

Projekt systemu kontroli dostępu RFID konfigurowanego aplikacją mobilną poprzez połączenie BT

Igor Elche

Streszczenie:

W tej pracy szczegółowo opisano proces stworzenia systemu kontroli dostępu służącego do zabezpieczenia stref teoretycznego obiektu przemysłowego lub biznesu przed nieautoryzowanym dostępem prowadzącym do awarii bezpieczeństwa.

System jest oparty na działaniu mikrokontrolera STM32L476RG, wykorzystującym technologię RFID do uwierzytelniania użytkowników i podejmowania decyzji nadania dostępu oraz została zaimplementowana technologia Bluetooth do komunikacji z aplikacją mobilną na systemie Android, służącą do konfigurowania systemu kontroli dostępu, uzyskania danych o historii logowań oraz zdalnego otwarcia zamku solenoidalnego za pomocą podłączonego sterującego przekaźnika i źródła zasilania.

Komunikacja w obie strony została zabezpieczona od przechwytywania używając algorytmu szyfrowania wiadomości RSA. Część programowa mikrokontrolera została opracowana w języku C posługując się środowiskiem programistycznym CubeMX. Aplikacja mobilna została napisana w framework'u Android Studio używając języka Java z wykorzystaniem API Bluetooth.

Algorytm działania opisano na schematach blokowych a połączenie modułów na schematach elektrycznych. W pracy pokazano trzy możliwości montażu urządzenia końcowego w oparciu o zintegrowaną płytę PCB, na płytce uniwersalnej lub stykowej.

Abstract:

This work describes in detail the process of creating an access control system used to protect zones of a theoretical industrial facility or business against unauthorized access leading to security failures.

The system is based on the STM32L476RG microcontroller, using RFID technology to authenticate users and make access decisions, and Bluetooth technology has been implemented for communication with a mobile application on Android, used to configure the access control system, obtain login history data and remotely open the solenoid lock using a connected control relay and power source.

Both-way communication is protected from interception using the RSA message encryption algorithm. The program part of the microcontroller was developed in C using the CubeMX programming environment. The mobile application was written in the Android Studio framework using Java using the Bluetooth API.

The operating algorithm is described in block diagrams and the connection of modules in electrical diagrams. The work shows three possibilities of mounting the end device based on an integrated PCB, on a universal board or a breadboard.

Spis treści

Wprowadzenie.....	2
Cel pracy.....	2
Zakres pracy.....	2
1. Przegląd rozwiązań.....	3
1.1. Kontroler dostępu autonomiczny DHI-ASI1201.....	3
1.2. Kontroler dostępu TTlock smartLock K2.....	4
2. Koncepcja urządzenia.....	6
2.1 Funkcjonalność.....	6
2.2 Analiza niezbędnych funkcji.....	7
3. Projekt części sprzętowej mikrokontrolera.....	9
3.1. Procesor.....	9
3.2. Moduł czytnika RFID.....	11
3.3. Moduł Bluetooth.....	12
3.4. Przekaźnik i zamek.....	14
3.4. Zasilanie.....	15
4. Projekt części programowej mikrokontrolera.....	16
4.1. Inicjalizacja sterownika.....	17
4.2. Obsługa modułu czytnika RFID.....	19
4.3. Obsługa modułu Bluetooth.....	21
4.4. Protokół komunikacyjny.....	23
4.5. Szyfrowanie i uwierzytelnianie.....	27
5. Projekt aplikacji mobilnej.....	31
5.1 Tworzenie aplikacji mobilnej.....	32
5.2. Interfejs graficzny.....	33
5.3. Bluetooth.....	35
5.4. Komunikacja z mikrokontrolerem.....	36
6. Budowa prototypu.....	39
6.1 Prototyp na płytce stykowej.....	40
6.2 Prototyp na uniwersalnej płytce drukowanej.....	41
6.1 Prototyp na płytce drukowanej.....	42
Podsumowanie.....	43
Bibliografia.....	44
Spis rysunków.....	45
Spis listingów.....	46
Spis tabel.....	47

1. Cel pracy

Celem pracy jest opracowanie projektu domowego systemu kontroli dostępu składającego się w części sprzętowej z mikrokontrolera STM32 sterującego elektrozamkiem, czytnika RFID oraz modułu BT. W części programowej ma być opracowany projekt aplikacji dla Androida z projektem protokołu komunikacyjnego, mechanizmami uwierzytelnienia i szyfrowania. Aplikacja ma umożliwiać konfigurację systemu kontroli dostępu, jak zarządzanie tagami RFID, przeglądanie historii wejśc oraz zdalne otwieranie zamka.

1.1 Zakres pracy

Projekt będzie oparty o mikrokontroler STM32-Nucleo-L476RG jako kontroler systemu kontroli dostępu oraz aplikacja mobilna Android jako interfejs człowiek-maszyna do interakcji z urządzeniem i wykorzystywać protokół komunikacyjny Bluetooth do komunikacji pomiędzy nimi, technologii RFID do autoryzacji użytkowników, standardy UART i SPI do komunikacji pomiędzy mikrokontrolerem a urządzeniami peryferyjnymi oraz algorytm szyfrowania RSA do ubezpieczenia komunikacji przed przechwytywaniem.

Praca składa się z rozdziałów:

1. Przegląd rozwiązań, w którym zostaną opisane dobre przykłady ofert rynkowych rozwiązujących problemy tej pracy.
2. Koncepcja urządzenia opisujący zasadę działania projektowanego urządzenia oraz wyszczególniający wszystkie funkcje, które trzeba będzie zaimplementować, aby zrealizować projekt w praktyce.
3. Projekt części sprzętowej mikrokontrolera który osobno opisuje każdy moduł fizyczny, z którego tworzone jest urządzenie, opisuje ich budowę, komunikację między sobą oraz zastosowanie w ramach tej pracy.
4. Projekt części programowej mikrokontrolera który opisuje realizację wszystkich opisanych funkcji urządzenia systemu kontroli dostępu w języku C wewnątrz oprogramowania CubeMX producenta mikrokontrolera ST.
5. Projekt aplikacji mobilnej, który opisuje implementację funkcji aplikacji mobilnej w języku JAVA w oprogramowaniu do tworzenia aplikacji mobilnych Android studio
6. Budowa prototypu opisujący różne sposoby tworzenia prototypu urządzenia.
7. Podsumowanie podsumowujący wyniki.

2. Koncepcja urządzenia

Celem tego rozdziału jest analiza wymagań dotyczących funkcjonalności urządzenia na bazie mikrokontrolera i aplikacji mobilnej, wybór technologii oraz opracowanie schematu części programowej projektu, która będzie realizować postawione zadania. Cały system zbudowany jest wokół mikrokontrolera. Konieczne jest zrozumienie różnicy między mikrokontrolerem a mikroprocesorem, ponieważ często są one mylone. Mikroprocesor jest elementem, który nie jest w stanie pracować samodzielnie – do jego działania potrzebne jest dosyć rozbudowane otoczenie, składające się z magistrali systemowej(magistrala danych, adresowa i sterująca) oraz pamięci programu, pamięci danych i układów wejścia/wyjścia, a mikrokontroler z kolei nie jest niczym innym jak mikroprocesorem zamkniętym w jednej obudowie wraz z całym niezbędnym do pracy otoczeniem, który może funkcjonować bez konieczności współpracy z układami zewnętrznymi [6].

2.1 Funkcjonalność

Stworzone w tej pracy urządzenie w połączeniu z aplikacją mobilną posiada pełną funkcjonalność wymaganą od systemu kontroli dostępu tej skali. Urządzenie sterowane przez mikrokontroler montuje się na chronionym obiekcie, podłącza do źródła zasilania i zamka, a czytnik RFID umieszcza się na stronie urządzenia dostępnej dla użytkowników, tak aby osoby nieupoważnione nie miały dostępu do urządzenia z mikrokontrolerem, ale miały dostęp do czytnika tagów i kart RFID. Urządzenie to pełni funkcję kontrolera dostępu, zamek można otworzyć poprzez przyłożenie znacznika RFID z zapisanym na nim UID użytkownika, któremu nadano wcześniej uprawnienia dostępu, lub bezpośrednio za pomocą przycisku w aplikacji mobilnej.

Urządzenie zapisuje w swojej pamięci informacje o UID użytkowników z uprawnieniami dostępu i przy każdym logowaniu rejestruje, kto i o której godzinie się zalogował, dzięki czemu administrator ma pełną информацию o historii dostępu. Za pomocą aplikacji dostępnej na urządzenia mobilne na platformie Android można skonfigurować system kontroli dostępu lub odbierać informacje z urządzenia. Rysunek 2.1 opisuje funkcjonalność aplikacji w zakresie przycisków.

Komunikacja pomiędzy mikrokontrolerem a aplikacją jest w pełni zabezpieczona za pomocą szyfrowania wiadomości RSA. Każde urządzenie generuje własną parę kluczy publicznych i prywatnych oraz wymienia się kluczami publicznymi.

Aplikacja posiada 9 przycisków i jedno pole wejściowe. Po podłączeniu do urządzenia żądane jest hasło, które sprawdza mikrokontroler. Przycisk otwierania drzwi zdalnie otwiera zamek. Za pomocą przełącznika trybu odczytu można całkowicie wyłączyć odczyt kart RFID, czyli uniemożliwić dostęp do nich, przydatny na przykład w nocy, kiedy wszyscy opuszczą chroniony obiekt. Korzystając z przycisków dodawania, usuwania i przeglądania UID użytkownika, można łatwo zarządzać uprawnieniami dostępu, wpisując w polu nazwy UID do dodawania lub usuwania, a przycisk przeglądania użytkowników pomaga dowiedzieć się, kto ma aktualnie przyznane prawa dostępu. Możliwe jest również zapisanie UID na kartę poprzez wpisanie nazwy, naciśnięcie przełącznika trybu zapisu, co z kolei spowoduje wyłączenie trybu odczytu, a potrzebny UID zostanie zapisany do każdego tagu przyłożonego do czytnika RFID. Przycisk historii pokazuje całą zarejestrowaną historię logowań użytkowników z dokładną godziną, datą i UID zalogowanego użytkownika.



Rys. 2.1 Funkcjonalność interfejsu graficznego wykonanej aplikacji mobilnej

2.2 Analiza niezbędnych funkcji

W oparciu o wymagania została opracowana część sprzętową urządzenia, która implementuje możliwość zapisu i odczytu tagów, sterowania zamkiem elektrycznym oraz komunikacji z urządzeniem mobilnym, rozpoznawania i realizacji otrzymywanych z niego poleceń oraz udostępniania przesyłanie danych pomiędzy nimi za pomocą szyfrowania. Jako algorytm szyfrowanie do bezpiecznej komunikacji pomiędzy adapterem bluetooth urządzenia i android aplikacją został wybrany Algorytm Rivesta-Shamira-Adlemana (RSA).

Aby spełnić wszystkie postawione nam zadania w formie zbliżonej do rzeczywistych produktów, urządzenie finalne musi spełniać następujące funkcje:

1. Zapisywanie w określonych blokach pamięci na znacznikach RFID identyfikatora użytkownika otrzymanego w poleceniu za pośrednictwem Bluetooth.
2. Odczytanie UID użytkownika z tagu RFID i porównanie ze strukturą użytkowników posiadających uprawnienia dostępu.
3. Dodawanie i usuwanie UID użytkowników ze struktury przechowującej je po otrzymaniu odpowiednich poleceń z aplikacji Android.
4. Otwarcie zamka, jeśli UID na tagu RFID odpowiada strukturze dozwolonych użytkowników.
5. Dodawanie nazw użytkowników wraz z odpowiednim znacznikiem czasu do struktury historii logowania, jeżeli otrzymali oni dostęp.
6. Obsługa zegara czasu rzeczywistego dla poprawnych znaczników czasu i korygowania ich danymi otrzymanymi z aplikacji na Androida.
7. Wysyłanie i odbieranie wiadomości poprzez interfejs szeregowy Bluetooth.

8. Generowanie prywatnego i publicznego kluczy RSA w oparciu o podane liczby pierwsze dla bezpiecznej komunikacji.
9. Mechanizm wymiany kluczy publicznych i przechowywania ich w strukturze.
10. Szyfrowanie danych kluczem prywatnym przed przesaniem poprzez interfejs szeregowy Bluetooth.
11. Odszyfrowanie zaszyfrowanych danych otrzymanych poprzez Bluetooth przy użyciu klucza publicznego aplikacji mobilnej.
12. Przechowywanie hasła wymaganego do dostępu do aplikacji mobilnej i sprawdzanie go po otrzymaniu.
13. Otwarcie elektrozamka po otrzymaniu odpowiedniego polecenia.
14. Włączenie lub wyłączenie trybu odczytu znaczników RFID po otrzymaniu odpowiedniego polecenia.
15. Kompilowanie w pakiet danych i wysyłanie listy UID lub historii logowania przez BT.

Android aplikacja musi spełniać następujące funkcje:

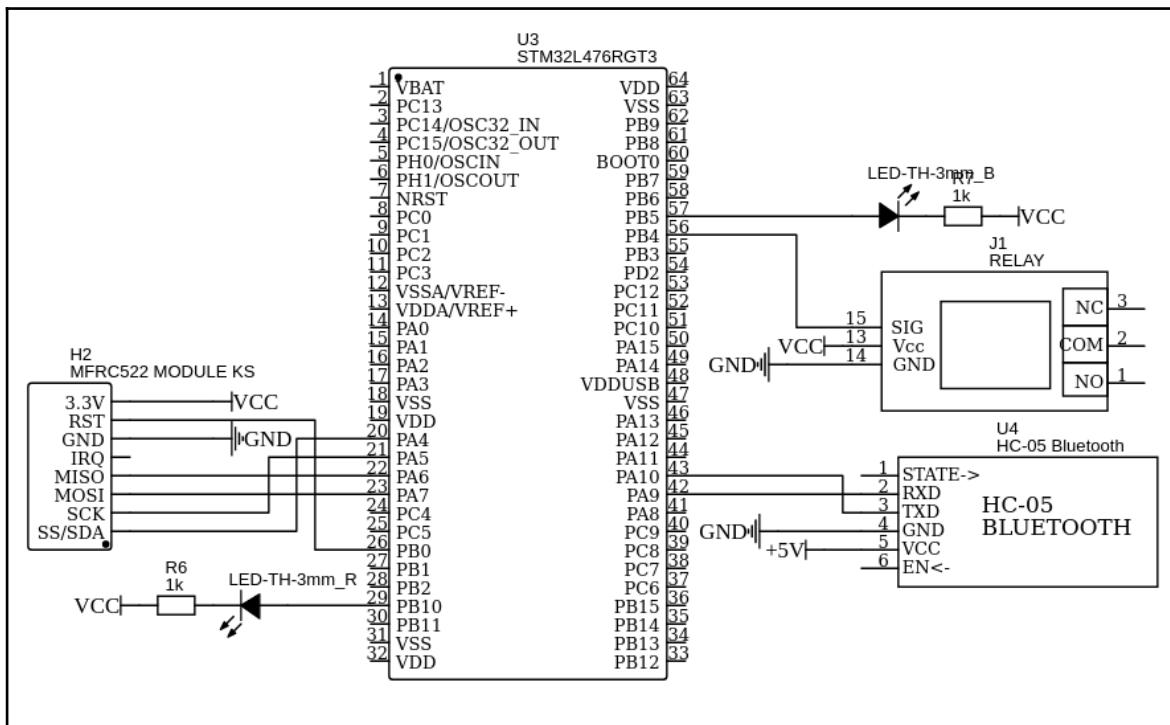
1. Zapytywanie lokalnego adaptera Bluetooth o sparowanych urządzeniach Bluetooth i Ustanowienie kanałów RFCOMM.
2. Łączenie się z adapterem bluetooth urządzenia poprzez wykrywanie usług.
3. Przesyłanie danych do i z sparowanego urządzenia.
4. Uwierzytelnianie hasła poprzez wysłanie wprowadzonego przez użytkownika hasła do mikrokontrolera i otrzymanie odpowiedzi.
5. Generowanie prywatnego i publicznego klucze RSA w oparciu o podane liczby pierwsze dla bezpiecznej komunikacji.
6. Mechanizm wymiany kluczy publicznych i przechowywania otrzymanego klucza w odpowiedniej zmiennej.
7. Szyfrowanie danych kluczem prywatnym przed przesaniem poprzez interfejs szeregowy Bluetooth.
8. Odszyfrowanie zaszyfrowanych danych otrzymanych poprzez Bluetooth przy użyciu klucza publicznego mikrokontrolera.
9. Zbieranie wiadomości do wysłania do odpowiednich pakietów z dodaniem bajtów do wiadomości o określonej koniecznej długości.
10. Demonstracja otrzymanych danych o historii i dopuszczonych użytkownikach w postaci okien.
11. Ustawianie pozycji przełączników odczytu/zapisu znacznika na podstawie otrzymanych danych o ich stanie.
12. Ciąg wejściowy UID użytkownika umożliwiający przesłanie go z komunikatem o zapisywaniu, usunięciu lub dodaniu do listy.
13. Wysłanie aktualnego czasu przez Bluetooth.

Ponieważ nie są określone wymagania dotyczące przechowywania UID użytkowników i historii dostępu w trwałej, niezależnej pamięci, urządzenie nie będzie ich obsługiwać. Zatem nazwy autoryzowanych użytkowników i historia dostępu będą przechowywane wyłącznie w dynamicznej pamięci RAM. Jeśli więc urządzenie straci zasilanie, wszystkie nazwy użytkowników i historia zostaną utracone.

3. Projekt części sprzętowej mikrokontrolera

W tej sekcji opisano szczegóły technicznej części pracy. Schemat połączeń elektrycznych, mikrokontroler sterujący całym systemem, peryferia i moduły, komunikacja pomiędzy nimi oraz metody zasilania. Poniższy schemat elektryczny (rys 3.1) przedstawia podłączenie modułów, które posłużą do stworzenia urządzenia kontroli dostępu do pinów mikrokontrolera. Na schemacie przekaźnik pokazany jest jako gotowy moduł, który jest powszechnie dostępny w sprzedaży, alternatywnie w części poświęconej tworzeniu prototypu urządzenia zostanie omówiony schemat tego modułu, który pozwala nam wbudować przekaźnik w płytę zamiast podłączania jako moduł osobny.

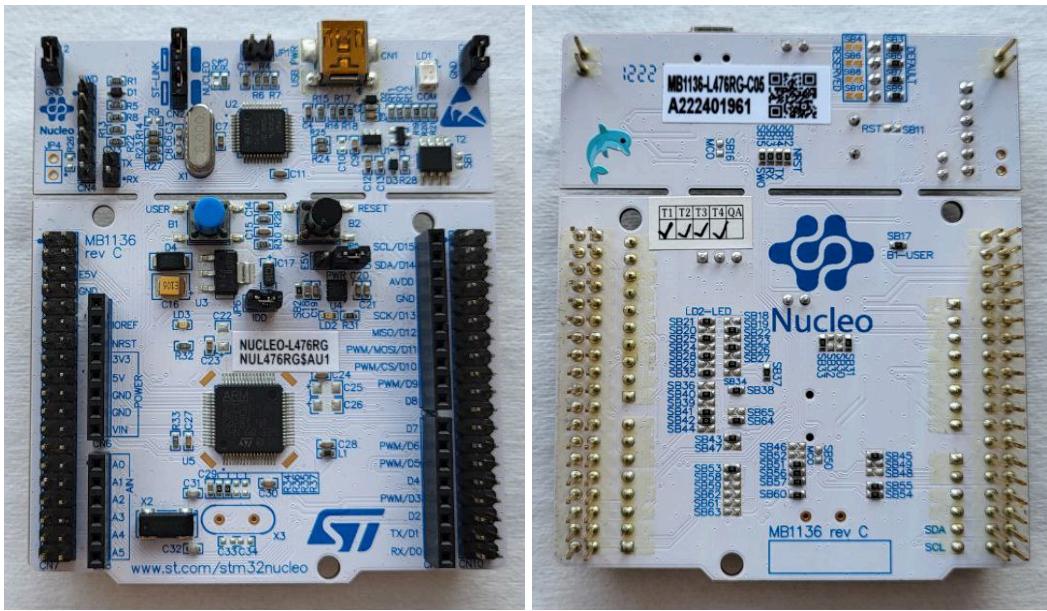
Użycie konkretnych pinów zostanie omówione w rozdziale poświęconym oprogramowaniu mikrokontrolera, należy jednak pamiętać, że można je zmieniać, gdyż mikrokontroler posiada redundantne interfejsy.



Rys. 3.1 Schemat elektryczny połączeń modułów peryferyjnych do STM32L476RG

3.1. Procesor

Do wykonania tej pracy wykorzystano płytę rozwojową STM32 NUCLEO-L476RG (Rys. 3.2) opartą na MCU STM32L476RG. Płyta posiada oscylator kwarcowy 32,768 kHz, złącza przedłużające ST Morpho zapewniające pełny dostęp do wszystkich wejść i wyjść STM32 w połączeniu ze złączem rozszerzeń ARDUINO® Uno V3, wbudowany debugger i programator ST-LINK z możliwością ponownego wyliczenia USB, pamięć masową, wirtualny port COM i port debugowania [7].



Rys. 3.2 Wygląd NUCLEO-L476RG

Urządzenia STM32L476xx to mikrokontrolery o ultraniskim poborze mocy oparte na wysokowydajnym 32-bitowym rdzeniu RISC Arm® Cortex®-M4, pracujące z częstotliwością do 80 MHz. Urządzenia STM32L476xx posiadają wbudowanej flash pamięci (do 1 MB, do 128 KB SRAM), elastyczny kontroler pamięci zewnętrznej dla pamięci statycznych, interfejs pamięci flash Quad SPI oraz szeroką gamę ulepszonych wejść/wyjść i urządzeń peryferyjnych podłączonych do dwóch magistral APB, dwóch magistrali AHB i 32-bitowej matrycy magistrali multi-AHB [8]. Wyposażone są również w standardowe i zaawansowane interfejsy komunikacyjne: trzy I2C(InterIntegrated Circuit), trzy SPI, trzy USART, dwa UART i jeden UART małej mocy, dwa SAI, jeden SDMMC, jeden CAN (Controller Area Network), jeden USB OTG o pełnej prędkości i jeden SWPMI (Single wire protocol master interface).

W celu optymalizacji zużycia energii mikrokontrolery STM32L4 wykorzystują dynamiczną adaptację napięcia wewnętrznego regulatora step-down w zależności od częstotliwości pracy sterownika: aby zapewnić wysoką wydajność, zwiększone jest napięcie, aby oszczędzać energię, częstotliwość i, dlatego zużycie jest zmniejszone [8]. Zegar czasu rzeczywistego RTC został zaprojektowany jako niezależny licznik ze sprzętową obsługą kalendarza. Można zachować sekundy, minuty i godziny, dzień tygodnia, datę, miesiąc i rok, prezentowane w formacie dziesiętnym binarnym BCD. Funkcja cyfrowej kalibracji pozwala na kompensację niedokładności rezonatora kwarcowego.

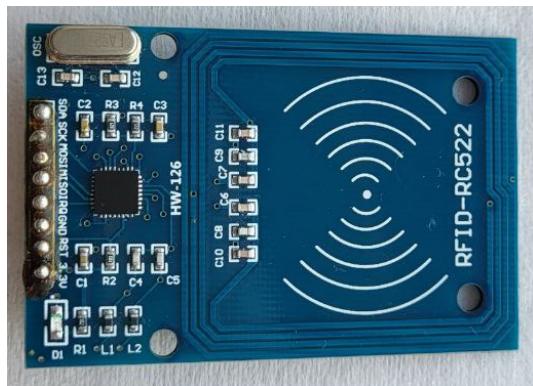
Kontroler zawiera trzy moduły SPI. Obsługuje pracę w trybie Master lub Slave z transmisją danych w trybie full duplex (na trzech liniach), half duplex (na dwóch liniach z dwukierunkową linią danych) i simplex (na dwóch liniach z jednokierunkową linią danych) z szybkością do 40 Mbit/s. Mikrokontroler posiada aż trzy moduły USART i dwa moduły UART, które zapewniają prędkość przesyłu danych do 10 Mbit/s. Możliwa jest współpraca z urządzeniami LIN, IrDA i kartami inteligentnymi. Oprócz standardowego USART, deweloper ma dostęp do LPUART małej mocy. Ze względu na oszczędność energii, ponieważ producent określa STM32L476 jako mikrokontroler o niskim poborze mocy, posiada wiele trybów związanych z zasilaniem: run, domyślny, sleep, w trybie uśpienia zatrzymuje się tylko procesor, Low-power run, minimalizując działanie regulatora prądu, Low-power sleep, Stop, Standby i Shutdown.

3.2. Moduł czytnika RFID

Czytniki RFID to urządzenia przeznaczone do bezdotykowego odczytu i przetwarzania danych z tzw. tagów RFID. System RFID składa się z trzech elementów: anteny lub cewki, transiwera (z dekoderem), transpondера (znacznika RF) zaprogramowanego elektronicznie z unikalnymi informacjami [9]. Czytnik, składający się z anteny i transiwera emisuje do otaczającej go przestrzeni fale radiowe o określonej częstotliwości, które są odbierane przez znacznik radiowy. Dzięki energii tych fal karta generuje energię elektryczną wystarczającą do zasilenia wbudowanego chipa, który w odpowiedzi emisuje sygnał radiowy z wszystym w kartę zaszyfrowanym kodem identyfikacyjnym. Kod ten z kolei jest odbierany przez czytnik, odszyfrowywany i przesyłany w przypadku opisanym w tej pracy do mikrokontrolera STM32L476RG.

Standardowe częstotliwości, na jakich pracują czytniki RFID: LF – 135 kHz, HF – 13,56 MHz, mikrofale – powyżej 433 MHz, 2,45 GHz i 5,8 GHz. Częstotliwość wpływa na szybkość przesyłania danych, zasięg odczytu oraz określa rodzaj barier zakłócających komunikację pomiędzy czytnikiem RFID a tagiem. Im wyższa częstotliwość, tym większa prędkość przesyłania danych i większa odległość odczytu, ale w przypadku mikrofal zwiększa się stopień absorpcji promieniowania przez materiały takie jak metal czy woda.

Większość znaczników RFID składa się z dwóch części. Pierwszy to układ scalony służący do przechowywania i przetwarzania informacji, modulowania i demodulowania sygnału o częstotliwości radiowej i anteny do odbioru i transmisji sygnałów. W tej pracy jako moduł RFID wykorzystano RC 522 pokazany na rysunku 3.3.



Rys. 3.3 Wygląd modułu RFID RC522

Moduł RC522 oparty jest na chipie MFRC522 firmy NXP, który zapewnia współpracę ze znacznikami HF (na częstotliwości 13,56 MHz) i obsługuje interfejsy SPI, UART oraz I2C [10], które znajdują się w module terminala. W obecnej pracy będzie on podłączony poprzez interfejs SPI, gdyż UART nie pozwala na podłączenie więcej niż jednego urządzenia typu slave, którym w przykładzie opisanym w tej pracy jest czytnik RFID RC522. Tworzone urządzenie musi mieć możliwość podłączenia wielu urządzeń, gdyż w większości zastosowań kontroli dostępu jest to standardowa praktyka, przynajmniej dlatego, że konieczne może być umieszczenie czytników po obu stronach drzwi lub sterowanie kilkoma drzwiami przez jeden główny mikrokontroler.

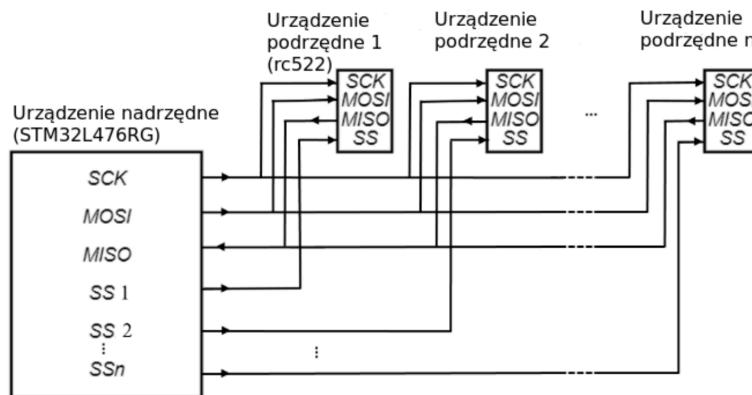
Interfejs SPI służy do prostego i niedrogiego szybkiego połączenia. Jest to synchroniczna magistrala 4-przewodowa. W mikrokontrolerach STM32 interfejs SPI można skonfigurować dla pracy z tylko jedną linią danych [11].

Szeregowy interfejs peryferyjny to synchroniczny, szeregowy, pełnodupleksowy interfejs wymiany danych pomiędzy urządzeniem nadzorowanym, w przypadku tej pracy

mikrokontrolerem STM32L476RG, a kilkoma urządzeniami podłączonymi, w przypadku opisanym w tej pracy modułami rfid rc522. Zawiera cztery linie cyfrowe (Rys. 3.4),

Linie cyfrowe SPI:

1. SCK – Serial Clock, linia taktowania przesyłająca sygnał zegara szeregowego z urządzenia nadzorowanego do urządzenia podłączanego.
2. MISO – Master Input Slave Output, ta linia przesyła dane z urządzeń podłączanych do urządzenia nadzorowanego.
3. MOSI – Master Output Slave Input, linia ta odpowiedzialna jest za przesyłanie danych z urządzenia master do urządzenia slave.
4. SS – Slave Select, na tej linii ustawiany jest poziom aktywnego sygnału, za pomocą którego urządzenie nadzorowane wybiera, z którym z urządzeń podłączanych chce się komunikować.

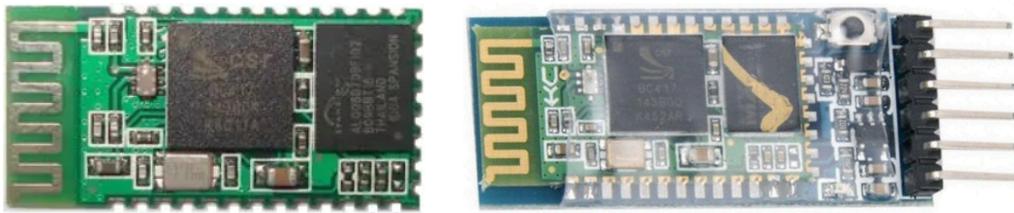


Rys. 3.4 Schemat połączeń poprzez interfejs SPI na przykładzie tej pracy

Urządzenie nadzorowane wysyła dane linią MOSI. Urządzenia podłączane odpowiadają poprzez linię MISO. Urządzenie nadzorowane generuje sygnał synchronizacji, który jest przesyłany linią SCK do urządzenia podłączanego. SS służy do adresowania urządzeń na odpowiednich liniach.

3.3. Moduł Bluetooth

HC-05 to moduł elektroniczny zbudowany w oparciu o mikroukład BC417 serii BlueCore4-Ext™ firmy Cambridge Silicon Radio [12], jest pokazany na rys. 3.5. Moduł ten pozwala w dość prosty sposób zrealizować transfer danych pomiędzy urządzeniem mobilnym, a urządzeniem mikroprocesorowym podłączonym do HC-05 (w tym przypadku mikrokontrolerem STM32L476RG). Moduły te produkowane są przez chińską firmę Guangzhou HC Information Technology w przystępnej cenie. Znaczące uproszczenie polega na ujawnieniu użytkownikowi końcowemu specyfiki protokołu radiowego. Cała wymiana danych odbywa się poprzez prostszy i bardziej znany interfejs przewodowy, w tym przypadku TTL RS232. W tej pracy używamy wersji na płytce drukowanej z wbudowanym adapterem bluetooth, w której główne piny sterujące są połączone z listwą pinów PLS, ponieważ jest to bardziej niezawodny sposób niż samodzielny montaż dodatkowego modułu bluetooth. Płytki modułu zawierają dwa mikroukłady w obudowie BGA: mikroukład BC417, układ sterujący Bluetooth i pamięć Flash 8 Mbit. Program odpowiedzialny za rolę i charakterystykę jest przechowywany w pamięci. Moduł



Rys. 3.5 Wygląd modułu hc-05 bez adaptera i z adapterem (źródło: [13])

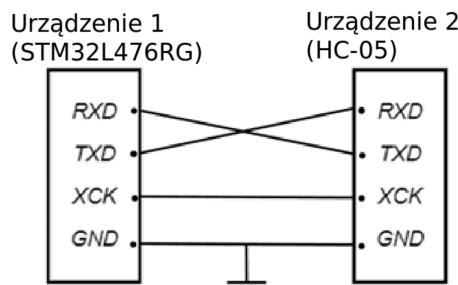
Specyfikacja HC-05:

- Napięcie zasilania: 3,3 V do 6,0 V,
- Prąd roboczy: 30mA,
- Zakres działania: maks. 10 m,
- Obsługiwana szybkość transmisji: 9600, 19200, 38400, 57600, 115200, 230400, 460800,
- Standard: Bluetooth 2.0,
- Zgodny ze standardem IEEE 802.15.1,

HC-05 posiada 6 standardowych pinów 2.54 mm:

- | | | |
|-------------------|---------------------------|-----------------------------|
| •VCC – zasilanie, | •GND – masa, | •TXD, RXD – interfejs UART, |
| •STATE – stan, | •EN – włącz/wyłącz moduł; | |

Moduł HC-05 podłączany jest do mikrokontrolera poprzez interfejs USART (rysunek 3.7). Interfejs USART, podobnie jak jego wyłącznie asynchroniczny odpowiednik UART, ma tę istotną cechę, że w odróżnieniu od SPI służy jedynie do połączenia ze sobą dwóch urządzeń, dlatego w niniejszym projekcie zaistniała konieczność wykorzystania obu interfejsów. W tej pracy nie ma potrzeby stosowania kilku modułów bluetooth, dlatego ta wada interfejsu USART nie dotyczy tej pracy. Chociaż moduł HC-05 posiada możliwość pracy w trybie synchronicznym, nadal będziemy korzystać z trybu asynchronicznego. W trybie asynchronicznym protokół ten wykorzystuje tylko dwa przewody, a mianowicie Rx i Tx. Ponieważ synchronizacja nie jest tutaj wymagana, oba urządzenia muszą, aby działać, korzystać z niezależnych wewnętrznych systemów taktowania. Linia Tx jednego urządzenia przesyła dane do linii Rx drugiego urządzenia i podobnie Tx tego ostatniego przesyła dane do Rx pierwszego urządzenia. W ten sposób następuje wymiana danych.



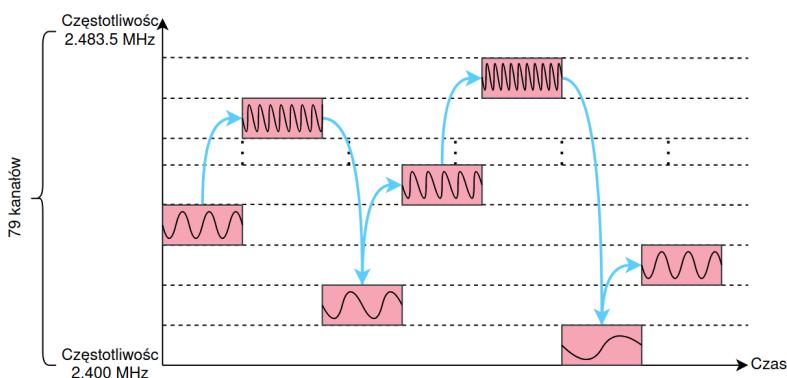
Rys. 3.6 Schemat połączenia poprzez interfejs USART na przykładzie tej pracy

W trybie asynchronicznym istnieje bardzo ważna cecha „szybkości transmisji”, która pomaga tym urządzeniom pozostać w trybie synchronizacji, ustalając prędkość wymiany danych. Szybkość przesyłania danych w obu urządzeniach musi być taka sama, aby zapewnić ich prawidłowe działanie. Szybkość transmisji zostanie ustawiona w procesie deklarowania peryferii na etapie tworzenia części programowej projektu.

W połączeniu z elementami pasywnymi i aktywnymi, BC417 tworzy system Bluetooth o zwiększonej szybkości przesyłania danych (EDR) do 3 Mbit na sekundę. Zastosowanie

stosu oprogramowania CSR Bluetooth zapewnia pełną zgodność systemu z wersją 2.0 specyfikacji transmisji danych i głosu. BC417 zawiera automatyczną kalibrację i wbudowane procedury kontroli testów (BIST), aby uprościć rozwój. Główną cechą HC-05 jest łatwość konfiguracji, jednakże maksymalny możliwy zasięg transmisji pakietów danych bez strat nie przekracza 20 metrów. Dlatego też zastosowanie modułów HC-05 jest wskazane pod warunkiem ich stosowania na krótkich dystansach.

Główna cecha tego modułu jest możliwość pracy w trybie Master i Slave. Organizacja komunikacji pomiędzy nadajnikiem (Master) i odbiornikiem (Slave) następuje dzięki zastosowaniu technologii FHSS, czyli pseudolosowej regulacji częstotliwości pracy. Proces transmisji danych przy zastosowaniu tej technologii praktycznie się nie zmienia, jednakże aby zapewnić stabilność komunikacji w warunkach dużego szumu eteru, częstotliwość nośna sygnału zmienia się okresowo w określonej, z góry ustalonej kolejności. Przykładowo na rysunku 3.6 można zaobserwować proces zmiany częstotliwości w zakresie 79 kanałów.



Rys. 3.7 Zasada działania technologii FHSS (opracowano na podstawie [14])

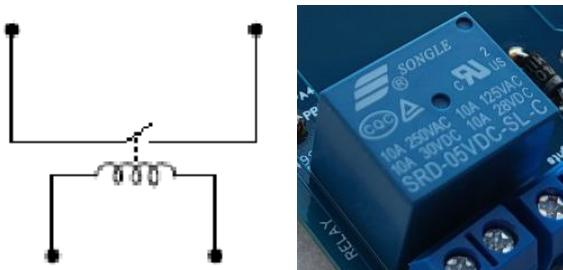
Unikalna sekwencja zmian częstotliwości musi być taka sama dla nadajnika i odbiornika. Dostrojenie odbiornika i nadajnika do tej samej pseudochaotycznej kombinacji zmian częstotliwości roboczej odbywa się poprzez przesłanie specjalnych sygnałów synchronizujących, które wyznaczają momenty początkowe kolejnego cyklu zmian częstotliwości.

3.4. Przekaźnik i zamek

Przekaźnik służy do dostarczania do podłączonych urządzeń wyższych napięć niż te, które można uzyskać z pinów GPIO mikrokontrolera. Ma w swojej konstrukcji elektromagnes, połączeniu którego uruchamia się mechanizm, który zamyka lub otwiera styki. Na rynku dostępnych jest wiele wersji przekaźników, różniących się układem styków, co z kolei odzwierciedla zachowanie przełączania po przyłożeniu napięcia do cewki przekaźnika. W przypadku tej pracy rozważany jest przekaźnik ze schematem styków „1A”, pokazanego na rys. 3.8, w nazwie którego 1 oznacza liczbę przełączanych grup, a litera A oznacza typ przełączania. Przekaźniki typu 1A mają jedną grupę normalnie otwartą, to znaczy po przyłożeniu napięcia do uzwojenia przekaźnika jeden styk jest zwarty.

Charakterystyki przekaźnika to:

- Napięcie cewki: 5 V
- Maksymalny prąd: 10 A / 15 A
- Maksymalne napięcie styków: 250 VAC przy 10 A / 125 VAC przy 10 A



Rys. 3.8 Schemat przekaźnika 1A (pracowano na podstawie [15]) i wygląd przekaźnika użytego w tej pracy

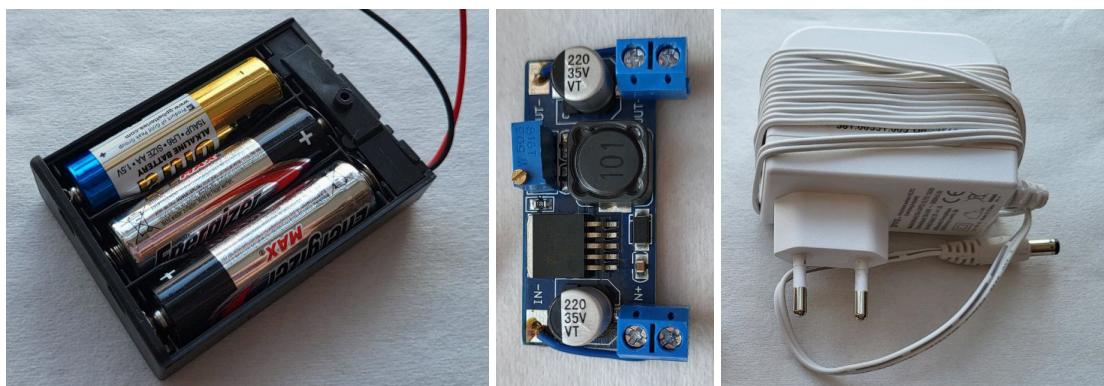
Zasilanie i zamek są podłączone po drugiej stronie przekaźnika, w tym przypadku może to być zamek 12 V lub 5 V (Rys. 3.9).



Rys. 3.9 Wygląd zamków 5V i 12V użytych w tej pracy

3.4. Zasilanie

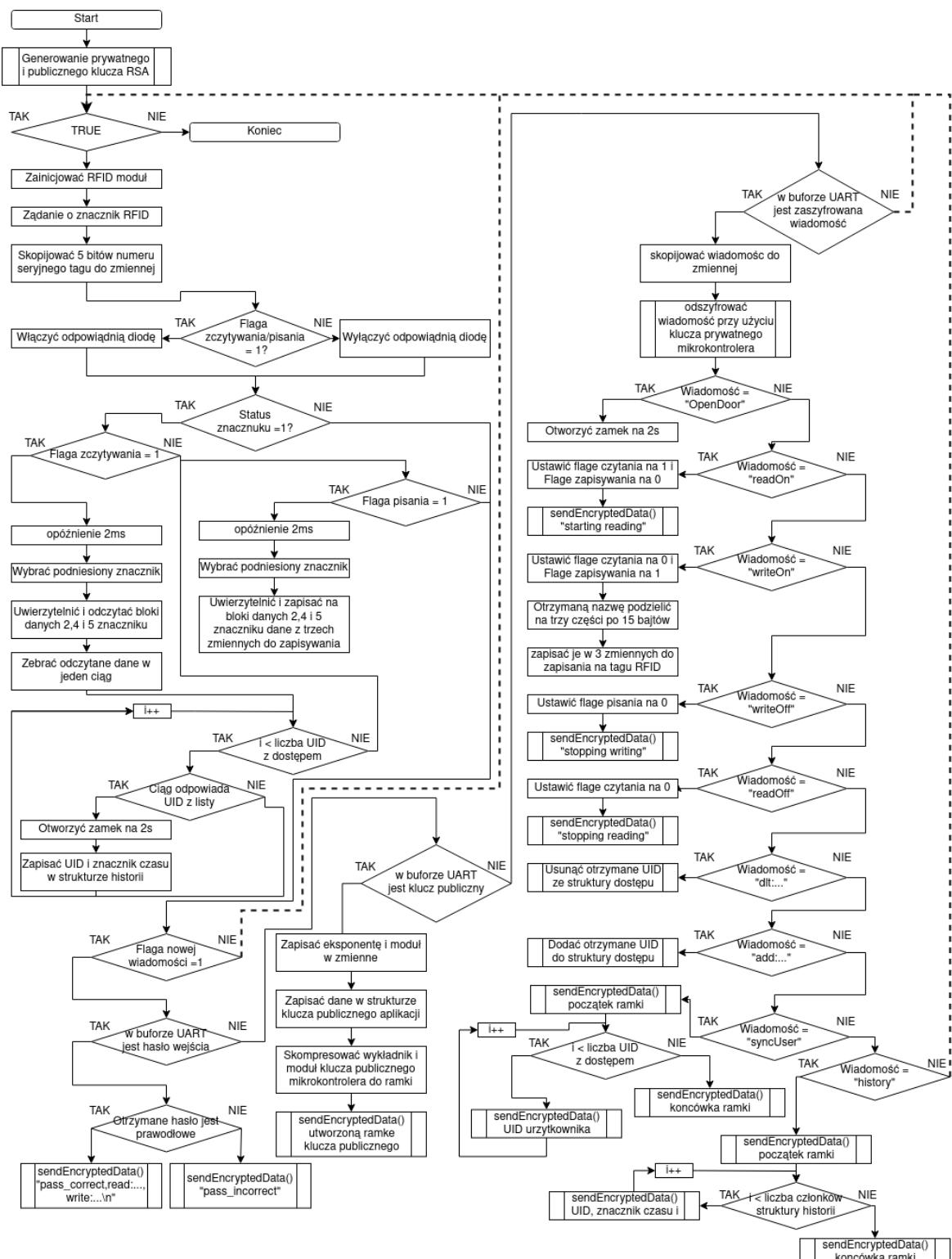
Istnieją dwie możliwości podłączenia urządzenia do zasilania (Rys. 3.10). Pierwsza opcja, przenośna, polega na podłączeniu mikrokontrolera bezpośrednio do zestawu akumulatorów wytwarzającego napięcie 5V. Podłącza się go po prostu do pinów E5V i GDN mikrokontrolera, a do przekaźnika podawane jest to samo napięcie 5V. Druga opcja jest bardziej skomplikowana, ponieważ większy zamek wymaga napięcia 12V. W tym przypadku podłączamy mikrokontroler poprzez przetwornice step-down LM2596 do zasilacza 5V i dostarczamy pełne 12V na przekaźnik sterujący zamkiem.



Rys. 3.10 Przedstawione opcje zasilania urządzenia

4. Projekt części programowej mikrokontrolera

W tej części omówione zostanie projektowanie programu mikrokontrolera, łącznie z inicjalizacją urządzeń peryferyjnych i programową realizacją wszystkich zadań przypisanych do urządzenia. Aby ułatwić zrozumienie algorytmu działania urządzenia, stworzono schemat blokowy programu mikrokontrolera (Rys. 4.1).



Rys. 4.1 Schemat blokowy oprogramowania mikrokontrolera

4.1. Inicjalizacja sterownika

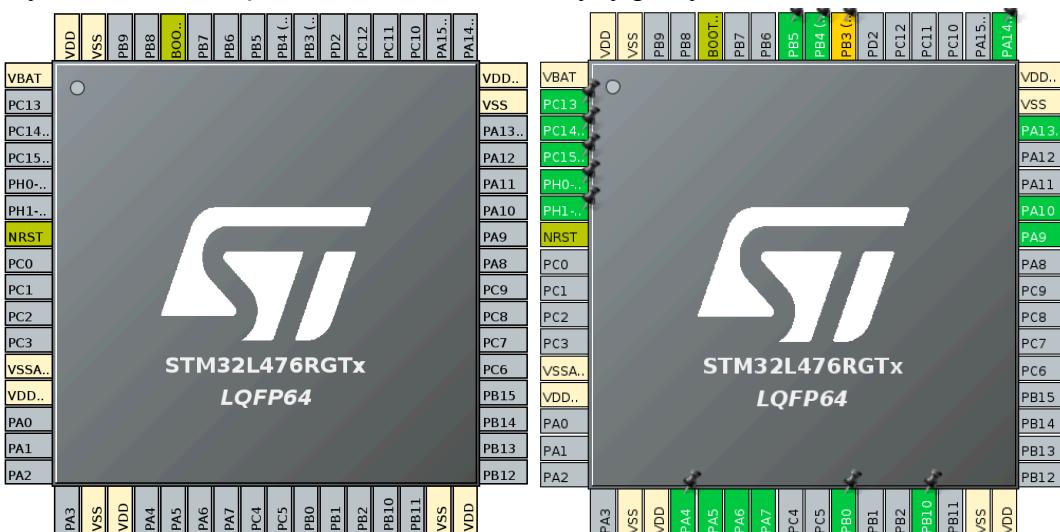
Do konfiguracji i generowania kodu wykorzystujemy środowisko programistyczne stworzone specjalnie dla procesorów ARM STMCubeMX oraz towarzyszącą mu bibliotekę HAL. Środowisko programistyczne CubeMX umożliwia interaktywny wybór jednego z wielu modeli STM32 na komputerze, określenie wejść mikrokontrolera do podłączenia sprzętu zewnętrznego oraz skonfigurowanie urządzeń peryferyjnych mikrokontrolera i ustawienie ich trybów pracy. W wyniku projektowania powstaje program roboczy mikrokontrolera.

STM32CubeMx wykorzystuje dwie biblioteki, starszą CMSIS i nową HAL (Hardware Abstraction Layer), warstwa abstrakcji sprzętu. Biblioteka HAL zawiera kilka folderów: folder Drivers zawiera sterowniki do różnych płyt deweloperskich opartych na STM32, folder Middlewares zawiera biblioteki do pracy ze sprzętem audio, port USB, a także biblioteki FatFs, FreeRTOS i inne, przystosowane do użytku w połączeniu ze sterownikiem HAL STM32L4xx. Po uruchomieniu programu STM32CubeMx można przejść do okna, w którym można wybrać model MK(Rys. 4.2).



Rys. 4.2 Wybór mikrokontrolera STM32L476RG-T3 w oprogramowaniu CubeMX

Po wybraniu modelu MK, po prawej stronie ekranu pojawi się obraz ze wszystkimi wyjściami mikrokontrolera(Rys. 4.3), a po lewej stronie pojawi się lista wszystkich dostępnych urządzeń peryferyjnych. Jeśli wybrać na interfejsie wejścia mikrokontrolera, pojawia się menu, za pomocą którego możemy skonfigurować urządzenia peryferyjne MK, na przykład ustawić częstotliwość taktowania i tryby pracy timerów.



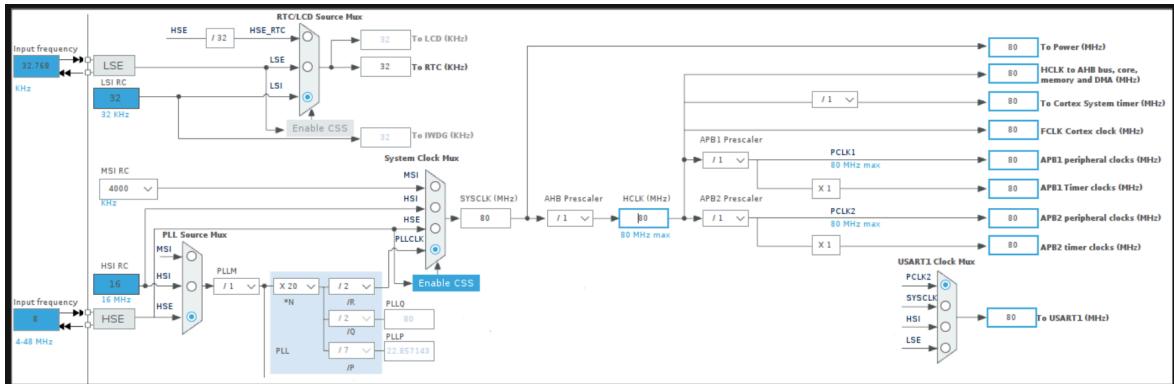
Rys. 4.3 Wizualizacja pinów mikrokontrolera przed i po inicjalizacji wszystkich peryferyjnych w oprogramowaniu CubeMX

Następnie należy wejść w pozycję menu Projekt i podać nazwę projektu oraz ścieżkę jego zapisania. Wybierając mikrokontroler, środowisko programistyczne pozwala na inicjowanie wszystkich standardowych urządzeń peryferyjnych, to jest zalecany wybór, ale

w przypadku potrzeby wybrania innej opcji, będzie konieczne ustawienie identyfikatora zintegrowanego środowiska programistycznego (Integrated Development Environment) w celu wygenerowania kodu programu. W tym celu w menu Projekt należy wybrać podmenu Ustawienia i w podpunkcie menu SYS ustawić typ interfejsu debugującego, zwykle szeregowego.

Następnie zostanie omówione wizualne konfigurowanie wyjść niezbędnych do wykonania zadań tej pracy, podłączenie wszystkich niezbędnych jednostek peryferyjnych i ich konfiguracja. zakładce Connectivity w Pinout & Configuration musimy podłączyć USART1 w trybie asynchronicznym, ustawiając jego prędkość transmisji na 9600 bitów na sekundę długość słowa na 8 bitów, koniecznie włączając przerwania globalne gdyż za ich pomocą będziemy mogli uzyskać dostęp do odbieranych poprzez transmisję Bluetooth danych i podłączyć piny mikrokontrolera PA9 jako kanał komunikacyjny TX interfejsu USART odpowiedzialny za wysyłanie komunikatów oraz pin PA10 jako kanał RX odpowiedzialny za odbieranie informacji. Można zastosować do tego zadania inne piny, które mogą być kanałami USART, są ich dwie pary, zgodnie z wyborem projektującego. Następnie zostały podłączone peryferii odpowiedzialne za komunikację z modułem RFID rc522, SPI1 w trybie Full-Duplex Master, ustawiając preskaler dla szybkości przesyłania danych na 8 bitów, nie uwzględniając przerwy globalnej lub sprzętowego sygnału NSS, format ramki powinien być Motorola, a rozmiar danych powinien wynosić 8 bitów, a jako piny mikrokontrolera niezbędne do korzystania z SPI deklarujemy PA5 jako SCK, PA6 jako MISO i PA7 jako MOSI. W zakładce Timery w Pinout & Configuration została podłączona zegar czasu rzeczywistego RTC niezbędny do utrzymania możliwości posiadania poprawnych znaczników czasu na zapisanej przez programę historii osób wchodzących przez podłączony zamek, został aktywowano źródło zegara i kalendarza, wybierając dogodny dla użytkownika format czasu - 24, asynchroniczny predivider należy ustawić na maksymalny 127 i ustawiając dowolny czasu, ponieważ nie będzie potrzebny dlatego że podczas procesu synchronizacji z aplikacją mobilną na Androidzie do mikrokontrolera przesyłane będą prawidłowe wskaźniki czasu rzeczywistego i daty . W zakładce piny GPIO należy ustawić 5 pinów jako GPIO_Output, wszystkie o niskim poziomie sygnału wyjściowego i wszystkie w trybie push-pull, w tym 2 dla diod LED sygnalizujących tryb odczytu i zapisu tagów rfid, jeden przekaźnik sterujący otwieraniem zamku solenoidalnego i dioda LED wskazująca to, PA4 jako SDA do komunikacji SPI z podłączonym modułem RFID rc522 i PB0 jako RST w tym samym celu. W zakładce System Core w Pinout & Configuration zostało podłączone RCC, używając rezonatorów kryształowo-ceramicznych zarówno dla zegarów high-speed, jak i low-speed.

Następnie wizualnie należy skonfigurować drzewo zegara (Rys. 4.4) za pomocą narzędzia Clock Wizard. Do taktowania wybrać HSE (High Speed Clock) i ustawić go na maksymalną częstotliwość, na jaką pozwala sterownik (80 Hz).



Rys. 4.4 Konfiguracja taktowanie w utilicie Clock Wizard

W zasadzie każde urządzenie, będące elementem mikrokontrolera STM32, wymaga do poprawnego działania włączenia dla niego sygnału zegarowego[8]. W praktyce pozwala to obniżyć pobór energii mikrokontrolera, ponieważ nawet porty wejścia/wyjścia wymagają włączenia sygnałów zegarowych[8].

Następnie, aby rozpocząć pracę z kodem, zostało wybrano środowisko programistyczne STM32CubeIDE i naciśnięto przycisk Generuj kod źródłowy oraz wybrano biblioteki niezbędne do pracy. Jeśli otworzysz folder z projektem, zobaczysz w nim nowe pliki i foldery. Folder Drivers będzie zawierał podłączone biblioteki, foldery Src i Inc będą zawierać pliki z kodem inicjującym. Plik definiuje funkcje MX_GPIO_Init() i MX_ADC1_Init(), które zawierają inicjalizację i konfigurację urządzeń peryferyjnych MK, które zostały wybrane podczas tworzenia projektu w Cube. Dalsze działania zostaną opisane wewnątrz środowiska programistycznego STM32CubeIDE.

Ważne jest wskazanie, że ze standardowych bibliotek w tworzonej aplikacji wykorzystane zostały:

•stdio.h •string.h •stdint.h •stdbool.h •stdlib.h •math.h

Stdint.h służy w funkcjach dodawania i usuwania UID do i ze struktur je przechowujących, stdlib.h służy do możliwości zwalniania dynamicznie alokowanej pamięci, unikając wyjątkowo niepożądanych wycieków pamięci, math.h do obliczeń matematycznych, które zajmujemy się generowaniem kluczy RSA i szyfrowaniem wiadomości, a pozostałe biblioteki są w większości miejsc wykorzystywane szerzej.

4.2. Obsługa modułu czytnika RFID

Aby opisać oprogramowanie służące do obsługi modułu RFID RC522, należy omówić powiązane ze sobą części kodu, które realizują następujące funkcje opisane w rozdziale 2:

1. Zapisywanie w określonych blokach pamięci na znacznikach RFID identyfikatora użytkownika otrzymanego w poleceniu za pośrednictwem Bluetooth.
2. Odczytanie UID użytkownika z tagu RFID i porównanie ze strukturą użytkowników posiadających uprawnienia dostępu.
3. Otwarcie zamka, jeśli UID na tagu RFID odpowiada strukturze dozwolonych użytkowników.
4. Dodawanie nazw użytkowników wraz z odpowiednim znacznikiem czasu do struktury historii logowania, jeżeli otrzymali oni dostęp.

Aby moduł rc522 mógł działać musimy posiadać bibliotekę, która będzie zawierała możliwości serwisowe gdyż moduł nie posiada mikrokontrolera osobistego zdolnego zautomatyzować jego pracę. Istnieje ogólnodostępna biblioteka dla mikrokontrolerów z rodziny Arduino opisująca ich specyficzną architekturę, składnię i język programowania C++. Do realizacji tej pracy istniała możliwość wykorzystać środowisko programistyczne Arduino, jednak z przeprowadzonych obserwacji wynika, że obecna praca wymaga rozwiązań niższego poziomu i języka programowania dostarczonego nam przez STMicroelectronics. Na tej podstawie dostępne są do wyboru dwa rozwiązania – ręczne przepisanie biblioteki lub skorzystanie z wcześniej przepisanej biblioteki. Ze względu na wydajność i szybkość zostało wybrano drugie podejście [16]. Biblioteka ta zawiera funkcje przetwarzania danych otrzymanych poprzez SPI, adresowania niezbędnego do uwierzytelnienia, czyli sprawdzenia próby dostępu do prezentowanej karty za pomocą klucza i wskazania pozytywnego statusu, jeśli wszystko się zgadza, oraz funkcje odczytu i zapisu klucza.

Syntaks korzystania z biblioteki pokazany jest w listingu 4.1. Najpierw musimy zainicjować wybrane urządzenia peryferyjne za pomocą funkcji MFRC522_Init(). Podczas

procesu testowania znaleziono konieczność przeprowadzenia inicjalizacji w każdym cyklu pracy programu, co wyeliminuje problemy z nieprawidłowo wybranym kluczem. Zalecane jest również ograniczenie każdego cyklu programu opóźnieniem, w tym przypadku użyto dwóch milisekund. Podnosząc znacznik RFID, program identyfikuje go w pętli głównej poprzez następujący kod programowy

Listing 4.1 Inicjalizacja, wybieranie, i autentykacja znaczniku RFID

```
while (1){
    MFRC522_Init();
    status = MFRC522_Request(PICC_REQIDL, str);
    status = MFRC522_Anticoll(str);
    memcpy(serNum, str, 5);
    if (status == MI_OK){
        MFRC522_SelectTag(str);
        MFRC522_Auth(PICC_AUTHENT1A, 24, KEY, serNum);
    ...
}}
```

w którym PICC_REQIDL jest rejestrem 0x26 odpowiadającym za monitorowanie aktywności anteny, a str jest zmienną buforową w formacie uint8_t, jak i bufor także status, który opisuje stan dostarczonego tagu i może wskazywać obecność, brak lub problem z dostarczonym tagiem. Po czym wybieramy znaleziony tag i przeprowadzamy uwierzytelnienie za pomocą klucza, który w tym przykładzie można zadeklarować jako uint8_t tablicę sześciu 0xFF. Przy uwierzytelnianiu, a także później przy zapisie i odczycie konieczne jest wskazanie bloku pamięci, do którego będą zapisywane informacje, w tym przypadku 24, w sumie jest 15 sektorów pamięci po 4 bloki pamięci każdy (Tab. 4.1). Należy uważać, aby nie zapisywać informacji do co czwartego bloku pamięci, zaczynając od bloku nr 3, gdyż w tych blokach zapisywane jest hasło.

Tab 4.1 Struktura pamięci transpondera RFID

Sektor 0	Blok producenta	Blok 1	Blok 2	blok kluczy
Sektor 1	Blok 4	Blok 5	Blok 6	blok kluczy
Sektor ...	Blok ...	Blok ...	Blok ...	blok kluczy
Sektor 15	Blok 60	Blok 61	Blok 62	blok kluczy

Następnie możemy zapisywać i odczytywać dane z karty, w Listingu 4.2 pokazane polecenia zapisywania ciągu W do dwudziestego czwartego bloku pamięci karty lub odczytywania z tego samego bloku do ciągu R.

Listing 4.2 Zapisywanie i odczytywanie bloku pamięci znaczniku RFID

```
status = MFRC522_Write((uint8_t)24 , W);
status = MFRC522_Read( 24, R);
```

Zgodnie z opisany powyżej schematem działania aplikacji należy uwierzytelnić transponder z włączonym trybem odczytu i odczytać zapisane na nim informacje z jego bloków pamięci, sprawdzić czy UID zapisany na karcie zgadza się z listą użytkowników dostępu, a jeżeli odpowiedź będzie pozytywna, otworzyć zamek i zapisać UID do struktury

przechowującej historię dostępu wraz ze znacznikami czasu. Aby to zaimplementować, musimy najpierw sprawdzić, czy karta jest przedstawiona i czy tryb odczytu jest włączony. Po czym wybieramy jeden raz tag i w przypadku tej pracy trzykrotnie przeprowadzamy uwierzytelnianie dla trzech różnych bloków pamięci i wczytujemy informacje z każdego z nich do trzech zmiennych, które następnie łączymy w jedną zmienną za pomocą funkcji memcpy() z biblioteki string.h, która kopiuje bajty do określonej komórki pamięci. Następnie znajdujemy długość samego UID w połączonej zmiennej i przeglądając w pętli wszystkie UID w strukturze, która je przechowuje, porównujemy je z odczytanym UID.

Po znalezieniu dopasowania wysyłamy sygnał na pin podłączony do przekaźnika i zamka, otwierając go i jednocześnie wpisując UID i znacznik czasu do struktury historii dostępu (Listing 4.3).

Listing 4.3 Mechanizm tworzenia nowego wpisu historii

```
snprintf((char*)myHistory[historyEntranceCounter].timeStamps,
sizeof(myHistory[historyEntranceCounter].timeStamps),
"%02d-%02d-%02d/%02d:%02d", date.Year, date.Month, date.Date,
time.Hours, time.Minutes);
snprintf((char*)myHistory[historyEntranceCounter].names,
sizeof(myHistory[historyEntranceCounter].names), "%s:",
myNames.names[i]);
historyEntranceCounter++;
```

Podobna, ale łatwiejsza procedura zapisywania nowych identyfikatorów UID w znaczniku. Tryb nagrywania włącza się, gdy otrzymamy przez Bluetooth polecenie zapisu wraz z nazwą, którą musimy zapisać, więc otrzymaną nazwę po prostu dzielimy na trzy zmienne po 15 bajtów każda i zapisujemy do trzech określonych bloków pamięci.

4.3. Obsługa modułu Bluetooth

Ten podrozdział poświęcony jest programowej obsłudze modułu bluetooth hc-05. Jako przykład prawidłowej implementacji kodu komunikacji w BLuetooth w tej pracy wzięto źródło [17]. Ponieważ USART był włączony podczas konfigurowania urządzeń peryferyjnych w aplikacji CubeMX, cały niezbędny kod jest generowany automatycznie. Podczas generowania projektu STM32CubeIDE tworzy funkcję MX_USART1_UART_Init która inicjuje USART1 zgodnie z ustawieniami (listing 4.4).

Listing 4.4 Funkcja MX_USART1_UART_Init inicjująca interfejs USART1

```
static void MX_USART1_UART_Init(void){
    huart1.Instance = USART1;
    huart1.Init.BaudRate = 9600;
    huart1.Init.WordLength = UART_WORDLENGTH_8B;
    huart1.Init.StopBits = UART_STOPBITS_1;
    huart1.Init.Parity = UART_PARITY_NONE;
    huart1.Init.Mode = UART_MODE_TX_RX;
    huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
    huart1.Init.OverSampling = UART_OVERSAMPLING_16;
    huart1.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
    huart1.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;}
```

Moduł HC-05 w odróżnieniu od rc522 posiada na swojej płytce pamięć oraz układ sterujący, który automatycznie nawiązuje i obsługuje komunikację Bluetooth. Program modułu zawiera zarówno jego nazwę, jak i hasło. W połączeniu z automatycznie wygenerowanym kodem biblioteki HAL i chipem modułu, on sam, bez żadnej interwencji użytkownika, może być dostępny dla wszystkich innych urządzeń Bluetooth w celu połączenia, to połączenie jest chronione hasłem i może odbierać informacje poprzez zapisanie ich w buforze i Wyślij informację.

W bibliotece HAL znajdują się dwie standardowe funkcje odpowiedzialne za odbieranie i przesyłanie danych, HAL_UART_Receive i HAL_UART_Transmit. Aby dowiedzieć się, czy znak został odebrany, funkcja odbierania danych w sposób ciągły odpytuje stan kontrolera UART w czasie oczekiwania. Dlatego ten tryb działania nazywany jest pracą „odpytywania”. Podczas odpytywania czas procesora jest marnowany bezproduktywnie. Aby tego uniknąć, używany jest tryb przerwania.

Przerwania (Interrupts) łączą mechanizm sprzętowy (hardware) i programowy (software) w celu zmuszania procesora do przerywania bieżącej pracy i wykonania określonego bloku instrukcji nazywanego programem obsługi Interrupt Service Routine [18]. Głównym celem wprowadzenia mechanizmu przerwań do rdzenia procesora jest wdrożenie asynchronicznego trybu działania programu i zrównoleglenie pracy poszczególnych urządzeń kompleksu obliczeniowego. Każdemu zdarzeniu wymagającemu przerwania towarzyszy sygnał przerwania żądania, informujący o tym komputer. Przetwarzanie przerwań odbywa się w trzech głównych etapach: zakończenie wykonywania bieżącego programu, przejście do wykonywania programu obsługi i powrót kontroli do przerwanego programu. Procedura obsługi przerwań ma za zadanie szybko reagować na zdarzenia, a w programie można zdefiniować wiele procedur obsługi, przykładowo w opisywanej pracy jest co najmniej przerwań USART i TIM. W związku z tym czas wykonania każdego z nich powinien być jak najkrótszy, aby nie opóźniać przetwarzania pozostałych przerwań. Aby zastosować przerwania w tej pracy, wcześniej było włączono globalne przerwania USART w rozdziale graficznej konfiguracji urządzeń peryferyjnych w CubeMX. W wyniku przeprowadzonych działań do kodu funkcji MX_USART1_Init() dodane zostało włączenie przerwania, a wyłączenie do kodu funkcji MX_USART1_DeInit(). Do pliku stm32f4xx_it.c dodano także procedurę obsługi tego przerwania (listing 4.5).

Listing 4.5 Obsługa przerwania USART1

```
void USART1_IRQHandler(void) {  
    HAL_UART_IRQHandler(&huart1);  
}
```

Ta procedura obsługi jest powiązana konkretnie z USART1 i wywołuje standardową bibliotekę obsługi przerwań USART, która wykorzystuje kontekst sterownika do przetwarzania bieżących zadań wymiany danych USART1. Przerwanie może zostać wywołane różnymi zdarzeniami związanymi z USART (odbiór bajtu, koniec wysyłania bajtu, błąd odbioru itp.). Używając flag przerwań, procedura obsługi biblioteki określa przyczynę wywołania i wykonuje niezbędne działania. W trybie przerwania można korzystać z funkcji przerwania odbioru/nadawania danych HAL_UART_Receive_IT() i HAL_UART_Transmit_IT(). Funkcje te nie „czekają” na zakończenie wymiany, a jedynie inicjują wymianę i przerywają przetwarzanie po zakończeniu odbioru/transmisji. Po zakończeniu wymiany zostanie wywołana procedura obsługi przerwań bibliotecznych, która na koniec jej wykonania wywoła funkcję callback HAL_UART_RxCpltCallback() lub HAL_UART_TxCpltCallback(), którą można zastąpić w celu wykonania potrzebnych nam funkcji (listing 4.6).

Listing 4.6 Callback odpowiedzialny za odbieranie wiadomości

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart) {
    if(huart->Instance == huart1.Instance)
    {
        HAL_UART_Receive_IT(&huart1, RX_BUFFER, BUFFER_LEN);
        newMessageFlag = 1;
    }
}
```

W celu obsługi komunikacji UART musimy zadeklarować trzy zmienne TX_BUFFER i RX_BUFFER, które ze względu na swoją nazwę stanowią bufory wiadomości do wysyłania i odbierania oraz flagę newMessageFlag, która będzie używana, aby kod, który powinien zostać wykonany po odebraniu określonego polecenie, na przykład otwarcie drzwi, zostanie wykonane tylko raz, w przeciwnym razie po odebraniu komunikatu polecenie będzie wykonywane aż do odebrania kolejnego polecenia. Funkcja Callback powinna wyglądać tak i znajduje się w głównym pliku aplikacji: Kod ten odpowiada za zapisanie odebranej wiadomości do bufora i oznaczenie flagi. Z kolei wiadomość wysyłana jest za pomocą funkcji HAL_UART_TRANSMIT().

Ważne jest, aby na końcu każdej wiadomości umieścić linię „\n”, gdyż dzięki temu identyfikatorowi tworzona aplikacja jest w stanie oddzielić wiadomości otrzymane poprzez interfejs szeregowy. Cecha ta jest bardzo istotna, gdyż dzięki niej mamy możliwość wysłania kilku wiadomości poprzez interfejs UART i dopiero po ostatniej dodajemy element końcowy, który docelowo wyświetli się u odbiorcy jako jedna wiadomość, możemy na przykład użyć to do iteracji po określonej tablicy i wysłania jej elementów, dodając pomiędzy nimi inne znaki w ramach jednej wiadomości, zastosowanie takiego algorytmu wykorzystywane jest przy przesyłaniu informacji o strukturach UID użytkowników ze strukturami dostępu i historii.

4.4. Protokół komunikacyjny

Bazując na opisanej powyżej podstawowej obsłudze wysyłania i odbierania wiadomości poprzez interfejs szeregowy USART, w tym rozdziale opiszemy jej zastosowanie w kodzie służącym do realizacji następujących funkcji opisanych w Sekcji 2 niezbędnych do osiągnięcia celu tej pracy:

- Dodawanie i usuwanie UID użytkowników ze struktury przechowującej je po otrzymaniu odpowiednich poleceń z aplikacji Android.
 - Wysyłanie i odbieranie wiadomości poprzez interfejs szeregowy Bluetooth.
 - Przechowywanie hasła wymaganego do dostępu do aplikacji mobilnej i sprawdzanie go po otrzymaniu.
 - Włączenie lub wyłączenie trybu odczytu znaczników RFID po otrzymaniu odpowiedniego polecenia.
 - Kompilowanie w pakiet danych i wysyłanie listy UID lub historii logowania przez BT.
 - Otwarcie elektrozamka po otrzymaniu odpowiedniego polecenia.

Najpierw musimy stworzyć struktury (listing 4.7), które będą przechowywać UID użytkowników i historię dostępu. Dodatkowo musimy opisać funkcje umożliwiające dodawanie do nich nowych elementów oraz usuwanie istniejących.

Listing 4.7 Inicjalizacja struktur przechowujących historię dostępu i UID

```
struct NamesArray {
    uint8_t names[MAX_NAMES][MAX_NAME_LENGTH];
```

```

        size_t count;
    };
    struct HistoryArray {
        uint8_t names[MAX_NAME_LENGTH];
        uint8_t timeStamps[TIMESTAMP_LENGTH];
    };
}

```

Tworząc strukturę NamesArray, określamy w niej dwa elementy, z czego jeden jest dwuwymiarową tablicą znaków przechowujących UID użytkowników, czyli inaczej jest to jednowymiarowa tablica ciągów UID użytkowników oraz zmienna count. W drugiej strukturze określmy tylko jednowymiarową tablicę znaków dla nazw użytkowników i jednowymiarową tablicę znaczników czasu. Różnica w wymiarach tablic pomiędzy obiema strukturami wynika z faktu, że będziemy ich używać inaczej. Strukturę UID użytkownika zadeklarujemy w kodzie tylko raz, a nazwy użytkowników zapiszemy w elementach tablicy nazw znajdującej się w tej strukturze oraz zadeklarujemy strukturę HistoryArray przeznaczoną do przechowywania historii dostępu jako tablicę struktur, czyli oznacza to, że nie będziemy pisać nazw i znaczników czasu w dwuwymiarowych tablicach wewnętrz struktury a jako nowych oddzielnych struktur przechowujących dwa ciągi znaków. Dzieje się tak dlatego, że w strukturze UID nie musimy porównywać ze sobą dwóch zmiennych, a w strukturze historii każdy znacznik kodu musi odnosić się do konkretnego UID użytkownika, który w tym momencie otworzył zamek. Ich deklaracja wewnętrz funkcji main pokazana w listingu 4.8 wraz z funkcją dodawania UID do struktury przechowującej ich.

Listing 4.8 Deklarowanie struktur i funkcja dodawania UID do struktury

```

void addName(struct NamesArray *array, const uint8_t *buffer, size_t
start, size_t length){
if (array->count < MAX_NAMES && length < MAX_NAME_LENGTH) {
    strncpy(array->names[array->count], buffer + start, length);
    array->names[array->count][length] = '\0';
    array->count++;
}
main(){
    struct HistoryArray myHistory[MAX_NAMES];
    struct NamesArray myNames = {{0}, 0};
}

```

Funkcja dodania UID pobiera strukturę UID, do której należy go dodać, bufor, w którym UID się znajduje, w przypadku aktualnej pracy jest to bufor otrzymany w ramach polecenia z aplikacji mobilnej i dlatego należy wskazać w kolejnym kroku dwa elementy z jakiego symbolu i z którego w całym poleceniu się znajduje nazwa, to sprawdzamy czy jest jeszcze miejsce w tablicy nazw i czy długość nazwy nie przekracza maksymalnej dozwolonej, po czym wpisujemy tę nazwę do struktury, a nie zapomniałem dodać znaku nutowego.

Funkcja usuwania UID (listing 4.9) użytkownika pobiera strukturę UID i UID, który ma zostać usunięty. Pętla iteruje po wszystkich nazwach w tablicy i jeśli ją znajdzie, po prostu używamy zagnieżdżonej pętli, aby iterować po wszystkich pozostałych identyfikatorach UID znajdujących się po usuniętym i miksować je w jeden, tak aby w strukturze nie było pustych elementów .

Listing 4.9 Funkcja usuwania UID

```
void deleteName(struct X *array, const uint8_t *nameToDelete) {  
    for (size_t i = 0; i < array->count; ++i) {  
        if (strcmp(array->names[i], nameToDelete) == 0) {  
            for (size_t j = i; j < array->count - 1; ++j) {  
                strcpy(array->names[j], array->names[j + 1]);}  
            array->count--;  
            break;}}}
```

Po opisaniu funkcji możemy przystąpić do omówienia głównej funkcjonalności części kodu związanej z modułem HC-05 – odbierania poleceń i odpowiadania na nie (listing 4.10). Opisując architekturę w prostych słowach, podczas odbierania wiadomości poprzez USART, callback zapisuje tę wiadomość do bufora i informuje resztę programu o tym, że otrzymano wiadomość poprzez ustawienie flagi newMessageFlag, po czym na przemian porównujemy początek otrzymanego pakietu informacji, który pokazuje typ żądania otrzymanego ze wszystkimi możliwymi oczekiwany typami żądań. Przykładowo, jeśli otrzymamy pakiet informacji „{openDoor}”, to porównujemy go ze wszystkimi możliwymi i ostatecznie odnajdujemy go wykorzystując tę część kodu:

Listing 4.10 Mechanizm reakcji na otrzymane polecenia

```
if(newMessageFlag == 1 && strncmp("{openDoor}",RX_BUFFER, 10)==0){  
    strcpy(buffer, "Opening Door");  
    HAL_UART_Transmit(&huart1,(void*)buffer,strlen(buffer), 1000);  
    HAL_GPIO_WritePin(GPIOB, Bluetooth_LED_Pin, GPIO_PIN_SET);  
    HAL_Delay(1000);  
    HAL_GPIO_WritePin(GPIOB, Bluetooth_LED_Pin, GPIO_PIN_RESET);  
    newMessageFlag = 0;}
```

Kod ten jest przykładem działania wszystkich innych podobnych części kodu odpowiedzialnych za reagowanie na wszystkie możliwe otrzymane polecenia. Jak widać, gdy otrzymamy to polecenie, natychmiast odsyłamy reakcję informującą, że polecenie zostało odebrane. Po czym wysyłamy sygnał na pin sterujący przekaźnikiem i zamkiem otwierającym go, następnie czekamy określona ilość czasu, wystarczającą aby użytkownik otworzył drzwi i ponownie zamknął zamek. Na sam koniec ponownie ustawiamy flagę odebranej wiadomości na 0, spełniając zadanie tej flagi.

Oprócz otwarcia drzwi musimy także obsłużyć komendy uruchomienia trybu czytania, zakończenia trybu czytania, dodania użytkownika, usunięcia użytkownika, żądania przesłania listy UID, synchronizacji zegara czasu rzeczywistego, wyłączenia i wyłączenia zapisu tagów, żądania wysłania historii, sprawdzanie hasła i wymiany klucza publicznego RSA, chociaż to ostatnie żądanie omówimy w specjalnie do tego przeznaczonej sekcji. Odbierając wiadomość o rozpoczęciu trybu odczytu musimy ustawić flagę odczytu na pozycję 1 i flagę zapisu na pozycję 0, upewniając się, że nie dojdzie do awarii się z komórkami pamięci na znaczniku RFID, dodatkowo zwracamy również odpowiedź, że pomyślnie włączono tryb odczytu. Gdy otrzymamy wiadomość o wyłączeniu trybu odczytu, robimy to samo, ale ustawiamy flagę na pozycję 0. Otrzymując żądanie o listę użytkowników, musimy odebrać pakiet danych, w którym zostaną zebrane wszystkie nazwy użytkowników. Aby to zrobić, wysyłamy początek pakietu, następnie iterujemy po wszystkich nazwach użytkowników i na koniec wysyłamy koniec wiadomości, która ostatecznie zostanie odebrana jako jedna wiadomość wysłana w częściach.

Listing 4.11 Mechanizm reakcji na otrzymane polecenia

```
char buffer1[] = "usr:";  
HAL_UART_Transmit(&huart1, (void*)buffer1, strlen(buffer1), 1000);  
for(i = 0; i < myNames.count; i++){  
    HAL_UART_Transmit(&huart1, (void*)myNames.names[i],  
    strlen(myNames.names[i]), 1000);  
    char buffer[] = " ";  
    HAL_UART_Transmit(&huart1, (void*)buffer, strlen(buffer), 1000);}  
    char buffer2[] = "\n";  
    HAL_UART_Transmit(&huart1, (void*)buffer2, strlen(buffer2), 1000);
```

Później opiszemy jak ten kod będzie działał w sytuacji gdy będziemy musieli wysłać zaszyfrowaną wiadomość. Różnica polega na tym, że zostanie ona podzielona na kilka wiadomości, ponieważ zaszyfrowana wiadomość jest znacznie dłuższa niż wiadomość w postaci zwykłego tekstu. Odpowiedź na zapytanie o historię działa identycznie, z tą tylko różnicą, że wysyłamy dane z innej struktury (listing 4.12).

Listing 4.12 Wysyłanie historii dostępu po żądaniu

```
snprintf(buffer,sizeof(buffer),"h:%s%s",myHistory[v].names,  
myHistory[v].timeStamps);
```

Otrzymując polecenie usunięcia użytkowników z listy wywołujemy funkcje opisane powyżej. Ale najpierw w kodzie reagowania na takie żądanie (listing 4.13) musimy obliczyć wielkość imienia. Następnie możemy przekazać UID wraz z otrzymanym buforem do funkcji dodającej. Lub w funkcji usuwania jeśli otrzymano polecenie usunięcia, ale wcześniej musimy zapisać samą nazwę do osobnej zmiennej, ponieważ funkcja usuwania działa inaczej i żąda konkretnej nazwy, a nie całej komendy.

Listing 4.13 Usunięcie UID po żądaniu

```
while( decryptedMessage[i]!='}') { i++; }  
nameSpaceCounter = i - 4;  
addName(&myNames, decryptedMessage, 4, nameSpaceCounter);  
    uint8_t nameToDelete[nameSpaceCounter + 1];  
memcpy(nameToDelete, decryptedMessage + 4, nameSpaceCounter);  
nameToDelete[nameSpaceCounter] = '\0';  
deleteName(&myNames, nameToDelete);
```

Komenda wyłączenia rejestracji tagów RFID (listing 4.14) polega na ustawieniu flagi rejestracji na 0, natomiast komenda włączenia rejestracji, którą otrzymujemy wraz z UID, który musimy zapisać na kartę, musi przyjąć nazwę z otrzymanej linii polecen i podzielić go na trzy części po 15 bajtów każda z których następnie zostanie zapisana w trzech różnych blokach pamięci znaczników RFID i włączą tryb rejestracji poprzez ustawienie flagi zapisu na 1.

Listing 4.14 Wyłączenia rejestracji tagów RFID po żądaniu

```
readFlag = 0; writeFlag = 1;i = 0;  
while( decryptedMessage[i]!='}') { i++; }  
nameSpaceCounter = i - 4;
```

```

memcpy(writeArrayDonor, decryptedMessage + 4, MAX_NAME_LENGTH);
memcpy(writeArray1, writeArrayDonor, subArraySize);
memcpy(writeArray2, writeArrayDonor+subArraySize, subArraySize);
memcpy(writeArray3, writeArrayDonor+2*subArraySize, subArraySize);

```

Otrzymując polecenie weryfikacji hasła, sprawdzamy otrzymane hasło podanym w programie hasłem i odsyłamy wiadomość pass_correct wraz z pozycjami flag odczytu i zapisu (listing 4.15), aby aplikacja mobilna mogła od razu ustawić przełączniki we właściwej pozycji po podczas łączenia, w rezultacie komunikat będzie wyglądał następująco: „pass_correct, read:%d, write:%d”. Otrzymując wiadomość z odczytami czasu i daty, wiemy, w jakiej pozycji powinny znajdować się elementy, dzięki czemu możemy je bezpośrednio wyodrębnić bezpośrednio na zmienne, tworząc z wyprzedzeniem nowe struktury czasowe poprzez zmianę formatu.

Listing 4.15 Weryfikacja hasła

```

RTC_TimeTypeDef new_time = {0};
eestlen = (RX_BUFFER[6]- '0')*10+(RX_BUFFER[7]- '0');
new_time.Hours = (RX_BUFFER[6]- '0')*10+(RX_BUFFER[7]- '0');
new_time.Minutes = (RX_BUFFER[9] - '0') * 10 + (RX_BUFFER[10] - '0');
new_time.Seconds = (RX_BUFFER[12]- '0')*10+(RX_BUFFER[13]- '0');
HAL_RTC_SetTime(&hrtc, &new_time, RTC_FORMAT_BIN);
RTC_DateTypeDef today;
today.Year = (RX_BUFFER[21] - '0') * 10 + (RX_BUFFER[22] - '0');
today.Month = (RX_BUFFER[18] - '0') * 10 + (RX_BUFFER[19] - '0');
today.Date = (RX_BUFFER[15] - '0') * 10 + (RX_BUFFER[16] - '0');
today.WeekDay = RTC_WEEKDAY_TUESDAY;
HAL_RTC_SetDate(&hrtc, &today, RTC_FORMAT_BIN);

```

4.5. Szyfrowanie i uwierzytelnianie

Wynikiem wdrożenia funkcjonalności programu w opisanych powyżej działach jest w pełni działający program do urządzenia kontroli dostępu opartego na tagach RFID i komunikacją Bluetooth. Komunikacja z tym urządzeniem może odbywać się poprzez prostą aplikację mobilną z terminaliem Bluetooth, którą można pobrać z preferowanego sklepu z aplikacjami. W tym terminalu po podłączeniu do modułu HC-05 można za pomocą prostych poleceń {OpenDoor}, {usr:Admin} w pełni sterować modułem. W kolejnym rozdziale jest opisane stworzenie aplikacji mobilnej, która będzie pełniła funkcję interfejsu człowiek-maszyna, gdyż nie od każdego użytkownika powinno być wymagane zapoznanie się z instrukcją obsługi pracy z opisanym w tym systemie kontrolą dostępu praca, a interfejs użytkownika znacznie ułatwi korzystanie z niego.

Jednak ważniejszą funkcją aplikacji jest możliwość wykonywania obliczeń kryptograficznych niezbędnych do generowania kluczy oraz szyfrowania i deszyfrowania komunikacji pomiędzy mikrokontrolerem a aplikacją. W terminalu niezwykle trudno byłoby ręcznie lub nawet za pomocą aplikacji strony trzeciej obliczyć klucze i szyfrować za każdym razem każdy bajt 10-bajtowego żądania otwarcia drzwi. Dlatego aplikacja mobilna pozwala mieć funkcjonalność kryptograficzną we wszystkie opisanej pracy, znacznie zabezpieczając cały system. Do zadania omawianego w tej pracy można zastosować dowolny z algorytmów szyfrowania. W pracy wybrano asymetryczny algorytm RSA. Choć jest on znacznie mniej wydajny w porównaniu na przykład do algorytmu

AES, różnice te równoważone są niewielkim rozmiarem wysyłanych pakietów oraz brakiem konieczności przesyłania klucza prywatnego, przez co system jest generalnie bezpieczniejszy i nie ma potrzeby stosowania drugiego algorytmu szyfrowania do przesłania tego klucza lub programowego ustawiania kluczy w obu urządzeniach. RSA, pomimo obfitości wielu innych i względnego wieku jego powstania, jest obecnie aktywnie wykorzystywany w prawie wszystkich hybrydowych kryptosystemach, a także do ochrony oprogramowania.

Algorytm ten został stworzony przez trzech naukowców z MIT (Massachusetts Institute of Technology), Rona Rivesta, Adiego Shamira i Leonarda Adlemana. Algorytm RSA jest ważny przez to, że różnicę w prostocie mnożenia dwóch liczb pierwszych i trudności w znalezieniu pierwotnych czynników. W przykładzie produkt dwóch liczby dwucyfrowych można łatwo znaleźć, ale czynniki pierwsze liczby trzycyfrowej są znacznie trudniejsza, a w większości przypadków zawiera nawet więcej trzech. Komputeru może zająć kilka lat na rozkład liczb wielkich. Etap pierwszy dowolnego algorytmu asymetrycznego polega na opracowaniu dwóch kluczy - publicznego i prywatnego. Klucze publiczne mogą być mogą być rozpowszechniane w jakikolwiek publiczny sposób, a ich utrata nie stwarza zagrożenia dla bezpieczeństwa, dlatego w tej pracy klucze publiczne generowane przez mikrokontroler i aplikację mobilną będą wymieniane w formacie tekstu jawnego. Klucz prywatny wygenerowany w połączeniu z kluczem publicznym jest nadal przechowywany na urządzeniu, które go utworzyło. Tylko klucz prywatny z określonej pary kluczy może odszyfrować wiadomość zaszyfrowaną kluczem publicznym z tej pary. Tej wiadomości nie można uzyskać przy użyciu samego klucza publicznego, działa to również w drugą stronę.

W skrócie algorytm działa w ten sposób, że mikrokontroler generuje klucze publiczny i prywatny z wybranych liczb pierwszych, następnie wysyła swój klucz publiczny w momencie nawiązania komunikacji i uwierzytelnienia aplikacji mobilnej i w tym samym momencie otrzymuje klucz publiczny aplikacji mobilnej. Do zaszyfrowania wiadomości, która ma zostać wysłana, mikrokontroler wykorzystuje klucz publiczny aplikacji, a odbierając od aplikacji wiadomość zaszyfrowaną kluczem publicznym otrzymanym od mikrokontrolera, odszyfrowuje ją swoim kluczem prywatnym. Dalej jest opisane, w jaki sposób zaimplementowano opisany algorytm.

W pracy tej nie wykorzystano specjalnej biblioteki kryptograficznej, lecz ręcznie napisano funkcje realizujące tzw. „książkowy” algorytm RSA, których równania są dostępne w źródle [19]. Na początek wybiera się dwie wystarczająco duże liczby pierwsze p i q i oblicza moduł n , gdzie.

$$n = p \times q \quad (1.1)$$

Następnie stosuje się funkcję Eulera do wartości n , która jest określona przez liczbę mniejszych liczb, z którymi argument nie ma wspólnych czynników

$$\varphi(n) = (p - 1)(q - 1) \quad (1.2)$$

Następnie wybierana jest liczba e , nazywana publicznym wykładownikiem, która byłaby względnie pierwsza z $\varphi(n)$ i spełnia warunek $1 < e < \varphi(n)$

Na koniec szukamy liczby „ d ” (wykładownik zamknięty), multiplikatywnej odwrotności liczby „ e ” modulo $f(n)$.

$$d \times e \equiv 1 \pmod{\varphi(n)} \quad (1.3)$$

W rezultacie na wyjściu tych obliczeń otrzymujemy dwie pary liczb, które są naszym kluczem publicznym i prywatnym. (e, n) jest kluczem publicznym i (d, n) jest kluczem prywatnym. W tym przypadku szyfrowanie będzie reprezentowane w następujący sposób:

$$C = M^e \times (\text{mod } n) \quad (1.4)$$

gdzie M to to liczba, czy w przykładzie tej pracy bajt szyfrowany. odszyfrowanie będzie wyglądać następująco. Odpowiednio, deszyfrowanie przeprowadza się w następujący sposób:

$$M = C^d \times (\text{mod } n) \quad (1.5)$$

Kod mikrokontrolera odpowiadający za obsługę generowania klucza (listing 4.16) i szyfrowania wraz z deszyfrowaniem wiadomości przed wysłaniem działa zgodnie z opisanymi powyżej obliczeniami. Najpierw są opisane funkcje, które muszą być zdefiniowane w specjalnym pliku biblioteki, ponieważ jest to dobra praktyka programistyczna struktrowania kodu. Zdefiniowanie dwóch losowych liczb pierwszych do utworzenia kluczy.

Listing 4.16 Generacja kluczy RSA przez mikrokontroler

```

Void rsa_gen_keys(struct
public_key_class *pub, struct
private_key_class *priv){
    long long p, q;
    generate_two_primes(&p, &q);
    long long e = (2 << 16) +1;
    long long d = 0;
    long long max = 0;
    long long phi_max = 0;
    max = p*q;
    phi_max = (p-1)*(q-1);
    d = ExtEuclid(phi_max,e);
    while(d < 0){
        d = d+phi_max;}
    pub->modulus = max;
    pub->exponent = e;
    priv->modulus = max;
    priv->exponent = d;}
```

Pierwsze dwie linie definiują dwie losowe liczby pierwsze tworzące klucze. W tej pracy wykorzystano 5-cyfrowe liczby pierwsze, ponieważ zapewniają one ponad optymalny poziom trudności w znalezieniu czynników początkowych, wystarczający do przeprowadzenia wszystkich niezbędnych manipulacji przed wygenerowaniem nowej pary kluczy przy następnym podłączeniu do telefonu komórkowego. Obie liczby muszą być tej samej długości, gdyż należy zadbać o równomierny rozkład bitów w module n . Aby je wygenerować wewnątrz funkcji `generate_two_primes` należy zdefiniować funkcję generującą jedną liczbę pierwszą, a następnie wywołać tę funkcję dla obu liczby, których potrzebujemy w pętli do `while`, dopóki $*p == *q$ nie będzie poprawną instrukcją. Tylko w niewielkim odsetku zdarzeń te dwie liczby mogą być identyczne, ale ta praca musi uwzględniać każdą możliwą przyczynę naruszenia bezpieczeństwa. Sama funkcja generująca jedną taką liczbę pierwszą na początku definiuje w dwóch zmiennych zakres, z którego ta liczba jest generowana, a następnie w pętli sprawdza, czy końcowa liczba jest pierwsza, za pomocą funkcji `Rand()` generuje liczbę z określonego zakresu w którym zmienna rozmiar określa wielkość wyjściowych liczb pierwszych i może być zmieniana w celu kontrolowania bezpieczeństwa algorytmu i szybkości obliczeń. W kolejnej linii wybierana jest wartość e za pomocą opisanego powyżej algorytmu, którą zwykle przyjmuje się jako równą $(2^{16}) + 1$. Następnie w kolejnych czterech liniach obliczane jest d dla klucza prywatnego przy użyciu rozszerzonego Algorytm euklidesowy, w którym

phi_max jest wartością funkcji Eulera modułu max, iloczynów p i q. Funkcja ExtEuclid (listing 4.17) oblicza multiplikatywną odwrotność d e modulo phi_max i wygląda następująco.

Listing 4.17 Funkcja obliczenia multiplikatywnej odwrotności

```
long long ExtEuclid(long long a, long long b){  
    long long x = 0, y = 1, u = 1, v = 0, gcd = b, m, n, q, r;  
    while (a!=0) {  
        q = gcd/a; r = gcd % a;  
        m = x-u*q; n = y-v*q;  
        gcd = a; a = r; x = u; y = v; u = m; v = n;  
    }return y;}
```

Główna część funkcji szyfrowania i deszyfrowania pokazana jest w listingu 4.18 i odpowiada algorytmowi opisanemu powyżej

Listing 4.18 Szyfrowanie i deszyfrowanie w STM32

```
long long *rsa_encrypt(const char *message, const unsigned long  
message_size, const struct public_key_class *pub){  
    long long *encrypted = malloc(sizeof(long long)*message_size);  
    for(i=0; i < message_size; i++){  
        if ((encrypted[i] = rsa_modExp(message[i], pub->exponent,  
        pub->modulus)) == -1)  
            return NULL; }  
    return encrypted; }  
  
char *rsa_decrypt(const long long *message, const unsigned long  
message_size, const struct private_key_class *priv){  
    char *decrypted = malloc(message_size/sizeof(long long));  
    char *temp = malloc(message_size);  
    for(i=0; i < message_size/8; i++){  
        if ((temp[i] = rsa_modExp(message[i], priv->exponent,  
        priv->modulus)) == -1){  
            free(temp);  
            return NULL; }  
    }  
    for(i=0; i < message_size/8; i++){  
        decrypted[i] = temp[i];  
    }free(temp);return decrypted; }
```

Ponadto wymagane jest sprawdzenie, czy bufore nie są puste. Po opisaniu wszystkich niezbędnych funkcji pozostaje tylko dodać w pliku głównym funkcję void sendEncryptedData(char *messageToSend) zawierającą wskaźnik do ciągu wiadomości, który chcemy wysłać poprzez interfejs Bluetooth, na przykład sendEncryptedData("opening drzwi"); lub sendEncryptedData(zmienna); po czym linia ta jest kopiowana do bufora, szyfrowana, a wynik szyfrowania przesyłany interfejsem UART, nie zapominając o dodaniu początku pakietu w postaci niezaszyfrowanych znaków „e:” i

końca „\n”, aby umożliwić aplikacji wie, że otrzymuje zaszyfrowaną wiadomość.

Wymiana kluczy po stronie mikrokontrolera (listing 4.20) odbywa się w ten sposób, że gdy otrzymamy klucz publiczny podzielony na wykładownik i moduł, musimy zapisać go w zmiennej klucza publicznego aplikacji i natychmiast w odpowiedzi wysłać utworzony przez nas klucz publiczny.

Listing 4.19 Przetwarzanie otrzymanego klucza publicznego

```
char e1_str[20], e2_str[20];
sscanf(RX_BUFFER + i, "e1:[^e]e2:[^m]", e1_str, e2_str);
long long multiplier_e2 = 1;
for (int k = 0; k < strlen(e2_str); k++) { multiplier_e2 *= 10; }
if(strlen(e2_str) > 6){
    pubAPP->exponent = numberpe = strtoll(e1_str, &endptr, 10);
}else{
    pubAPP->exponent = numberpe = strtoll(e1_str, &endptr, 10) *
    multiplier_e2 + strtoll(e2_str, &endptr, 10);}
```

Liczba otrzymana jako exponent lub modulus jest podzielona na dwie, ponieważ często jest znacznie większa niż maksymalna pojemność long int, więc pojawiają się błędy, dlatego jest potrzeba dzielić je podczas przejścia i łączyć ponownie niżej. Po wpisywaniu do dwóch zmiennych jest obliczany mnożnik dla e2 lub m2 wtedy, gdy nie jest puste żebądź wiedzieć ich rozmiar. Po tym jest przeprowadzane obliczanie, przez ile należy pomnożyć drugą liczbę, aby znajdowała się zaraz po pierwszej po czym są łączone i zapisywane do struktury kluczu publicznego. Identyczna metoda jest do rozebrania i zebrania m1 i m2. Następnie wysyłamy komponenty klucza publicznego mikrokontrolera w ramce key{e:...m:...}\n.

5. Projekt aplikacji mobilnej

Android Studio to zmodyfikowane środowisko programistyczne dla aplikacji mobilnych Android, w którym Java jest oficjalnie uznawana za wiodący język programowania. Proces instalacji i opcje Android Studio są dość proste i instynktownie zrozumiałe dla większości użytkowników. Android Studio używa Gradle jako systemu komplikacji. Gradle to system do automatycznego tworzenia aplikacji i wykorzystuje DSL na języku Groovy do przedstawienia konfiguracji. Gradle określa, które komponenty aplikacji nie uległy zmianie, a które zadania nie wymagają ponownego uruchomienia.

Z opisanych w drugiej części pracy, która dotyczy koncepcji projektowanego urządzenia, aplikacja musi mieć możliwość formułowania i wysyłania następujących pakietów danych z poleceniami:

- Otwórz drzwi;
- Włącz tryb czytania tagów;
- Wyłącz tryb czytania tagów
- Dodaj określony UID;
- Usuń określony UID;
- Zapisz podany UID w tagu;
- Zapytaj o listę UID;
- Zapytaj o historię dostępu;

Dodatkowo aplikacja musi utworzyć parę kluczy publicznych i prywatnych, wymienić klucze publiczne z mikrokontrolerem, mieć możliwość szyfrowania wysyłanych wiadomości kluczem publicznym mikrokontrolera oraz odszyfrowywania otrzymanych wiadomości swoim kluczem prywatnym.

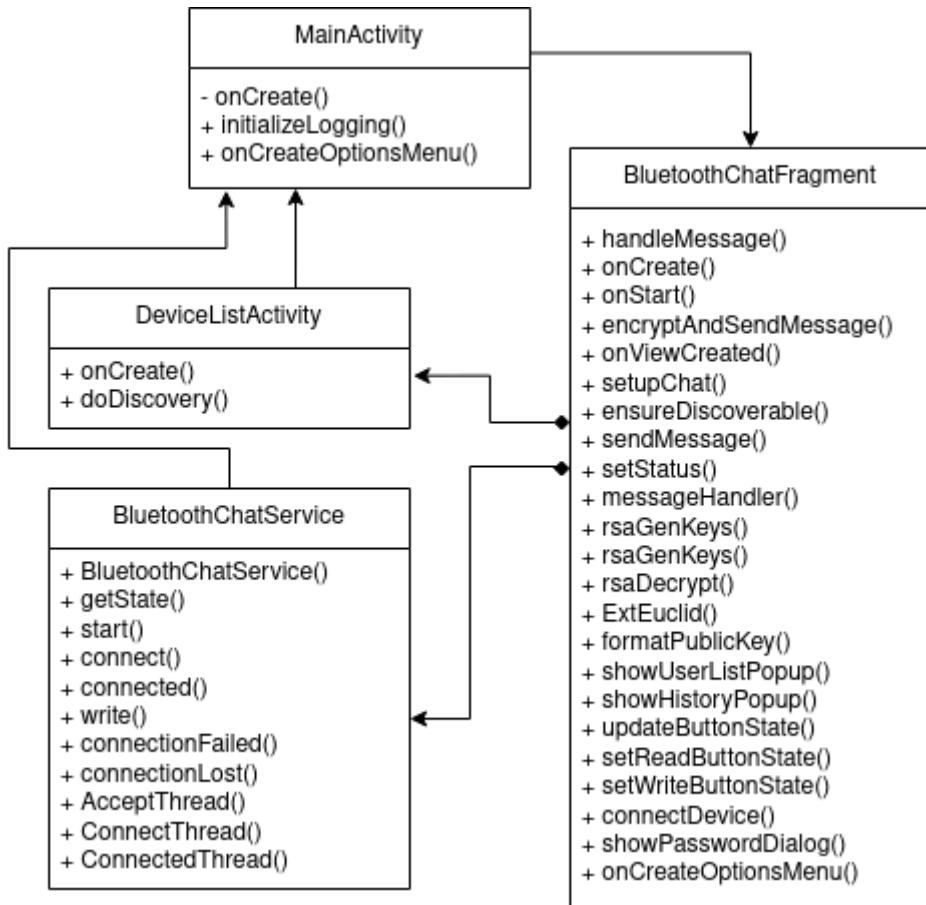
5.1 Tworzenie aplikacji mobilnej

Tworząc nowy projekt w Android Studio należy wybrać najniższą wersję systemu operacyjnego Android potrzebną do uruchomienia aplikacji. Następnie Android Studio prosi o wybranie szablonu aplikacji spośród dostępnych, uproszczenia wybrano szablon Empty Activity. Po lewej stronie okna Android Studio znajduje się pełna architektura katalogów i plików utworzonego projektu. Architektura każdej aplikacji opracowanej w Android Studio zawiera następujące ważne komponenty:

1. Plik `AndroidManifest.xml`, który zawiera podstawowe informacje o aplikacji wymagane przez system Android takie jak nazwę pakietu, komponenty aplikacji, activity, service, uprawnienia wymagane do działania aplikacji, minimalny i docelowy poziom API oraz używane funkcje urządzenia, takie jak obecność kamery w urządzeniu itp. Plik manifestu określa również ikonę aplikacji, nazwę, motyw aplikacji itp.
2. Pliki z rozszerzeniem `.java` znajdujące się w folderze `java`. Pliki te reprezentują moduły oprogramowania, których kod jest napisany w języku programowania Java;
3. Pliki z rozszerzeniem `.xml`, znajdujące się w folderze `res/layout` i posiadające opcje dla okien i elementów sterujących używanych w aplikacji, w formacie XML;
4. Plik `string.xml` znajdujący się w folderze `res/values` służy do zapisywania stałych tekstowych używanych w aplikacji.

Głównymi składnikami aplikacji na Androide są Activity, Service, Content provider, App Widget i Process. Activity to komponent aplikacji, który wyświetla ekran i z którym użytkownicy mogą wchodzić w interakcję. Activity może zawierać jeden lub więcej fragmentów. Fragment jest częścią interfejsu użytkownika w Activity i służy do ponownego wykorzystania. Process służy do tego, aby złożone zadania obliczeniowe były wykonywane w osobnym wątku i nie blokowały wątku głównego. Główny wątek aplikacji powinien być używany wyłącznie do interakcji użytkownika z interfejsem graficznym. Service to komponent aplikacji, który działa w wątku w tle i nie zawiera interfejsu użytkownika i może wchodzić w interakcję z głównym procesem aplikacji jedynie za pomocą komunikatów. Intent to obiekt przesyłania wiadomości aplikacji, którego można użyć do zażądania akcji od komponentu aplikacji. Intent są używane głównie w trzech przypadkach: do rozpoczęcia operacji, do uruchomienia usługi i do wysyłania wiadomości rozgłoszeniowych. Wszystkie komponenty systemu są zadeklarowane w pliku `AndroidManifest`, dzięki czemu system Android wie, które komponenty istnieją.

Projekt aplikacji mobilnej będzie składał się z czterech głównych klas, które współdziałają ze sobą, są one pokazane na rys. 5.1. Do głównych funkcji aplikacji mobilnej należy komplikacja pakietów danych do komunikacji z mikrokontrolerem sterującym zamkiem, szyfrowanie tych pakietów danych, obsługa otrzymanych pakietów danych i w razie potrzeby ich odszyfrowanie. Dlatego najważniejszą klasą tej aplikacji i jej większością jest utrzymanie i ustanawianie komunikacji Bluetooth. Ponieważ nasze urządzenie mobilne przejmuje rolę Master, cała praca spada na jego barki. Komunikacja będzie obsługiwana przez klasę `BluetoothChatService`, natomiast klasa `BluetoothChatFragment` będzie odbierać, wysyłać, szyfrować i deszyfrować pakiety, dzięki czemu będą się w niej znajdować obsługa wszystkich przycisków i pól i wszystkie funkcje użytkownika związane z komunikacją Bluetooth. Taki jest schemat i zasada działania aplikacji stworzonej w tej pracy.

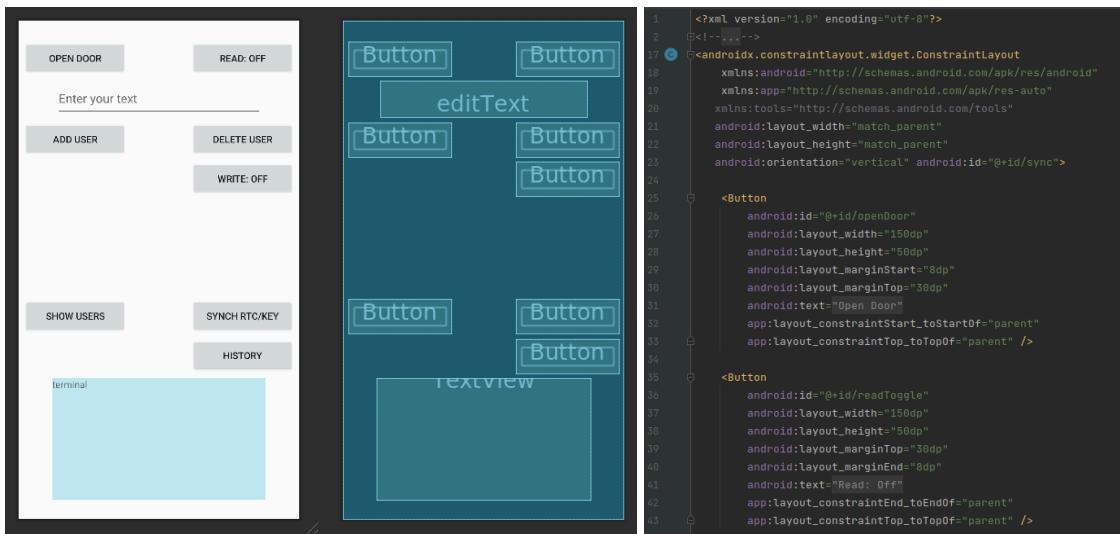


Rys. 5.1 Schemat klas i metod aplikacji

`MainActivity` jest podklassą `Activity`, klasą główną, której instancja tworzona jest podczas uruchamiania aplikacji. Klasa ta odpowiada za konfigurację interfejsu użytkownika, zarządzanie fragmentami, w tym tworzenie `BluetoothChatFragment` i dodawanie do kontenera fragmentów za pomocą `FragmentTransaction`, obsługę menu opcji i inicjowanie logowania do aplikacji czatowej Android Bluetooth. Klasy związane z nawiązaniem komunikacji Bluetooth zostaną opisane w rozdziale poświęconym tej komunikacji, natomiast funkcjonalność przycisków, pól i związane z nimi funkcje zostaną opisane w odrębnym rozdziale.

5.2. Interfejs graficzny

Do opracowania interfejsu mobilnego w systemie Android używany jest Extensible Markup Language, zwykle w skrócie XML. Standard XML definiuje zbiór podstawowych reguł leksykalnych i składniowych służących do konstruowania języka opisywania informacji za pomocą prostych znaczników. Standard definiuje metajęzyk, w oparciu o który poprzez wprowadzenie ograniczeń w strukturze i treści dokumentów definiowane są specyficzne dla domeny języki znaczników danych. Istnieją dwa tryby pracy z plikami XML: graficzny i przeglądanie kodu (Rys. 5.2). Projektowanie formularza okiennego odbywa się poprzez umieszczenie na nim różnych elementów sterujących, czyli View, takich jak przyciski opcji, pola tekstowe, przełączniki i przyciski, znajdujących się w zakresie narzędzi znajdujących się po lewej stronie edytora formularza okna.



Rys. 5.2 Demonstracja trybów pracy z XML na przykładzie menu głównego aplikacji

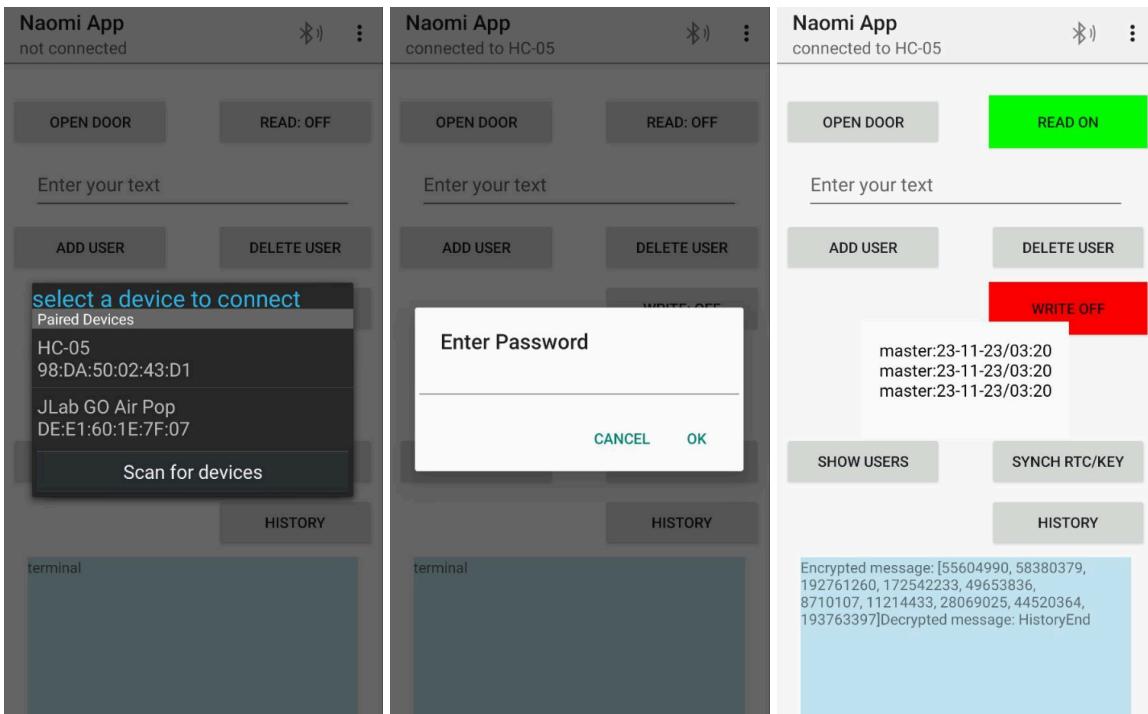
Na przykładzie powyższego okna głównego interfejsu tworzonego programu można zauważyc, że w tworzonym programie wykorzystywane są takie elementy View jak Button (przycisk), editText (pole wprowadzania tekstu), TextView (napis). View button to przycisk z napisem lub obrazkiem, po kliknięciu na który powinna nastąpić jakaś akcja. Właściwość tekst zawiera słowo wyświetlane na przycisku, a właściwość onClick określa sposób obsługi akcji, która ma miejsce po kliknięciu przycisku myszą. editText to jednowierszowe tło do wprowadzania słowa, które jest przechowywane we właściwości tekst. Właściwość inputType pozwala wybrać typ znaków wejściowych. Aplikacja mobilna musi mieć możliwość wysyłania następujących poleceń:

- Otwórz drzwi;
 - Wyłącz tryb czytania tagów;
 - Dodaj określony UID;
 - Usuń określony UID;
 - Zapisz podany UID w tagu;
 - Zapytaj o listę UID;
 - Zapytaj o historię dostępu;

Dlatego musimy stworzyć następujące elementy interfejsu:

- Przycisk wysyłający polecenie otwarcia drzwi
 - Przełącznik trybu odczytu tagów
 - Pole wejściowe UID do użytku w trybie zapisu i zarządzania listą UID
 - Przełącznik trybu nagrywania powiązany z polem wejściowym
 - Przycisk dodawania UID z listy powiązany z polem wejściowym
 - Przycisk usunięcia UID z listy powiązany z polem wejściowym
 - Przycisk żądania historii dostępu
 - Przycisk żądania listy UID
 - Przycisk synchronizacji kluczy publicznych i aktualnego czasu

Dodatkowo musimy udostępnić wyskakujące okienka odpowiedzialne za wyświetlanie otrzymanej historii dostępów i listy UID oraz co najważniejsze za wybranie podłączonego urządzenia i wyświetlenie okna wpisania hasła w celu autoryzacji, ich wygląd w ostatecznej aplikacji pokazano na rysunku 5.3.



Rys. 5.3 Okienka wybrania podłączonego urządzenia, autoryzacji i historii

5.3. Bluetooth

Dla aplikacji opartych na systemie Android istnieje specjalny interfejs API (Interfejs programowania aplikacji) – oficjalny framework dostarczony przez Android zawierający zestaw klas i metod umożliwiających aplikacji bezprzewodowe łączenie się z innymi urządzeniami Bluetooth, umożliwiając funkcje bezprzewodowe typu punkt-punkt i wielopunkt. API definiuje obiekt `BluetoothAdapter` dla wszystkich operacji za pomocą protokołu Bluetooth, który reprezentuje własny adapter Bluetooth urządzenia z systemem Android. Interfejs API Bluetooth systemu Android zapewnia interakcję z urządzeniami Bluetooth za pośrednictwem funkcji wywołania zwrotnego (callback). Istnieje oficjalny przewodnik [20], dotyczący korzystania z tego API, został on wykorzystany przy tworzeniu tej pracy i wszelkie najdrobniejsze szczegóły nie zostaną omówione, ponieważ funkcje zostały udostępnione przez programistów Androida.

Przede wszystkim należy zadeklarować w pliku `AndroidManifest.xml` wszystkie uprawnienia niezbędne do działania bluetooth. Są to te same uprawnienia, o które aplikacje proszą użytkowników po uruchomieniu. Musimy zadeklarować uprawnienia:

- `BLUETOOTH`; • `BLUETOOTH_ADMIN`; • `BLUETOOTH_SCAN`
- `BLUETOOTH_ADVERTISE`; • `BLUETOOTH_CONNECT`;

Poniżej opisano zawartość klasy `BluetoothChatService` i `BluetoothChatFragment`, które zawierają funkcjonalność nawiązania połączenia z adapterem bluetooth hc-05 (listing 5.1). Na początku zdefiniowany jest adapter, który przygotowuje nową sesję Bluetooth, Następnie aplikacja upewnia się, że Bluetooth jest włączony. Wywoła metod `isEnabled()`, aby sprawdzić, czy metod Bluetooth jest aktualnie włączona i jeżeli nie, to włącza go, a Po kliknięciu przycisku Połączenia Bluetooth powinniśmy otrzymać listę podłączonych urządzeń.

Listing 5.1 Nawiązanie połączenia z adapterem bluetooth hc-05

```
mAdapter = BluetoothAdapter.getDefaultAdapter();
Intent enableIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
pairedDevices = bluetoothAdapter.getBondedDevices();
```

Dla połączenia dwóch urządzeń, jedno musi działać jako serwer, w naszym przypadku aplikacja mobilna, trzymając otwarte gniazdo BluetoothServerSocket, którego celem jestazda serwera jest nasłuchiwanie przychodzących żądań połączenia i udostępnianie podłączonego gniazda Bluetooth po zaakceptowaniu żądania. Aby skonfigurować gniazdo serwera i zaakceptować połączenie, najpierw aplikacja tworzy gniazdo za pomocą listenUsingRfcommWithServiceRecord(), potem zaczyna nasłuchiwać żądań połączeń, wywołując metod accept(). Po pomyślnym nawiązaniu połączenia aplikacja powinna odbierać strumienia dla połączeń wychodzących i przychodzących, InputStream i OutputStream które można otrzymać za pomocą getInputStream() i getOutputStream() odpowiednio.

5.4. Komunikacja z mikrokontrolerem

Cała funkcjonalność opisanych powyżej przycisków, przełączników i pola, które mają generować i wysyłać żądania do mikrokontrolera, a także przetwarzanie otrzymanych odpowiedzi wraz z szyfrowaniem, deszyfrowaniem i tworzeniem do tego kluczy RSA, są opisane w klasie BluetoothChatFragment i omówione w tej sekcji. W metodzie onViewCreated() przyciski są inicjowane poprzez znalezienie odpowiednich widoków w układzie fragmentu przy użyciu identyfikatorów zasobów. Przyciski działają poprzez setOnClickListener i metodę onClick, zastosowaną w następujący sposób na przykładzie przycisku otwierającego zamek, pokazanego w listingu 5.2. Kliknięcie tego przycisku powoduje wysłanie polecenia {openDoor}. Jest to realizowane poprzez funkcję szyfrującą i obliczającą wiadomości, co opisano w dalszej części.

Listing 5.2 Kod przycisku wysyłającego żądanie otwarcia drzwi

```
openDoor.setOnClickListener(new View.OnClickListener() {@Override
public void onClick(View view) {
String messageToEncrypt = "openDoor";
encryptAndSendMessage(messageToEncrypt);}});
```

Przyciski zapytają o historię dostępu i listę UID po prostu wysyłają odpowiednie komunikaty, podobnie jak przycisk odblokowania.

Przełącznik readToggle (listing 5.3) to prosty przycisk, którego kolor i tekst zmieniają się po każdym naciśnięciu z „czytanie włączone” na „czytanie wyłączone” lub można je przełączyć ręcznie, co jest niezbędne, aby w momencie połączenia się z mikrokontrolerem otrzymać z niego informację o tryby czytania i zapisywania, synchronizując z nim położenie obu przełączników. Oprócz tego przełączania, naciśnięcie przycisku powoduje wysłanie polecenia {ReadOn} lub {ReadOff}.

Listing 5.3 Kod przełącznika trybu odczytywania

```
if (readToggle.getText().equals("Read On")) {
    updateButtonState("read", "0");
```

```

readToggle.setBackgroundColor(Color.RED);*/
String messageToEncrypt = "readOff";

```

Wszystkie trzy przyciski wysyłające żądania związane z wprowadzonym w linii UID działają na tej samej zasadzie, uwzględniają tekst wpisany w tę linię, jeśli w tej linii znajduje się coś, w ramach wysyłanego polecenia. Przykład w listingu 5.4 pokazuje część kodu przycisku usuwania UID, ale wystarczy zamienić „Dlt:” na „add:”, aby wysłać żądanie dodawania UID i połączyć ten algorytm z poprzednim algorytmem przełączania, wstawiając ten warunek do środka wyłączenie trybu nagrywania i zastąpienie początku polecenia „wOn:”

Listing 5.4 Formatowanie żądania usuwania UID z listy

```

String userName = editText.getText().toString();
if (!userName.isEmpty()) {
    String messageToEncrypt = "Dlt:" + userName + "}";
    encryptAndSendMessage(messageToEncrypt);
}

```

Przycisk synchronizacji czasu i kluczy publicznych (listing 5.5) tworzy SimpleDateFormat w celu sformatowania bieżącego czasu w formacie (godzina:minuta:sekunda/dzień-miesiąc-rok). następnie bieżący czas jest uzyskiwany za pomocą metody new Date() i ramka jest tworzony poprzez dołączenie sformatowanego czasu do ciągu znaków w formacie „{time: currentTime}” z dodawaniem spacji, aby uzyskać całkowitą długość 500 znaków. Po czym wiadomość jest wysyłana.

Listing 5.5 Wysyłanie czasu rzeczywistego i klucza publicznego RSA

```

SimpleDateFormat sdf = new SimpleDateFormat("hh:mm:ss/dd-MM-yy",
Locale.getDefault());
String currentTime = sdf.format(new Date());
String originalMessage = "{time:" + currentTime + "}";
int spacesToAdd = 500 - originalMessage.length();
StringBuilder paddedMessage = new StringBuilder(originalMessage);
for (int i = 0; i < spacesToAdd; i++) {
    paddedMessage.append(" ");
}
sendMessage(paddedMessage.toString());

```

Ale to nie jest koniec funkcjonalności tego przycisku, ponieważ po wysłaniu czasu i po oczekiwaniu opóźnienia specjalnie zdefiniowanego przez handler w sekundę, klucze RSA są generowane używając do tego metody rsaGenKeys(pub, priv), po czym klucz publiczny jest formatowany do postaci ciągu znaków, formatowany w ramkę, dopełniany do 500 znaków i wysyłany do mikrokontrolera. Funkcja generowania klucza jest pełną kopią tej samej funkcji, z której korzysta mikrokontroler, jedynie przepisaną w języku JAVA używanym w aplikacjach mobilnych. Języki są na tyle podobne, że chociaż proces tłumaczenia jest pełen nieprzewidywalnych błędów, nie jest trudny. Różnice polegają głównie na tym, że klucze są przechowywane w klasach, a nie w strukturach, oraz na zastosowaniu typu zmiennej BigInteger (listing 5.6). Funkcje szyfrowania i deszyfrowania również działają w ten sam sposób.

Listing 5.6 Klasy kluczy RSA

```

public static class PublicKey {

```

```

    public BigInteger modulus;
    public BigInteger exponent;
public PublicKey(BigInteger modulus, BigInteger exponent) {
    this.modulus = modulus;
    this.exponent = exponent;
}

```

Również wygenerowanie nowej pary kluczy jest powodowane prawidłową autoryzacją (listing 5.7), która jest realizowana prośbą o podanie hasła i zapytaniem mikrokontrolera o poprawności tego hasła. Ta para kluczy jest pierwszą i w zasadzie jedyną parą kluczy używaną w sesji aplikacji (generowanie pary kluczy podczas synchronizacji ma miejsce w przypadkach, w których integralność klucza została utracona z powodu niedoskonałości w kanale komunikacyjnym). Po poprawnym wpisaniu hasła oba urządzenia generują własne, osobne i losowe pary kluczy, których klucze publiczne są natychmiast wymieniane. Klucz publiczny może być dystrybuowany bez ryzyka dla bezpieczeństwa, więc nie ma ryzyka eksmisji go w postaci zwykłego tekstu.

Listing 5.7 Okno wprowadzania hasła

```

AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
builder.setTitle("Enter Password");
final EditText input = new EditText(getActivity());
input.setInputType(InputType.TYPE_CLASS_TEXT |
InputType.TYPE_TEXT_VARIATION_PASSWORD);
builder.setView(input);

```

Metoda encryptAndSendMessage(listing 5.8), która jak wspomniano powyżej szyfruje i wysyła wysłaną do niej wiadomość najpierw wywołuje metodę rsaEncrypt, przekazując wiadomość i klucz publiczny (pubSTM) jako parametry i otrzymuje od niej tablicę BigInteger reprezentującą zaszyfrowaną wiadomość. Potem metoda iteruje po tablicy i konwertuje każdą BigInteger na ciąg znaków, którzy są następnie konkatenowane w jeden ciąg za pomocą StringBuilder, dopełniane do 500 bajtów i wysyłane poprzez Bluetooth.

Listing 5.8 Szyfrowanie i wysyłanie wiadomości do mikrokontrolera

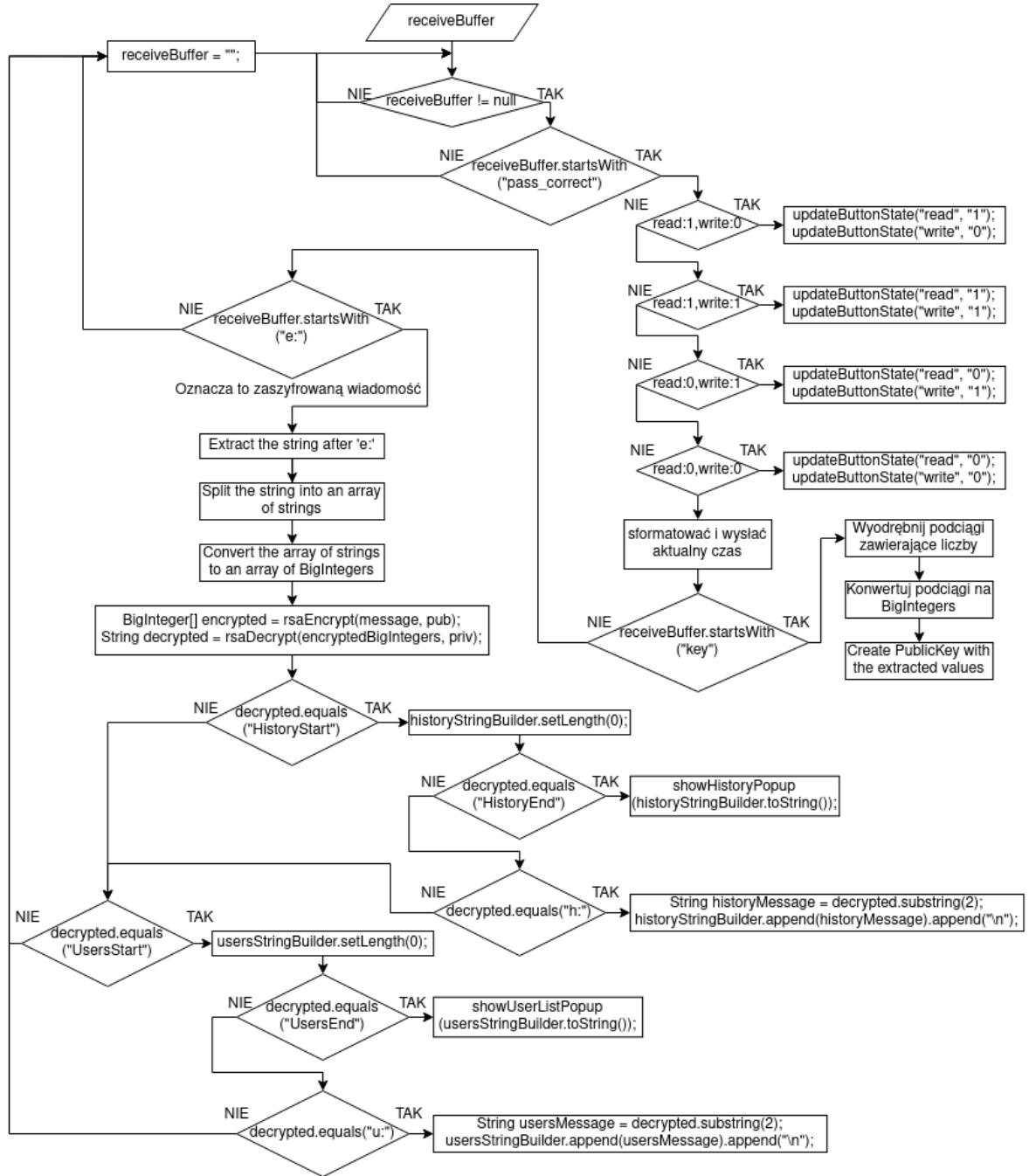
```

private void encryptAndSendMessage(String message){
    BigInteger[] encrypted = rsaEncrypt(message, pubSTM);
    StringBuilder messageBuilder = new StringBuilder();
    for (BigInteger bigInteger : encrypted) {
        String paddedBigIntString = String.format("%010d", bigInteger);
        messageBuilder.append(paddedBigIntString);
    }
    String encryptedString = messageBuilder.toString();
    ...
}

```

Ale zdecydowanie najważniejszą częścią tej sekcji jest metoda odbierania wiadomości za pośrednictwem komunikacji Bluetooth. Kod, który reaguje na zmiany w buforze odebranego komunikatu, następnie decyduje, co zrobić z odbieraną wiadomością, jest to w formacie schematu blokowego pokazane na rys. 5.4. Jeśli jest to wiadomość jawną tekstową zawierającą otrzymany publiczny klucz mikrokontrolera, pozycji przełączników lub odpowiedź autoryzacyjną, należy wykonać z góry określone działania, czy to zapisać klucz do klasy prywatnej, ustawić przełączniki w należące stany, czy odpowiedzieć na

autoryzację. Jeśli aplikacja otrzymuje zaszyfrowaną wiadomość, to musi najpierw odszyfrować ją przy użyciu własnego klucza prywatnego, a następnie podjąć niezbędne działania odpowiadające polecomenom zawartym w tej wiadomości.

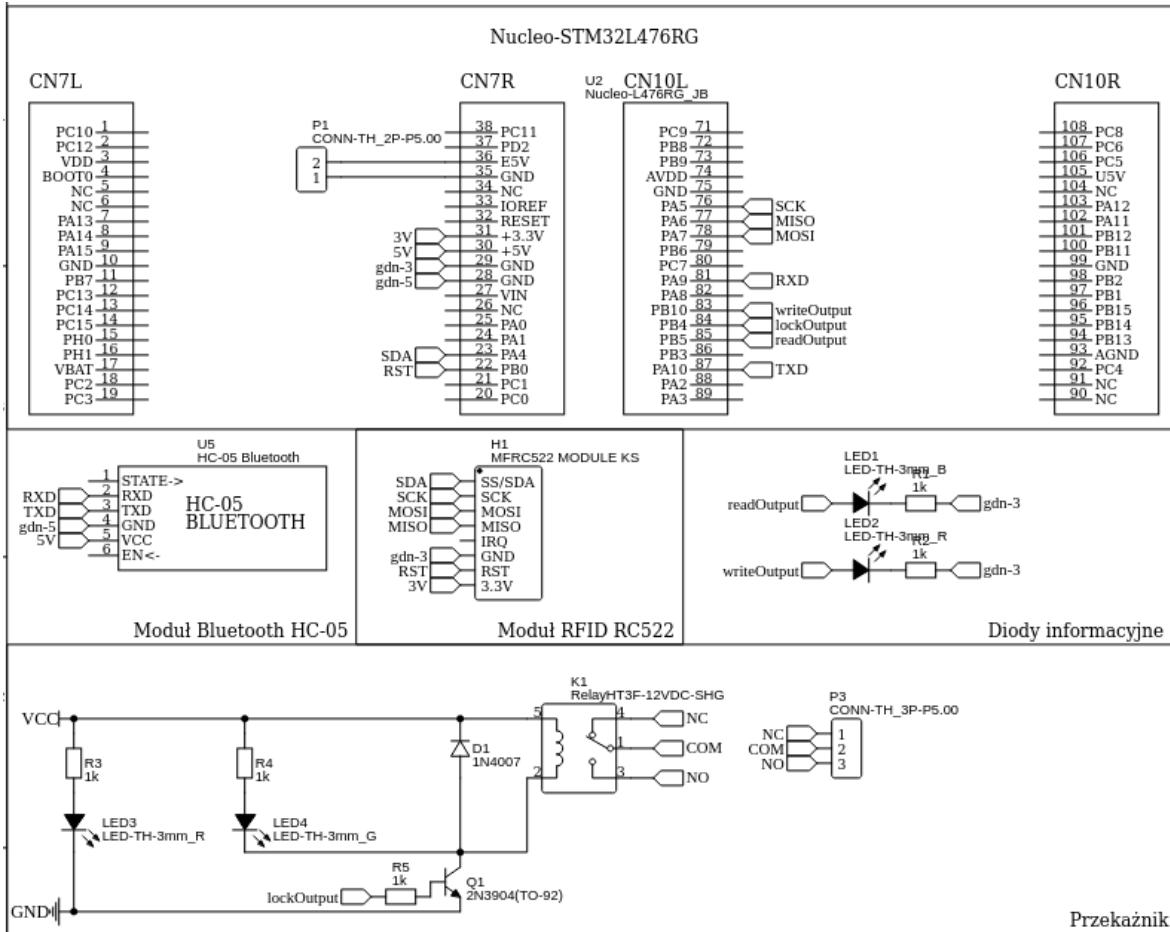


Rys. 5.4 Schemat blokowy algorytmu reakcji na odebraną wiadomość

6. Budowa prototypu

W tym rozdziale omówiono trzy główne sposoby tworzenia prototypu projektowanego urządzenia. Stworzenie prototypu polega na połączeniu mikrokontrolera i komunikujących z nim modułów i diod LED za pomocą przewodów lub elementów przewodzących. Niezależnie od zastosowanej metody, wszystkie opierają się na schemacie połączeń elektrycznych opisany w trzeciej części tej pracy. Schemat ten opisuje podłączenie

urządzeń peryferyjnych do pinów samego mikrokontrolera, jednak ponieważ w tej pracy projektujemy urządzenie w oparciu o płytę rozwojową STM32-Nucleo-L476RG, ważne jest opisanie schematu elektrycznego przy tworzeniu prawdziwego prototypu za pomocą piny tej płytka. Rysunek 6.1 poniżej opisuje właśnie taki schemat i za pomocą etykiet pokazuje, które piny mikrokontrolera należy do czego podłączyć. Dodatkowo poniższy schemat przedstawia zaprojektowany obwód przekaźnika, który jest używany, ten schemat on potrzebny w trzeciej opcji tworzenia prototypu, projektowania płytki drukowanej, ponieważ pożądane jest posiadanie przekaźnika na samej płytce.

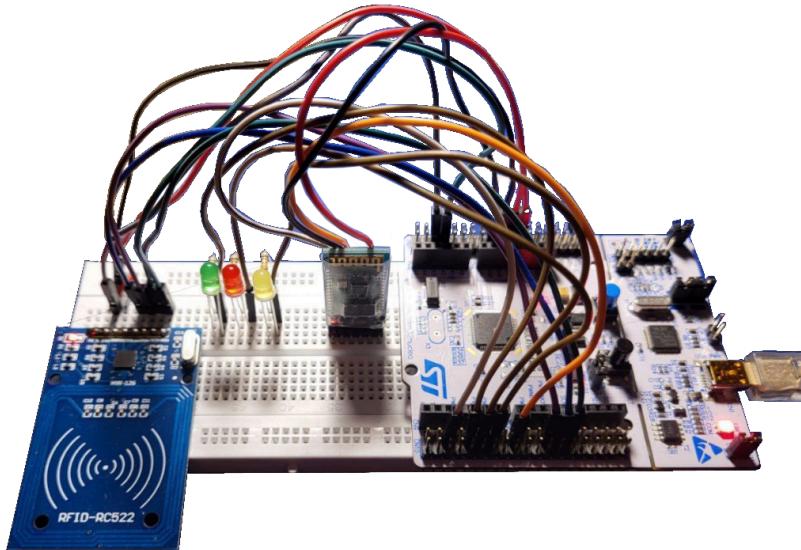


Rys. 6.1 Schemat elektryczny połączeń modułów peryferyjnych do STM32L476RG

6.1 Prototyp na płytce stykowej

Pierwszą i najłatwiejszą metodą zaprojektowania urządzenia jest użycie płytki stykowej. Jest najłatwiejszy, bo nie wymaga lutowania czegokolwiek, wystarczy sama płytka i komplet kabli GPIO, za pomocą których podłączamy mikrokontroler do urządzeń. Najważniejszą zaletą tej metody jest niezaprzeczalna szybkość i łatwość wykonania takiego prototypu. Kolejnym plusem jest łatwość poprawiania błędów. Nie ma potrzeby ponownego lutowania źle podłączonego kabla, wystarczy go odłączyć i podłączyć. Ale jednocześnie wady tej metody są oczywiste, kruchosć konstrukcji i chaotyczna organizacja drutów w bardziej złożonych projektach. Pierwszą wadą jest to, że jeden zły ruch może spowodować rozłączenie wszystkich przewodów, ponieważ nic ich nie trzyma. Drugą wadą jest to, że w bardziej skomplikowanych projektach często prawie niemożliwe jest

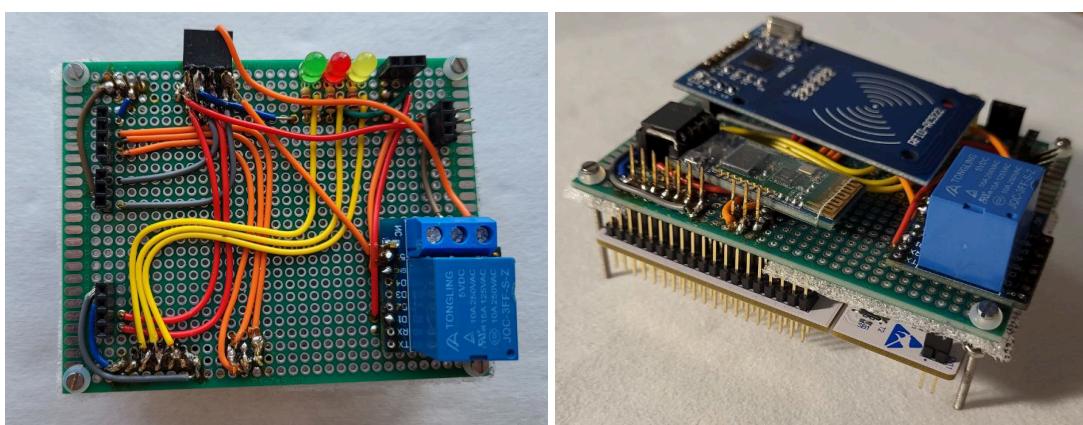
zrozumienie, który przewód przechodzi z którego wyjścia do którego wejścia, nawet wielokolorowe zestawy przewodów nie pomagają, ponieważ zawsze jest kilka przewodów tego samego koloru używany. Ten minus jest oczywisty na zdjęciu poniżej (rys. 6.2), który pokazuje ukończony projekt tej pracy na płytce stykowej.



Rys. 6.2 Urządzenie zbudowane na płytce stykowej

6.2 Prototyp na uniwersalnej płytce drukowanej

Drugą metodą jest zastosowanie uniwersalnych płyt drukowanych. Ta płytka jest prostokątną matrycą metalizowane otworów w rastrze 2,54 mm i te otwory nie są ze sobą połączone za pomocą predefiniowanych ścieżek. W praktyce oznacza to możliwość poprowadzenia ścieżek cynkiem, czyli jak w tej pracy, przy użyciu przewodów spawanych i podłączenie tych przewodów albo do samych pinów modułu, albo połączenie ich za pomocą żeńskich złączy GPIO, co pozwala na swobodne wkładanie i wyjmowanie moduł. Wykorzystując takie złącza można łatwo stworzyć swego rodzaju "shield", czyli rozszerzenie, które będzie instalowane bezpośrednio na istniejącym GPIO mikrokontrolera, jak pokazano na zdjęciu poniżej (rys. 6.3).



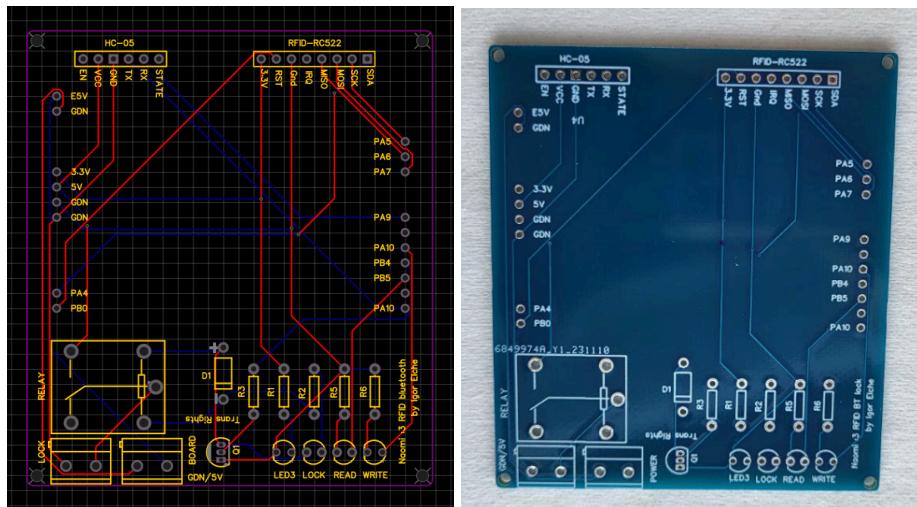
Rys. 6.3 Urządzenie zbudowane na płytce uniwersalnej

6.1 Prototyp na płytce drukowanej

Ostatnią i najbardziej profesjonalną opcją budowy prototypu urządzenia jest zaprojektowanie i zamówienie płytki drukowanej (PCB). Płytnka drukowana to powierzchnia dielektryczna, na którą nałożone są ścieżki przewodzące i przygotowane miejsca do montażu elementów elektronicznych. Każda płytka drukowana składa się z kilku głównych elementów:

- Podłoże dielektryczne (podstawa płytki)
- Przewodniki (ścieżki i podkładki)
- Powłoka dielektryczna przewodów (maskowanie)
- Pola stykowe komponentów
- Otwory montażowe i przelotowe
- Maska lutownicza

Projekt układu płytka drukowanej wykonano w programie EasyEDA (Rys. 6.4)



Rys. 6.4 Projekt płytka drukowanej i płytka drukowana wykonana fabrycznie

Ta metoda wykonania prototypu urządzenia jest lepsza od poprzedniej pod względem czasu potrzebnego na jego wykonanie, ponieważ podłączenie elementów elektrycznych w specjalnym programie jest znacznie łatwiejsze niż żmudne lutowanie każdego kabla, ale w zamian wymaga dużo czasu oczekiwania na produkcję i wysyłkę gotowych płyt, ponieważ są one produkowane w Chinach. 5 takich płytka można zamówić w chwili pisania tego tekstu za 7 euro. Prototyp urządzenia projektowanego w tej pracy wykonany na płytce drukowanej demonstruje rys. 6.5.



Rys. 6.5 Urządzenie zbudowane na płytce uniwersalnej

Podsumowanie

W pracy opracowano i zaprojektowano system kontroli dostępu składający się z mikrokontrolera, urządzeń peryferyjnych konfigurowanych za pomocą aplikacji mobilnej. Wszystkie założone cele zostały osiągnięte. Ta praca stanowi prezentację wszystkich kroków niezbędnych do stworzenia takiego systemu, począwszy od zaczarowania modułów RFID i Bluetooth podłączonych do mikrokontrolera i umożliwienia mu wykorzystania jego funkcjonalności, protokołów komunikacyjnych z nimi oraz zastosowanych przekaźników, zamków i sposobów łączenia systemu do zasilania. W pracy opisano konfigurację peryferii mikrokontrolera, uruchomienie oraz podstawowe funkcje programowego sterowania modułami i zegarami.

Opisano w pracy kompletny algorytm działania oprogramowania tworzonego urządzenia, po czym opisano sposób realizacji każdej z funkcji, które składają się na jedno całe urządzenie zdolne do odczytu i zapisu znaczników RFID, odbierania i wysyłania komunikatów poprzez Bluetooth, reagowania na polecenia otrzymywanych poprzez Bluetooth, prowadzenie logu zalogowanych osób oraz listy użytkowników mających dostęp, dodawanie i usuwanie z tej listy, generowanie kluczy RSA, wymiana klucza publicznego z aplikacją mobilną, szyfrowanie wiadomości wychodzących i odszyfrowywanie wiadomości przychodzących.

Powstała aplikacja mobilna posiadająca pełną funkcjonalność niezbędną do konfiguracji i zarządzania systemem kontroli dostępu: dodawania i usuwania użytkowników, włączania trybu zapisu lub odczytu, przeglądania historii dostępów i listy użytkowników z dostępem oraz zdalnego otwierania zamków. Stworzono i opisano trzy wersje prawdziwego prototypu urządzenia, od płytki stykowej po profesjonalnie drukowanej płytki. Stworzono niezbędne schematy elektryczne podłączenia modułów do mikrokontrolera i płytki rozwojowej oraz schematy blokowe funkcjonowanie części programowej mikrokontrolera i aplikacji mobilnej.

Bibliografia

1. <https://www.mordorintelligence.com/industry-reports/global-access-control-market-industry>
2. https://material.dahuasecurity.com/uploads/cpq/prm-os-srv-res/smart/formal/Product/HQ/DHI-ASI1201E/Datasheet/ASI1201E_ASI1201E-D_datasheet_20221105.pdf
3. <https://www.ttlock.com/#/productionCenter>
4. <https://ttlock-eu.com/shop/ttlock-k-2/>
5. <https://play.google.com/store/apps/details?id=com.tongtongsuo.app>
6. Paweł Hadam "Projektowanie systemów mikroprocesorowych", Warszawa, Polska, 2004
7. https://www.st.com/resource/en/data_brief/nucleo-g071rb.pdf
8. <https://www.st.com/resource/en/datasheet/stm32l476je.pdf>
9. Mandeep Kaur, Manjeet Sandhu, Neeraj Mohan i Parvinder S. Sandhu, "RFID Technology Principles, Advantages, Limitations & Its Applications", International Journal of Computer and Electrical Engineering, Vol.3, No.1, February, 2011 1793-8163
10. <https://pdf1.alldatasheet.com/datasheet-pdf/view/227840/NXP/RC522.html>
11. Krzysztof Paprocki "Mikrokontrolery STM32 w praktyce", Legionowo, Polska, 2009
12. BlueCoreTM4-External Product Data Sheet
13. <https://www.technophiles.com/hc-05-pinout-specifications-datasheet/>
14. M. A. Hosany and L. Chooramun, "Performance Analysis of a Data Communication System Employing Bluetooth and Wi-Fi Standards," 2018 International Conference on Intelligent and Innovative Computing Applications (ICONIC), Mon Tresor, Mauritius, 2018, pp. 1-6, doi: 10.1109/ICONIC.2018.8601283.
15. Charles Platt "Encyclopedia of Electronic Components Volume 1" Sebastopol, Canada, 2013.
16. https://s28.picofile.com/file/8463407384/RC522_and_RFID_Tags.rar.html
17. <https://deepbluembedded.com/stm32-hc-05-bluetooth-module-examples/>
18. Maciej Szumski "Mikrokontrolery STM32 w systemach sterowania i regulacji", Legionowo, Polska, 2018
19. [https://en.wikipedia.org/wiki/RSA_\(cryptosystem\)](https://en.wikipedia.org/wiki/RSA_(cryptosystem))
20. <https://developer.android.com/guide/topics/connectivity/bluetooth#java>