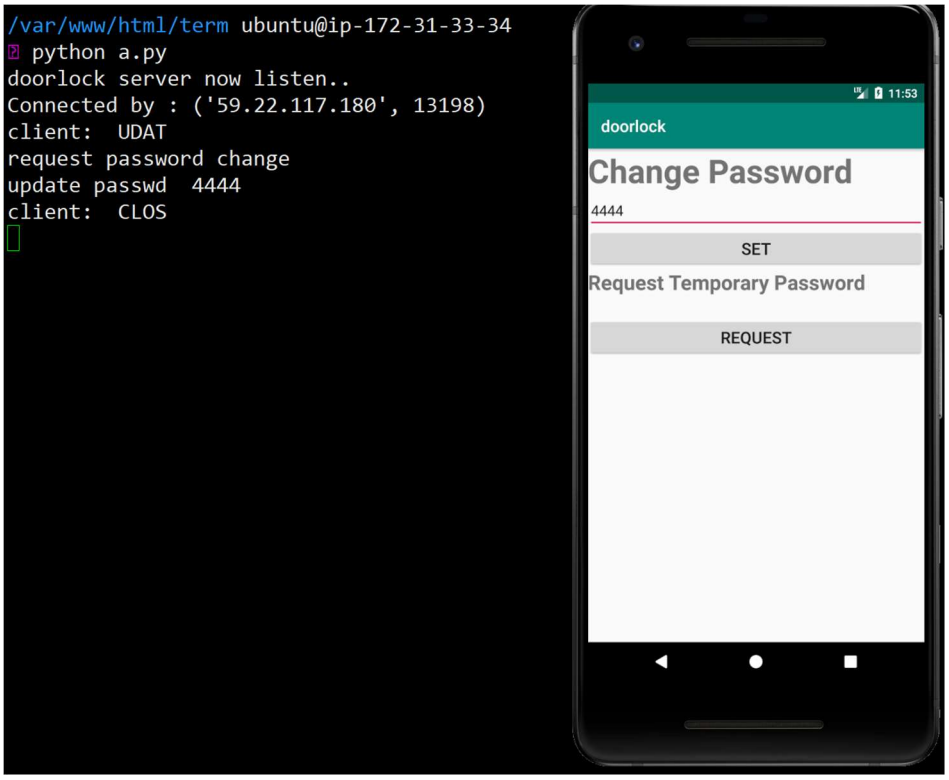


본 프로젝트에서는 안드로이드 앱을 이용하여 도어락의 비밀번호 재설정, 임시 비밀번호 발급 등의 기능을 수행할 수 있다. 사용된 안드로이드 SDK 버전은 23이며, 소켓 통신을 이용한다. 본 프로젝트가 어떻게 활용될 수 있는지에 대한 가능성을 보여주기 위함 임으로 세부적인 설정은 진행하지 않았다(기기 인증 등). 안드로이드 앱 제작의 특성상 소스코드의 일부만을 기록한다.

7.1) Password Reset

```
changePwBtn.setOnClickListener(new View.OnClickListener(){
          @Override
          public void onClick(View v){
              String str = textpw.getText().toString();

              if(!isPassword(str)) {
                  Toast.makeText(MainActivity.this, "Password must be 4
digits", Toast.LENGTH_SHORT).show();
                  return;
              }

              try{
                  BackgountTask bg = new BackgountTask();
                  String buf = "UDAT";

                  String ret = bg.execute(buf, str).get();
                  if(ret.equals("fail")) Toast.makeText(MainActivity.this,
"Server is now up. But something wrong in communication.",
Toast.LENGTH_SHORT).show();
                  else Toast.makeText(MainActivity.this, "Success",
Toast.LENGTH_SHORT).show();

                  bg.cancel(true);
              }catch (Exception e){
                  Toast.makeText(MainActivity.this, "Something wrong in
socket communication", Toast.LENGTH_SHORT).show();
              }
          }
      });
```

패스워드 변경 버튼을 눌렀을 경우 서버로 업데이트 요청을 보낸다. 서버가 정상적인 응답을 했다면, 변경할 비밀번호를 전송한다.

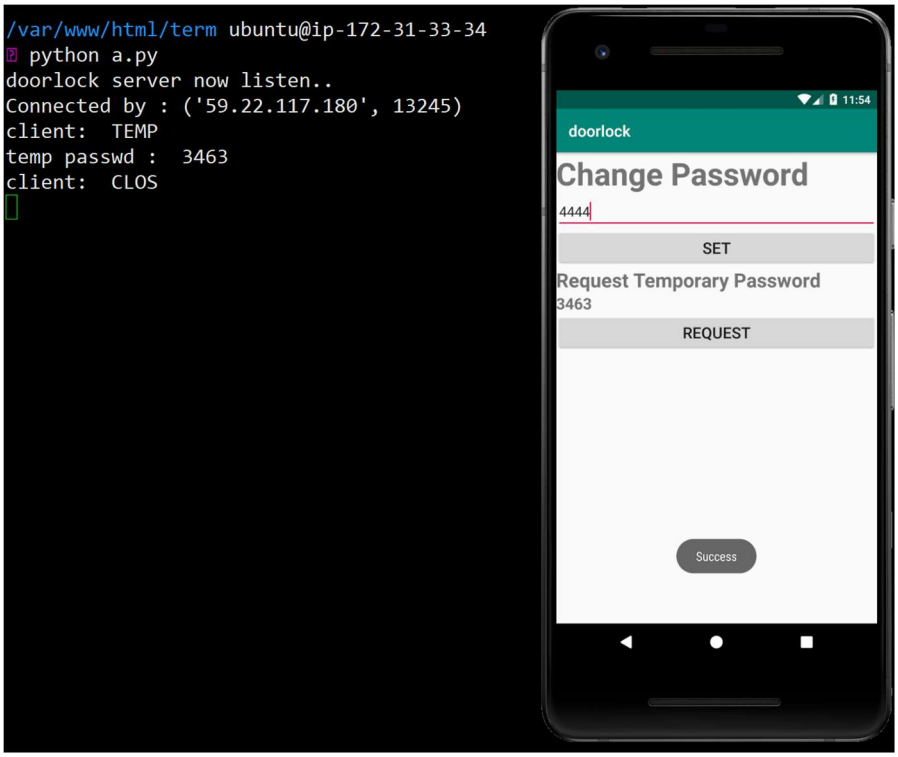


패스워드 변경 요청을 보냈을 때 서버의 기록과 응답은 위와 같다.

7.2) Request Temporally Password

```
getTempPwBtn.setOnClickListener(new View.OnClickListener(){

          @Override
          public void onClick(View v) {
              try{
                  BackgountTask bg = new BackgountTask();
                  String buf = "TEMP";

                  String ret = bg.execute(buf).get();
                  if(ret.equals("fail")) Toast.makeText(MainActivity.this,
"Server is now up. But something wrong in communication.",
Toast.LENGTH_SHORT).show();
                  else Toast.makeText(MainActivity.this, "Success",
Toast.LENGTH_SHORT).show();

                  temppw.setText(new String(ret));

                  bg.cancel(true);
              }catch (Exception e){
                  Toast.makeText(MainActivity.this, "Something wrong in
socket communication", Toast.LENGTH_SHORT).show();
              }
          }
      });
```

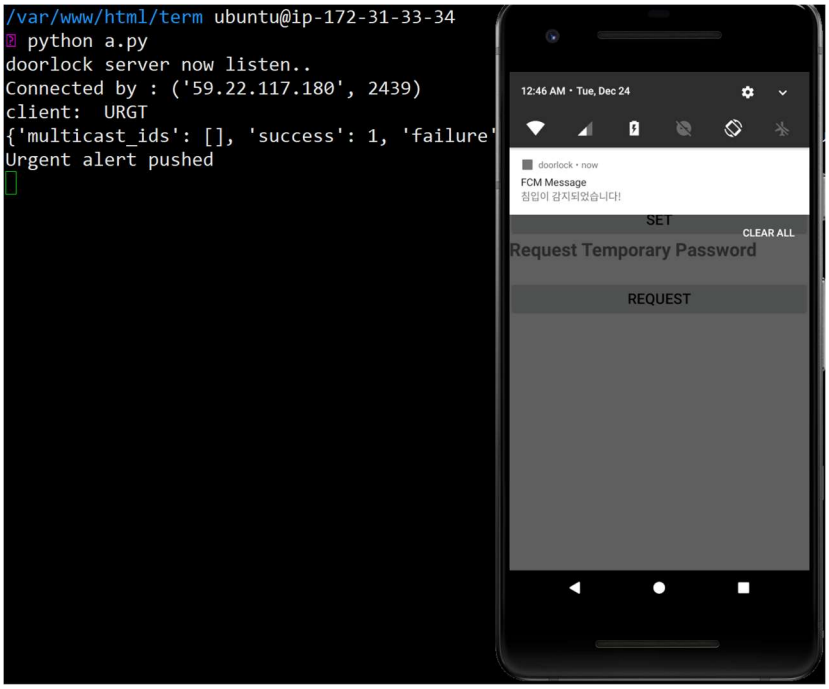
임시비밀번호 발급 버튼을 눌렀을 경우 서버로 임시비밀번호 발급을 요청한다. 서버의 응답이 정상적일 경우 화면에 임시비밀번호를 띄워준다.



임시비밀번호 발급 요청을 보냈을 때 서버의 기록과 응답은 위와 같다.

7.3) Urgent Alert

긴급 상황이 발생했을 경우 도어락은 서버로 긴급상황을 알리고, 서버는 앱으로 긴급상황을 푸시한다.



긴급상황에 대한 푸시 알람은 위와 같다.

5. 결과

결과로 작성된 코드는 아래의 주소를 통해 확인할 수 있다.

https://github.com/gomsoup/embed_2019/tree/master/term

위 주소의 코드로 보드의 동작을 확인하였을 때, 시나리오와 동일하게 동작함을 볼 수 있었다.