Sprawozdanie z laboratorium

Przedmiot: Urządzenia peryferyjne

Temat: Karty mikroprocesorowe (GSM / SIM)

Data: 16.10.2025

Autorzy:

Marcel Cieśliński, nr indeksu 280871

Mateusz Bonifatiuk, nr indeksu 280902

1. Cel ćwiczenia

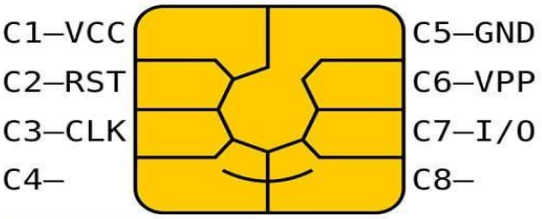
Celem laboratorium było poznanie zasad działania kart mikroprocesorowych stosowanych w systemie GSM oraz praktyczne zapoznanie się z ich obsługą poprzez interfejs PC/SC i bibliotekę Winscard. W ramach zajęć należało uruchomić aplikację umożliwiającą komunikację z kartą SIM, wysyłać polecenia APDU w formacie HEX oraz napisać własny program, który odczytuje podstawowe dane z karty – takie jak lista kontaktów (ADN) i wiadomości SMS (EF_SMS).

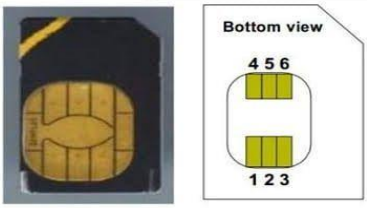
2. Zagadnienia teoretyczne

Karta SIM to mikroprocesorowy układ scalony zawierający trzy główne typy pamięci: ROM, RAM i EEPROM. ROM przechowuje system operacyjny, RAM służy do operacji tymczasowych, a EEPROM zawiera dane użytkownika takie jak kontakty, SMS-y czy klucze autoryzacyjne.

Struktura plików karty SIM obejmuje plik główny MF, katalogi DF (np. DF TELECOM) oraz pliki danych EF (np. EF_ADN – książka telefoniczna, EF_SMS – wiadomości).

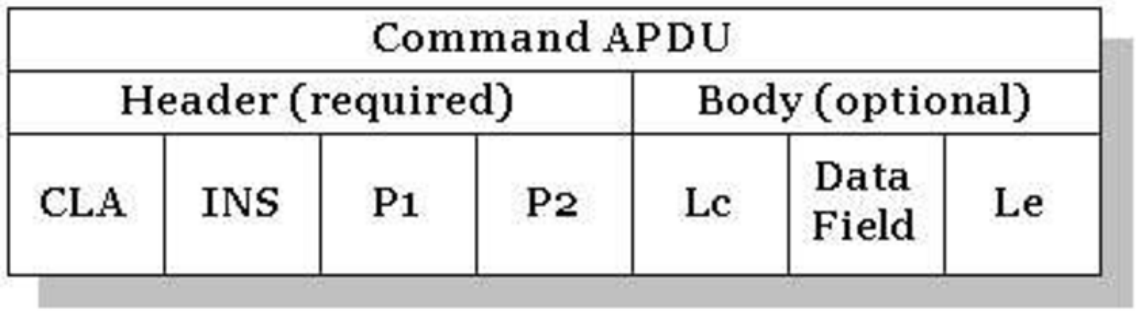
Najważniejsze styki karty to: C1 (VCC – zasilanie), C2 (RST – reset), C3 (CLK – zegar), C5 (GND – masa), C7 (I/O – linia danych).





1	VCC
2	RESET
3	CLOCK
4	GND
5	VPP
6	DATA

Po podłączeniu karty następuje przesłanie ATR (Answer To Reset), zawierającego informacje o protokole transmisji (T=0 lub T=1) oraz parametrach pracy. Komendy APDU (Application Protocol Data Unit) służą do komunikacji między kartą a aplikacją, mają strukturę [*rysunek1*].



Rysunek 1

3. Przebieg ćwiczenia

W pierwszej części wykorzystano program SIMCardManager,SmartX do odczytu zawartości kart SIM, uzyskując listę kontaktów oraz podstawowe informacje zapisane w pamięci karty. W drugiej części opracowano własny program konsolowy w języku C++, umożliwiający komunikację z kartą poprzez funkcje biblioteki Winscard.

Program pozwalał na: wyświetlenie dostępnych czytników, wybór urządzenia, wysyłanie dowolnej komendy APDU w formacie HEX oraz odczyt plików EF_ADN i EF_SMS. Odpowiedzi karty były prezentowane w formacie HEX oraz ASCII.

4. Sekwencja testów i komunikacja APDU

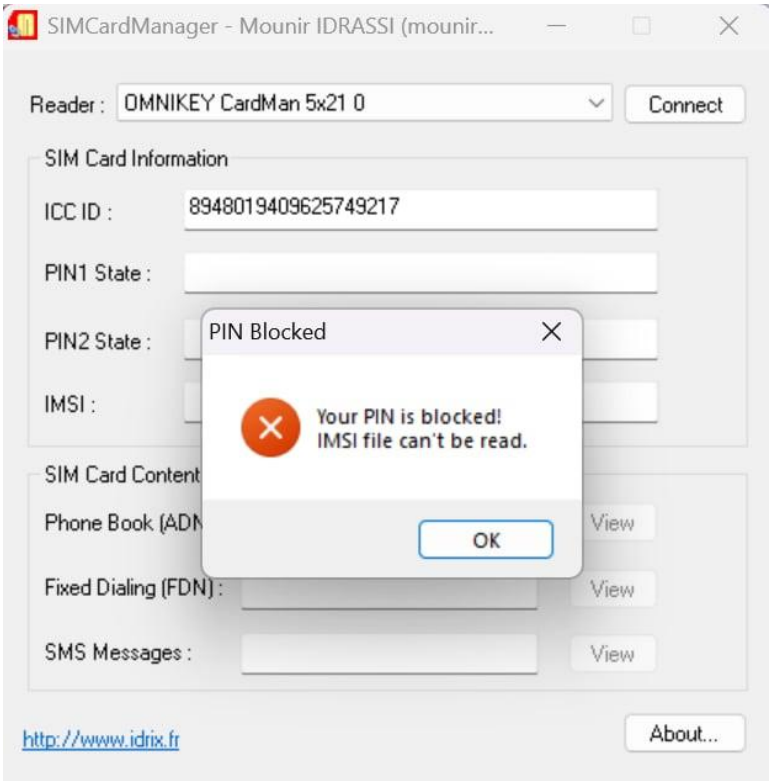
Przeprowadzono testy komunikacji z kartą SIM przy użyciu programu Xsmart. Odczyt danych następował za pomocą sekwencji poleceń APDU, które umożliwiają wybór katalogów i odczyt plików EF_SMS oraz EF_ADN.

Krok	Komenda (HEX)	Opis	Odpowiedź	Interpretacja
1	A0 A4 00 00 02 3F 00	Wybór katalogu MF	9F XX / 90 00	Dostęp do struktury karty
2	A0 A4 00 00 02 7F 10	Wybór DF TELECOM	9F XX / 90 00	Przejście do katalogu danych użytkownika

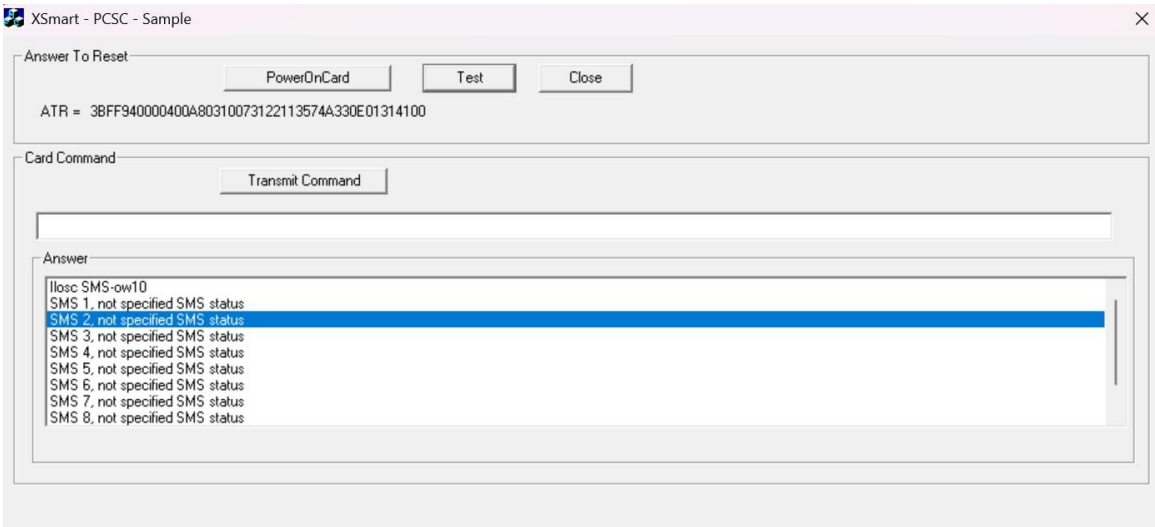
3	A0 A4 00 00 02 6F 3C	Wybór EF_SMS	9F XX	Znaleziono plik SMS
4	A0 C0 00 00 0F	GET RESPONSE	90 00	Odczyt struktury pliku
5	A0 B2 01 04 B0	Odczyt rekordu SMS	90 00	Dane wiadomości w formacie binarnym

5. Zadanie 1

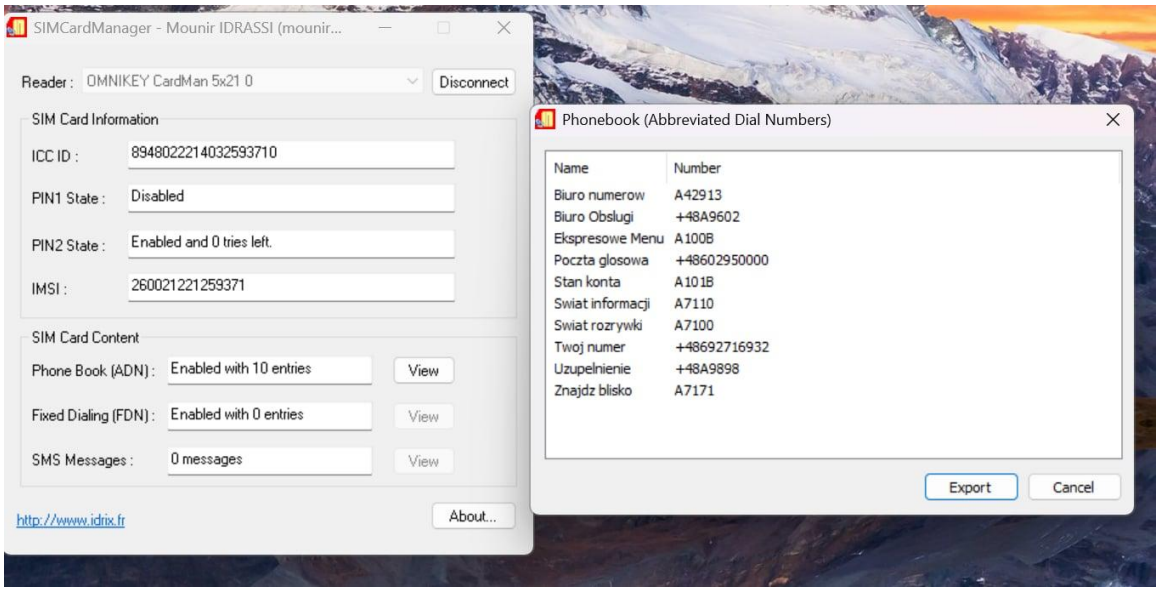
Po podłączeniu czytnika do komputera przystąpiliśmy do testowania pierwszej karty SIM w programie **SIMCardManager**.
Pierwsza karta okazała się **zablokowana**, przez co nie było możliwe odczytanie danych z katalogów ani wiadomości SMS *(Rysunek 2, Rysunek 3)*.
W celu kontynuacji pomiarów **wymieniono kartę SIM na inną**, która została poprawnie rozpoznana przez czytnik.
Dzięki temu udało się **uzyskać dostęp do zawartości pamięci karty**, w tym do **listy kontaktów** niestety karta nie miała w EF żadnych zapisanych SMS-ów *(Rysunek 4).*



Rysunek2



Rysunek 3



Rysunek4

6.Zadanie 2 – program w c++

Do realizacji zadania opracowaliśmy własny program w języku **C++**, wykorzystując środowisko programistyczne **Microsoft Visual Studio** oraz bibliotekę **WinSCard** umożliwiającą komunikację z interfejsem **PC/SC**.
Program nawiązuje połączenie z usługą systemową **Smart Card Resource Manager**, a następnie **pobiera listę dostępnych czytników** i umożliwia użytkownikowi **wybór jednego z nich**. Po nawiązaniu połączenia z kartą SIM, aplikacja **pobiera ATR (Answer To Reset)** – informację identyfikującą kartę i protokół transmisji.

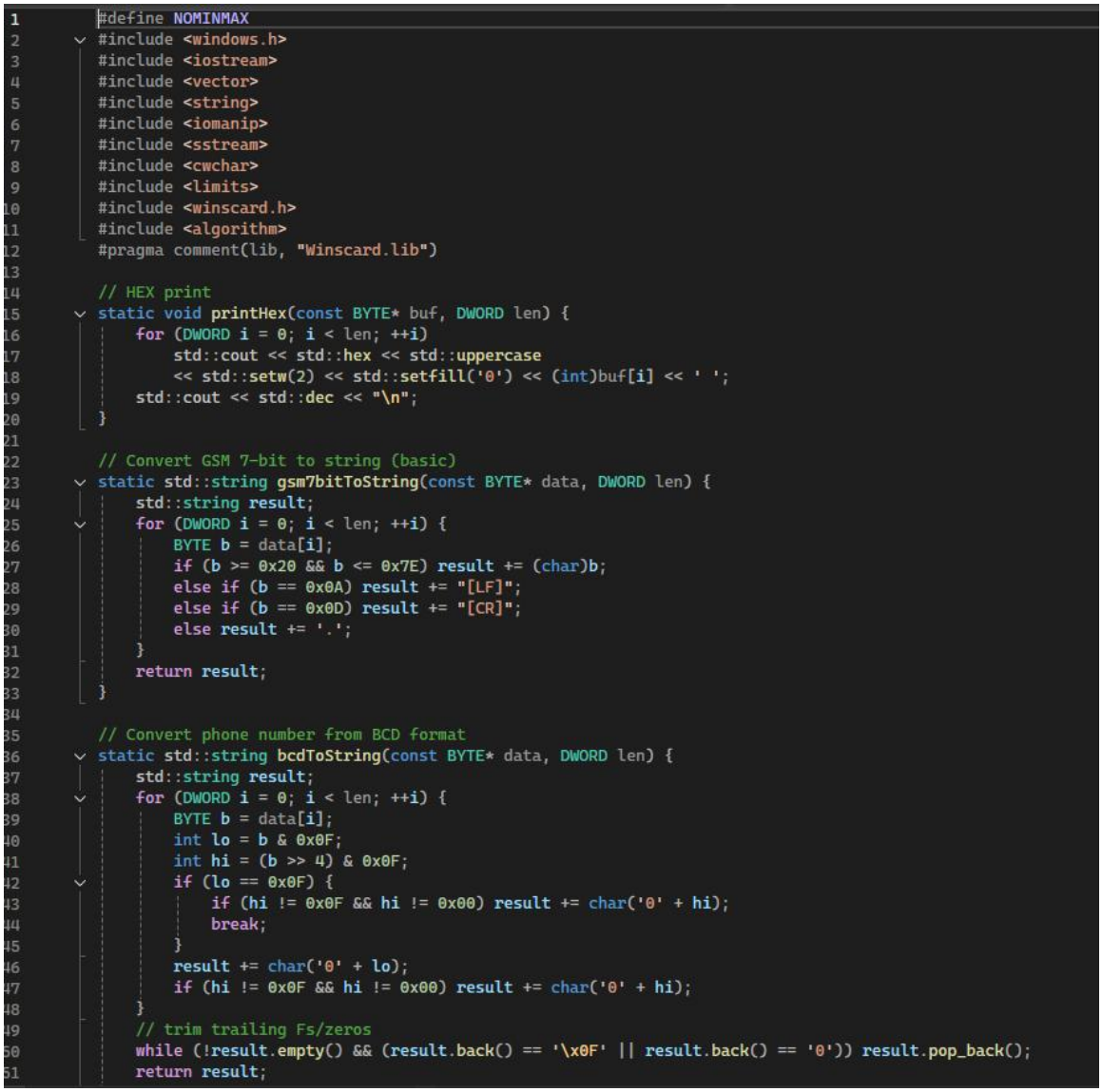
Kolejnym krokiem jest **wysyłanie poleceń APDU** do karty SIM, które umożliwiają przechodzenie pomiędzy strukturami plików:

- SELECT MF (3F00) – wybór pliku głównego,

- SELECT DF TELECOM (7F10) – wybór katalogu z danymi użytkownika,
- SELECT EF_ADN (6F3A) – wybór pliku zawierającego książkę telefoniczną.

Po uzyskaniu dostępu do pliku EF_ADN program wykorzystuje komendę READ RECORD, aby **odczytać w pętli kolejne rekordy kontaktów.**
Otrzymane dane w formacie **szesnastkowym (HEX)** są następnie **konwertowane na postać czytelną ASCII**, co pozwala na wyświetlenie imion i numerów telefonów zapisanych w pamięci karty.

Dodatkowo aplikacja umożliwia wysyłanie **dowolnych komend APDU wprowadzanych przez użytkownika**, a także próbę odczytu wiadomości **SMS (EF_SMS)** w sposób analogiczny do kontaktów.



```
#define NOMINMAX
#include <windows.h>
#include <iostream>
#include <vector>
#include <string>
#include <iomanip>
#include <sstream>
#include <cwchar>
#include <limits>
#include <winscard.h>
#include <algorithm>
#pragma comment(lib, "Winscard.lib")

// HEX print
static void printHex(const BYTE* buf, DWORD len) {
    for (DWORD i = 0; i < len; ++i)
        std::cout << std::hex << std::uppercase
        << std::setw(2) << std::setfill('0') << (int)buf[i] << ' ';
    std::cout << std::dec << "\n";
}

// Convert GSM 7-bit to string (basic)
static std::string gsm7bitToString(const BYTE* data, DWORD len) {
    std::string result;
    for (DWORD i = 0; i < len; ++i) {
        BYTE b = data[i];
        if (b >= 0x20 && b <= 0x7E) result += (char)b;
        else if (b == 0x0A) result += "[LF]";
        else if (b == 0x0D) result += "[CR]";
        else result += '.';
    }
    return result;
}

// Convert phone number from BCD format
static std::string bcdToString(const BYTE* data, DWORD len) {
    std::string result;
    for (DWORD i = 0; i < len; ++i) {
        BYTE b = data[i];
        int lo = b & 0x0F;
        int hi = (b >> 4) & 0x0F;
        if (lo == 0x0F) {
            if (hi != 0x0F && hi != 0x00) result += char('0' + hi);
            break;
        }
        result += char('0' + lo);
        if (hi != 0x0F && hi != 0x00) result += char('0' + hi);
    }
    // trim trailing Fs/zeros
    while (!result.empty() && (result.back() == '\x0F' || result.back() == '0')) result.pop_back();
    return result;
```

```
        std::cout << std::hex << std::setw(2) << std::setfill('0') << (int)data[i] << ' ';
    std::cout << std::dec << "\n----------------------------------------\n";
}

// Display SMS in readable format
static void displaySMS(const BYTE* data, DWORD len, int index) {
    if (len < 2) return;

    std::cout << "SMS #" << index << ":\n";

    BYTE status = data[0];
    std::cout << "  Status: ";
    switch (status) {
    case 0x00: std::cout << "Nieprzeczytany"; break;
    case 0x01: std::cout << "Przeczytany"; break;
    case 0x03: std::cout << "Wysłany"; break;
    case 0x07: std::cout << "Nie wysłany"; break;
    default:   std::cout << "Nieznany (0x" << std::hex << (int)status << std::dec << ")"; break;
    }
    std::cout << "\n";

    // Attempt to parse sender (heuristic)
    for (DWORD i = 1; i + 2 < len; ++i) {
        if (data[i] == 0x91 || data[i] == 0x81) {
            BYTE numberLen = data[i + 1];
            if (i + 2 + numberLen <= len) {
                std::string sender = bcdToString(&data[i + 2], numberLen);
                if (!sender.empty()) {
                    std::cout << "  Nadawca: +" << sender << "\n";
                    break;
                }
            }
        }
    }

    if (len >= 7) {
        std::cout << "  Timestamp (raw): ";
        for (DWORD i = (len >= 7 ? len - 7 : 0); i < len; ++i)
            std::cout << std::hex << std::setw(2) << std::setfill('0') << (int)data[i] << ' ';
        std::cout << std::dec << "\n";
    }

    // Find text start (heuristic)
    DWORD textStart = 0;
    for (DWORD i = 10; i + 1 < len; ++i) {
        if (data[i] != 0x00 && data[i] != 0xFF) { textStart = i; break; }
    }
    if (textStart > 0 && textStart < len) {
        DWORD textLen = std::min<DWORD>(len - textStart, 160);
        std::string message = gsm7bitToString(&data[textStart], textLen);
        std::cout << "  Treść: " << message;
```

```
            if (textLen < (len - textStart)) std::cout << "...";
            std::cout << "\n";
        }

        std::cout << "  Dane HEX: ";
        for (DWORD i = 0; i < std::min<DWORD>(len, 32); ++i)
            std::cout << std::hex << std::setw(2) << std::setfill('0') << (int)data[i] << ' ';
        std::cout << std::dec << "\n----------------------------------------\n";
    }

    // Helper: transmit and print (for custom APDU)
    static LONG transmitAPDU_and_print(SCARDHANDLE card, DWORD proto, const std::vector<BYTE>& apdu) {
        const SCARD_IO_REQUEST* io = (proto == SCARD_PROTOCOL_T1) ? SCARD_PCI_T1 : SCARD_PCI_T0;

        BYTE resp[512]{};
        DWORD respLen = sizeof(resp);
        LONG rc = SCardTransmit(card, io, apdu.data(), (DWORD)apdu.size(), nullptr, resp, &respLen);
        if (rc != SCARD_S_SUCCESS) {
            std::cout << "SCardTransmit blad: 0x" << std::hex << rc << std::dec << "\n";
            return rc;
        }

        std::cout << "=> APDU: ";
        for (BYTE b : apdu) std::cout << std::hex << std::setw(2) << std::setfill('0') << (int)b << ' ';
        std::cout << std::dec << "\n";

        if (respLen >= 2) {
            DWORD dataLen = respLen - 2;
            BYTE sw1 = resp[respLen - 2], sw2 = resp[respLen - 1];

            std::cout << "<= DATA: ";
            for (DWORD i = 0; i < dataLen; ++i)
                std::cout << std::hex << std::setw(2) << std::setfill('0') << (int)resp[i] << ' ';
            std::cout << std::dec << "\n";

            std::cout << "ASCII: ";
            for (DWORD i = 0; i < dataLen; ++i) {
                unsigned char ch = resp[i];
                std::cout << ((ch >= 32 && ch <= 126) ? (char)ch : '.');
            }
            std::cout << "\n";

            std::cout << "STATUS: " << std::hex << std::setw(2) << std::setfill('0')
                << (int)sw1 << ' ' << (int)sw2 << std::dec << "\n";

            // GET RESPONSE handling if 61 xx
            if (sw1 == 0x61 && sw2 != 0x00) {
                BYTE le = sw2;
                std::vector<BYTE> getResp = { 0x00, 0xC0, 0x00, 0x00, le };
                std::cout << "  (GET RESPONSE " << std::hex << (int)le << std::dec << ")\n";
                respLen = sizeof(resp);
```

```
            rc = SCardTransmit(card, io, getResp.data(), (DWORD)getResp.size(), nullptr, resp, &respLen);
            if (rc == SCARD_S_SUCCESS && respLen >= 2) {
                DWORD dlen = respLen - 2;
                std::cout << "GET<= DATA: ";
                for (DWORD i = 0; i < dlen; ++i) std::cout << std::hex << std::setw(2) << std::setfill('0') << (int)resp[i] << ' ';
                std::cout << std::dec << "\n";
                std::cout << "GET<= STATUS: " << std::hex << (int)resp[respLen - 2] << ' ' << (int)resp[respLen - 1] << std::dec << "\n";
            }
            else {
                std::cout << "GET RESPONSE blad: 0x" << std::hex << rc << std::dec << "\n";
            }
        }
    }
    else {
        std::cout << "<= Pusta odpowiedz (respLen=" << respLen << ")\n";
    }
    return SCARD_S_SUCCESS;
}

// Read phonebook contacts (improved: CLA=00, handle 6C XX, log SW)
static void readPhonebook(SCARDHANDLE card, DWORD proto) {
    std::cout << "\n=== ODCZYT KSIĄŻKI TELEFONICZNEJ ===\n";

    // Use CLA=00 selects (more compatible)
    std::vector<BYTE> selectMF = { 0x00, 0xA4, 0x00, 0x00, 0x02, 0x3F, 0x00 };
    std::vector<BYTE> selectDF = { 0x00, 0xA4, 0x00, 0x00, 0x02, 0x7F, 0x10 };
    std::vector<BYTE> selectADN = { 0x00, 0xA4, 0x00, 0x00, 0x02, 0x6F, 0x3A };

    transmitAPDU_and_print(card, proto, selectMF);
    transmitAPDU_and_print(card, proto, selectDF);
    transmitAPDU_and_print(card, proto, selectADN);

    std::cout << "\n--- KONTAKTY ---\n";
    const SCARD_IO_REQUEST* io = (proto == SCARD_PROTOCOL_T1) ? SCARD_PCI_T1 : SCARD_PCI_T0;

    for (int i = 1; i <= 250; i++) { // iterate more records, break on 6A83 or other stop
        std::vector<BYTE> readRec = { 0x00, 0xB2, (BYTE)i, 0x04, 0x00 }; // Le=0 -> card may respond 6C xx
        BYTE resp[512]; DWORD respLen = sizeof(resp);
        LONG rc = SCardTransmit(card, io, readRec.data(), (DWORD)readRec.size(), nullptr, resp, &respLen);
        if (rc != SCARD_S_SUCCESS) {
            std::cout << "Blad przy SCardTransmit: 0x" << std::hex << rc << std::dec << "\n";
            break;
        }
        if (respLen < 2) {
            std::cout << "Pusta odpowiedz dla rekordu " << i << "\n";
            continue;
        }

        BYTE sw1 = resp[respLen - 2], sw2 = resp[respLen - 1];

        // handle 6Cxx (correct Le)
```

```
    const SCARD_IO_REQUEST* io = (proto == SCARD_PROTOCOL_T1) ? SCARD_PCI_T1 : SCARD_PCI_T0;

    for (int i = 1; i <= 250; i++) { // iterate more records, break on 6A83 or other stop
        std::vector<BYTE> readRec = { 0x00, 0xB2, (BYTE)i, 0x04, 0x00 }; // Le=0 -> card may respond 6C xx
        BYTE resp[512]; DWORD respLen = sizeof(resp);
        LONG rc = SCardTransmit(card, io, readRec.data(), (DWORD)readRec.size(), nullptr, resp, &respLen);
        if (rc != SCARD_S_SUCCESS) {
            std::cout << "Blad przy SCardTransmit: 0x" << std::hex << rc << std::dec << "\n";
            break;
        }
        if (respLen < 2) {
            std::cout << "Pusta odpowiedz dla rekordu " << i << "\n";
            continue;
        }

        BYTE sw1 = resp[respLen - 2], sw2 = resp[respLen - 1];

        // handle 6Cxx (correct Le)
        if (sw1 == 0x6C) {
            readRec.back() = sw2; // set Le to correct length
            respLen = sizeof(resp);
            rc = SCardTransmit(card, io, readRec.data(), (DWORD)readRec.size(), nullptr, resp, &respLen);
            if (rc != SCARD_S_SUCCESS) {
                std::cout << "Blad SCardTransmit po 6C: 0x" << std::hex << rc << std::dec << "\n";
                break;
            }
            if (respLen >= 2) {
                sw1 = resp[respLen - 2]; sw2 = resp[respLen - 1];
            }
        }

        if (sw1 == 0x90 && sw2 == 0x00 && respLen > 2) {
            displayContact(resp, respLen - 2, i);
        }
        else {
            std::cout << "Rekord " << i << " SW1SW2= "
                << std::hex << std::setw(2) << std::setfill('0') << (int)sw1 << ' '
                << std::setw(2) << (int)sw2 << std::dec << "\n";
            if (sw1 == 0x6A && sw2 == 0x83) { // record not found / out of range
                std::cout << "Koniec rekordow (6A83)\n";
                break;
            }
            if (sw1 == 0x69 && sw2 == 0x82) {
                std::cout << "Brak uprawnien (PIN zablokowany?)\n";
                break;
            }
            // other SW -> continue or break depending
            if (i > 50) break; // safety
        }
    }
}
```

```cpp

    // Read SMS messages (improved identical strategy)
static void readSMS(SCARDHANDLE card, DWORD proto) {
    std::cout << "\n=== ODCZYT WIADOMOŚCI SMS ===\n";

    std::vector<BYTE> selectMF = { 0x00, 0xA4, 0x00, 0x00, 0x02, 0x3F, 0x00 };
    std::vector<BYTE> selectDF = { 0x00, 0xA4, 0x00, 0x00, 0x02, 0x7F, 0x10 };
    std::vector<BYTE> selectSMS = { 0x00, 0xA4, 0x00, 0x00, 0x02, 0x6F, 0x3C };

    transmitAPDU_and_print(card, proto, selectMF);
    transmitAPDU_and_print(card, proto, selectDF);
    transmitAPDU_and_print(card, proto, selectSMS);

    std::cout << "\n--- WIADOMOSCI SMS ---\n";
    const SCARD_IO_REQUEST* io = (proto == SCARD_PROTOCOL_T1) ? SCARD_PCI_T1 : SCARD_PCI_T0;

    for (int i = 1; i <= 250; i++) {
        std::vector<BYTE> readRec = { 0x00, 0xB2, (BYTE)i, 0x04, 0x00 };
        BYTE resp[512]; DWORD respLen = sizeof(resp);
        LONG rc = SCardTransmit(card, io, readRec.data(), (DWORD)readRec.size(), nullptr, resp, &respLen);
        if (rc != SCARD_S_SUCCESS) {
            std::cout << "Blad przy SCardTransmit: 0x" << std::hex << rc << std::dec << "\n";
            break;
        }
        if (respLen < 2) {
            std::cout << "Pusta odpowiedz dla SMS " << i << "\n";
            continue;
        }

        BYTE sw1 = resp[respLen - 2], sw2 = resp[respLen - 1];

        if (sw1 == 0x6C) {
            readRec.back() = sw2;
            respLen = sizeof(resp);
            rc = SCardTransmit(card, io, readRec.data(), (DWORD)readRec.size(), nullptr, resp, &respLen);
            if (rc != SCARD_S_SUCCESS) { std::cout << "Blad SCardTransmit po 6C\n"; break; }
            if (respLen >= 2) { sw1 = resp[respLen - 2]; sw2 = resp[respLen - 1]; }
        }

        if (sw1 == 0x90 && sw2 == 0x00 && respLen > 2) {
            displaySMS(resp, respLen - 2, i);
        }
        else {
            std::cout << "SMS " << i << " SW1SW2= "
                << std::hex << std::setw(2) << std::setfill('0') << (int)sw1 << ' '
                << std::setw(2) << (int)sw2 << std::dec << "\n";
            if (sw1 == 0x6A && sw2 == 0x83) { std::cout << "Koniec rekordow (6A83)\n"; break; }
            if (sw1 == 0x69 && sw2 == 0x82) { std::cout << "Brak uprawnien (PIN zablokowany?)\n"; break; }
            if (i > 50) break;
        }
    }
```

```cpp
}

// Send custom APDU command (narrow strings)
static void sendCustomAPDU(SCARDHANDLE card, DWORD proto) {
    std::cout << "\n=== WYSYŁANIE WŁASNEJ KOMENDY APDU ===\n";

    std::string line;
    std::cout << "Podaj komendę APDU w HEX (np. 00 A4 00 00 02 3F 00):\n> ";
    if (!std::getline(std::cin, line) || line.empty()) {
        std::cout << "Pusta komenda.\n";
        return;
    }

    std::stringstream ss(line);
    std::string tok;
    std::vector<BYTE> apdu;
    while (ss >> tok) {
        try {
            unsigned int val = std::stoul(tok, nullptr, 16);
            apdu.push_back(static_cast<BYTE>(val & 0xFF));
        }
        catch (...) {
            std::cout << "Bledny format. HEX oddzielony spacjami.\n";
            return;
        }
    }

    if (apdu.empty()) { std::cout << "Pusta komenda.\n"; return; }

    // Use transmitAPDU_and_print for output and GET RESPONSE handling
    transmitAPDU_and_print(card, proto, apdu);
}

int main() {
    std::cout << "=== PROGRAM DO ODCZYTU DANYCH Z KART SIM ===\n\n";

    // 1) KONTEKST
    SCARDCONTEXT ctx;
    LONG st = SCardEstablishContext(SCARD_SCOPE_USER, nullptr, nullptr, &ctx);
    if (st != SCARD_S_SUCCESS) {
        std::cout << "Blad: nie moge nawiazac kontekstu (Smart Card service).\n";
        return 1;
    }

    // 2) LISTA CZYTNIKOW
    LPWSTR readers = nullptr;
    DWORD len = SCARD_AUTOALLOCATE;
    LONG r = SCardListReadersW(ctx, nullptr, (LPWSTR)&readers, &len);
    if (r != SCARD_S_SUCCESS) {
        std::cout << "Nie wykryto podlaczonych czytnikow (lub brak sterownikow).\n";
        SCardReleaseContext(ctx);
```

```
388         return 1;
389     }
390
391     std::vector<std::wstring> lista;
392     const wchar_t* p = readers;
393     while (*p) {
394         lista.emplace_back(p);
395         p += wcslen(p) + 1;
396     }
397     if (lista.empty()) {
398         std::cout << "Lista czytnikow pusta.\n";
399         SCardFreeMemory(ctx, readers);
400         SCardReleaseContext(ctx);
401         return 1;
402     }
403
404     // a) Wyświetl nazwę podłączonego urządzenia-czytnika
405     std::wcout << L"\n(a) Podlaczone urzadzenie-czytnik:\n";
406     for (size_t i = 0; i < lista.size(); ++i)
407         std::wcout << L"  [" << i << L"] " << lista[i] << L"\n";
408
409     // b) Wybór czytnika (use getline to avoid mixing)
410     std::string line;
411     size_t idx = 0;
412     while (true) {
413         std::cout << "\n(b) Wybierz numer czytnika: ";
414         if (!std::getline(std::cin, line)) { std::cout << "Blad wejsciowy\n"; return 1; }
415         try {
416             idx = static_cast<size_t>(std::stoul(line));
417             if (idx < lista.size()) break;
418         }
419         catch (...) {}
420         std::cout << "Niepoprawny numer. Sprobuj ponownie.\n";
421     }
422     std::wcout << L"Wybrano: " << lista[idx] << L"\n";
423
424     // 4) POLACZENIE
425     SCARDHANDLE card;
426     DWORD proto = 0;
427     LONG c = SCardConnectW(ctx, lista[idx].c_str(),
428         SCARD_SHARE_SHARED,
429         SCARD_PROTOCOL_T0 | SCARD_PROTOCOL_T1,
430         &card, &proto);
431     if (c != SCARD_S_SUCCESS) {
432         std::cout << "Nie moge polaczyc z karta. Kod: 0x" << std::hex << c << std::dec << "\n";
433         SCardFreeMemory(ctx, readers);
434         SCardReleaseContext(ctx);
435         return 1;
436     }
437     std::cout << "Polaczono. Protokol: "
438         << ((proto & SCARD_PROTOCOL_T0) ? "T0 " : "")
```

```
439             << ((proto & SCARD_PROTOCOL_T1) ? "T1 " : "") << "\n";
440
441         // 5) ATR
442         BYTE atr[64] = {};
443         DWORD atrLen = sizeof(atr);
444         DWORD state = 0, proto2 = 0;
445         LONG s = SCardStatusW(card, nullptr, nullptr, &state, &proto2, atr, &atrLen);
446         if (s == SCARD_S_SUCCESS) {
447             std::cout << "ATR: ";
448             printHex(atr, atrLen);
449         }
450         else {
451             std::cout << "Nie udalo sie pobrac ATR. Kod: 0x" << std::hex << s << std::dec << "\n";
452         }
453
454         // MENU GŁÓWNE (use getline for choices)
455         int choice = 0;
456         do {
457             std::cout << "\n=== MENU GLOWNE ===\n";
458             std::cout << "1. Wyslij wlasna komende APDU (HEX)\n";
459             std::cout << "2. Odczytaj ksiazke telefoniczna\n";
460             std::cout << "3. Odczytaj wiadomosci SMS\n";
461             std::cout << "4. Wyjscie\n";
462             std::cout << "Wybierz opcje: ";
463
464             if (!std::getline(std::cin, line)) break;
465             try { choice = std::stoi(line); }
466             catch (...) { choice = -1; }
467
468             switch (choice) {
469             case 1: sendCustomAPDU(card, proto); break;
470             case 2: readPhonebook(card, proto); break;
471             case 3: readSMS(card, proto); break;
472             case 4: std::cout << "Zamykanie programu...\n"; break;
473             default: std::cout << "Niepoprawny wybor.\n"; break;
474             }
475
476         } while (choice != 4);
477
478         SCardDisconnect(card, SCARD_LEAVE_CARD);
479         SCardFreeMemory(ctx, readers);
480         SCardReleaseContext(ctx);
481         return 0;
482     }
483
```

7. Wyniki i obserwacje

Podczas realizacji ćwiczenia program napisany w języku **C++ z wykorzystaniem biblioteki WinSCard.h** poprawnie nawiązywał połączenie z czytnikiem **OMNIKEY 5321v2** oraz z włożoną kartą SIM.
Wykorzystano komendy **SELECT MF**, **SELECT DF TELECOM**, **SELECT EF_ADN** oraz **GET RESPONSE**, które zwracały poprawne kody statusu **(90 00)**, co potwierdzało prawidłową komunikację z kartą i dostęp do odpowiednich plików systemu plików SIM.

Podczas pierwszych prób wykorzystano kartę SIM, która okazała się **zablokowana**, przez co nie było możliwe odczytanie danych IMSI ani zawartości plików EF. W efekcie nie udało się uzyskać dostępu do listy kontaktów ani wiadomości SMS.
Po **zmianie karty SIM** w czytniku program ponownie nawiązał połączenie i poprawnie odczytał informacje z karty. W menu programu możliwe było przeglądanie **listy kontaktów (EF_ADN)** oraz **wiadomości SMS (EF_SMS)**.

Zaobserwowano jednak, że przy odczycie niektórych rekordów pojawiały się **nieprawidłowe lub nieczytelne dane**. Było to spowodowane różnicami w długości rekordów i formatami zapisu danych (m.in. **GSM 7-bit** oraz **UCS2**), które wymagały dodatkowej interpretacji w kodzie programu.
Zauważono również, że **uruchomienie dwóch programów korzystających z czytnika jednocześnie (np. SIMCardManager oraz własnego programu)** powodowało błędy transmisji i niemożność uzyskania odpowiedzi od karty – dlatego do testów używano wyłącznie jednej aplikacji jednocześnie.

Całość potwierdziła poprawne działanie biblioteki PC/SC, nawiązanie połączenia z czytnikiem oraz prawidłową strukturę i logikę transmisji komend **APDU**.

8. Wnioski

Własny program oparty na bibliotece **WinSCard** poprawnie nawiązywał połączenie z czytnikiem kart oraz przesyłał komendy **APDU**, co potwierdziło prawidłową komunikację z kartą SIM.
Podstawowe polecenia, takie jak **SELECT MF** czy **GET RESPONSE**, zwracały status **90 00**, świadczący o poprawnym wykonaniu operacji.

Problemy pojawiły się podczas próby odczytu danych z plików **EF_ADN** (książka telefoniczna) oraz **EF_SMS** (wiadomości SMS). Funkcje odpowiedzialne za te operacje zwracały niepoprawną liczbę rekordów oraz dane w formacie nieczytelnym, co sugeruje, że karta mogła być częściowo zablokowana lub wymagała autoryzacji poprzez kod **PIN/PUK**. Nieprawidłowy odczyt mógł być również spowodowany różnicami w implementacji standardów zapisu danych (**GSM 7-bit** i **UCS2**) pomiędzy poszczególnymi kartami SIM.

Ostatecznie potwierdzono, że program poprawnie realizuje połączenie i komunikację z kartą SIM, jednak pełny odczyt danych wymagałby rozszerzenia implementacji o obsługę autoryzacji użytkownika oraz dokładniejsze parsowanie struktur rekordów SIM.

9. Bibliografia

- ForensicsWare – Digital Forensics: SIM Card Analysis
- Ambimat Electronics – Smart Card Selected APDU Commands
- Skilled Engineer – SIM Card Pinouts and Electrical Connections
- Dokumentacja Microsoft: WinSCard API Reference
- Zdjęcie opisu styków karty Sim: https://ar.pinterest.com/pin/850758185867424894/
- Zdjęcie APDU struktury: https://www.researchgate.net/figure/Command-APDU-structure-SOURCE-9_fig1_220369444