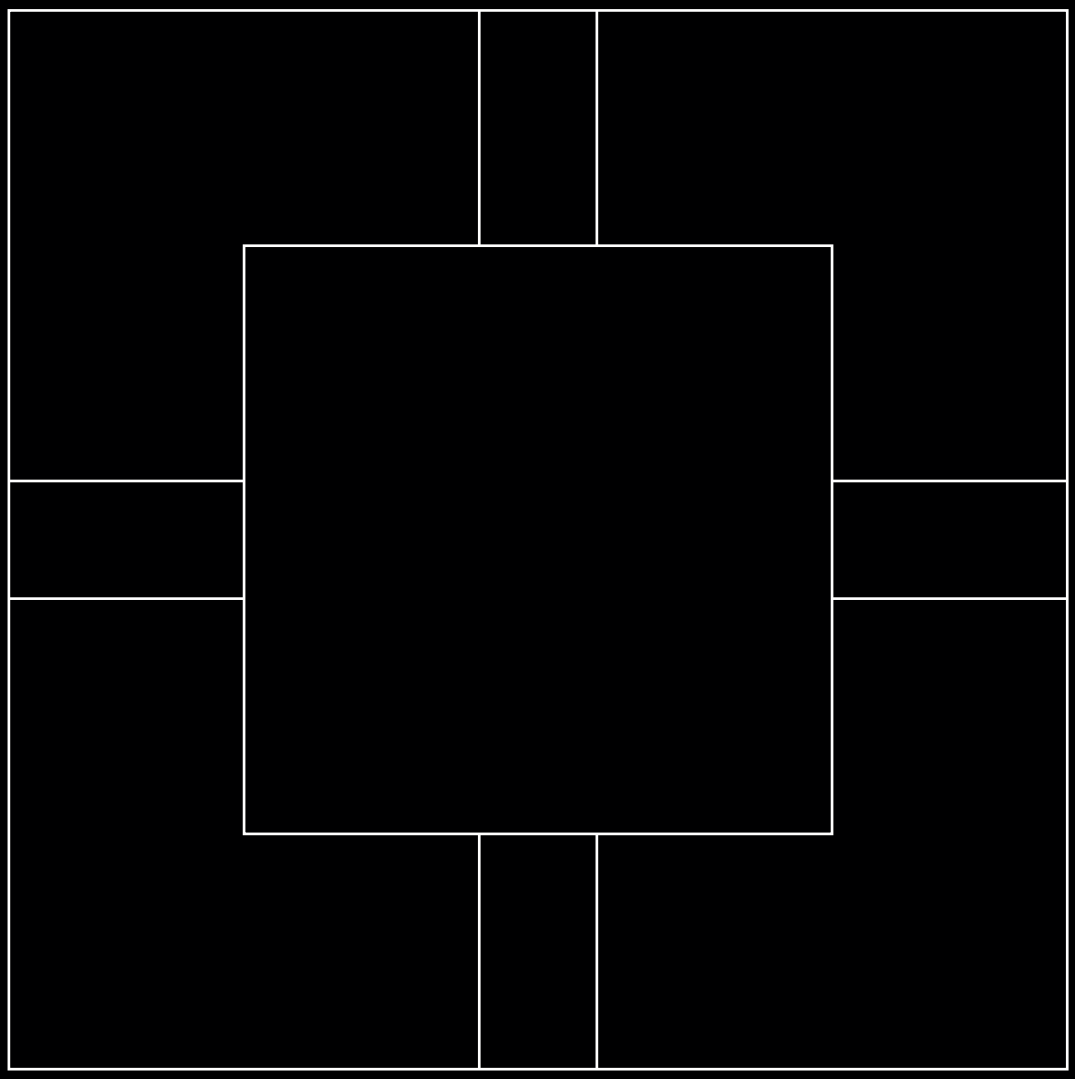
Monografie Politechniki Łódzkiej

Bezpieczne systemy wbudowane Zastosowania

Krzysztof Lichy, Kamil Lorenc



Krzysztof Lichy, Kamil Lorenc

Bezpieczne systemy wbudowane. Zastosowania

Łódź 2016

Bezpieczne systemy wbudowane Zastosowania

Krzysztof Lichy, Kamil Lorenc

Monografie Politechniki Łódzkiej
Łódź 2016

Recenzenci:
prof. dr hab. inż. Liliana Byczkowska-Lipińska
dr hab. inż. Piotr Napieralski

Redaktor Naukowy Wydziału Fizyki Technicznej,
Informatyki i Matematyki Stosowanej:
dr hab. inż. Aneta Poniszewska-Marańda

Projekt okładki:
dr inż. Krzysztof Guzek

WYDAWNICTWO POLITECHNIKI ŁÓDZKIEJ
90-924 Łódź, ul. Wólczańska 223
tel. 42-631-20-87, 42-631-29-52
fax 42-631-25-38
e-mail: zamowienia@info.p.lodz.pl
www.wydawnictwa.p.lodz.pl

ISBN 978-83-7283-884-1

Reprodukcja z materiałów dostarczonych przez Autorów

Nakład 50 egz. Ark. 6,5 wyd. Ark. druk. 8,0. Papier offset. 80 g, 70 x 100
Wykonano w Drukarni „Quick-Druk" s.c. 90-562 Łódź, ul. Łąkowa 11
Nr 2239

Spis treści

Wstęp

System wbudowany to dedykowany system komputerowy złożony z odpowiednio dobranych komponentów programowych i sprzętowych. Jego przeznaczeniem jest wykonywanie określonej aplikacji programowej. Ze względu na wszechobecne dążenie do opracowywania „inteligentnych" urządzeń systemy wbudowane mają szerokie zastosowania w wielu dziedzinach.

Obecnie można zaobserwować, że systemy wbudowane po latach przebywania niejako w cieniu komputerów osobistych przeżywają swoisty rozkwit związany między innymi ze znacznym spadkiem cen podzespołów. Współcześnie, takie urządzenia, jak telewizor wyposażone są w podzespoły pozwalające im konkurować o uwagę użytkownika z komputerami stacjonarnymi lub laptopami. Spadek cen spowodował masową dostępność tanich, łatwych do nauki platform deweloperskich, ułatwiających wejście w świat budowy systemów wbudowanych.

W pracy, na przykładzie systemu lokalizującego i „proof of concept" menadżera haseł wbudowanego w popularną kartę chipową, zostanie przedstawione spektrum możliwości stosowania systemów wbudowanych.

1. Wprowadzenie w problematykę

Celem pracy jest przedstawienie zastosowania systemów wbudowanych do budowy systemu lokalizującego pojazdy. Autorzy zwrócili uwagę na to, że ceny dostępnych obecnie na rynku gotowych rozwiązań zdecydowanie nie odzwierciedlają kosztów użytych w nich podzespołów. Moduł GPS jest obecny niemal na każdym kroku, przy częstych podróżach używa się nawigacji GPS. Często nawigacja jest wbudowana jako element w samochodzie, można ją znaleźć smartfonie. To samo dotyczy modemów GSM i telefonów komórkowych. Wszystko to powoduje, że ceny samych modułów są bardzo niskie. Wobec tego ceny modułów lokalizacyjnych powinny być niskie. Jednak gdy przeanalizuje się ceny oferowanych systemów lokalizacyjnych, to okazuje się, że nie mają one żadnego przełożenia w cenach podzespołów.

Biorąc pod uwagę powyższe czynniki postanowiono przedstawić projekt i implementację „autorskiego systemu wbudowanego" lokalizującego z zastosowaniem popularnych elementów. „Autorski system" jest przeznaczony do lokalizacji pojazdów. W pracy zostały przedstawione następujące etapy budowy systemu:

- analiza dostępnego sprzętu, jego zalet oraz wad,
- zaprojektowanie schematu elektronicznego systemu,
- implementacja oprogramowania obsługującego system (firmware)

Niewątpliwą zaletą takiego systemu jest zastosowanie systemów wbudowanych, które pozwalają na zastosowanie do procesu budowy oprogramowania niskopoziomowych funkcji udostępnianych przez sam procesor. Pozwala to jednocześnie na głębsze zrozumienie zasad budowy wszystkich etapów przekształcania kodu źródłowego w kod maszynowy, konieczność nieprzerwanego działania systemu oraz opracowania odpowiednich metod przywracania systemu z sytuacji wyjątkowych.

2. Zagadnienie lokalizacji

2.1 Zarys historyczny problematyki

Początki urządzeń służących do wyznaczania własnej pozycji datuje się na co najmniej VIII wiek naszej ery, gdy żeglarze arabscy zaczęli wykorzystywać astrolabium, będące w zasadzie okrągłą płytą z oznaczonymi ciałami niebieskimi, do określania własnej szerokości geograficznej. W XVIII wieku zostało ono zastąpione przez nieco bardziej skomplikowane sekstanty, które stosowano do obserwacji ciał niebieskich do czasy zastosowania fali elektromagnetycznej w komunikacji radiowej. W nawigacji morskiej zaczęto stosować radiolatarnie, co umożliwiło określanie pozycji znacznie dokładniej i znosiło ograniczenie w określaniu jedynie szerokości geograficznej. Zasadniczą wadą tego typu nawigacji był zasięg co powodowało, że na pełnym morzu, z dala od portów, wciąż w użyciu były prymitywne sekstanty.

Prawdziwa rewolucja nastąpiła dopiero w ostatnich latach zeszłego stulecia, gdy władze amerykańskie udostępniły tworzony przez siebie od lat siedemdziesiątych system nawigacji satelitarnej GPS. Od tamtego momentu nawigacja GPS upowszechniła się do tego stopnia, że dziś jest dostępna dla każdego. Mniej więcej w tym samym czasie co GPS do powszechnego użytku weszły systemy bezprzewodowej komunikacji na małe odległości takie jak GSM czy WiFi, rewolucjonizując styl życia społeczeństw.

Ze względu na stopień rozwoju technologii bezprzewodowych, dzisiaj niemal każdy może przy użyciu niewielkiego urządzenia (takiego jak na przykład smartfon) zlokalizować wybrany obiekt, za cenę jedynie zainstalowanego na nim oprogramowania.

Początkowo systemy lokalizacji obejmowały tylko określanie własnej pozycji, nie zdobyły jednak dużej popularności. Możliwość lokalizacji zewnętrznych obiektów uzyskało dużo większe zainteresowanie. Od pewnego czasu w gronie stałych zainteresowanych lokalizacją zewnętrznych obiektów były służby specjalne oraz właściciele dużych flot pojazdów. Grono użytkowników tej technologii stale rośnie. Najpierw firmy motoryzacyjne

na rynku amerykańskim i na europejskim zaczęły oferować systemy automatycznego wzywania pomocy w razie wypadku, korzystające z systemu GPS oraz sieci komórkowych. W Unii Europejskiej system taki, zwany eCall, stanie się za parę lat obowiązkowy [10]. Powoduje to szybkie upowszechnianie się systemów nawigacji a wraz z ich rozwojem powiązanych z nimi usług np. usługi lokalizacji skradzionych pojazdów. Istnieją również zamknięte systemy na bazie rozwiązań własnościowych np. system LoJack stosujący nadajnik na zarezerwowanej dla niego częstotliwości.

Specyfika branży powoduje dużą standaryzację rozwiązań lokalizacyjnych i utrzymuje niewielkie zainteresowanie systemami niezależnych producentów. Przekłada się to na ich cenę. Niestety tak duże ujednolicenie systemów lokalizacyjnych spowoduje zapewne, w dalszej perspektywie, ich całkowitą nieprzydatność jako zabezpieczeń antykradzieżowych. Złodziej, kradnąc samochód, będzie potrzebował rozpracowania tylko jednego systemu, który na pewno będzie stosował system GPS do określania pozycji oraz sieci komórkowej do komunikacji ze światem. Może to spowodować, że producenci, pomimo technicznych możliwości, będą oferować klientom jedynie obligatoryjne usługi automatycznej pomocy (eCall), a za dodatkowe funkcje każą sobie dopłacać, na przykład w formie abonamentu, który dzięki braku realnej konkurencji będzie miał wysokie koszty.

Z powyższych powodów, budując od podstaw system lokalizacji pojazdów, należy skonstruować go w taki sposób, aby umożliwiał wprowadzanie nowych funkcji. Bardzo ważne jest również, aby cena podzespołów była jak najniższa i nie ograniczała możliwości rozbudowy. Należy również zwrócić uwagę aby system lokalizacji spełnił wszystkie wymagania potencjalnych użytkowników. System taki musi spełniać kryteria autonomiczności, to znaczy musi on działać poprawnie bez jakiekolwiek ingerencji ze strony użytkownika. Jest to niezbędne, gdyż typowy użytkownik sprzętu elektronicznego ma tendencję do braku zainteresowania się szczegółami otaczających go sprzętów. Stąd jeżeli hipotetycznie lokalizator, który zainstalował we własnym pojeździe wymagałby od niego ciągłego sprawdzania czy sprzęt działa poprawnie. Kolejną bardzo istotną, choć dość często pomijaną kwestią jest zagadnienie zasilania takiego układu. Każdy pojazd posiada wprawdzie akumulator, jednak poleganie jedynie na nim może okazać się niewystarczające. Dobry system powinien więc posiadać własne, niezależne źródło zasilania, a zasilanie z pojazdu pobierać

jedynie w momencie doładowywania wewnętrznej baterii. Najlepiej aby w czasie ładowania, silnik pojazdu był włączony. Jak widać, ze względu na powyższe ograniczenia zaprojektowanie a następnie budowa dobrego systemu lokalizacyjnego nie jest trywialnym zagadnieniem.

2.2 Alternatywne metody określania pozycji

2.2.1 System nawigacyjny GPS (Global Positioning System)

GPS jest obecnie najpopularniejszym systemem nawigacyjnym na świecie. Jego budowa rozpoczęła się w latach siedemdziesiątych XX wieku, jako projekt wojskowy pod auspicjami Departamentu Obrony USA. Celem projektu było dostarczenie niezawodnego systemu lokalizacji bazującego na konstelacji satelitów. Jego podstawowymi odbiorcami miały być Marynarka Wojenna i Lotnictwo Armii Stanów Zjednoczonych Ameryki.

Główną i niepodważalną zaletą jest globalny zasięg rozwiązania zapewniany przez ponad trzydzieści satelitów krążących nad Ziemią. Istotną zaletą jest również brak opłat za wykorzystanie oraz bardzo niski koszt sprzętu. Do zalet systemu należy również duża dokładność w ustalaniu pozycji (rząd kilku metrów zależnie od liczby „widzianych" satelitów).

Największą wadą rozwiązania jest łatwość i podatność sygnału na zakłócenie. Związane jest to z zastosowaniem zaledwie dwóch stałych częstotliwości oraz brak szyfrowania sygnału. Pewnym problemem jest również czas potrzebny na uzyskanie tzw. fixa, czyli ustalenie pozycji, szczególnie zaraz po włączeniu odbiornika, gdzie nie ma on żadnych informacji o pozycji satelitów (*cold fix*).

2.2.2 Inne systemy nawigacji satelitarnej

System GPS nie jest jedynym istniejącym obecnie systemem nawigacji satelitarnej. Swoje własne systemy rozwijają obecnie wszystkie liczące się mocarstwa. Spośród konkurentów GPS, jedynie dwóch osiągnęło już globalną zdolność lokalizacyjną – rosyjski GLONASS oraz chiński BDS. Ten pierwszy jest nawet stosowany w większości współczesnych odbiorników. Użycie odbiornika działającego dwu- lub nawet więcej zakresowo daje szansę na wyeliminowanie łatwości zakłócenia. Niestety należy się spodziewać, że dostępne komercyjnie odbiorniki nie będą odporne

na zakłócenie tylko jednego z pasm, skutecznie uniemożliwiając określenie pozycji, pomimo dostępności satelit GLONASS. Warto nadmienić, że w budowie znajdują się już kolejne systemy takie jak indyjski IRNSS czy europejski Galileo. Poza dywersyfikacją źródeł i częstotliwości sygnału, co niewątpliwie sprawia, że znalezienie odbiornika zdolnego korzystać z systemów innych niż GPS może być bardzo dużą zaletą, cała reszta wad i zalet systemu GPS ma zastosowanie także w tym przypadku.

2.2.3 Możliwość zastosowania sieci GSM

Metoda ta była dość popularna w czasach, gdy większość telefonów nie była jeszcze wyposażona w moduł GPS do ustalania przybliżonej pozycji, szczególnie przez aplikacje mapowe. Dziś jest ona już raczej zmarginalizowana. Stosuje ona identyfikator stacji bazowej sieci komórkowej, w postaci parametrów LAC i CID, określających operatora oraz ID komórki wewnątrz jego sieci. Do zalet takiego rozwiązania należy zaliczyć praktycznie zerowy koszt (warunek to posiadanie modułu GSM/3G) . Zaletą jest również stosunkowo duża odporność na zakłócenia sygnału (celowe lub przypadkowe). Jest to wynik zastosowania bardzo szerokiego pasma częstotliwości współczesnych modemów, przykładowo: GSM900, GSM1800, UMTS2100, LTE2600 etc.) . Nie ma również problemu z czasem określenia pozycji. Jest on natychmiastowy ze względu na to, że modem zawsze jest podłączony do jakiejś stacji bazowej. Niestety rozwiązanie to obarczone jest dwoma istotnymi wadami. Ze względu na specyfikę protokołu istnieje możliwość podstawienia fałszywej stacji bazowej przy użyciu urządzeń typu IMSI Catcher z fałszywymi parametrami LAC/CID w pobliżu celu, co spowoduje, że urządzenie lokalizujące połączy się z BTS-em o najsilniejszym sygnale – a więc fałszywym. Mamy również do czynienia z stosunkowo niewielką dokładnością. Określenie, do której komórki jest podłączony użytkownik w danym momencie poza terenem zabudowanym (gdzie sieć nadajników jest zdecydowanie rzadsza) może owocować dokładnością rzędu kilku kilometrów.

2.2.4 Bezpośrednie zastosowanie stacji bazowych

Sieci komórkowe mogą zostać użyte także w znacznie szerszym zakresie niż tylko i wyłącznie użycie identyfikatora aktualnej stacji bazowej.

Teoretycznie możliwe jest kilkukrotne przełączenie modemu (nazwanego w nomenklaturze GSM ME - Mobile Equipment) pomiędzy różnymi stacjami bazowymi (BTS), zapamiętując jednocześnie siłę sygnału, określoną przez parametr RSSI (Received signal strength indication). Pozwoliłoby to na wykonanie triangulacji. Konieczna byłaby jednak znajomości bardzo dokładnej lokalizacji wszystkich stacji bazowych na terenie całego kraju. Niestety, problemem może okazać się oprogramowanie modemu, które w standardzie GSM 07.07 oraz GSM 07.05 nie określa nawet komendy manualnego wybierania stacji bazowej. Z tego względu, aby wykonać triangulację, należałoby poznać potencjalne interfejsy służące producentowi do debugowania oprogramowania modemu, a być może nawet dokonać inżynierii wstecznej samego firmware, co oprócz problemów prawnych generowałoby olbrzymi narzut czasowy potrzebny do przeprowadzenia całej operacji. Zalety takiego rozwiązania to zdecydowanie zwiększona, w porównaniu do użycia pary LAC/CID, dokładność określonej pozycji. Podobnie jak w poprzednim przypadku rozwiązanie jest odporne na zakłócenia. Wady rozwiązania są również podobne. Istnieje możliwość postawienia fałszywej stacji bazowej, warto jednak zauważyć, że istnieje teoretyczna możliwość wykrycia stacji bazowej niepasującej do otoczenia. Istotną wadą jest również czas potrzebny na zbudowanie prototypu i konieczność zidentyfikowania modemu pozwalającego na manipulację w oprogramowaniu.

2.2.5 Zastosowanie istniejących sieci standardu 802.11

Dziś niemal na każdym kroku istnieją dziesiątki działających w okolicy sieci WiFi. Każda taka sieć przedstawia się dużą ilością parametrów, z czego dwa podstawowe, BSSID (Basic service set identification) oraz SSID (service set identifier) w zasadzie pozostają w czasie działania niezmienne. SSID jest wybieralny przez użytkownika, a więc od czasu do czasu może ulegać zmianom. BSSID, będący faktycznie adresem MAC punktu dostępowego, powinien być z definicji przypisany do konkretnego producenta i jednocześnie unikalny w skali globu. Dziś, ze względu na „wysyp" taniego chińskiego i niecertyfikowanego w Stanach Zjednoczonych czy Unii Europejskiej sprzętu sieciowego warunek unikalności jest często niespełniony. Mimo to prawdopodobieństwo trafienia dwóch identycznych adresów jest bliskie zeru. Pewnym problemem, szczególnie,

w krajach rozwijających się, jest odsprzedaż sprzętu, gdzie ten sam access point zaczyna działać w zupełnie innym miejscu niż dotychczas, co może doprowadzić do zmylenia wszelkich systemów lokalizacyjnych zbudowanych na sztywnych algorytmach (na przykład uśredniających pozycję takiego punktu). Problemem jest także dokładność określonej w ten sposób pozycji, nawet pomijając powyższe problemy, gdyż określenie dokładnej pozycji samego punktu dostępowego jest dość trudne oraz wymaga dużej ilości obserwacji. Z tego względu utrzymanie dobrej i aktualnej bazy danych wymaga ogromnych nakładów. Na to może pozwolić Google, który buduje swoją bazę niejako przy okazji tworzenia zdjęć do usługi Street View. Z tego względu baza ta nie będzie sprawdzała się w terenie słabo zurbanizowanym, gdzie samochód Google z dużym prawdopodobieństwem nie trafił.

Drugim serwisem lokalizacji jest mało znany szerszej publiczności serwis WiGLE, bazujący o obserwacje wolontariuszy. Tutaj, pomimo większych szans na pokrycie terenów mało uczęszczanych, istnieje ryzyko przemycenia do bazy fałszywych danych. Niestety, nawet zakładając idealne pokrycie świata danymi oraz zlokalizowania punktów dostępowych z dokładnością do metra, system lokalizujący bazujący o sieci WiFi, wciąż nie będzie mógł konkurować z dokładnością systemu GPS, ze względu na ogromną ilość czynników wpływających na siłę sygnału obserwowanych sieci, która wydaje się być najlepszym wyznacznikiem odległości używanej do ewentualnej triangulacji. Siła ta bowiem może zależeć przykładowo od ilości, grubości czy materiału, z jakiego składają się przeszkody terenowe zasłaniające obserwatorowi dany punkt. Z tego względu pozycja określona na bazie sieci WiFi jest z góry obarczona błędem, który może dochodzić do kilku metrów. Główną zaletą takiego podejścia to możliwość zastosowania bardzo popularnej infrastruktury. Do zdecydowanych wad zaliczyć należy konieczność zależności od zewnętrznych baz danych oraz potrzebę zbudowania i utrzymania własnej. Koszty takiego rozwiązania byłby trudne do oszacowania. Pamiętać należy również o dokładności, która choć zdecydowanie wyższa niż w przypadku sieci GSM, to jednak wciąż ustępująca systemom nawigacji satelitarnej. Istotnym problemem jest również brak możliwości skorzystania w obszarach niezurbanizowanych.

2.2.6 Własna sieć nadajników

Każdy sposób zastosowania komunikacji bezprzewodowej niesie za sobą ryzyko celowego zakłócenia (ang. *jamming*) lub sfałszowania (ang. *spoofing*). Aby system lokalizacyjny nadawał się do zastosowania jako zabezpieczenie antykradzieżowe (metoda odzyskania skradzionego pojazdu) należy użyć systemu, którego nie spodziewa się potencjalny złodziej. Dobrym pomysłem wydaje się użycie całkowicie własnego systemu działającego w zakresie trudnej do przewidzenia częstotliwości, która nie pozwoli na wprowadzenie zakłóceń. Niestety, takie rozwiązanie wymagałoby budowy własnej infrastruktury złożonej z dziesiątków lub setek nadajników (gęstość takiej sieci zależy od użytej częstotliwości, im niższa tym mniej nadajników, ale też trudniej o koncesję). Ograniczenia tej metody oznaczają, że w praktyce niemożliwe jest, aby na jakimś terenie powstało ich więcej niż kilka. Z drugiej strony warto jednak zauważyć, że żadna z wymienionych wcześniej metod tak naprawdę nie umożliwiała uwierzytelnienia się stacji wobec lokalizatora, a zatem wszystkie istniejące systemy są podatne na podszycie się intruza i w konsekwencji ustalenie przez system błędnej lokalizacji. Ponieważ taki system może zostać opracowany od podstaw, daje możliwość zdefiniowania silnych algorytmów weryfikujących tożsamość elementów systemu. Zalety takiego systemu to potencjalna możliwość wbudowania odporności na fałszowanie transmisji i kłopot z dostępem do dokumentacji rozwiązania dla osób chcących skompromitować system. Zdecydowaną wadą rozwiązania jest konieczność poniesienia znacznych nakładów finansowych potrzebnych na zaprojektowanie, zbudowanie i utrzymanie sieci własnych nadajników.

2.2.7 Wybór metody do zastosowania

Ponieważ wszystkie opisane wyżej metody mają swoje indywidualne wady i zalety, żadna z nich nie może być uznana za uniwersalne rozwiązanie wszystkich problemów, jakie niosą różne zastosowania urządzeń lokalizujących pojazdy. Z tego względu żaden system nie może ograniczyć się wyłącznie do jednej metody. Sensowną niezawodność, szczególnie w dziedzinie zabezpieczeń antykradzieżowych, gdzie jest ona poddana największej próbie, można uzyskać łącząc ze sobą jak największą liczbę metod

o całkowicie odmiennej charakterystyce. Aby dokonać właściwego wyboru, poniżej znajduje się zestawienie podstawowych cech wszystkich powyższych systemów.

Tabela 1. Porównanie metod lokalizacji

Metoda	Koszty	Opłaty	Dokładność	Zasięg	Zakres częstotliwości	Wiarygodność [1]	Łatwość implementacji	Suma
GPS	**4**	**4**	**3**	**3**	0	0	**5**	19
Alt. systemy naw. satelitarnej	3	**4**	**3**	2	2	0	1	15
LAC/CID	3	1	0	2	**3**	1	**5**	15
Triangulacja GSM	1	1	2	2	**3**	2	0	11
Triangulacja 802.11	2	**4**	2	2	2	2	2	16
Sieć nadajników	0	0	**3**	2	**3**	**3**	0	11
Własny nadajnik	1	**4**	0	0	**3**	**3**	2	13
Maks.	4	4	3	3	3	3	5	25

Jak łatwo zauważyć, zdecydowanym i niekwestionowanym zwycięzcą okazał się system GPS, który pomimo wielu wad okazał się najlepszym rozwiązaniem. Warto również odnotować dobry wynik niedokładnej metody korzystającej z parametrów LAC i CID oraz używającej sieci standardu 802.11. Z przeprowadzonej analizy wynika jasno, że każdy system lokalizacji musi za punkt wyjścia obrać lokalizację GPS. Dopiero w przypadku specjalizowania do lokalizacji skradzionych pojazdów stosować inne metody ze wskazaniem na użycie sieci 802.11.

2.3 Problem przekazania zebranych danych

Poza kwestią określenia własnej pozycji, system lokalizacyjny powinien być w stanie przekazać zebrane dane na zewnątrz. Aby to umożliwić konieczna jest forma komunikacji bezprzewodowej. Istnieje kilka metod przekazywania danych na zewnątrz.

2.3.1 Możliwości sieci telefonicznej GSM

Standardowym i oczywistym sposobem przesyłania danych jest zastosowanie sieci telefonicznej GSM. Niewątpliwą zaletą tego rozwiązania jest bardzo dobre pokrycie terenu nadajnikami sieci komórkowej. Jest to również relatywnie tania metoda. Zauważyć należy, że zastosowanie wiadomości SMS znacznie ogranicza czas potrzebny na otrzymanie odpowiedzi od systemu oraz może generować pewne koszty przy częstym sprawdzaniu pozycji. Z tego względu nie nadaje się do stałego monitoringu pozycji pojazdu, a jedynie do okazjonalnego zastosowania. Aby zastosować sieć GSM należy posiadać modem GSM, którego cena, pomimo niskich cen korzystających z nich telefonów komórkowych pozostaje na relatywnie wysokim poziomie. Warto odnotować, że jeśli chodzi o odporność na zakłócenia to sytuacja jest identyczna jak w przypadku lokalizacji do zakłócenia komunikacji wystarczy IMSI Catcher. Zalety:

- duży zasięg,
- niewielkie koszty obsługi systemu.

Wady:

- cena modemu GSM,
- ograniczona ilość informacji możliwych do przesłania przez minutę,
- szybko rosnące koszty wraz ze wzrostem intensywności wykorzystania,
- łatwość zakłócenia transmisji.

2.3.2 Sieci GSM oraz połączenie GPRS

Współcześnie każdy modem GSM ma możliwość transmisji w technice GPRS. Dlatego możliwe jest rozszerzenie metod komunikacji co daje niemal ciągłe przesyłanie danych pomiędzy systemem a jego klientem. Niestety GPRS ma wadę, którą jest konieczność połączenia systemu z zewnętrznym adresem IP. Konieczny jest serwer wirtualny, własny serwer lub zastosowanie (bardzo często darmowego) hostingu i dostępnego na nim protokołu http. Należy zwrócić uwagę, że zastosowanie tego protokołu generuje znaczny i niepotrzebny narzut na ilość przesyłanych danych. Podsumowując rozwiązanie to nie nadaje się dla użytkowników domowych, ale powinno dobrze sprawdzać się do obsługi floty pojazdów.

Zalety:

- duży zasięg,

- niezbyt częste wymagane interwencje użytkownika,
- możliwość przesyłania sporej ilości danych.

Wady:

- relatywnie wyższe koszty obsługi systemu,
- cena modemu GSM,
- konieczność utrzymywania infrastruktury,
- łatwość zakłócenia transmisji.

2.3.3 Wnioski

Liczba praktycznych wariantów komunikacji na zewnątrz jest znacznie bardziej ograniczona. Ciekawym rozwiązaniem wydaje się nadajnik SOS, lecz jest to raczej metoda do zastosowania, gdy wszystkie inne zawiodą. Można byłoby pokusić się również o przeanalizowanie użycia operatorów Internetu satelitarnego lub sieci telefonii satelitarnej takiej jak Iridium. Ze względu na ograniczenia sprzętowe w pracy to rozwiązanie nie jest rozważane.

Spośród przedstawionych rozwiązań najlepsze wydaje się być zrealizowanie komunikacji w oparciu o wiadomości SMS, a komunikację przez GPRS pozostawić, jako ewentualne rozszerzenie.

2.4 Wybór zastosowania systemu lokalizacji

Przedstawione zostały powyżej zastosowania systemów lokalizacyjnych, które pozwolą na opracowanie „autorskiego systemu lokalizacji".

Najbardziej rozpowszechnionym obecnie zastosowaniem systemu nawigacji jest zarządzanie flotami pojazdów. Zarządca firmy transportowej lub operator pogotowia ratunkowego powinni mieć stały dostęp do informacji o lokalizacji podległych im pojazdów. System taki wymaga, aby pojazdy składające się na flotę w krótkich odstępach informowały centralę o swojej pozycji. Ważna jest więc pojemność kanału komunikacyjnego. Nieistotna zaś staje się odporność na zakłócenia szczególnie celowe, gdyż w teorii system taki nie powinien zostać celem ataku.

Drugim zastosowaniem mogłoby być dostarczanie usług podobnych do usług znajdowania zagubionych telefonów. W tym przypadku wystarczyłoby, aby system stosował jedynie najprostsze rozwiązania, a jego cena byłaby dużą zaletą. Warto także opracować przejrzysty interfejs w postaci

aplikacji mobilnej, która po naciśnięciu przycisku, przy użyciu otrzymanej pozycji oraz pozycji telefonu określonej poprzez wbudowany GPS, pokazałaby kierunek, w którym należy podążać. Rozwinięciem tego zastosowania jest wykorzystanie systemu do znajdowania skradzionego pojazdu. W tym przypadku, ze względu na wagę sprawy, należy zadbać o niezawodność oraz odporność na zakłócenia systemu. Należy więc zapewnić, jak najwięcej metod określania pozycji i jak najmniej standardowych kanałów komunikacji.

Ostatnim wartym odnotowania zastosowaniem jest zastosowanie lokalizatorów, jako urządzeń szpiegowskich. Są one obecnie na dużą skalę stosowane przez służby specjalne wszystkich liczących się krajów. System taki posiada własny obwód zasilający umożliwiający działanie niezależnie od instalacji auta co skraca czas jego montażu. Jest to problem bardzo istotny ze względu na możliwość kilkudniowego działania bez ładowania. Systemy tego typu, oprócz służb czy policji pozostają też w kręgu zastosowań prywatnych agencji detektywistycznych.

Oczywiście systemy takie od dawna pozostają też w kręgu zastosowań wojska, jednak ze względu na całkowicie inne standardy, jakie muszą spełniać systemy wojskowe oraz trudność wejścia na ten rynek, został on w tych rozważaniach pominięty.

2.5 Istniejące rozwiązania systemów lokalizacji

Poniżej przedstawiono kilka przykładów systemów lokalizacyjnych używanych do różnych zastosowań.

2.5.1 CalAmp LMU-900

LMU-900 jest urządzeniem, którego przeznaczeniem jest między innymi zarządzanie małymi flotami pojazdów. W celu ustalania pozycji został on wyposażony w 50 kanałowy odbiornik GPS, co jest standardem w low-endowych nawigacjach samochodowych. Specyfikacja na stronie producenta jest co prawda dość ogólnikowa, ale można założyć, że nie obsługuje on alternatywnych do GPS systemów nawigacji. Jeśli chodzi o komunikację ze światem zewnętrznym to odbywa się ona poprzez sieci komórkowe. Sprzęt ten obsługuje wszystkie ważniejsze technologie –

GSM 850/1900 (wersja amerykańska) jak też 900/1800 (wersja europejska), a ponadto popularny w USA CDMA oraz UMTS2100, co pozwala mu na zastosowanie najlepszej dostępnej w okolicy formy komunikacji. Przesyłanie danych odbywa się zarówno za pomocą wiadomości SMS jak i datagramów UDP. Cena urządzenia nie została podana [3].

2.5.2 DT-GPS-TRACKER MINI

Jest to produkt zamknięty w hermetycznej obudowie, umożliwiającej montaż na zewnątrz pojazdu, wyposażony z baterię zdolną zasilać urządzenie przez 6 dni. Możliwe jest także podłączenie go na stałe do akumulatora samochodu. Do komunikacji wykorzystuje sieć GSM (850/800/1800/1900) oraz wiadomości SMS. Umożliwia określenie pozycji na żądanie, w określonych odstępach czasu, po przekroczeniu granic ustalonej strefy lub przekroczeniu prędkości. Pozycja urządzenia jest określana przy pomocy wbudowanego modułu GPS. Jego cechy powodują, że nadaje się głównie do monitoringu małych flot lub jako akcesorium szpiegowskie. Nie wykazuje jednak cech potrzebnych do zastosowania do ustalania pozycji skradzionych pojazdów. Cena wynosi około sześćset złotych [4].

2.5.3 TRACKER Cat 5 Plus

Cat 5 Plus jest urządzeniem, którego głównym zadaniem ma być ustalanie pozycji skradzionych pojazdów. Podstawowym środkiem komunikacji i lokalizacji jest para GSM/GPS. Żeby dodatkowo uodpornić urządzenie na wykorzystanie jammerów posiada także własną technologię komunikacji bezprzewodowej pomiędzy pojazdami wyposażonymi w moduły jego producenta. Urządzenia te tworzą razem sieć w topologii siatki (ang. *mesh network*). Każdy węzeł komunikuje się z innymi węzłami, przesyłając swój identyfikator, po czym identyfikator ten jest przesyłany do centrali w celu porównania go z listą skradzionych pojazdów. Taki sposób rozwiązania problemu zakłóceń transmisji pozwala na budowę odpornego systemu, eliminując konieczność utrzymywania drogiej infrastruktury. Cena rozwiązania jest stosunkowo wysoka i wynosi około osiemset GBP, plus koszty abonamentu [5].

2.5.4 FBI Tracking Device

Jest to urządzenie stosowane przez amerykańskie FBI do kontroli operacyjnej podejrzanych. Urządzenie składa się z kilku części, największa z nich stanowi źródło zasilania całego układu. Kolejna mieści moduły elektroniczne, a ostatnia antenę GPS. Układ jest zasilany przy pomocy ogniw Li-SOCl2 o ogromnej jak na tak mały układ pojemności 13 Ah. Pozycja urządzenia jest wyznaczana przy pomocy standardowego, cywilnego modułu GPS. Moduł komunikacyjny natomiast nadaje na częstotliwości ok. 433 MHz, co jest dość nietypowe, gdyż z pewnością znacznie ogranicza zasięg urządzenia, z drugiej strony nie powodując możliwych do wykrycia przez potencjalnego inwigilowanego transmisji GSM [6].

3. Autorski system wbudowany

3.1 Założenia projektu

Systemy lokalizujące mają trzy główne zastosowania. Są to: zarządzanie flotami pojazdów, zabezpieczenia antykradzieżowe oraz jako akcesoria szpiegowskie. Ze względu na ograniczone zasoby, tworzony system ma stanowić punkt wyjścia dla systemów dostosowanych do każdego z tych zastosowań. Urządzenie takie musi więc zaoferować podstawową funkcjonalność. Proponowany system lokalizacji powinien zastosować system GPS oraz komunikować się przy pomocy wiadomości SMS. Pozwala to na zarządzanie małymi flotami pojazdów lub pojedynczymi pojazdami, gdzie można sobie pozwolić na określanie pozycji przykładowo w odstępach godzinnych.

Jednak, aby sprostać wymaganiom współczesnego rynku, tworzone systemy muszą stale być rozwijane o nowe funkcje, spełniając kolejne oczekiwania klientów. Do osiągnięcia tego celu system musi wykazywać łatwość rozbudowy o kolejne funkcje. Jedną z funkcji, którą warto brać pod uwagę jest możliwość zastosowania binarnych wiadomości SMS, aby ograniczyć ilość znaków koniecznych do przesłania wymaganych danych.

Ceny dostępnych na rynku komercyjnych rozwiązań są mocno wygórowane. Zupełnie nie odpowiada to cenom sprzętu, z którego takie systemy się składają lub mogłyby składać. Wydaje się więc zasadne zwrócenie szczególnej uwagi na ceny stosowanych podzespołów. Maksymalne obniżenie ceny komponentów pozwoli może zwiększyć liczbę potencjalnych klientów takich systemów.

Ze względu na wymaganą elastyczność ważne jest również, żeby opracowany system zachował maksymalną niezależność sprzętową i aby jego kod był przenośny. Należy w miarę możliwości trzymać cały zależny od sprzętu kod w odseparowanych od reszty plikach, które będą jedynie udostępniały reszcie systemu pewien ustandaryzowany interfejs.

Aby umożliwić systemowi dalszy rozwój koniecznością jest pisanie kodu w sposób czytelny wraz z dokumentacją. Dzięki temu, gdyby zaszła potrzeba wprowadzenia do projektu nowego programisty, można będzie zrobić to maksymalnie wydajnie, oszczędzając przy tym sporo czasu.

Ważnym zagadnieniem jest problem zasilania systemu. Wiąże się z tym zarówno rozwiązanie kwestii podłączenia układu do akumulatora pojazdu lub posiadania własnego źródła zasilania. Należy również zwrócić uwagę na odpowiednie optymalizację działania programu, tak aby maksymalnie zmniejszyć pobór prądu, a tym samym wydłużyć czas pracy na baterii. Ze względu na znaczące wykroczenie poza ramy informatyki kwestie zapewnienia odpowiedniego źródła zasilania zostaną w niniejszym opracowaniu pominięte. Dobre rozwiązanie tego problemu wymagałoby zaprojektowanie własnej płytki i przeniesienie nań użytych układów. Aktualny natomiast pozostaje problem odpowiedniego zaprojektowania oprogramowania.

Ostatnim ważnym założeniem przy budowie systemu jest jego pełna niezależność od interwencji użytkownika. System musi sam wykrywać swoje błędy, a w razie wystąpienia poważnego błędu, takiego jak na przykład rzucenie przez procesor wyjątku dostępu do pamięci, dokonać automatycznego restartu. W przypadku systemów wymagających wniesienia opłat miesięcznych lub po wyczerpaniu środków na karcie prepaid, warto by system przypomniał użytkownikowi o konieczności dokonania opłaty oraz ewentualnie udostępniał interfejs do dokonywania takich wpłat.

Ze względu na założenie poczynione na samym początku tego rozdziału, praca ta będzie traktować jedynie o części systemu znajdującej się po stronie obserwowanego pojazdu. Pominięta zostaje więc kwestia zapewnienia wygodnego interfejsu użytkownika. Interfejs taki bardzo zależy od zastosowania do jakiego zostanie przeznaczony taki system, niemożliwe więc jest opracowanie interfejsu uniwersalnego. Przedstawiony program w pracy jest opracowany na potrzeby badań stanowi raczej bazę do dalszej jego specjalizacji.

3.2 Elementy wykonanego systemu

3.2.1 Użyty sprzęt

Przy wyborze sprzętu głównym czynnikiem była jego cena. Ponieważ w systemie użyty został GPS a do komunikacji SMS zbudowany system składa się z następujących elementów: odbiornik GPS, modem GSM i procesor kontrolujący działanie systemu. W wyniku analizy dostępnych na rynku rozwiązań, zdaniem Autorów najbardziej interesujący sprzęt, jeżeli chodzi o modemy GSM, oferuje tajwańska firma SIMCom. Produkty tej

firmy cechuje spora gama różnych modeli oraz łatwa dostępność tanich płytek deweloperskich, które mogą posłużyć do szybkiej budowy prototypu systemu. Jednym z dostępnych w ofercie tej firmy produktów jest układ o nazwie SIM908. Komunikacja z nim odbywa się przy pomocy komend AT, co jest standardem wśród modemów GSM. Część z używanych komend została zdefiniowana w standardzie GSM (GSM 07.05 oraz GSM 07.07), co sprawia, że przeportowanie systemu, tak aby można było inny modem, będzie to wymagało przepisanie jedynie części kodu odpowiedzialnego za komunikację. Ponadto modem umożliwia komunikację poprzez protokół GPRS, a więc ewentualna rozbudowa systemu o przesyłanie danych właśnie z jego użyciem nie będzie wymagała wymiany stosowanego sprzętu. Największą jednak zaletą tego układu jest wbudowany odbiornik GPS. Pozwala to zdecydowanie obniżyć koszty dzięki wyeliminowaniu konieczności stosowania zewnętrznego odbiornika. Użyta płytka deweloperska zawiera wlutowany układ SIM908, co umożliwia zasilanie jej ze źródła o różnym zakresie napięć, co pozwala na stosowanie dowolnego zasilacza lub włączenie jej bezpośrednio w instalację samochodu przez gniazdo zapalniczki. Posiada też dwa połączenia UART, jedno do wydawania komend AT i odbierania odpowiedzi oraz drugie służące do debugowania układu, oba wyprowadzone na standardowe złącza 0.1″ (2,54 mm), ułatwiając prototypowanie. Do płytki dołączona jest zewnętrzna antena GSM oraz aktywna antena GPS. Wadą użytej płytki deweloperskiej jest konieczność jej ręcznego włączenia, co uniemożliwia zastosowanie jej w gotowym produkcie. Ze względu na brak możliwości bezpośredniego oprogramowania układu SIM908, spowodowanego nieudostępnieniem przez producenta potrzebnych do tego narzędzi i dokumentacji, konieczne jest zastosowanie zewnętrznego procesora, który będzie komunikował się z modemem i zarządzał pracą całego systemu. Istnieje fizyczna możliwość bezpośredniego programowania, ponieważ układ działa pod kontrolą procesora z rdzeniem ARM. Ze względu na najkorzystniejszą cenę, wytypowany został układ ESP8266. Jest to jeden z najtańszych dostępnych na rynku procesorów 32-bitowych, dodatkowo wyposażony we wbudowany moduł WiFi. Jego najważniejszą zaletą jest, wynikająca głównie z ceny, popularność i duża dostępność dokumentacji i materiałów opracowanych przez użytkowników. Układ ten posiada małą trwałość użytej pamięci flash Wada ta wymusiła zmianę pamięci na lepszą

co spowodowało wzrost ceny układu. Problemy wspomniane wyżej spowodowały konieczność znalezienia alternatywy dla układu ESP8266. Wybrany został łatwo dostępny w kraju układ STM32-F4 Discovery, będący płytką deweloperską o ogromnych możliwościach, wyposażoną w procesor ARM z rdzeniem Cortex M4, produkcji ST o taktowaniu 168 MHz. Możliwości tego układu zdecydowanie przewyższają potrzeby tworzonego systemu. Dzięki użyciu tej płytki możliwe jest dalsze portowanie programu na inne urządzenia na bazie procesorów STM32, co ułatwi użycie innych procesorów z rdzeniem ARM. Otwiera to więc możliwość zastosowania całej rodziny procesorów. Niewątpliwą wadą tego rozwiązania jest marnowanie potencjału użytego sprzętu, a także związana z tym wysoka cena podzespołu, która nie jest uzasadniona realnymi potrzebami. Całkowity koszt podzespołów zastosowanych w systemie powinien zamknąć się w granicach 200-300 złotych. W przypadku użycia ESP8266 oraz SIM908 bez płytki deweloperskiej będzie to ok. 200 zł za sztukę[1]. A więc zapewnia to spełnienie założeń projektowych dotyczących ceny.

3.2.2 Zastosowane technologie

- Język programowania – **C**

Całość systemu została wykonana w języku C. Wybór ten był podyktowany uniwersalnością tego języka oraz bezproblemowym działaniem na systemach wbudowanych. Kompilatory języka C są dostępne na praktycznie wszystkie wykorzystywane obecnie platformy, przy czym do kompilacji projektu został użyty GNU Compiler Collection (GCC), dostępny zarówno na platformę ARM, jak i wykorzystywaną przez ESP8266 - Tensilica Xtensa. Użycie języka C pozwala stworzyć program, który zostanie skompilowany bezpośrednio do kodu maszynowego użytej platformy co pozwala na uniknięcie narzutu spowodowanego użyciem wirtualnej maszyny czy też kompilacji Just In Time (JIT).

[1] Modem SIM908 – ok. 125 PLN; ESP8266 – ok. 12 PLN; PCB na zamówienie – ok. 60 PLN.

- Użyte biblioteki

System ze względu na konieczność zachowania jego maksymalnej przenośności na nowe platformy ma maksymalnie ograniczone korzystanie z zewnętrznych bibliotek. Jedyną istotną zależnością jest newlib będący lekką implementacją standardowej biblioteki języka C. Nie implementuje on oczywiście wszystkich znajdujących się w libc funkcji, jednak daje dostęp do najważniejszych i najpopularniejszych z nich, pozwalając na w miarę komfortowe programowanie, korzystając ze znanych narzędzi na nawet najprostszych systemach. Dużą zaletą użycia newliba jest jego przenośność. Do działania na całkowicie nieznanej wcześniej platformie wymaga on, oczywiście oprócz właściwego kompilatora, implementacji jedynie kilku funkcji, odpowiedzialnych między innymi za alokację pamięci oraz obsługę plików.

Oprócz newlib, system korzysta z funkcji udostępnianych przez platformy, na które został przeznaczony. W przypadku obu platform, wszystkie funkcje zależne od platformy są umieszczone w oddzielnych plikach, stanowiących warstwę pośredniczącą w komunikacji systemu ze sprzętem. Głównym elementem platformy STM32 jest udostępniana przez producenta biblioteka StdPeriph-Driver, z której korzystają moduły składające się na warstwę pośrednią.

Ostatnim elementem jest opracowana w ramach projektu biblioteka do obsługi wiadomości SMS.

- System budowania

System budowania projektu stosuje o program Make oraz kompilator wywodzący się z pakietu GCC. Plik Makefile głównego projektu pozwala obecnie na kompilację jedynie na platformę STM32F4 i wymaga biblioteki udostępnianej przez jej producenta.

Ponadto całkowicie odrębnym projektem jest biblioteka libsms, która może zostać użyta na dowolnej platformie i jest zależna jedynie od biblioteki standardowej języka C.

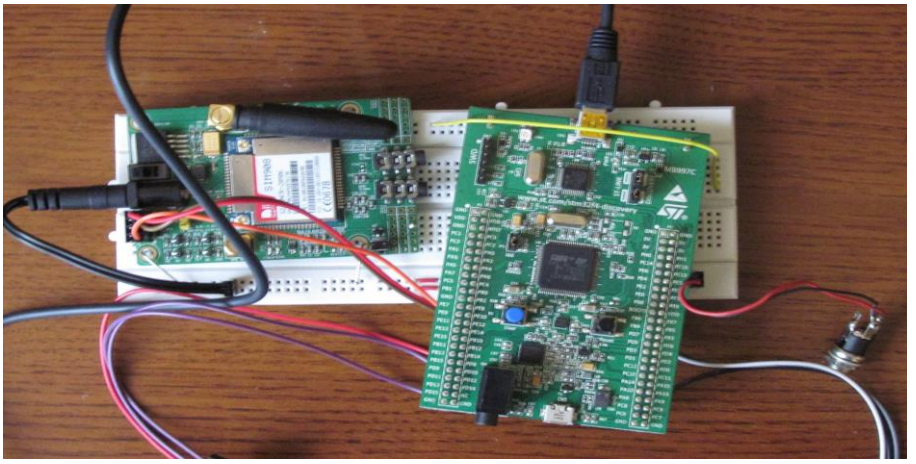
- Narzędzia pomocnicze

Najważniejszym narzędziem pomocniczym opracowanym w ramach tej pracy jest program „sms” będący referencyjną implementacją biblioteki libsms. Pozwala on na wszechstronną manipulację wiadomościami SMS zapisanymi w formie PDU. Dodatkowym jego zadaniem, poza sprawdzaniem poprawności tworzonego kodu, jest ułatwienie zapoznania się z formatem wiadomości SMS oraz wizualizacja prawdziwych wiadomości (otrzymanych przy użyciu modemu) w formie czytelnej dla człowieka.

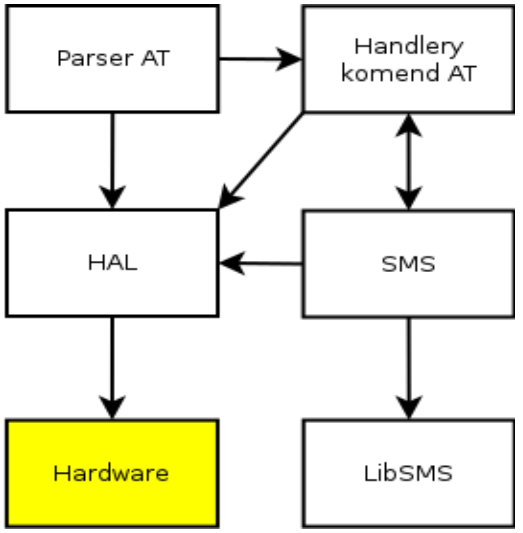
Ponadto, w celu lepszego zaprezentowania formatu PDU wiadomości SMS opracowano autorski program ułatwiający analizę binarnych formatów plików. Ze względu na jego wczesny etap rozwoju nie jest on jednak jeszcze dostępny. Będzie on rozwijany w kolejnych wersjach rozwiązania.

3.3 Opis prototypu

Od strony sprzętowej system składa się z dwóch podzespołów. Pierwszy z nich zawiera procesor, który jest odpowiedzialny za zarządzanie całym systemem, drugi natomiast stanowi jednocześnie modem GSM oraz odbiornik GPS. Komunikacja odbywa się poprzez protokół UART i polega na wysyłaniu komend AT przez układ zarządzający do modemu oraz odbiór wysłanych przez niego odpowiedzi.



Rys. 1. Działający prototyp urządzenia



Rys. 2. Schemat ideowy działania systemu

Oprócz modułów zaznaczonych na rysunku 2 istnieje jeszcze główny moduł, którego zadaniem jest inicjalizacja wszystkich pozostałych modułów systemu oraz wywoływanie w pętli metod udostępnianych przez nie, w szczególności odbiór pojedynczych linii z UART (poprzez HAL) a następnie przekazywanie ich do parsera AT. Ponieważ moduł ten musi „wiedzieć" o istnieniu wszystkich pozostałych modułów, nie został on ujęty na schemacie (rys.2), aby nie zaciemniać pozostałych relacji. Na schemacie pominięto również użytą bibliotekę standardową newlib.

HAL (Hardware Abstraction Layer) ma za zadanie stanowić pomost pomiędzy funkcjonalnościami udostępnianymi przez sprzęt, a pozostałą częścią systemu. Budowa warstwy pośredniej ma na celu maksymalne uniezależnienie systemu od użytego sprzętu. Z tego względu jest wykorzystywany przez wszystkie inne moduły.

Parser AT stanowi implementację ogólnych reguł jakimi rządzą się komendy oraz odpowiedzi AT. Do jego zadań należy między innymi dostarczenie interfejsu do kolejkowania oraz późniejszego wysyłania komend do zdalnego systemu. Odpowiada również za odbiór odpowiedzi, wstępne określenie typu odebranej linii oraz jej wstępne sparsowanie. Jeśli typ jest ustandaryzowany zostanie przekazany do tak przygotowanej (lub nie) linii

do właściwej funkcji odpowiedzialnej za jej dalszą obróbkę (tzw. handlera). W tym celu moduł ten przechowuje wskaźnik do listy handlerów, które powinny być zainicjalizowane przez zewnętrzny moduł (rys. 2 na schemacie jako ’Handlery komend AT’).

Moduł handlerów składa się głównie z funkcji odpowiedzialnych za parsowanie konkretnych odpowiedzi przekazanych przez parser AT. Przechowuje również strukturę zawierającą aktualny stan modemu umożliwiając innym modułom wykorzystanie go do własnych celów. Komendami wartymi odnotowania są komendy odpowiedzialne za odbiór wiadomości SMS. Weryfikują one poprawność otrzymanego PDU po czym przekazują go do znajdującego się w module SMS parsera.

Moduł SMS odpowiedzialny jest za odbiór oraz wysyłkę SMS. Odebrane wiadomości są przekazane do niego z modułu handlerów. Wewnątrz są one parsowane oraz sprawdzane pod kątem poprawności komend. Jeśli wiadomość zostanie rozpoznana jako komenda, wykonywane jest odpowiednie działanie. Moduł ten jest również odpowiedzialny za przechowywanie informacji na temat już obsłużonych wiadomości, oszczędzając czas potrzebny na parsowanie wiadomości nierozpoznanych jako poprawne komendy oraz zapobiegając wielokrotnemu obsłużeniu tej samej komendy. Przy wysyłce odpowiedzi moduł korzysta z danych zgromadzonych w strukturze stanu modemu (moduł handlerów). Parsowanie wiadomości przez ten moduł odbywa się poprzez zastosowanie specjalnie utworzonej na potrzeby systemu zewnętrznej biblioteki libsms.

Biblioteka libsms udostępnia narzędzia potrzebne do obsługi PDU wiadomości SMS, który zawiera dane wiadomości przechowywane w formacie stosowanym do przesyłania wiadomości przez sieć GSM. Zastosowanie PDU umożliwia pełną kontrolę nad wszystkimi parametrami wiadomości. Pozwala to na opracowanie w przyszłości własnego formatu danych na bazie binarnych SMS-ów, zamiast stosowania standardowego siedmiobitowego alfabetu. Umożliwia to optymalizować rozmiar przesyłanych danych, a zatem rozszerzyć ilość danych możliwych do zmieszczenia na jednej stronie wiadomości SMS bez podnoszenia kosztów. Biblioteka libsms została początkowo opracowana jako część programu do odczytu informacji z PDU, możliwego do zastosowania na komputerach klasy PC.

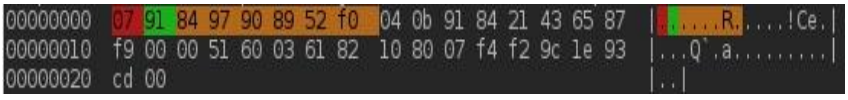
4. Implementacja

4.1 Biblioteka do obsługi PDU SMS

Biblioteką opracowaną ramach projektu jest biblioteka libsms. Ma ona za zadanie parsowanie PDU SMS do postaci umożliwiającej łatwe odczytanie poszczególnych wartości zapisanych wewnątrz, a także dostarczenie zestawu narzędzi, które pomagają w takim odczycie. W tym celu opracowana jest struktura przechowująca kontekst danego PDU. Możliwa jest też budowa pustego kontekstu, która może być wypełniona własnymi danymi i odwrócenie całego procesu, czego efektem będzie nowe PDU, gotowe do wysłania przez modem. W ramach budowy biblioteki opracowana została także pomocnicza aplikacja, której celem jest testowanie na bieżąco tworzonych funkcji, a także wizualizacja danych zawartych w PDU. Pomaga to w lepszym zrozumieniu zawartych w nim danych. Aplikacja ta nie jest jednak wykorzystywana w ramach systemu lokalizacji, a więc jej opis pominięto.

4.1.1 Format PDU SMS

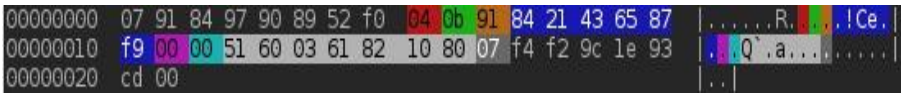
Każda wiadomość poprzedzona jest nagłówkiem opisującym adres centrum SMS (SMSC). Jest to pośrednik pomiędzy abonentami sieci komórkowej, trzymający wiadomości w czasie, gdy abonent będący adresatem wiadomości jest niedostępny. Po stronie interfejsu AT modemu istnieje możliwość podania pustego nagłówka SMSC, co spowoduje, że wybrany zostanie domyślny adres, przechowywany na karcie SIM nadawcy.



Rys. 3. Przykładowe PDU z zaznaczonym nagłówkiem SMSC

Nagłówek składa się z trzech wartości. Pierwsza to długość pozostałej części nagłówka, a więc w przypadku, gdy nagłówek nie zawiera adresu centrum SMS, pole to przyjmuje wartość 0. Drugie pole opisuje typ podanego adresu. Wartość 0x91 oznacza, że podany numer zawiera prefiks kraju oraz używa linii cyfrowej (ISDN). Ostatnie z pól przechowuje adres,

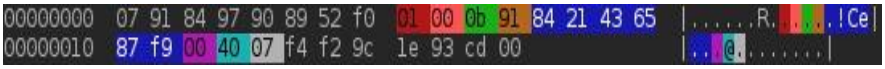
przy czym jest on zapisany jako upakowany kod BCD, gdzie połówki bajtu (z ang. *nibble*) są zapisane w odwrotnej kolejności (littleendian). Wartość równa 15 (0xf) oznacza, że dana połówka jest niewykorzystana. Warto zauważyć, że nagłówek SMSC nie jest liczony do długości wiadomości przez komendy AT zgodne ze standardem GSM 07.05.



Rys. 4. Przykładowa wiadomość SMS-DELIVER

Właściwa część wiadomości zaczyna się od wartości opisującej typ wiadomości. Od typu zależy format dalszej części wiadomości. W tym przypadku jest to wiadomość typu SMS- DELIVER, a więc wiadomość jaką otrzymuje odbiorca. Kolejne trzy wartości opisują adres nadawcy (Originator Address - OA). Jego format jest niemal identyczny jak adres SMSC, z różnicą w zapisie długości adresu. Tutaj liczona jest ilość cyfr adresu przed upakowaniem. Następna wartość to PID (Protocol IDentifier). Jako, że w przypadku wiadomości po stronie użytkowników, pole to ma zawsze wartość równą zeru. Następne pole to DCS (Data Coding Scheme), które opisuje między innymi alfabet użyty do zakodowania właściwej części wiadomości. Kolejne pole to znacznik czasowy wiadomości (ang. *timestamp*). Nie jest on jednak istotny z punktu widzenia tego systemu. Ostatnie oznaczone pole zawiera długość właściwej części wiadomości. Zaraz za nim znajduje się właściwa wiadomość, zakodowana zgodnie z alfabetem wskazanym przez DCS.

Standard GSM definiuje trzy różne sposoby kodowania wiadomości. Pierwszym i zarazem najczęściej używanym jest alfabet opracowany specjalnie na potrzeby wiadomości SMS (opisany w GSM 03.38[8]). Jest to alfabet ze słowami siedmiobitowymi, upakowanymi na ośmiobitowych znakach. Dzięki temu zakodowana nim wiadomość może być nieco dłuższa niż gdyby zastosowano łatwiejsze w obsłudze słowa ośmiobitowe (160 znaków siedmibitowych do 140 ośmiobitowych). Drugi dostępny alfabet jest opisany przez standard UCS-2, będący kompatybilnym z UTF-16 (jednak UTF-16 nie jest komaptybilne z UCS-2). Trzecia opcja to wykorzystanie słów ośmiobitowych, co pozwala na przesłanie danych o dowolnym formacie. Są to tak zwane binarne SMS-y.



Rys. 5. Przykładowa wiadomość SMS-SUBMIT

W przypadku wiadomości typu SMS-SUBMIT, czyli przygotowanej do wysłania występuje kilka drobnych różnic względem wiadomości SMS-DELIVER. Tutaj zaraz za typem wiadomości znajduje się pole MR (Message Reference). Pole to można jednak pominąć, gdyż jest ono automatycznie ustawiane w momencie wysłania wiadomości. Format adresu jest identyczny jak w poprzednim przypadku, jednak oznacza on oczywiście numer adresata wiadomości (Destination Address). Nie występuje tu znacznik czasowy wiadomości, zamiast niego znajduje się opcjonalne pole opisujące czas po jakim wiadomość straci ważność w przypadku niedostarczenia [7].

4.1.2 Serializacja PDU

Zaprezentowany wyżej format PDU nie może zostać opisany przez strukturę w języku C. Pierwszym ograniczeniem jest brak możliwości zdefiniowania wewnątrz struktury tablicy o nieokreślonym rozmiarze lub rozmiarze określonym przez inny element tej struktury. Wyjątkiem jest umieszczenie dynamicznej tablicy na końcu danej struktury, która pozwala na wystąpienie kilku tablic o zmiennym rozmiarze wewnątrz całego PDU. Powoduje to konieczność napisania funkcji swym zadaniem zbliżonej do serializacji, znanej z języków wyższego poziomu. Opracowanie struktury zawierającej dane łatwiej dostępne dla procesora niż czyste PDU daje okazję do zapisania pól takich jak adresy w formie nieupakowanej, co ułatwia ich późniejsze zastosowanie.

Niemożliwe jest opracowanie jednej struktury opisującej PDU, niezależnie od jego typu. Biorąc jednak pod uwagę fakt, że większość pól powtarza się niezależnie od typu PDU oraz znając sposób w jaki wysokopoziomowa deklaracja struktur w języku C przekłada się na układ danych w pamięci, możliwe jest uniezależnienie większej części struktury od typu konkretnego PDU.

```
struct sms_ctx
{ uint8_t
    sca_length;
    /*(...)*/
};

struct sms_submit_ctx
{ struct sms_ctx
    common_ctx; uint8_t mr;
    uint8_t da_length;
    uint8_t da_number_type;
    char *da_number;
};

struct sms_deliver_ctx
{       struct       sms_ctx
    common_ctx;      uint8_t
    oa_length;       uint8_t
    oa_number_type;     char
    *oa_number;
    struct scts scts;
};
```

Rys. 6. Listing: Definicja kontekstu

Dzięki takiej deklaracji użycie struktury, jako składowej innej struktury, powoduje de facto wstawienie jej zawartości w wybrane miejsce. Możliwe jest używanie pamięci zaalokowanej na przykład sms_submit_ctx (a więc mającej jej rozmiar) jako, struktury sms_ctx, umożliwia przynajmniej części kodu bycie niezależnym od typu PDU. Dzięki temu poniższy kod wypisze właściwą wartość pola sca_length, chociaż w momencie wywołania nie musi on znać prawdziwego typu zaalokowanej struktury.

```
 struct sms_submit_ctx *submit_ctx =
param; struct sms_ctx *ctx = (struct
sms_ctx*) submit_ctx; printf("%d\n",
          ctx->sca_length);
```

Rys. 7. Listing : Inicjalizacja struktur kontekstu

Daje to pewną namiastkę dziedziczenia znanego z języków wyższego poziomu. Niestety, nawet stosując powyższą metodę, kod języka C będzie musiał bardzo często rozgałęziać się w zależności od typu używanego PDU. Szczególnie uciążliwe stają się procesy serializacji i deserializacji

PDU. W praktyce wymusza to stosowanie konstrukcji switch lub if/else w każdym fragmencie różniącym się pomiędzy typami PDU.

```
(*ctx)->pid = *((uint8_t*)cursor++);
(*ctx)->dcs = *((uint8_t*)cursor++);
if(pdu_mti == SMS_DELIVER)
{ memcpy(&(((struct sms_deliver_ctx*)(*ctx))-
    >scts), cursor, sizeof(struct scts));
    cursor += sizeof(struct scts);
}
else if((pdu_mti == SMS_SUBMIT) && (((*ctx)->pdu_type &
    PDU_TYPE_VPF_MASK) == PDU_TYPE_VPF_REL))
{
    cursor += 1;
    // TODO: get
    vpf
}
else if((pdu_mti == SMS_SUBMIT) && (((*ctx)->pdu_type &
    PDU_TYPE_VPF_MASK) == PDU_TYPE_VPF_ABS))
{
    cursor += 7;
    // TODO: get
    vpf
}
(*ctx)->udl = *((uint8 t*)cursor++);
```

Rys. 8. Listing : Fragment kodu deserializującego PDU

Jak można zauważyć w tym konkretnym zadaniu paradygmat strukturalny przegrywa ze znacznie elastyczniejszym i nowocześniejszym paradygmatem obiektowym, z którego można korzystać w języku C++. Pomimo uproszczenia kodu, powoduje on tylko nieznaczny spadek wydajności względem dobrze napisanego kodu w języku C.

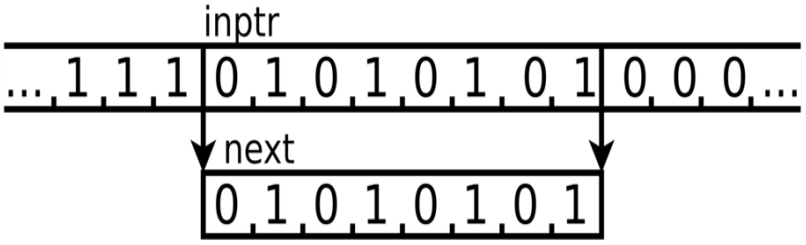
4.1.3 Konwersja GSM 03.38 <=> UTF-8 [8]

Jak wspomniano wyżej standard GSM 03.38 bazuje na słowach siedmiobitowych. Ponadto zdefiniowany jest alfabet częściowo niekompatybilny z używanym powszechnie standardem ASCII. Generuje to dwa problemy. Pierwszym jest wypakowanie pojedynczego słowa do zmiennej

ośmiobitowej. Drugim natomiast konwersja tak odczytanego znaku do formatu ułatwiającego późniejsze operowanie wiadomościami. Warto również zauważyć, że ponieważ tworzona biblioteka musi odczytywać wiadomości i pozwalać na tworzenie nowych, należy zaimplementować zarówno konwersję z oraz i na format GSM.

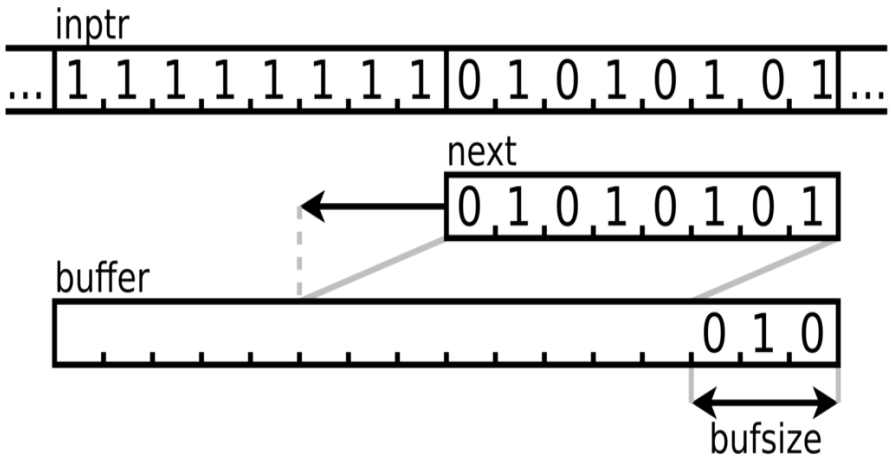
Aby więc rozwiązać pierwszy z problemów, należy z wspomnianego bufora pobierać po siedem bitów co pozwala na jego dalsze dekodowanie. Implementowany algorytm, dla którego minimalną jednostką procesora jest bajt (osiem bitów), musi pobierać naraz po osiem bitów z bufora wejściowego i przekazywać dalej po siedem bitów. Wskaźnik iterujący po buforze wejściowym został oznaczony jako *inptr*. Wyjściem tej części dekodera jest zmienna *character*. Jako pośrednik stosowany jest jest bufor o zmiennej długości z zakresu 0-16 bitów (w praktyce długość nie przekracza 14 bitów). Długość bufora przechowywana jest w zmiennej *bufsize*. W najprostszym przypadku na starcie algorytmu bufor jest pusty (a więc *bufsize* jest równy 0). Nie jest to jednak regułą, gdyż zależy od obecności w wiadomości nagłówków (UDH - User Data Header). Aby lepiej zaprezentować działanie algorytmu, przyjęty został bufor, w którym znajdują się już trzy bity - 0, 1, 0.

Pierwszym etapem każdej iteracji jest sprawdzenie ilości bitów znajdujących się aktualnie w buforze. Jeżeli znajduje się w nim co najmniej 7 bitów, oznacza to, że możliwe jest ich pobranie i przekazanie danych. W przedstawionym przypadku (rys.9) są w buforze jedynie 3 bity, a więc konieczne jest pobranie danych z wejścia.



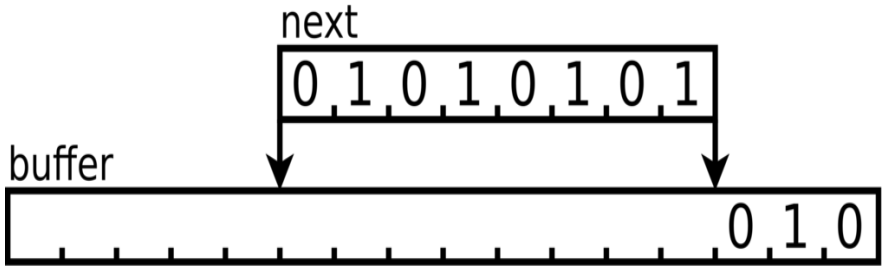
Rys. 9. Konwersja GSM 03.38 <=> UTF-8 krok 1

Osiem bitów znajdujących się pod wskaźnikiem *inptr* jest zapisywanych w zmiennej wejściowej – *next*, po czym wskaźnik ten jest przesuwany do przodu, aby wskazywał na kolejną porcję danych (rys. 10).



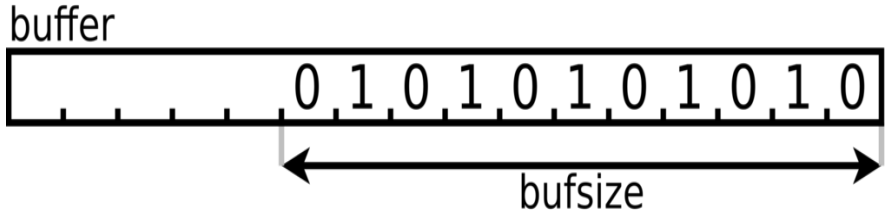
Rys. 10. Konwersja GSM 03.38 <=> UTF-8 krok 2

Następnie dane znajdujące się w zmiennej wejściowej należy przesunąć w lewo o obecną długość bufora.



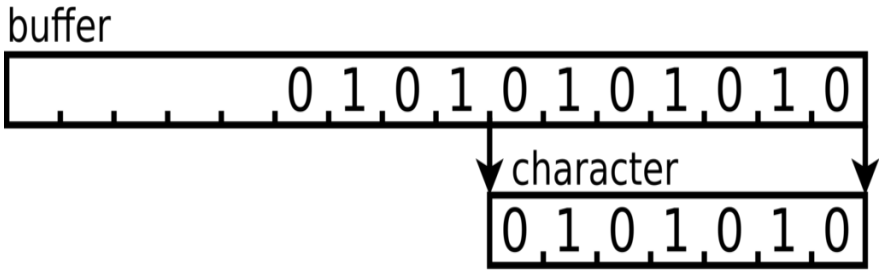
Rys. 11. Konwersja GSM 03.38 <=> UTF-8 krok 3

Następnie zawartość tej zmiennej zostaje wstawiona do bufora.



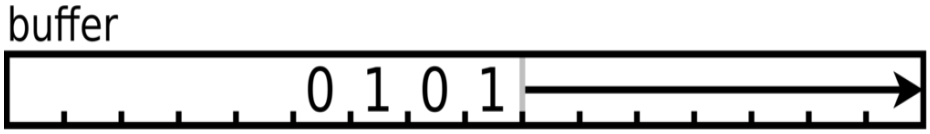
Rys. 12. Konwersja GSM 03.38 <=> UTF-8 krok 4

A długość bufora zostaje powiększona o 8.



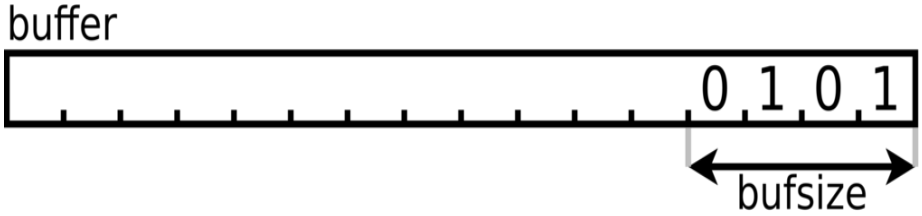
Rys. 13. Konwersja GSM 03.38 <=> UTF-8 krok 5

Ponieważ ilość danych, jakie są obecnie w buforze, jest większa niż 7, to możliwe jest pobranie z niego następnego symbolu. Jest on już gotowy do dalszych działań konwersji z alfabetu GSM 03.38.



Rys. 14. Konwersja GSM 03.38 <=> UTF-8 krok 6

Po pobraniu danych z bufora, należy przesunąć pozostałe znajdujące się w nim dane o 7 miejsc w prawo oraz zmniejszyć rozmiar bufora o 7.



Rys. 15. Konwersja GSM 03.38 <=> UTF-8 krok 7

Następnie całość wykonywana jest od początku. Algorytm działa do czasu wypakowania z bufora wejściowego założonej z góry liczby symboli.

W przypadku kodowania danych do GSM 03.38, algorytm działa niemal identycznie, z tą różnicą, że wstawiał będzie po 7 bitów do bufora, natomiast pobierał po 8 bitów.

Rozwiązanie drugiego z problemów z pozoru wydaje się dość proste. Wystarczy zmapować jedne symbole na drugie i gotowe. Niestety po przeanalizowaniu alfabetu zdefiniowanego w opisanym standardzie, okazuje się, że niektóre z symboli nie znajdują się w tablicy ASCII. Wymusza to więc użycie standardu UTF-8, który zawiera wszystkie wymagane symbole. Niektóre z symboli niestety nie mieszczą się na pierwszej stronie tablicy UTF-8, a więc wymagają użycia co najmniej dwóch bajtów. W przypadku kodowania danych UTF-8 do formatu GSM trzeba więc dokonać mapowania tablica => bajt.

Założeniem pracy jest maksymalna przenośność oraz fakt, że, program jest budowany na systemy o ograniczonych zasobach sprzętowych, konieczne jest znalezienie kompromisu pomiędzy czasem procesora zajmowanym przez enkoder, rozmiarem programu oraz zużyciem RAM. Istotna jest też prostota implementacji rozwiązania. W tym celu przedstawionych zostanie kilka możliwych rozwiązań problemu.

- Użycie konstrukcji switch/case

Najprostszym rozwiązaniem wydaje się być użycie konstrukcji switch/case. Implementacja tej metody jest banalnie prosta. Niestety ma ona podstawową wadę polegającą na powtórzeniu niemal identycznego kodu, w tym przypadku, 128 razy. Wiąże się z tym również całkowity brak zoptymalizowania, ponieważ taki kod będzie wymagał wykonania średnio kilkudziesięciu skoków dla każdego kodowanego znaku. Powoduje to, że jest ona zasadna wyłącznie w przypadku sprawdzania kilku warunków.

- Użycie pętli for

Drugim prostym rozwiązaniem problemu jest użycie pętli for do iterowania się po całej mapie dostępnych znaków w celu znalezienia indeksu konkretnego znaku. Spowoduje to jednak dramatyczny spadek wydajności funkcji kodującej, ze względu na konieczność wykonania średnio 64 przejść pętli dla każdego kodowanego znaku.

- Mapowanie przy użyciu tablicy

```
static const uint8_t reverse_charmap[] =
{
    [0x40] = 0x00, //"@"
    [0xc2a3] = 0x01,//"£"
    [0x24] = 0x02, //"$"
    [0xc2a5] = 0x03,//"¥"
    [0xc3a0] = 0x7f //"à"
}
```

Rys. 16. Listing : Fragment kodu deserializującego PDU

Standard C pozwala na zdefiniowanie jedynie niektórych elementów tablicy. Zwiększa to czytelność definicji. Niestety jak łatwo przekonać się poprzez obserwację rozmiaru aplikacji przed i po zdefiniowaniu tablicy widać, że pomimo użycia jedynie niektórych elementów, alokowana jest pamięć potrzebna na zmieszczenie wszystkich elementów. Powoduje to, że dla przypadku, w którym ostatni element miałby indeks równy 1000, zostanie zaalokowane w sekcji rodata 1000 elementów. Niewątpliwą zaletą takiego podejścia jest minimalizacja ilości instrukcji wymaganych do konwersji jednego znaku. Rozwiązanie to sprawdziłoby się więc na systemach z dużą ilością dostępnej pamięci oraz kodujących w ten sposób spore ilości danych.

- Użycie słownika

Wartym do przeanalizowania rozwiązaniem jest użycie konstrukcji słownika znanego z języków wysokiego poziomu. Dobrze zaimplementowany słownik powinien pozwolić na niemal natychmiastowe odnalezienie poszukiwanego elementu, oszczędzając przy tym (w porównaniu do poprzedniej propozycji) sporo pamięci operacyjnej oraz nie zwiększając nadmiernie rozmiaru programu. Niestety rozwiązanie to ma pewne wady. Przede wszystkim zaimplementowanie wydajnego słownika nie jest rzeczą prostą, więc najlepszym wyjściem byłoby skorzystanie z gotowego rozwiązania. Niestety gotowe rozwiązanie dorzuciłoby niepotrzebnych zależności do całego projektu.

- Rozwiązanie bazujące na liczeniu sumy CRC znaku

Rozwiązanie to charakteryzuje się prostotą implementacji. Niestety, w przypadku systemów wbudowanych zmarnowanie ok. 64 KiB pamięci

operacyjnej jest wręcz niedopuszczalne. Szczególnie w przypadku układu ESP8266 posiadającego mocno ograniczone zasoby pamięci operacyjnej i pamięci flash przechowującej kod i dane programu.

Z tego względu ważne jest ograniczenie rozmiaru tablicy potrzebnej do zmapowania znaków Unicode do GSM 03.38. Dobrym rozwiązaniem jest zastosowanie funkcji skrótu. Prostym rodzajem skrótu jest algorytm CRC (Cyclic Redundancy Check). Ponieważ należy utrzymać użycie pamięci na jak najniższym poziomie, zastosowany został wariant CRC8, gdzie wynik działania mieści się na jednym bajcie. Aby utrzymać czas obliczania sumy kontrolnej na możliwie najniższym poziomie, zastosowany został algorytm oparty o predefiniowaną tablicę. Wymaga to zdefiniowania tablicy o rozmiarze 256 bajtów. Drugą tablicą jakiej wymaga powyższa metoda jest tablica mapująca wartość sumy CRC na konkretne słowo alfabetu GSM. Teoretycznie tablica taka wymagałaby tylko kolejnych 256 bajtów. Niestety, ze względu na fakt, że suma CRC ma jedynie 8 bitów długości w praktyce występuje sporo konfliktów. To powoduje, że końcowa tablica jest faktycznie macierzą 256×3. Mimo tej drobnej niedoskonałości razem daje to ok. 1 KiB wymaganej pamięci.

Tabela 2. Porównanie algorytmów kodowania do GSM 03.38

Algorytm	Czas [s]	Użycie RAM [KiB]	Rozmiar aplik. [B]
Switch/case	1,880485679 (1,47)	92 (1,05)	14400 (1,17)
For	15,009642611 (11,74)	**88 (1,00)**	**12320 (1,00)**
Tablica	**1,278683285 (1,00)**	144 (1,64)	65392 (5,31)
Słownik	1,770141813 (1,38)	92 (1,05)	13856 (1,12)
CRC	1,994351710 (1,56)	92 (1,05)	13976 (1,13)

Teoretycznie więc ostatnia metoda może konkurować jedynie ze słownikiem. Sprawdzono jak poszczególne algorytmy sprawdzają się

w praktyce. Poniższa tabela zawiera zestawienie szybkości działania, zużycia pamięci operacyjnej oraz rozmiaru programów wykorzystujących zaproponowane algorytmy. W nawiasach podano wartość względną (*wynik = aktualny/najlepszy*, a więc im bliżej jedności tym lepiej). Do przeprowadzenia badania zastosowano zbiór losowych znaków o rozmiarze 64 MiB.

Jak można zauważyć, niemal wszystkie wyniki okazały się bardzo podobne. Oczywiście, zgodnie z przewidywaniami, czas wykonania pętli for okazał się być prawie dwunastokrotnie wyższy od reszty. Drugim przewidzianym odstępstwem jest ilość pamięci zużytej przez wariant z tablicą. Z tych powodów obie metody nie nadają się do zastosowania w praktyce. Pozytywnie zaskakuje czas wykonania algorytmu switch/case. Niestety ze względów opisanych wyżej on również musi zostać odrzucony. Zaskoczeniem był też dobry wynik algorytmu z użyciem słownika. Pomimo użycia najprostszej możliwej implementacji, okazało się, że jest on w stanie skutecznie konkurować z zaproponowanym algorytmem korzystającym z CRC. Jest to spowodowane tym, że oba algorytmy, pomimo, że na pierwszy rzut oka wydają się różne, tak naprawdę są niemal bliźniaczo do siebie podobne.

Jedyną różnicą jest sposób liczenia sumy kontrolnej użytej do budowy tablicy mieszającej. W przypadku słownika, jako hash, zostało użyte modulo z wartości klucza. W drugim przypadku była to wartość funkcji CRC. W drugim przypadku konieczne jest więc wykonanie większej ilości operacji. Słownik ma też przewagę nieco mniejszego zużycia pamięci. Z drugiej strony zastosowana implementacja wymaga inicjalizacji polegającej na wypełnieniu tablicy danymi. Jeżeli jednak taka inicjalizacja będzie się odbywać jedynie raz w trakcie działania całego programu, to nie stanowi to problemu.

4.2 Warstwa komunikacji ze sprzętem

Ważnym aspektem działania całego systemu jest zależność systemu od użytego sprzętu. Złe zaplanowanie warstwy komunikacyjnej powoduje znaczne utrudnienie przeniesienia systemu na inny sprzęt. W przypadku opisanego systemu warstwa ta (zwana HAL) składa się z następujących modułów: hw, persistance, syscalls, timer oraz uart.

Pierwszy z nich – **hw** odpowiada za inicjalizację całego sprzętu, udostępniając funkcję, która powinna być wykonana jako pierwsza w funkcji main(). Drugim zadaniem tego modułu jest przechwycenie funkcji obsługujących wyjątki i doprowadzenie do restartu całego systemu w przypadku, gdy taki wyjątek wystąpi.

Moduł **persistance** ma na celu implementację funkcji zapisu i odczytu danych z nośnika trwałego, w tym przypadku pamięci flash. Ma to na celu przechowanie danych o obsłużonych wiadomościach SMS, pomiędzy uruchomieniami urządzenia i pozwala zachowywać wszystkie odebrane wiadomości w pamięci karty SIM. Użytkownik tego modułu musi jednak uważać, aby zapis do pamięci flash odbywał się tylko wtedy, gdy naprawdę jest potrzebny, gdyż każdy zapis powoduje na procesorze STM32 blokadę procesora do czasu jego zakończenia i stwarza ryzyko przepełnienia rejestru UART, a zatem zgubienia znacznej ilości otrzymywanych z niego danych.

Moduł **syscalls** zawiera implementację wywołań systemowych, wymaganych przez bibliotekę newlib. Pozwala to na użycie tej biblioteki niezależnie od wykorzystywanego sprzętu. Warto wspomnieć, że obecnie wewnątrz tego modułu zaimplementowane są tylko trzy funkcje: read, write oraz sbrk. Pozostałe nie są wykorzystywane przez system, a więc ich pełna implementacja okazała się zbędna. Każda z funkcji zwraca właściwy dla siebie błąd. Implementacja read oraz write pozwoliła natomiast na zastosowanie funkcji printf i pokrewnych do komunikacji przez UART lub wysyłania informacji przydatnych do debugowania systemu na dedykowany w tym celu port. Obecnie działa to w ten sposób, że deskryptory stdin oraz stdout są używane do komunikacji przez UART, natomiast stderr odpowiada za wysyłanie logów na port UART dedykowany specjalnie temu celowi.

Moduł **timer** odpowiada za udostępnienie cyklicznych wywołań funkcji. Udostępnia on funkcję pozwalającą innym składnikom systemu dodanie własnych funkcji do listy. Istnieje również możliwość usunięcia dowolnego timera z zewnątrz. Ponadto każdy timer poprzez zwracaną przez siebie wartość może nakazać temu modułowi usunięcie siebie z listy timerów. Główny timer jest wywoływany co sekundę, natomiast rejestrowane przez niego zadania mogą określić czas co jaki mają być wywoływane z dokładnością do jednej sekundy.

Uart jest najbardziej rozbudowanym wśród modułów warstwy HAL. Odpowiada bezpośrednio za odbiór i wysyłkę danych przy pomocy protokołu UART. Ponieważ procesor STM32 nie posiada wbudowanej kolejki do przechowywania otrzymywanych danych, konieczne jest udostępnienie własnej implementacji takiej kolejki. Implementacja ta znajduje się w oddzielnym module, a więc pozostaje poza zakresem tego rozdziału i może być zastosowana do innych celów w pozostałych częściach systemu. Warto jedynie odnotować, że użyto tu bufora cyklicznego obsługującego przepełnienia. W takim przypadku najstarsze przechowywane dane zostają nadpisane przez nowo otrzymane.

Ponieważ parser AT, który jest jedynym konsumentem otrzymanych danych operuje na pełnych liniach, moduł ten udostępnia funkcję pobierającą ze wspomnianego bufora jedną pełną linię w przypadku, gdy jest ona dostępna. Dzięki temu sam bufor może pozostać ukryty dla parsera.

4.2.1 Przenośność kodu

Dzięki takiemu rozplanowaniu warstwy komunikacyjnej, do poprawnego przeportowania systemu na nową platformę sprzętową wymagane jest jedynie zaimplementowanie funkcji udostępnianych przez opisane wyżej moduły. Docelowy system powinien jednak posiadać co najmniej dwa porty UART. Nie jest to co prawda konieczne, ale zdecydowanie przyspieszy wykrycie problemów, jakie mogą wystąpić w trakcie takiej operacji. Wszystkie testowane do tej pory platformy, a były to ESP8266 oraz STM32-F4 Discovery w przypadku całego systemu oraz dodatkowo AMD64 dla biblioteki libsms były systemami co najmniej 32-bitowymi. Wszystkie z nich korzystały także z konwencji little-endian przy zapisie zmiennych. To drugie jest szczególnie istotne w przypadku biblioteki libsms, która operuje na otrzymanych z zewnątrz danych binarnych. W przypadku zastosowania systemu big-endian należy więc zwrócić szczególną uwagę na obsługę PDU SMS.

Ważnym, aczkolwiek łatwym do przeoczenia aspektem, jest konieczność wykonania resetu urządzenia w przypadku każdej sytuacji wyjątkowej. W przypadku układu ESP8266 takie działanie jest zaimplementowane domyślnie, jednak drugi z wykorzystanych układów, przynajmniej w przypadku użytych bibliotek udostępnionych przez producenta w swoim pakiecie firmware, w przypadku każdego nieprzewidzianego przez programistę

błędu wykonuje pętlę nieskończoną. Konieczne jest więc przechwycenie jak największej liczby błędów i wykonanie zamiast tego restartu.

Opisane rozwiązania mające ułatwić przenoszenie kodu zostały wypróbowane w praktyce. Początkowo cały system był budowany dla układu ESP8266. Niestety problemy z samym układem spowodowane najprawdopodobniej zastosowaniem w nim słabej jakości pamięci flash wymusiły przejście na bardziej stabilny sprzęt. Cały proces nie okazał się skomplikowany. Najwięcej problemów sprawiało zaprogramowanie warstwy sprzętowej, nie zaś jej integracja z resztą systemu. Można więc uznać, że HAL spełnił przewidziane dla niego zadanie. Obecnie układ ESP8266 nie jest obsługiwany przez system, czego powodem jest brak rozwiązania opisanego tu problemu z jego działaniem, jednak przywrócenie tej obsługi wymaga przede wszystkim niewielkiej modyfikacji głównego modułu programu. Nie powinno więc ono stanowić problemu, gdyby zaszła taka potrzeba.

4.3 Parser komend AT

4.3.1 Format komend i odpowiedzi

Każda z komend wydawanych modemowi musi rozpoczynać się znakami „AT” lub „at” oraz kończyć się znakiem powrotu karetki (CR). W praktyce występują dwa formaty komend. Pierwszy z nich określa komendę jako ciąg w formacie „AT<x><n>” lub „AT&<x><n>”, gdzie „<x>” oznacza komendę, a „<n>” jej argumenty. Ten rodzaj komend jest jednak wykorzystywany przede wszystkim przez komendy opisane standardem V.25, który w przypadku modemów GSM służy głównie do wykonywania połączeń. Ponieważ żadna z tych komend nie jest wykorzystywana w tworzonym systemie, nie będą one przedmiotem dokładniejszego opisu.

Drugi format – nazywany rozszerzonym (extended syntax) jest powszechnie używany przez komendy związane ze standardem GSM. Jest on również wykorzystywany przez rozszerzenia katalogu komend tworzone przez producentów modemów. Także używany tutaj modem marki SIMCom posiada własne rozszerzenie oparte o ten format. Format ten oferuje kilka typów komend:

- Test – „AT+<x>=?” – zwraca zakresy wartości dla komendy Write
- Read – „AT+<x>?” – odczytuje wartość parametru/parametrów
- Write – ”AT+<x>=<n>” – ustawia wartość parametru/parametrów
- Execute – „AT+<x>” – wykonuje komendę

Wysyłając odpowiedź, modem zawsze poprzedza ją i kończy parą znaków CRLF. Każda z wykonywanych komend powinna zakończyć się wysłaniem statusu wykonania w postaci ciągu „OK”, „ERROR”, „CME ERROR” lub „CMS ERROR”. Dopóki żadna z tych odpowiedzi nie zostanie wysłana, co oznacza, że poprzednia komenda jeszcze się nie zakończyła, system, określany w terminologii GSM jako TE lub DTE – (Data) Terminal Equipment, nie może wysyłać następnych komend.

Spora część komend, szczególnie tych opisanych w standardach GSM 07.05 oraz GSM 07.07 odpowiada ciągami zbliżonymi składniowo. Przykładami takich komend są „AT+CREG?” (GSM 07.07) odpowiadające ciągiem „+CREG: 2,5,"1234","ABCD"” czy „AT+CMGF?” (GSM 07.05) odpowiadające przykładowo „+CMGF: 0”. Warto tu również wspomnieć, że część komend posiada tak zwane Unsolicited Result Codes (URC) czyli odpowiedzi niezainicjowane przez pytanie użytkownika. Czasem, jak w przypadku komendy „AT+CREG” format takiej odpowiedzi różni się od formatu odpowiedzi zamówionej. Część komend jednak odpowiada ciągami, których format jest całkowicie unikalny. Dobrym przykładem jest komenda „AT+CMGL=<stat>,<mode>” służąca do odczytania wiadomości zgromadzonych na karcie SIM. Odpowiada ona najpierw linią opisującą aktualną wiadomość w formacie podobnym do większości innych komend ze standardu GSM, a w następnej linii wypisująca PDU wiadomości zapisane szesnastkowo. Format ten powtarza się kolejno dla każdej znalezionej wiadomości. Na koniec wysyłana jest linia „OK” mówiąca o powodzeniu wykonania [9].

4.3.2 Ogólna część parsera AT

Ponieważ niektóre cechy odpowiedzi wysyłane przez modem powtarzają się istnieje możliwość ukrycia części specyfikacji przed funkcjami, które są bezpośrednio odpowiedzialne za obsługę konkretnych komend.

Upraszcza to kod znajdujący się wewnątrz tych funkcji (zwanych handlerami) i zapobiega powtórzeniom fragmentów. Rozróżnia się dwa rodzaje handlerów. Pierwszy ma za zadanie obsłużyć odpowiedzi (oraz kody URC) o formacie „+<x>: <p1>,<p2>,...". Drugi odpowiada za obsługę każdej niepustej linii, która nie będzie spełniać reguł poprzedniego kodu.

Główną częścią parsera AT jest funkcja *at_parseline()*, której zadaniem jest przetworzenie kolejnej odebranej z modemu linii. Sam parser nie ma możliwości żądać kolejnej linii, musi więc przechowywać stan w wewnętrznej strukturze. Struktura ta zawiera również wskaźnik do listy handlerów obu typów. Początkowo są one jednak puste i do konkretnej implementacji należy ich wypełnienie.

Cały proces od początku, a więc od momentu wydania przez system polecenia wysłania komendy przebiega następująco. Pierwszym krokiem jest przekazanie komendy do kolejki komend typu FIFO. Komenda przekazana najwcześniej będzie wysłana jako pierwsza. Wysłanie komendy może odbyć się jedynie, gdy wewnętrzny stan parsera nie przechowuje informacji o parsowaniu żadnych komend. Oznacza to, że ostatnia wykonywana komenda zakończyła się sukcesem i można bezpiecznie przejść do wysyłki następnej. W tym momencie zostaje zainicjowana komenda wysyłająca. Zdejmuje ona pierwszą komendę z kolejki, po czym wypisuje ją na standardowe wyjście co jest równoznaczne z transmisją do modemu. Dokonuje także zapisu komendy w strukturze przechowującej wewnętrzny stan parsera. Dzięki temu zostaje zablokowana możliwość wysłania kolejnej komendy zanim obecna nie zostanie wykonana.

Taka metoda pilnowania ma jednak jedną wadę. Istnieje zagrożenie przepełnienia bufora UART. Może to spowodować, że system nigdy się nie dowie o sukcesie bądź porażce wykonania komendy. Należy więc wprowadzić zewnętrzny kontroler (watchdog) pilnujący, aby żadna z komend nie wykonywała się zbyt długo. Watchdog monitoruje więc stan parsera, a gdy po ustalonym czasie nie zauważy zmian w polu aktualnie wykonywanej komendy, dokona resetu stanu parsera. Powoduje to ryzyko wykonania komendy jeszcze zanim zakończone zostanie wykonanie poprzedniej. Dzięki odpowiednio dużej wartości timeoutu można to ryzyko zminimalizować.

Po wysłaniu komendy do modemu, chwilę później powinna zacząć spływać odpowiedź. Po chwili pierwsza linia odpowiedzi powinna zostać

przekazana do parsera. Pierwszym zadaniem parsera jest zakwalifikowanie otrzymanej linii. Na początek sprawdza on czy linia pasuje do formatu opisanego w pierwszym akapicie. Jeśli tak, dokonuje ekstrakcji komendy na jaką jest to odpowiedź. Następnie dokonuje sparsowania parametrów odpowiedzi. Jest to zrealizowane poprzez rozbicie wartości oddzielonych przecinkami w tablicę ciągów w stylu C (char**) przy czym koniec jest oznaczany poprzez adres wypełniony samymi binarnymi jedynkami – ”(void*)-1”. . Warto tu zwrócić uwagę na pominięcie sprawdzenia aktualnie wykonywanej komendy. Dzieje się tak, ponieważ ten typ handlerów odpowiadać może zarówno na odpowiedzi zamówione (Solicited) jak i również niezamówione (URC). Aktualnie zapamiętana w stanie komenda nie jest więc ważna. Jeśli zostanie odnaleziony właściwy handler zostaje on wywołany jako parametr z listą argumentów.

Jeśli natomiast otrzymana linia nie spełnia tych reguł (a więc nie jest ani SRC, ani też URC), zostaje sprawdzona pod kątem bycia jedną ze znanych wartości, wśród których są między innymi „OK”, „ERROR” , które kończą wykonanie aktualnej komendy oraz kilka innych wartości jakie modem może przesłać, szczególnie w trakcie procesu inicjalizacji. Pozwala to stwierdzić, kiedy modem będzie gotowy do pracy. Na tym etapie eliminowane są również puste linie.

Jeśli linia nie zostanie „skonsumowana” na żadnym z poprzednich etapów, ostatnim etapem jest wysłanie jej do drugiego typu handlera (nazwanego plain handlerem). Dopiero tu sprawdzana jest wartość aktualnie wykonywanej komendy i jeśli jest zarejestrowany dla niej plain handler, zostaje mu przekazana cała linia. Handler ten może zadecydować czy obsłużenie danej linii się udało poprzez zwracaną wartość.

Jeśli linia zostanie rozpoznana jako SRC lub URC, a nie ma zarejestrowanego dla niej handlera parser zwraca błąd mówiący o braku handlera. Jeśli linia przejdzie przez wszystkie sprawdzenia i nie zostanie skonsumowana, zostanie zwrócony błąd, który mówi o jej nieznanym formacie.

4.3.3 Handlery odpowiedzi

Moduł ten zawiera implementacje znanych odpowiedzi. Jest on również odpowiedzialny za przechowywanie stanu modemu, przy czym jako stan należy rozumieć takie parametry jak stan odblokowania karty SIM lub stan połączenia z siecią GSM. Struktura ta przechowuje również dane

nieokreślające stanu samego modemu, lecz stan parsowania konkretnej komendy. Tak jest między innymi w przypadku komendy listującej wiadomości SMS z karty SIM. To tutaj znajduje się także lista obsługiwanych komend w postaci mapowania komenda-handler. Dotyczy to zarówno handlerów SRC/URC jak i plain handlerów. Korzystanie tego modułu wymaga aby go najpierw zainicjalizować. W trakcie inicjalizacji obie listy handlerów zostają zarejestrowane wewnątrz stanu parsera.

Najważniejszym z zaimplementowanych obecnie handlerów jest ten odpowiedzialny za odczyt PDU otrzymanych wiadomości SMS. Zanim jednak zostanie on wywołany, wywoływany jest handler obsługujący SRC odpowiedniej komendy. Za odczyt wiadomości są odpowiedzialne dwie komendy. Komenda CMGR (odczytuje wybraną wiadomość) oraz CMGL (listuje wszystkie zgromadzone wiadomości). W obu przypadkach SRC zawiera między innymi identyfikator wiadomości w pamięci SIM oraz jej długość. Obie wartości są potrzebne do odczytania wiadomości przez wspomniany handler. Muszą więc być zapamiętane w stanie modemu do czasu obsłużenia wiadomości. W momencie otrzymania przez modem nowej wiadomości, do TE wysyłane jest URC o kodzie CMTI (nieposiadające odpowiednika w komendach) przekazując przez niego id wiadomości, które mogą zostać zastosowane do odczytania pojedynczej wiadomości.

Taka konstrukcja obsługi komend pozwala na łatwe zaimplementowanie kolejnych komend, gdyby zaszła konieczność obsługi kolejnych z nich. Na przykład komendy odblokowującej kartę SIM poprzez podanie kodu PIN. Istnieje dzięki temu też możliwość prostego przeniesienia systemu na modem innej firmy. Większość użytych komend jest ustandaryzowana przez standard GSM, pozostałą część (głównie komendy obsługujące GPS) można łatwo dostosować do nowego interfejsu.

4.4 Interfejs komunikacyjny

Komunikacja pomiędzy systemem lokalizującym a jego użytkownikiem odbywa się za pomocą wiadomości SMS.

Aby odróżnić komendę oraz odpowiedź od zwykłych wiadomości, otrzymywanych od operatora i nie poddawać ich zbędnej obróbce, każda poprawna komenda powinna rozpoczynać się od ciągu „cmd>”, natomiast każda poprawna odpowiedź – „ans>”.

W tej chwili istnieje jedna dostępna komenda – „cmd>getloc”. Powoduje ona ustalenie aktualnej pozycji przy pomocy odbiornika GPS, a następnie odesłana w odpowiedzi do użytkownika. Poniżej przedstawiono format takiej odpowiedzi:

```
ans>,1234,ABCD⏎
   ,1929.123456,5111,654321,123.000000,0.123456,9
0.654321
```

gdzie:

- `1234` – oznacza LAC (Location Area Code), opisuje grupę komórek sieci GSM
- `ABCD` – CID (Cell ID), opsiuje identyfikator BTS (Base Transceiver Station), do którego podłączony jest modem
- `1929.123456` – długość geograficzna ustalona przez GPS
- `5111.654321` – szerokość geograficzna
- `123.000000` – wysokość nad poziomem morza
- `0.123456` – prędkość
- `90.654321` – kierunek poruszania

Długość i szerokość geograficzna została opisana w formacie: `DDMM.mmmmmm`, gdzie

- `DD` – stopnie
- `MM` – pełne minuty
- `.mmmmmm` – milionowe części minuty

Dodatnia wartość obu liczb oznacza odpowiednio szerokość północną (N) oraz długość wschodnią (E). Poprawny zapis obu wartości powinien więc w tym przypadku wyglądać tak:
$51°11.654321N,19°29.123456E$.

Do celów ustalenia pozycji włączony jest moduł GPS, który w czasie oczekiwania na zdarzenia pozostaje wyłączony. Odpowiedź zostanie odesłana dopiero po ustaleniu dokładnej pozycji (złapaniu fixa GPS). W przypadku, gdyby ustalenie pozycji okazało się z jakichś względów niemożliwe, wiadomość zwrotna nie nadejdzie aż do chwili powodzenia operacji lokalizacji, a moduł GPS pozostanie cały czas włączony.

Aktualnie funkcja obsługująca komendy SMS nie przewiduje zastosowania binarnych wiadomości SMS. Docelowo należałoby napisać dodatkową funkcję, której zadaniem byłaby obsługa wyłącznie tego typu wiadomości.

Niestety rozbudowa tej części systemu jest nieco trudniejsza niż dodawanie obsługi nowych komend. Jest to spowodowane podziałem obsługi komendy SMS na kilka etapów. W momencie otrzymania wiadomości SMS jest ona parsowana i przekazywana modułowi SMS. Moduł ten, na podstawie otrzymanej komendy decyduje jakie operacje należy wykonać. W przypadku żądania wysłania lokalizacji konieczne jest zastosowanie GPS do określenie aktualnej pozycji. W tym celu należy włączyć moduł GPS i poczekać na określenie pozycji. W momencie określenia pozycji przez GPS rozpoczyna się wywoływanie funkcji odpowiedzialnej za stworzenie PDU wiadomości SMS zawierające przewidziane dane. Następnie tworzona jest komenda wysyłająca PDU, zawierająca stworzone PDU zapisane heksadecymalnie. Trzecim etapem jest wysłanie komendy do modemu i odczekanie na jej pomyślne wykonanie. Ze względu na specyfikę komendy wysyłającej wiadomość SMS, konieczne jest odczekanie na pojawienie się znaku zachęty po wysłaniu żądania wysłania wiadomości i dopiero następnie transmisja samego PDU.

Spośród powyższych kroków, dwa pierwsze są zależne od obsługiwanej komendy, a wywołanie drugiego może nastąpić jedynie, gdy spełnione zostaną warunki specyficzne dla danej komendy. Z tego względu możliwości łatwej rozbudowy modułu są ograniczone.

4.5 Mechanizmy uodparniające na awarie

4.5.1 Automatyczny reset systemu po błędzie

Aby osiągnąć pełną niezawodność systemu, tworząc go należy przewidzieć każdy błąd jaki może wystąpić w trakcie jego działania. Ponieważ

jednak zawsze istnieje ryzyko, że twórca systemu coś przeoczy, należy prawidłowo obsłużyć także błędy, których powód może być trudny do ustalenia. W przypadku systemów autonomicznych, a do takich można zaliczyć opisany tutaj system lokalizacyjny, najlepszym wyjściem z takich nieprzewidzianych sytuacji wyjątkowych jest restart całego systemu. Oczywiście nie rozwiąże to wszystkich problemów związanych z zaistniałą sytuacją wyjątkową, jednak daje szansę na dalsze działanie systemu, bez wymagania od użytkownika interwencji. Obsłużenie wyjątku poprzez restart może również w niektórych przypadkach doprowadzić do nieskończonej pętli restartów. Mimo wszystko jest on lepszym wyjściem niż standardowa, w przypadku procesorów z serii STM32, blokada procesora przy użyciu nieskończonej pętli. W przypadku opisywanego systemu konieczność wymuszenia resetu przez programistę nie występowała na platformie ESP8266.

4.5.2 Tryb odzyskiwania

Modem SIM908 podczas inicjalizacji wysyła do TE kilka informacji. Jedną z nich jest informacja o gotowości do działania. Znajdują się tam także informacje o stanie blokady karty SIM oraz rejestracji w sieci GSM. System z modułami zarządzającym i modemem powinny wystartować razem, stosując te dane, aby niepotrzebnie nie odpytywać o nie modem. Jednak ze względu na możliwość restartu urządzenia w przypadku wystąpienia błędu system nie będzie w stanie odebrać tych danych. Konieczne jest więc zaimplementowania pewnego rodzaju trybu odzyskiwania, który zostałby uruchomiony tylko jeśli po ustalonym czasie nie zostanie wykryty żaden z komunikatów startowych.

Algorytm startowy zaczyna się w momencie startu systemu. Stan modemu jest wówczas nieznany. Dlatego stan wewnątrz systemu zostaje opisany jako niezdefiniowany. W przypadku prawidłowego uruchomienia po chwili powinny zacząć spływać pierwsze dane z modemu. Gdyby jednak tak się nie stało, startuje timer, który ma zadanie monitorować stan systemu. Jedną z pierwszych linii, jakie powinny zostać otrzymane przez system jest linia „RDY", oznaczająca początek sekwencji startowej.

W momencie jej otrzymania stan systemu zostaje ustawiony na *MOD_STARTING*. Następnie powinno pojawić się kilka wiadomości URC opisujących stan modemu. Ostatnią linią sekwencji powinna być linia „Call

Ready", która oznacza, że modem jest gotowy do pracy. W tym momencie stan systemu zostaje oznaczony jako *MOD_GATHERING*. Powoduje to, że system sprawdza, które dane na temat stanu modemu zostały otrzymane i w razie potrzeby wysyła komendy, które mają zadanie zebrania reszty danych. Kiedy zebrane zostaną wszystkie potrzebne dane, system przechodzi do stanu normalnej pracy.

Gdyby jednak zdarzyło się, że linia „RDY" nie została otrzymana po ustalonym czasie, system przechodzi do stanu odzyskiwania. Odpytuje w nim modem o wszystkie potrzebne mu dane. Po chwili modem powinien odpowiedzieć na wszystkie żądania, a wymagane pola w stanie modemu powinny zostać wypełnione. Po tej operacji system przechodzi do trybu normalnej pracy.

4.5.3 Watchdog wykonania komend

Istnieje ryzyko, że odpowiedź na wydaną przez system komendę nie zostanie przez niego poprawnie odebrana może to spowodować sytuację podobną do zakleszczenia, gdzie system będzie oczekiwał na dane, które już nigdy nie nadejdą. Aby temu zapobiec opracowany został watchdog (punkt 4.3.2), Monitorujący czas wykonania komend.

4.5.4 Monitor skrzynki odbiorczej SMS

Istotnym zadaniem systemu jest także kontrola skrzynki odbiorczej SMS. Ryzyko przepełnienia bufora UART można wyeliminować poprzez zastosowanie kontroli przepływu, jednak ze względu na ograniczenia płytki deweloperskiej modemu SIM908, takie rozwiązanie okazuje się niemożliwe do wykonania. Należy więc co jakiś czas upewnić się, że wszystkie przechowywane na karcie SIM wiadomości zostały już obsłużone. Do tego celu został zastosowany moduł timerów. Pozwala on na wygodne ustalenie interwału wywołania timera. Czas ten jest w tym wypadku dość istotny, ponieważ z jednej strony za częste wywołania komendy listującej wiadomości doprowadziłyby do niepotrzebnego zużycia energii. Z drugiej zaś strony zbyt rzadkie wywołanie tej procedury skutkowałoby znacznymi opóźnieniami w odsyłaniu odpowiedzi w przypadku nieotrzymania informacji o nadchodzącej wiadomości. Dobrym kompromisem pomiędzy obomа stronami wydaje się ustalenie interwału na 5 minut. Nie powinno to

skutkować nadmiernym opóźnieniem w odsyłaniu odpowiedzi, gdyż w skrajnych przypadkach określenie pozycji przy użyciu GPS także może zająć nawet kilka minut, szczególnie w przypadku poruszania się po mieście, gdy moduł GPS nie miał jeszcze okazji złapania pierwszego fixa, a więc wymagany jest tak zwany cold fix.

4.5.5 Potwierdzanie wysłania wiadomości

Otrzymane wiadomości nie są przez system kasowane z pamięci karty SIM. Zamiast tego utrzymywana jest lista wiadomości obsłużonych przez system. W przypadku wiadomości niebędących prawidłowymi komendami możliwe jest natychmiastowe oznaczenie ich jako obsłużone. Spowoduje to, że przy następnym listowaniu wszystkich wiadomości, te nieistotne będą odrzucane jeszcze zanim ich treść zostanie odczytana, jedynie na podstawie ich identyfikatora w pamięci SIM. Pozwala to na oszczędzenie czasu procesora.

W przypadku prawidłowych komend, oznaczenie ich jako obsłużonych już na tym etapie obsługi niesie niebezpieczeństwo, że podczas awarii systemu wiadomość nie zostanie w ogóle obsłużona. Żeby temu zapobiec należy wstrzymać się z oznaczeniem wiadomości. Dopiero w momencie otrzymania od modemu informacji o wysłaniu wiadomości do sieci GSM, możliwe jest oznaczenie wiadomości jako obsłużonej.

Niestety takie rozwiązanie wciąż nie zapewnia pełnej odporności na błędy. Rozwiązuje to jedynie ewentualne błędy systemu lub modemu GSM. Do ogniw, które mogą zawieść działanie systemu należą sieć GSM (konkretnie centrum SMS) oraz adresat wiadomości. Należy mieć na uwadze możliwość zażądania od adresata potwierdzenia odbioru wiadomości. Umożliwia to wyeliminowanie potencjalnych problemów z siecią GSM lub sprzętem adresata. Istnieje jednak ewentualność, że adresat nie wysyła potwierdzeń otrzymanych wiadomości. W przedstawionym w pracy systemie obsługa raportów potwierdzających odbiór wiadomości nie została opracowana. Jest to jednak jeden z najpilniejszych kierunków rozwoju opisanego systemu i będzie brany pod uwagę przy pracach nad kolejnymi wersjami systemu.

5. Autorski system lokalizacyjny – wnioski

Opisane rozwiązanie pozwala na skuteczną rywalizację z dostępnymi na rynku komercyjnymi systemami. Zostało to osiągnięte dzięki zastosowaniu tanich komponentów. Oczywiście opracowany system nie jest gotowy do komercyjnego zastosowania ponieważ obejmuje on głównie zadania programistyczne z pominięciem elementów elektronicznych.

Na szczególną uwagę zasługuje fakt, że pełnym sukcesem zakończyło się osiągnięcie kryterium przenośności, co zostało sprawdzone w trakcie budowy systemu. Niestety nieco gorzej wypadła elastyczność, która działa dobrze w przypadku rozbudowywania obsługi modemu GSM i możliwości zamiany stosowanego modemu. Obsługa wiadomości SMS stoi na dość wysokim poziomie i w razie potrzeby pozwala na zastosowanie w pełni możliwości tego protokołu. W jej ramach powstała samodzielna biblioteka oraz referencyjna aplikacja co pozwala na zastosowanie ich w ramach zupełnie innego zadania.

Opracowany system jest autonomiczny. Została zaimplementowana obsługa najczęściej występujących błędów co pozwala na bezawaryjne działanie. Interwencja użytkownika jest wciąż wymagana w przypadku korzystania ze wszystkich środków z karty SIM, co może zostać poprawione w ramach rozbudowy obsługi modemu. W stopniu niesatysfakcjonującym został natomiast rozwiązany problem oszczędzania energii. Ponieważ wykonany został tylko prototyp. System z racji swojej podstawowej funkcjonalności ma bardzo szerokie możliwości rozwoju. Różne cele wymagają nieco odmiennych cech od takiego systemu. Przykładowo obsługa floty pojazdów wymagałaby przede wszystkim skupienia się na dodatkowych kanałach komunikacyjnych takich jak pakietowe połączenie internetowe oraz opracowanie kompleksowego oprogramowania zarządzającego. Użycie do celów szpiegowskich wymagałoby uwzględnić poważny problem zarządzania energią oraz miniaturyzacji. W każdym z tych zastosowań wspólnym mianownikiem jest oszczędność energii. Drugim kierunkiem jest rozbudowa protokołu komunikacyjnego wykorzystującego wiadomości SMS oraz dodanie połączeń pakietowych. Takie rozwiązanie można by było zastosować do obsługi konta abonenta sieci komórkowych.

6. Systemy wbudowane stosowane w bezpieczeństwie

Systemy wbudowane mogą zostać zastosowane do ochrony poświadczeń pozwalających na uwierzytelnienie się do systemów informatycznych. Co czyni je interesującym dla współczesnych użytkowników. Tradycyjny model uwierzytelnienia polega na podaniu przez użytkownika jego nazwy użytkownika - loginu oraz hasła, stanowiącego według współczesnego rozumienia procesu uwierzytelniania „coś co wiesz" (ang. *something you know* [19]). W początkowym okresie istnienia sieci Internet, informacja ta w momencie wymiany między użytkownikiem a serwerem, na którym ów użytkownik się uwierzytelnia nie była w żaden sposób chroniona. Najprawdopodobniej było to spowodowane hermetycznością sieci u zarania jej istnienia. Z czasem pojawiało się coraz więcej zagrożeń, lecz ciągle hasła były przesyłane bez żadnej ochrony. Dużą zmianę w stosowanych zabezpieczeniach można było zauważyć dopiero w ostatnich latach. Złożyło się na to wiele czynników, ale jednym z kluczowych było upowszechnienie się sieci bezprzewodowych. W nich podsłuch przesyłanych danych jest bardzo łatwy.

Szyfrowanie przesyłanych danych jest już standardem, a jeżeli przesyłane dane są danymi wrażliwymi, to zgodnie z nowymi unijnymi regulacjami ich odpowiednie zabezpieczenie jest obowiązkowe pod groźbą kar finansowych [20]. Wraz z wyeliminowaniem problemu przechwycenia danych logowania na etapie ich przesyłania oraz znaczącym wzrostem świadomości dostawców usług, ciężar walki o bezpieczeństwo przesunął się w stronę urządzenia, z którego korzysta użytkownik.

Ponieważ zapewnienie odpowiedniego poziomu bezpieczeństwa urządzeń końcowych okazuje się znacznie trudniejsze niż infrastruktury serwerowej popularność zyskało podejście do weryfikacji tożsamości zwane uwierzytelnianiem dwuskładnikowym (ang. *two-factor authentication*). Podejście to dodaje do wspomnianego „czegoś co wiesz", kolejny składnik – „coś co masz" (ang. *something you have*). Najczęstszą implementacją tego składnika są kody przesyłane przez SMS-y. Ponieważ w niektórych przypadkach nawet to staje się niewystarczające istnieje jeszcze

trzeci czynnik – „coś czym jesteś” (ang. *something you are*). Tak zwane rozwiązania biometryczne. [15]

Niestety, pomimo ciągłego wzrostu popularności wieloskładnikowego uwierzytelniania, wciąż istnieją serwisy, które z niego nie korzystają. Rozwiązaniem tego problemu może być system wbudowany opisany poniżej. Rozwiązany zostanie problem niewystarczającego bezpieczeństwa uwierzytelniania opartego o hasło, tam, gdzie nie jest dostępne uwierzytelnianie dwuskładnikowe lub użytkownik nie ma możliwości bądź też nie chce takiego uwierzytelniania aktywować. Głównym celem opisanego poniżej systemu jest przechowywanie haseł użytkownika, tak aby mógł on użyć ich do potwierdzenia swojej tożsamości. Jednak nie jest w stanie ich w żaden sposób poznać i uniemożliwić ich wyciek nawet w przypadku infekcji urządzenia, z którego ktoś korzysta złośliwym oprogramowaniem. Pozwoli to na połączenie wspomnianego „czegoś co znasz” z „czymś co masz”, tak że faktycznie użytkownik będzie w posiadaniu jedynie drugiego ze składników. Do implementacji bezpiecznego kontenera na hasła użyta zostanie karta inteligentna (ang. *smart card*) zgodna z technologią JavaCard (rys. 17).

Rys. 17. Karta używana w projekcie (źródło: własne)

Karty takie są od lat standardową platformą do implementacji kryptografii w oparciu o klucze niedostępne dla użytkownika. Dzięki temu powinny one zapewniać bezpieczeństwo przechowywanych danych nawet w sytuacji fizycznego posiadania urządzenia przez atakującego lub możliwości uruchomienia przez niego własnego kodu.

Ponieważ obecnie najpopularniejszym protokołem zapewniającym bezpieczeństwo warstwy transportowej jest Transport Layer Security

(TLS), do implementacji bezpiecznego systemu wymiany została użyta najnowsza karta.

Efektem opracowanego systemu jest wyeliminowanie niebezpieczeństwa wycieku hasła po stronie użytkownika. Czas jaki i zajmuje proces logowania wzrasta od czterech do kilkunastu sekund, bez pogorszenia jakości generowania liczb pseudolosowych, kluczowych dla pierwszego etapu zestawiania bezpiecznej sesji.

Jednym z celów postawionym przez Autorów jest zbadanie możliwości zaimplementowania standardu TLS od strony klienta na platformie JavaCard. Aby to osiągnąć w pracy została przeprowadzona analiza interfejsów udostępnianych przez platformę JavaCard oraz wybranych urządzeń implementujących ten standard pod kątem udostępnianych algorytmów kryptograficznych. Następnie przeanalizowano protokół TLS pod kątem wymagań koniecznych do spełnienia, aby nawiązać i utrzymać bezpieczne połączenie i opracowano implementację wybranej części standardu. Ze względu na pośrednictwo komputera pomiędzy kartą a serwerem, konieczne jest również zaimplementowanie aplikacji po stronie komputera PC. Powstałe rozwiązanie została przebadane pod kątem odporności na znane problemy bezpieczeństwa ze szczególnym uwzględnieniem tych, wymienionych w dokumentacji standardu TLS oraz generatora liczb pseudolosowych stanowiącego podstawę bezpieczeństwa wszystkich stosowanych algorytmów. Przeprowadzona została ocena przydatności opracowanego systemu dla użytkowników końcowych. Przeanalizowano w szczególności niedogodności i skomplikowania, jakie mogą wyniknąć z powodu wzrostu czasu potrzebnego na cały proces.

7. Problem zarządzania hasłami

7.1 Zarys problemu

7.1.1 Identyfikacja zagrożeń

Hasła stanowią najstarszą i jednocześnie najskuteczniejszą metodę weryfikacji tożsamości użytkownika. Obecnie niemal wszystkie serwisy przechowują hasła w formie tzw. hashu lub skrótu, najczęściej będącego efektem działania jednej z funkcji z rodziny SHA, z dodatkowym ciągiem, zwanym solą, uniemożliwiającym masowe łamanie haseł przez wygenerowanie skrótów wszystkich możliwych kombinacji i porównanie ze skrótami, które atakujący chce złamać. Dzięki wzrostowi świadomości programistów i administratorów skutkującej stosowaniem opisanych metod, złamanie hasła po stronie serwera stało się dość trudnym zadaniem, osiągalnym jedynie dla najbardziej zdeterminowanych atakujących, dysponujących potężną mocą obliczeniową.

Powody wymienione wyżej, oraz fakt, że siłowe łamanie hasła przez ciągłe odpytywanie serwera metodą brute force jest stosunkowo łatwe do wyeliminowania przez administratorów, sprawiają, że jedynym etapem, na którym możliwe jest poznanie hasła przez atakującego, przy założeniu, że użytkownik przestrzegał wszystkich dobrych praktyk odnośnie haseł, jest chwila od rozpoczęcia wpisywania hasła przez użytkownika, aż do otrzymania hasła przez serwer oczekujący na uwierzytelnienie.

Słabymi punktami są zatem użytkownik, jego urządzenie oraz protokół zabezpieczający przesyłane dane. Warto jednak zaznaczyć, że pod warunkiem dobrze skonfigurowanego serwera, ten ostatni został właściwie wyeliminowany.

7.1.2 Zalecenia odnośnie haseł

Najsłabszym ogniwem całego łańcucha zostaje więc użytkownik. Z tego powodu powstały różne zalecenia i standardy, mające na celu opisanie dobrych praktyk, a niekiedy wręcz warunków jakie musi spełniać hasło, aby zostać uznane za dobre czy dopuszczalne. Pierwszymi przejawami wymuszonego podnoszenia jakości haseł było wprowadzanie przez kolejne serwisy minimalne długości haseł. Potem doszła do tego konieczność stosowania cyfr i znaków specjalnych.

Dobrym przykładem polityki bezpieczeństwa są rekomendacje tworzone przez organizację OWASP[20] dotyczące złożoności hasła obejmujące następujące warunki:

- hasło musi posiadać co najmniej po jednym znaku z minimum trzech poniższych klas:
  - wielkie litery,
  - małe litery,
  - cyfry,
  - znaki specjalne (w tym spacja),
- posiadać minimum 10 znaków,
- posiadać maksimum 128 znaków,
- żaden znak nie może powtarzać się więcej niż dwa razy.

Ministerstwo Spraw Wewnętrznych i Administracji [23] opracowało polską normę o ochronie danych osobowych gdzie zostały także określone wymagania dotyczące polityki haseł. Określają one, że dla pewnych zbiorów danych konieczna jest zmiana hasła przynajmniej raz na 30 dni. Samo hasło musi składać się z minimum 8 znaków, w tym małych i wielkich liter oraz cyfr lub znaków specjalnych.

Warto też zauważyć, że ze względu na powszechność wycieków baz danych z hasłami została powszechnie przyjęta zasada stosowania różnych haseł do różnych serwisów. Ma to zmniejszyć skalę zniszczeń i zapobiec włamaniom na konto bankowe lub skrzynkę pocztową jeśli dojdzie do wycieku danych.

7.1.3 Konsekwencje polityki haseł

Wprowadzenie polityki haseł, szczególnie jeżeli jest ona bardzo restrykcyjna (wymusza ciągłe zmiany haseł) ma swoje konsekwencje. Ze względu na naturę ludzką, bardzo prawdopodobne jest, że użytkownicy będą próbować na różne sposoby ominąć niedogodności wprowadzone przez politykę haseł. Przykładem bardzo złego hasła, spełniającego wszystkie wymagania OWASPu jest ’Passw0rd12’. Gdy dodatkowo włączona zostanie konieczność zmiany hasła np. co 30 dni, z dużym prawdopodobieństwem spora część użytkowników doda na końcu aktualny numer miesiąca. Inną metodą jest zapisanie hasła na kartce i przyklejenie jej do monitora.

Wszystkie te rozwiązania nie spełniają swojego zadania dla korzystającego z haseł użytkownika.

7.2 Istniejące menadżery haseł

Ze względu na niemożliwość zapamiętania ogromnej ilości haseł jakie każdy aktywny użytkownik sieci musiałby mieć powstały programy zwane menadżerami haseł. Pozwalają one przechowywać wszystkie hasła w jednej bazie, chronionej za pomocą jednego możliwie jak najsilniejszego hasła. Pozwala to na łatwą zmianę hasła, które akurat wyciekło do sieci, albo prowadzenie polityki comiesięcznej zmiany najważniejszych haseł bez konieczności ciągłego uczenia się długiego zestawu nowych haseł.

Podejścia do tworzenia tego typu programów są jednak zupełnie różne i nie da się wymienić jednego, które miałoby same zalety.

7.2.1 KeePass

KeePass jest najstarszym z prezentowanych rozwiązań. Jego cechą charakterystyczną jest przechowywanie lokalnie bazy danych na komputerze lub urządzeniu mobilnym użytkownika. KeePass dostępny jest jako aplikacja desktopowa na platformę .NET, kompatybilną z Mono. Aplikaja działaja zarówno na systemie Windows jak i Linux. Dostępne są również nieoficjalne implementacje, które działają na systemach Android i IOS.

Baza danych jest zaszyfrowana algorytmem AES w wersji 256-bitowej, a klucz szyfrujący generowany jest za pomocą algorytmu AES-KDF, bazującego na iteracjach algorytmu AES, na podstawie wpisanego przez użytkownika hasła głównego.

Silne szyfrowanie połączone z faktem przechowywania bazy na urządzeniu użytkownika powoduje, że KeePass należy do najbezpieczniejszych rozwiązań. Dużą zaletą jest też fakt, że całość oprogramowania jest oparta o wolne licencje, a kod jest otwartym źródłem, co umożliwia niezależnym badaczom analizę implementacji algorytmów, zwiększając bezpieczeństwo całego systemu.

Logowanie za pomocą KeePassa polega na wybraniu z listy właściwego wpisu i opcji 'Auto-Type', powodującej minimalizację aplikacji i zaemulowanie wciśnięć klawiszy. Takie rozwiązanie wraz z brakiem integracji z najpopularniejszymi przeglądarkami powoduje, że dla mniej technicznych użytkowników może się on okazać zbyt niewygodny w obsłudze.

Fakt lokalnego przechowywania bazy danych powoduje jednak, że użytkownik jest odpowiedzialny za kopie zapasowe bazy danych. Powoduje to ryzyko utraty danych w razie zaniedbania obowiązku przechowywania danych.. Co więcej taki model prowadzi do utrudnienia synchronizacji bazy pomiędzy różnymi urządzeniami i w razie synchronizowania poprzez kopiowanie naraża użytkownika na jeszcze większe prawdopodobieństwo utraty części danych przez zastąpienie nowszej wersji starszą.

Jest jeszcze jeden aspekt negatywny, rzadko zauważany przez użytkowników wybierających menadżer haseł jest to ryzyko wycieku całej bazy danych w przypadku zainfekowania ich komputera lub smartfona złośliwym oprogramowaniem. W szczególności dotyczy to oprogramowania z funkcją keyloggera, które pozwoli w takiej sytuacji na poznanie przez atakującego głównego hasła i kradzież wszystkich przechowywanych wewnątrz haseł. Także sama procedura wpisywania hasła do serwisu stanowi łatwy cel dla keyloggerów, ze względu na emulację klawiatury [15].

7.2.2 LastPass

LastPass oferuje zupełnie inne podejście niż KeePass. Jego baza danych jest zapisana w chmurze dostawcy usługi. Producent udostępnia aplikacje na popularne systemy mobilne oraz wtyczki do najpopularniejszych przeglądarek.

Bezpieczeństwo jest bazuje, tak jak w przypadku KeePassa, o AES-256. Różnica polega na sposobie generowania klucza z hasła głównego. Zastosowano tu algorytm PBKDF2. Jak zapewnia producent, pomimo przechowywania danych w jego chmurze, nie ma on dostępu do danych, a jedynie do ich zaszyfrowanej wersji, a zarówno hasło główne jak i klucze szyfrujące nigdy nie trafiają na jego serwer, a są używane lokalnie. Cała infrastruktura jest jednak własnością producenta i nikt z zewnątrz nie ma wglądu w sposób jej działania. Niemożliwe jest też postawienie własnej infrastruktury do przechowywania danych.

Rozwiązania chmurowe pozwalają LastPassowi wyeliminować problem synchronizacji danych między urządzeniami. Ponadto integracja z systemami mobilnymi i przeglądarkami pozwala nawet mało zaawansowanym użytkownikom z powodzeniem nauczyć się jego obsługi.

Dobra integracja z systemem użytkownika zmniejsza powierzchnię ataku za pomocą złośliwego oprogramowania, a szczególnie, jeśli nie jest

ono ukierunkowane w atak na samego LastPassa. W przypadku ataku skierowanego może dojść do wycieku hasła głównego oraz każdego wpisanego przez sam menadżer hasła. Sytuację ratuje możliwość skonfigurowania uwierzytelniania dwuskładnikowego, w tym bazującego o kody SMS lub karty inteligentne. Dzięki temu hasło główne nie wystarczy do wycieku całej bazy [16].

7.2.3 Mooltipass

Całkowicie inne podejście prezentuje Mooltipass. Mooltipass nie jest on programem działającym na urządzeniu użytkownika ani usługą chmurową, a fizycznym urządzeniem. Podobnie jak LastPass, oferuje on bardzo dobrą integrację z najpopularniejszymi systemami mobilnymi i przeglądarkami.

Algorytm szyfrujący jest podobny do poprzednich rozwiązań. Baza danych jest jednak przechowywana poza komputerem użytkownika ale nigdy nie trafia do chmury. Klucze szyfrujące natomiast przechowywane są na karcie inteligentnej, zabezpieczone kodem PIN, który po trzykrotnym błędnym wpisaniu blokuje kartę na zawsze. Taka architektura całkowicie zabezpiecza przed niepowołanym dostępem. Jedyną możliwością kradzieży całej bazy danych jest fizyczna kradzież zarówno urządzenia jak i karty oraz poznanie przez złodzieja kodu PIN zabezpieczającego kartę.

Hasła są wpisywane po potwierdzeniu na interfejsie urządzenia chęci przeprowadzenia takiej operacji za pomocą interfejsu USB HID, poprzez emulację klawiatury. Wybór serwisu może odbywać się zarówno, poprzez wtyczkę do przeglądarki, jak i interfejs urządzenia. Oba rozwiązania pozwalają na kompatybilność z każdym produkowanym obecnie urządzeniem.

Wadą takiego rozwiązania jest konieczność noszenia dedykowanego urządzenia. Jego obecność może być też problematyczna w przypadku korzystania z urządzeń mobilnych, w szczególności smartfonów.

Niestety, jeżeli chodzi o bezpieczeństwo wprowadzanych haseł, także i ten menadżer zapewnia go tak długo jak urządzenie użytkownika nie będzie zainfekowane złośliwym oprogramowaniem. Inaczej, pozostaje bezradne, uniemożliwiając jednak wyciek całej bazy, a jedynie aktualnie używanych poświadczeń [17].

7.2.4 Podsumowanie

Różnorodność dostępnych menadżerów haseł pozwala użytkownikowi na swobodny wybór pomiędzy większym bezpieczeństwem, wygodą i kompatybilnością. Żadne z opisanych rozwiązań nie oferuje jednak wszystkich trzech cech jednocześnie.

Wszystkie trzy rozwiązania mają też jeden, wspólny słaby punkt w poziomie oferowanego bezpieczeństwa, nie chronią przed atakującym, który uzyskał już dostęp do komputera użytkownika, jedynie starają się minimalizować poniesione przez niego straty.

Najbliżej modelu pełnej ochrony znajduje się obecnie Mooltipass, przechowujący bazę na dedykowanym do tego urządzeniu i chroniący dostępu do jego zawartości. Dzięki czemu jedyne dane jakie mogą z niego wyciec to aktualnie wprowadzane hasła. Użycie karty inteligentnej, jako elementu całego procesu, wydaje się rozwiązaniem, które ma największy potencjał rozwoju. Niestety zastąpienie, całego dedykowanego urządzenia, służącego jednocześnie jako czytnik kart, magazyn danych i interfejs dla komputera, czytnikiem kart dostępnym na rynku nie zapewni całkowitego bezpieczeństwa.

Potrzebny zatem jest jeszcze jeden element, aby opracować system umożliwiający pełne zabezpieczenie procesu logowania do dowolnego serwisu internetowego. Konieczne jest zabezpieczenie warstwy transportującej hasło z magazynu danych do sesji TLS, zabezpieczającej połączenie przez sieć.

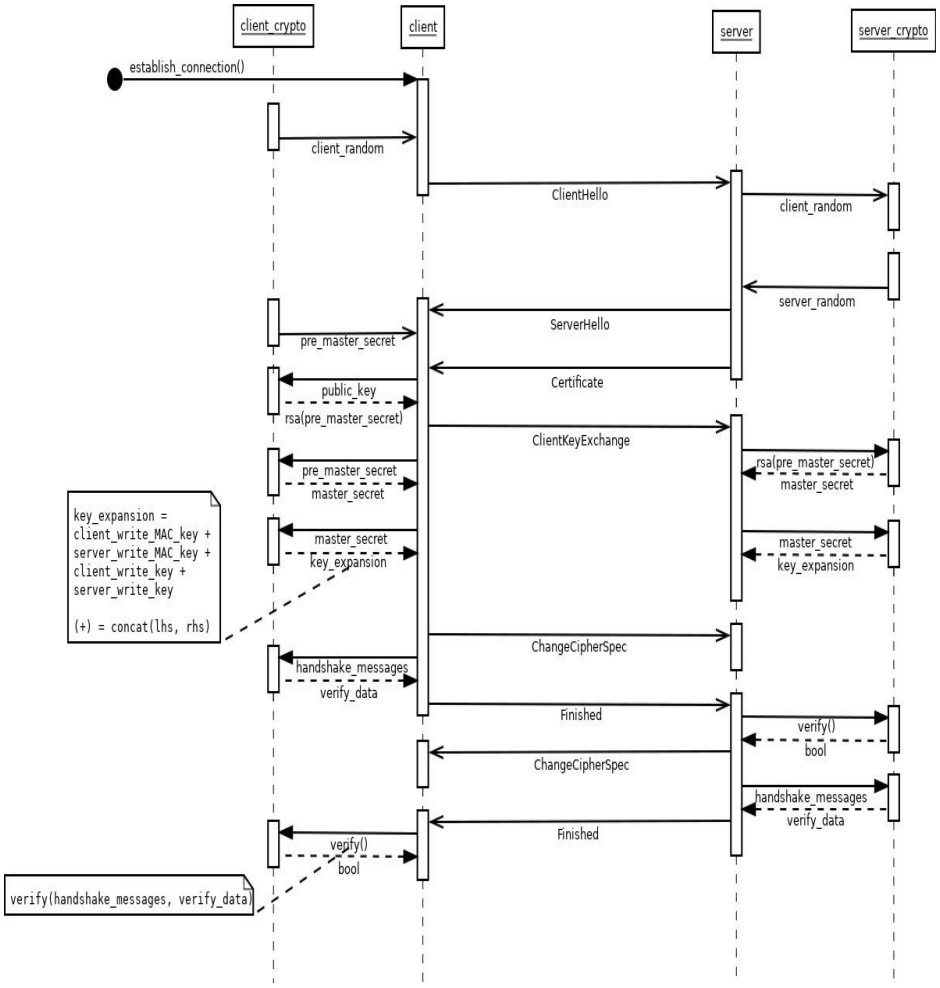
7.3 Bezpieczeństwo warstwy transportowej – TLS

Zadaniem protokołu TLS jest zapewnienie bezpieczeństwa i integralności komunikacji, a także umożliwienie stronom potwierdzenia swojej tożsamości. W przypadku procesu logowania, szczególnie ważne jest zweryfikowanie tożsamości serwera, aby żaden atakujący nie był w stanie podszyć się pod niego.

7.3.1 Zestawianie połączenia

Protokół zestawienia, który musi zrealizować wiele zadań jest skomplikowanym procesem zestawiania połączeń. Jego przybliżoną zasadę działania podano na rys. 18. Bardziej szczegółowe informacje można znaleźć w [18].

Komunikacja jest inicjowana przez klienta, zaraz po pomyślnym zestawieniu połączenia TCP, które zapewnia transport pakietów TLS. Sam pakiet jest podzielony na dwie warstwy. Pierwszą jest warstwa rekordu, umożliwiająca odbiorcy poznanie długości całego pakietu. Jest ona przesyłana w postaci niezaszyfrowanej, nawet po przejściu stron na komunikację szyfrowaną. W niższej z warstw strona może wysłać jeden z czterech rodzajów komunikatów: alert, handshake, application data oraz change cipher spec.



Rys. 18. Schemat ustanawiania połączenia TLS (źródło: własne)

Komunikaty typu handshake umożliwiają wymianę danych potrzebnych do wygenerowania parametrów połączenia. Większość pakietów tego typu jest przesyłana w formie niezaszyfrowanej, po ustaleniu parametrów szyfrowania sprawdzana jest ich integralność. Pierwszym z komunikatów tego typu jest ClientHello. Zawiera on między innymi oferowane przez klienta zestawy algorytmów szyfrujących oraz blok losowych danych, które nieco później będą stosowane do wygenerowania kluczy szyfrujących. Lista zestawów algorytmów musi natomiast zawierać obowiązkowo jeden, podstawowy zestaw - *TLS RSA WITH AES 128 CBC SHA*.

Odpowiedzią serwera na ten pakiet jest odesłanie komunikatu ServerHello. Zawiera on zestaw algorytmów szyfrujących wybranych spośród zaoferowanych przez klienta. Ponadto, także serwer generuje losowy blok danych, dzięki czemu losowość wygenerowanych kluczy zależy od obu stron komunikacji i słabość jednej z nich nie powinna wpływać na bezpieczeństwo całego połączenia.

Zaraz po przesłaniu wiadomości powitalnych, serwer powinien przesłać klientowi swój łańcuch certyfikatów. Służy do tego komunikat Certificate. Jest to etap krytyczny z punktu widzenia bezpieczeństwa komunikacji. Podstawienie przez atakującego fałszywego certyfikatu lub wyciek powiązanego z nim klucza prywatnego powodują, że cały proces traci zarówno poufność jak i integralność.

Proces ten wykracza jednak poza działanie protokołu TLS, wkraczając w zagadnienia infrastruktury klucza publicznego oraz protokołu X.509. Z punktu widzenia protokołu TLS ważne jest jedynie wypakowanie z certyfikatu serwera klucza publicznego, w postaci wartości modulus (n) oraz exponent (e).

Następnie serwer wysyła komunikat ServerHelloDone oznaczający, że powitanie zostało zakończone i nieprzenoszący żadnych danych.

Ponieważ klient otrzymał już wszystkie dane umożliwiające skuteczną komunikację z serwerem za pomocą szyfrowanego kanału komunikacji etap nawiązywania połączenia mógłby zbliżać się do końca. Ponieważ jednak szyfrowanie asymetryczne nie należy do najwydajniejszych, dane w protokole TLS są szyfrowane za pomocą szyfrów symetrycznych. Obecnie najczęściej stosowanym jest algorytm AES. Należy także oprócz poufności zapewnić integralność danych. Z tego względu proces generowania kluczy jest wieloetapowy i skomplikowany.

Pierwszym jego etapem jest wymiana kolejnego bloku losowych danych, które przesyłane są w formie zaszyfrowanej za pomocą odebranego klucza publicznego i wysłane przez klienta wewnątrz komunikatu ClientKeyExchange. Dane te zwane są PreMasterSecret. Jedyną funkcją PreMasterSecret jest wygenerowanie kolejnego klucza MasterSecret, który jest generowany na podstawie PreMasterSecret po obu stronach komunikacji, po czym PreMasterSecret jest kasowany z pamięci. Proces przekształcania jednego sekretu w drugi jest wykonywany za pomocą funkcji zwanej PseudoRandomFunction (w skrócie PRF). Przyjmuje ona trzy parametry: sekret, ziarno (ang. *seed*) oraz etykietę (ang. *label*). Sama funkcja działa na zasadzie wielokrotnego wywoływania funkcji HMAC, dzięki czemu pozwala rozwinąć podany sekret do bloku o dowolnej długości. W przypadku MasterSecret, jako etykieta używany jest ciąg znaków „master secret". Ziarnem są oba bloki losowych danych wymienione w wiadomościach powitalnych.

Ciąg ten nie jest jednak jeszcze ostatecznym kluczem. Na jego podstawie, z tymi samymi losowymi danymi, które użyte są jego wygenerowaniu, ale w odwrotnej kolejności jest wygenerowany dopiero ostateczny blok danych będący kluczami. Do wygenerowania bloku użyta jest ta sama funkcja PRF, z etykietą" key expansion".

W tym momencie oba końce znają następujące klucze:

- sekret HMAC klienta,
- sekret HMAC serwera,
- klucz szyfrujący klienta,
- klucz szyfrujący serwera.

Następnym komunikatem jest change cipher spec, nakazujący przejście na szyfrowaną komunikację ze strony od klienta do serwera.

Zaraz po nim klient wysyła szyfrowany komunikat Finished, którego zadaniem jest weryfikacja nowo wygenerowanych parametrów połączenia oraz zapewnienie integralności wymienionych do tej pory danych. Stosowana jest po raz kolejny funkcja PRF, tym razem ze skrótem strumienia składającego się z zawartości wszystkich dotychczasowych pakietów typu handshake jako ziarno. Dodatkowo wiadomość ta jest podpisana za pomocą algorytmu HMAC, co pozwala na weryfikację zarówno poprawności odszyfrowania jak i integralności całego komunikatu.

Analogiczną czynność wykonuje serwer, wysyłając kolejno pakiety change cipher spec oraz Finished, gdzie w tym ostatnim do strumienia, którego skrót jest obliczany i dodawany jest komunikat klienta Finished. Gdy jego poprawność zostanie zweryfikowana, połączenie można uznać za otwarte. W przeciwnym razie strona, której nie udał się jakikolwiek poprzedni etap, jest zobowiązana do wysłania komunikatu alert, powodującego zerwanie połączenia. Warto zauważyć, że alert nie może pojawić się na żadnym z poprzednich etapów, aby uniemożliwić atakującym analizę, na którym etapie wystąpił błąd, uniemożliwiając specyficzne rodzaje ataków.

7.3.2 Wymiana danych

Wymiana danych odbywa się za pomocą pakietów typu application data. Są one zaszyfrowane przez wynegocjowany algorytm szyfrowania symetrycznego, najczęściej AES w trybie CBC (*Cipher Block Chaining*). Ponadto do wiadomości dodana jest wartość funkcji HMAC umożliwiająca weryfikację integralności danych.

7.3.3 Wymagania odnośnie implementacji TLS

Podstawowym wymaganiem, jakie stawia standard RFC, odnośnie implementacji protokołu TLS jest dostępność zestawu *TLS RSA WITH AES 128 CBC SHA*. Szyfrowanie wartości PreMasterSecret odbywa się tu za pomocą algorytmu RSA z kluczem o długości zależnej od klucza publicznego serwera, a więc pozostającej poza kontrolą klienta.

Szyfrowanie symetryczne odbywa się przy pomocy algorytmu AES z blokiem o długości 128 bitów, w trybie CBC czyli szyfrem blokowym. Z tego względu szyfrowane dane muszą zostać dopełnione tzw. paddingiem. Nie jest on jednak kompatybilny z najpopularniejszym standardem PKCS#7. Zamiast tego definiuje swój własny format paddingu. Z tego względu istnieje bardzo duże prawdopodobieństwo, że wybrana implementacja AES nie będzie miała dla niego wsparcia i wystąpi konieczność opracowania własnej i użycia biblioteki w trybie bez paddingu.

Ostatnim elementem zestawu jest algorytm podpisu, zwany MAC (ang. *Message Authentication Code*). MAC używa implementacji na bazie sum kontrolnych *Hash-based MAC*, w skrócie HMAC. Podstawowy zestaw, jako podstawę, definiuje SHA-1. Warto jednak zauważyć, że zanim

strony ustalą parametry, HMAC jest używany jeszcze w funkcji PRF. W tym przypadku standard definiuje jeden dostępny algorytm HMAC, który nie może zostać zmieniony -SHA-256.

Podsumowując, do prawidłowej implementacji protokołu TLS wymagane są implementacje następujących algorytmów kryptograficznych:

- RSA o zmiennej długości klucza,
- AES 128-bitowy w trybie CBC,
- padding zgodny z SSL,
- HMAC-SHA-1,
- HMAC-SHA-256.

Ponadto do zapewnienia całkowitego bezpieczeństwa wymagana jest implementacja standardów ASN.1 oraz X.509, do manipulacji certyfikatami i przechowywania listy zaufanych certyfikatów, a więc pełna implementacja infrastruktury klucza publicznego (PKI).

7.3.4 Szczegóły kryptograficzne

Aby zapewnić pełne bezpieczeństwo połączenia konieczne jest zwrócenie szczególnej uwagi na pewne drobne szczegóły. Ich pominięcie może skutkować pozyskaniem przez atakującego wartości master secret, jakiegoś z kluczy szyfrujących lub podpisujących oraz poznanie części lub całości przesyłanych danych bez znajomości kluczy. Standard TLS definiuje między innymi następujące szczególnie wrażliwe przypadki.

Pierwszym z newralgicznych punktów jest generator liczb pseudolosowych. Użycie niewystarczająco dobrze wylosowanej wartości pre master secret może spowodować przewidzenie tej wartości przez atakującego. Zwrócić należy uwagę na wartość ziarna, jakim inicjalizowany jest generator. Przydatne może się też okazać zbadanie entropii wygenerowanych nim bloków.

Drugim ważnym etapem jest weryfikacja autentyczności certyfikatu przesłanego przez serwera oraz jego centrum certyfikacji (CA) . Jeżeli chodzi o sam certyfikat, to ważne jest też, aby zawarty w nim klucz publiczny miał odpowiednią długość. Obecnie zalecaną długością jest 2048 bitów. Coraz częściej spotyka się także klucze o długości 4096 bitów. Ważnym parametrem jest data ważności certyfikatu. Klient musi zweryfikować, że

nie znajduje się ona ani w przyszłości, ani w przeszłości. Należy także dokonać sprawdzenia czy certyfikat nie figuruje na liście unieważnionych certyfikatów (ang. *Certificate Revocation List - CRL*), której adres powinien być dołączony do certyfikatu. Ważne jest także, aby weryfikować wersję protokołu używaną przez drugą stronę. Jest to szczególnie istotne w przypadku klientów wspierających SSLv2, który nie posiadał obrony przed atakiem obniżającym wersję protokołu. Obrona taka została szczegółowo opisana w standardzie TLS.

Istotnym szczegółem łatwym do przeoczenia jest weryfikacja, jeśli został już opracowany atak po jego pominięciu [10] lub w przypadku podpisu wykonanego algorytmem RSA. Konsekwencją braku takiej weryfikacji jest podrobienie podpisu.

Należy również zwrócić uwagę na sposób generowania wektora IV dla trybu CBC algorytmu szyfrowania. Kluczowe jest, aby atakujący nie miał szans przewidzieć wartości wektora. W przeciwnym razie atakujący ma możliwość potwierdzenia, że zaszyfrowany blok ma konkretną zawartość [11].

Weryfikacja wartości HMAC może prowadzić do wycieku pewnych informacji. Z tego względu należy zapewnić identyczność czasu weryfikacji kodu HMAC niezależnie od poprawności paddingu odszyfrowanego pakietu.

Kluczowa jest weryfikacja wszystkich aspektów poprawności wiadomości Finished. Jej nagłówek zawiera informację o wersji protokołu, jakiej żąda druga strona. Wersja ta jest szyfrowana i podpisana, więc może zostać uznana za wiarygodną. Wartość HMAC pozwala sprawdzić, że wiadomość dotarła nienaruszona. Wartość verify data natomiast pozwala sprawdzić integralność całości odbytej wcześniej komunikacji. Dzięki temu niemożliwe jest wymuszenie na stronach komunikacji zastosowania starszej wersji protokołu lub wybranie słabego zestawu szyfrów. W przypadku błędu na którymkolwiek z tych etapów proces musi być kontynuowany, aby zapobiec atakom timingowym.

Dopiero po zakończeniu weryfikacji błąd może być zwrócony drugiej stronie. Nie może ona jednak poznać powodu tego błędu. Aby zastosować wszystkie możliwości protokołu TLS odnośnie ochrony informacji, powinno się stosować zestawy oparte o szyfry DHE (z ang. *Diffie-Hellman*

Ephemeral). Zapewni to tzw. *Perfect Forward Secrecy*, dzięki czemu wyciek kluczy długoterminowych nie powoduje wycieku kluczy poszczególnych sesji zakończonych w przeszłości.

Weryfikacja wartości HMAC jest ważna ze względu na zapewnienie integralności danych. Zapobiega również próbom usunięcia wybranych pakietów z komunikacji lub zmiany ich kolejności. Jest to możliwe dzięki zastosowaniu numeru sekwencyjnego dla kolejnych pakietów. Może wystąpić przypadek, że mimo zmiany numeru sam pakiet nie został zmodyfikowany. Wówczas wiadomość zostanie zweryfikowana negatywnie przez drugą stroną co w konsekwencji doprowadzi do zerwania połączenia.

Warto podkreślić, że TLS najpierw podpisuje wiadomość, a dopiero później ją szyfruje. Metoda ta nosi nazwę *authenticate-then-encrypt* i w przeciwieństwie do metody *encrypt-then-authenticate*, zapewnia bezpieczeństwo jedynie wobec niektórych kombinacji algorytmów szyfrujących i podpisujących. Jednym z przypadków, gdzie bezpieczeństwo to zostało udowodnione jest jednak tryb CBC bezpiecznych szyfrów blokowych [18].

7.4 Bezpieczny kontener danych - JavaCard

7.4.1 Smart card według standardu ISO 7816

Standard ISO/IEC 7816 opisuje wszystkie aspekty funkcjonowania kart inteligentnych (ang. *smart cards*), począwszy od ich fizycznych wymiarów, przez opis złącz elektrycznych aż po interfejs USB-ICC, który pozwala na podłączenie karty do złącza USB.

Przy tworzeniu aplikacji, działających na karcie, najważniejszy jest standard ISO/IEC 7816-4, opisujący format komunikatów, wymienianych pomiędzy kartą, a komputerem, zwanych APDU (ang. *Appication Protocol Data Unit*).

APDU wymieniane są parami. Komunikacja zawsze inicjowana jest przez komputer, a zadaniem karty jest odpowiedź na przysłane żądanie. Format żądania jest następujący:

$$\boldsymbol{CLA} \quad \boldsymbol{INS} \quad \boldsymbol{P_1 P_2} \quad \boldsymbol{L_C} \quad \boldsymbol{D_0 \ldots} \quad \boldsymbol{D_{Lc-1}} \quad \boldsymbol{L_e}$$

Przy czym każdy element ma długość 8 bitów (jeden bajt). Znaczenie poszczególnych pól w pakiecie jest następujące:

- *CLA* oznacza klasę. Najstarszy bit tego pola oznacza przynależność do klasy własnościowej lub międzybranżowej. Pozostałe bity mają przypisane znaczenie jedynie w klasie międzybranżowej. Dla klas własnościowych istnieje pełna dowolność w ich używaniu.
- *INS* oznacza komendę, jaką ma wykonać aplikacja zainstalowana na karcie. Sam standard alokuje wiele z wartość do konkretnych komend. Przykładem takiej komendy jest komenda *SELECT*, do której przypisano wartość 0xA4.
- P_1 oraz P_2 służą jako parametry dla wybranej komendy i ich znaczenie leży w gestii projektującego komendę.
- Wartość L_c oznacza długość pola danych. Ponieważ jest to wartość jednobajtowa, to maksymalna długość danych jaka jest możliwa do przesłania za pomocą protokołu wynosi 255 bajtów.
- Bajty D_0 - D_{L_c-1} stanowią pole danych, o długości określonej przez pole L_c. Ich znaczenie jak w przypadku parametrów, jest zależne od projektanta komendy.
- Pole L_e określa maksymalną spodziewaną długość danych w odpowiedzi otrzymanej z karty.

Na tak wysłane APDU, karta ma obowiązek odpowiedzieć APDU odpowiedzi mającym następujący format:

$$L_e \quad D_0 \ldots \quad D_{L_e-1} \quad SW_1 \quad SW_2$$

- L_e oznacza ilość bajtów danych przesłanych przez kartę.
- D_0 - D_{L_e-1} działają analogicznie do danych w APDU żądania.
- SW_1 oraz SW_2 stanowią dwubajtowy kod odpowiedzi, gdzie niektóre wartości są zdefiniowane w standardzie. Najczęściej spotykanymi z nich są:
  - 0x9000 – sukces,
  - 0x6E00 – nieobsługiwane *CLA*,
  - 0x6D00 – nieobsługiwane *INS*.

7.4.2 Cechy JavaCard

JavaCard jest jedną z platform języka Java, obok Java Enterprise (EE), Standard (SE) oraz Micro (ME). Jest to technologia umożliwiająca uruchamianie aplikacji napisanych w języku Java (zwanych apletami) na kartach chipowych. Początki technologii sięgają połowy lat dziewięćdziesiątych. Głównymi zastosowaniami są karty SIM oraz karty bankomatowe debetowe i kredytowe.

Warto pamiętać, że JavaCard, która jest środowiskiem języka Java, nie zapewnia kompatybilności bajtkodu z innymi maszynami np. JVM. Bajtkod jest optymalizowany tak, aby zajmował jak najmniej miejsca i dostosowywał się do działania w systemie wbudowanym, o ograniczonych zasobach.

Cechą charakterystyczną technologii jest zapewnienie izolacji apletów. Dzięki temu możliwe jest zainstalowanie kilku apletów na jednej karcie. Nie należy obawiać się, że aplet pełniący funkcję legitymacji studenckiej, zarządzanej przez uczelnię, wejdzie w posiadanie danych innego apletu, będącego interfejsem karty kredytowej.

Ważną cechą jest zarządzanie czasem życia apletów. Definiuje ono kilka podstawowych operacji na aplecie:

- install – odpowiada za instalację apletu, w szczególności stworzenie instancji oraz zarejestrowanie apletu w systemie przez metodę register,
- register – rejestruje aplet w systemie,
- select – pozwala na inicjalizację sesji,
- process – przetwarza APDU wysyłane do apletu,
- deselect – pozwala na zakończenie sesji.

Taka architektura ma swoje konsekwencje. Szczególnie duży wpływ na rozwiązania, jakie programiści muszą stosować w opracowywanych przez siebie apletach ma fakt tworzenia obiektu apletu jedynie w funkcji install (raz przez cały czas życia karty).

Specyfika działania karty chipowej, która jest programowana jedynie raz przez dostawcę polega na tym, że muszą zostać przedsięwzięte środki zapewniające integralność danych.

Jest to szczególnie ważne, w przypadku zaniku napięcia w trakcie przesyłania danych np. kopiowania danych do bufora. Aby temu zapobiec, JavaCard jest nastawiona na model transakcyjny newralgicznych operacji,

takich jak kopiowanie. Oczywiście, API oferuje również operacje w formie nieatomowej, pozwalając programiście podjąć decyzję, której funkcji użyć. Specyfiką stosowania API sposób zarządzania pamięcią. Obiekty tworzone przez operator new są alokowane w pamięci EEPROM.

API udostępnia alokację danych w pamięci RAM, na żądanie programisty. Ponadto, JavaCard nie wymusza implementacji garbage collectora, przez co raz zbudowane obiekty istnieją przez cały czas życia apletu.

Nowsze wersje standardu JavaCard Connected Edition, wersja 3.x udostępniają nieco inne podejście do pisania apletów niż opisane wyżej podejście klasyczne (JavaCard Classic, wersje 2.x i niższe). Standard ten przewiduje połączenie apletu z Internetem.

7.4.3 Implementacja API JavaCard - JavaCard Open Plat-form (JCOP)

JCOP jest implementacją udostępnianą przez NXP, na bazie JavaCard w wersji 2.2.2, a więc niewyposażoną jeszcze w udogodnienia webowe. Jest to jednak obecnie najtańsza i najbardziej dostępna implementacja standardu. Karta taka może mieć przykładowo 36 KiB pamięci EEPROM oraz około 2 KiB dostępnej pamięci RAM. Udostępnia całkiem spory zestaw funkcji kryptograficznych. Dużą zaletą implementacji dostępnych w bibliotece jest szybkość ich działania.

7.4.4 Biblioteka kryptograficzna JCOP

JCOP udostępnia klasy do obsługi szyfrowania zarówno symetrycznego jak i asymetrycznego. Dostępne są algorytmy takie, jak AES (w trybach CBC i ECB, bez paddingu), DES (również CBC i ECB, bez paddingu lub padding zgodny z ISO 9797) czy RSA (bez paddingu lub padding zgodny z PKCS#1).

Możliwe jest także podpisywanie wiadomości algorytmami takimi jak AES CBCMAC, ECDSA-SHA czy RSA-SHA. Niedostępne są natomiast podpisy w oparciu o algorytmy HMAC.

Dostępne długości kluczy dla algorytmów szyfrowania to: dla algorytmu AES 128, 192 oraz 256 bitów; dla algorytmu RSA - 512 do 2048 bitów. W przypadku RSA - 512 niemożliwe jest zastosowanie klucza o długości 4096, będącego w coraz powszechniejszym zastosowaniu.

JCOP posiada generator liczb pseudolosowych, umożliwiający generowanie losowych bloków, które mogą być użyte w zastosowaniach kryptograficznych.

Jeżeli chodzi o funkcje skrótu, to dostępne są algorytmy takie jak: MD5, SHA-1, SHA-224 (przez niestandardową klasę) oraz SHA-256. Ponadto możliwe jest użycie algorytmów SHA-1, SHA-224 oraz SHA-256 w formie preinicjalizowanej, umożliwiając przekazanie skrótu niepełnych danych z zewnątrz i doliczenie reszty danych już na karcie. Ostatnią z dostępnych funkcji jest obliczanie wartości funkcji CRC-16 danych.

7.4.5 Ograniczenia

Ze względu na fakt, że system jest systemem wbudowanym, JavaCard, a także jego implementacja JCOP, mają względem standardowego środowiska języka Java bardzo specyficzne ograniczenia. Wspomniana wcześniej optymalizacja bajtkodu pociągnęła za sobą ograniczenia w języku oraz API. Najbardziej widocznymi ograniczeniami jest brak klasy String oraz typu int. Ponadto nie można stosować enumeracji oraz wielu bardziej zaawansowanych konstrukcji języka. Dużym ograniczeniem względem nowoczesnego podejścia do pisania aplikacji w Javie jest korzystanie ze środowiska wersji piątej tego języka (JDK 1.5)[26]. Spośród wymienionych największe konsekwencje ma brak typu int w większości implementacji. Wymusza to na programiście rzutowanie zmiennych na typ o mniejszym rozmiarze po każdej operacji, takiej jak dodawanie. Powoduje to momentami ogromny narzut na objętość kodu i zmniejsza jego czytelność.

Specyfika zarządzania pamięcią ma duże konsekwencje na sposób budowy aplikacji zgodnych z JavaCard. Ponieważ, ze względu na fakt alokacji obiektów domyślnie w pamięci EEPROM, alokowanie danych często zmiennych w ten sposób spowoduje zbyt szybkie zużycie tej pamięci (producent podaje ok. 0,5 mln cykli zapisu jako jej trwałość). Przez to programista musi zadbać, które dane mają być przechowywane na tej lub innej pamięci. Istnieje możliwość zaalokowania bufora w pamięci RAM, jednak mała ilość dostępnej pamięci tego typu spowoduje jej szybkie wyczerpanie. Jest to problem, szczególnie ważny z powodu braku garbage collectora na większości implementacji. Z tego względu, wypracowanie metody sprawnego i bezpiecznego zarządzania pamięcią leży w gestii programisty.

Konieczność budowy wszystkich potrzebnych obiektów z funkcji install, lub konstruktorów przez nią wywołanych ma duży wpływ na architekturę całej aplikacji. Pomimo, że używany język Java jest językiem zorientowanym obiektowo, to de facto nie jest możliwe korzystanie z wszelkich dobrodziejstw tego podejścia, takich jak np. wzorce projektowe. Niemożliwe, ze względu na specyfikę języka Java, jest ukrycie wskaźników do faktycznych obiektów w pamięć lub zastosowanie programowania strukturalnego znanego z języków z rodziny C/C++/ Cały aplet sprowadza się więc do manipulacji na blokach danych, o nieopisanej strukturze.

Ostatnim ważnym ograniczeniem jest brak dostępności niektórych funkcji kryptograficznych. Szczególnie istotny jest mały wybór metod dopełniania danych do pełnych bloków w przypadku szyfrów blokowych takich jak AES czy RSA. Sprawia to, że z dużym prawdopodobieństwem to programista będzie musiał poprawnie dokonać dopełnienia danych paddingiem.

JCOP nie implementuje żadnych algorytmów podpisu opartych o HMAC. Dostępne są jednak funkcje skrótu. Powoduje to jednak, że ich implementacja również będzie należała do zadań programisty. Z punktu widzenia algorytmicznego nie jest to oczywiście zadanie trudne, ale powoduje konieczność operowania na potencjalnie dużych blokach danych, zużywając pamięć EEPROM lub wyczerpując pamięć RAM. Nie obędzie się również bez spadku wydajności spowodowanego brakiem sprzętowego wsparcia, które byłoby dostępne, jeżeli JCOP zapewniałby wsparcie dla HMAC.

8. Projekt systemu

Podstawą do projektowanego systemu jest karta chipowa, umożliwiająca uruchamianie apletów zgodnych z technologią JavaCard. Technologia ta służy do budowy bezpiecznego kontenera na hasła i przechowywania kluczy kryptograficznych aktualnie otwartego połączenia. Jedyną metodą komunikacji karty ze światem zewnętrznym jest protokół ISO 7816. Komunikacja z serwerem odbywa się przy pomocy protokołu TLS w wersji 1.2.

8.1 Ogólna architektura

Zdefiniowany zestaw technologii umożliwia implementację protokołu TLS na jeden z dwóch sposobów. Pierwszy z nich zakłada implementację całości protokołu TLS po stronie karty. Drugim podejściem jest implementacja tylko tej części protokołu, której implementacja jest konieczna po stronie karty, ze względu na zastosowanie parametrów kryptograficznych połączenia. Oba te podejścia mają pewne zalety i wady.

W pierwszym przypadku, gdy karta jest odpowiedzialna za całość implementacji, aplikacja kliencka staje się jedynie swego rodzaju serwerem proxy. Karta natomiast przejmuje rolę klienta. Jest to rozwiązanie bezpieczniejsze ze względu na standard. Jego twórcy nie zakładają, że część protokołu będzie implementowana w innym środowisku niż reszta. Zapewnia to duże uproszczenie logiki, niezbędnej do zaimplementowania. Rozwiązanie to posiada wady. Jedną z nich jest konieczność implementacji protokołu sieciowego w mocno ograniczonym środowisku, jakim jest język Java na platformę JavaCard. Dużymi brakami są brak rzeczywistej obiektowości i typów enumeracyjnych, szczególnie przydatnych w przypadku protokołów sieciowych. Problemy te nie wykluczają jednak teoretycznej możliwości opracowania takiej implementacji. Elementem, wykluczającym jest ilość dostępnych zasobów. Ponieważ jest to system wbudowany, każdy zaoszczędzony bajt może zdecydować o możliwości uruchomienia aplikacji na konkretnym urządzeniu. Tak więc każde rozwiązanie zakładające mniejsze zużycie pamięci będzie lepsze. To samo dotyczy potrzebnej mocy obliczeniowej, która byłaby zużywana na operacje, które z powodzeniem może wykonać komputer użytkownika.

Drugi wariant zakłada implementację na karcie jedynie operacji kryptograficznych potrzebnych do utrzymania poufności kluczy szyfrujących i uwierzytelniających. Pozwola to na redukcję do minimum zasobów stosowanych na platformie JavaCard, w szczególności alokowania dużych bloków pamięci pod pakiety protokołu TLS. Wadą tego podejścia jest skomplikowanie protokołu komunikacyjnego między kartą a komputerem. Szczególnie uwidacznia się to w procesie ustanawiania połączenia TLS.

Z opisanych powodów, szczególnie konieczności ograniczania zużycia zasobów karty do minimum, podstawą implementacji został drugi z wariantów.

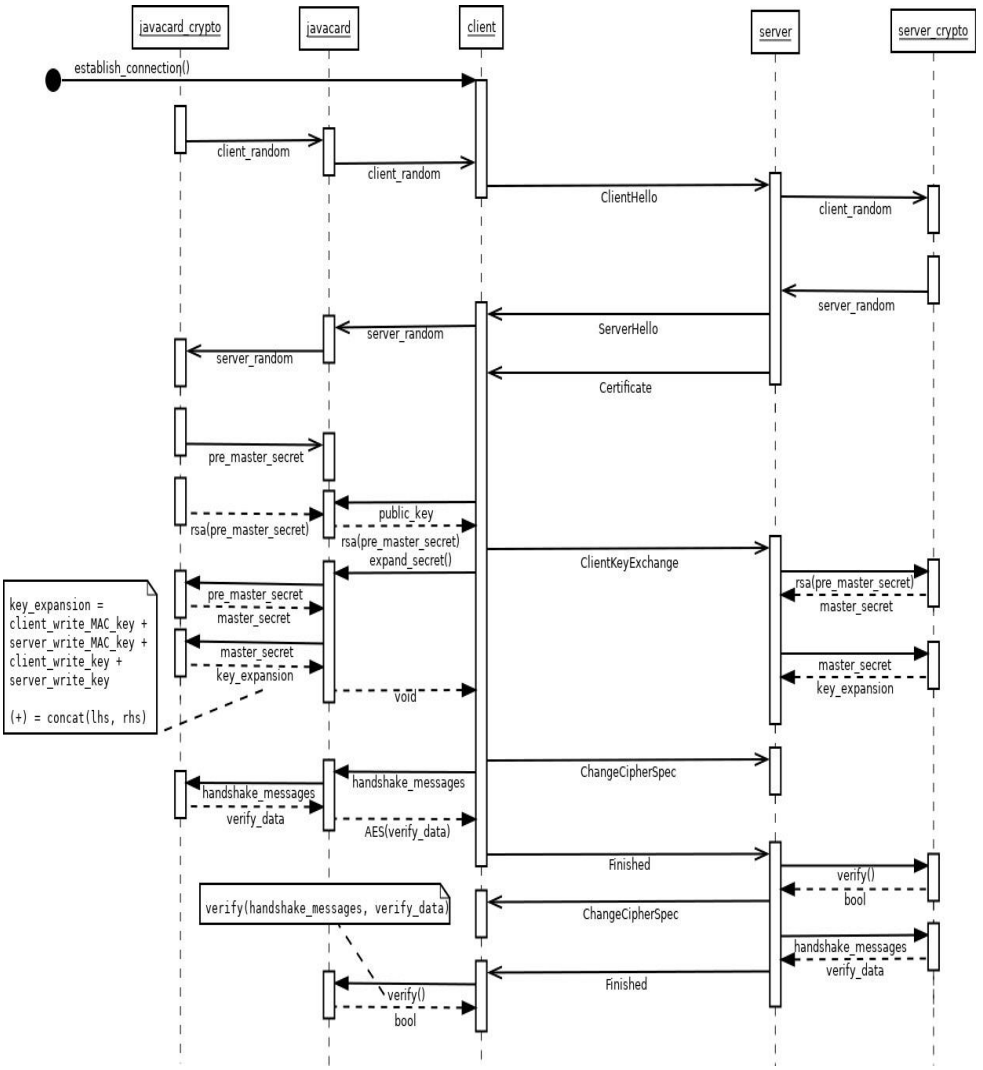
8.2 Wymiana kluczy TLS

Najważniejszym etapem procesu logowania jest ustanowienie połączenia TLS. Na rysunku 19 przedstawiony został cały proces w przypadku, gdy komputer klienta odpowiada za przechowywanie parametrów połączenia. Celem etapu projektowania było zaproponowanie procesu, gdzie klient nie będzie miał fizycznego dostępu do tych parametrów.

Rysunek 19 przedstawia reprezentację przepływu danych w projektowanym systemie. Centralnym punktem jest karta chipowa (oznaczona jako *javacard*). To na niej spoczywa przechowywanie wszystkich istotnych parametrów połączenia, takich jak klucze kryptograficzne. Aby zachować pełne bezpieczeństwo systemu, wszystkie istotne dane są z karty do komputera klienta przesyłane w sposób zaszyfrowany, gotowy do przesłania do docelowego serwera za pomocą protokołu TCP.

Projekt nie zakłada implementacji samego protokołu TLS po stronie karty. Pakiety, gotowe do przesłania w strumieniu TCP są wciąż tworzone po stronie klienta. Szczególnie uciążliwym ograniczeniem (roz.7.4.5), utrudniającym bądź wręcz uniemożliwiającym pełną implementację protokołu jest pamięć. Pierwszym problemem jest ilość dostępnej pamięci operacyjnej niecałe 2 KiB, co wymusza dynamiczne zarządzanie pamięcią, nawet przy założeniu, że protokół nie zostanie w pełni zaimplementowany po stronie karty. Drugim problemem jest żywotność pamięci EEPROM. Producent ocenia ją na około pół miliona cykli zapisu. Dla celów przygotowania rozwiązania typu Proof of Concept nie stanowi to problemu. Natomiast

po wdrożeniu systemu zbyt intensywnie wykorzystującego tego typu pamięć, mogłoby dojść do znaczącego skrócenia żywotności całego systemu. Aby więc możliwie ograniczyć skutki obu czynników, tworzenie pakietów odbywa się po stronie klienta.



Rys. 19. Proponowany schemat ustanawiania połączenia TLS z użyciem karty chipowej (źródło: własne)

Najpoważniejszym elementem przedstawionej architektury jest implementacja pakietu Finished. Jego zawartością jest wyjście funkcji PRF, której ziarnem jest skrót wszystkich wysłanych do tej pory komunikatów, w większości zawierających dane niezaszyfrowane. Co więcej jest to pierwszy pakiet, po zmianie parametrów połączenia, a więc używający świeżo wynegocjowanych kluczy kryptograficznych, zarówno do zaszyfrowania swojej zawartości, jak i do jej podpisania. Konieczne jest dostarczenie skrótu przez klienta. W pewnych sytuacjach może to stanowić lukę bezpieczeństwa.

Także drugi pakiet Finished stanowi problem. W jego przypadku wartość handshake messages dodatkowo zawiera poprzedni pakiet Finished, oraz potencjalnie także inne pakiety, wysłane po nim. Powoduje to, że po pierwsze musi zostać zapewniony interfejs do wysyłania tych pośrednich pakietów i po drugie wartość handshake messages pierwszego z pakietów musi zostać przechowana, dodając do niej sam pakiet, w celu obliczenia nowego skrótu. Innym wyjściem jest pominięcie weryfikacji pakietu. Może to nieść konsekwencje dla bezpieczeństwa transmisji.

Pozostała część wymiany może zostać niemal w całości przeniesiona na kartę. Na wejściu należy jedynie podać wartości server random, klucz publiczny serwera oraz skrót wszystkich komunikatów, otrzymując na wyjściu w odpowiednich momentach zaszyfrowany asymetrycznie pre master secret oraz symetrycznie pakiet Finished.

Przedstawiony schemat nie zakłada weryfikacji certyfikatu, dostarczanego przez serwer, a jedynie przesłanie klucza publicznego w postaci dwóch liczb: eksponenta i modulusu. Powodem takiej decyzji jest wykraczanie obsługi certyfikatów poza problematykę tej pracy. Prawidłowa implementacja wymagałaby budowy obsługi formatów takich jak X.509 czy ASN.1, poprawną implementację infrastruktury klucza publicznego, w tym obsługę list CRL, przechowywanie certyfikatów centrów certyfikacji. Wszystkie te elementy musiałyby znajdować się po stronie karty, ze względu na możliwość przeprowadzania ataków Man in the Middle po stronie komputera klienta. Należałoby także zaprojektować odpowiednie mechanizmy zarządzania listami zaufanych centrów certyfikacji. Wszystkie te elementy mogłyby stanowić odrębne opracowanie. Skutkiem decyzji o pominięciu tych problemów jest natomiast rzeczywisty brak możliwości

zapewnienia bezpieczeństwa komunikacji. Dlatego też przedstawiony system nie będzie w pełni gotowy do wdrożenia jako rozwiązanie dla klienta końcowego a pozostanie jedynie demonstracją możliwości zaimplementowania pełnego systemu w przyszłości (tzw. *Proof of Concept, PoC*).

8.3 API komunikacji między kartą a klientem ISO 7816

Aby przesłać wymagane dane z i do karty, należy użyć protokołu ISO 7816. Narzuca to pewne ograniczenia na przepływ danych. Najważniejszym z tych ograniczeń jest długość pojedynczej paczki danych do 256 bajtów, a w praktyce 255 bajtów plus dwubajtowy kod błędu w stronę od karty oraz 254 bajtów plus parametry w stronę przeciwną. Ponieważ niektóre z przesyłanych komunikatów wymagają przepływu większych ilości danych, należy znaleźć sposób na ominięcie tych ograniczeń. Przy okazji warto również zwiększyć ilość danych przesyłanych w pojedynczym pakiecie do 256 bajtów. Dzięki temu dane zaszyfrowane algorytmami RSA oraz AES nie będą musiały być dzielone w połowie bloku. W przypadku AES mogłoby to być rozwiązane przez ograniczenie przesyłanych danych do 240 bajtów, co jednak wiązałoby się z marnowaniem kilkunastu bajtów na pakiet. Dla RSA jest to jeszcze ważniejsze, gdyż jeden z pakietów będzie musiał przenieść klucz publiczny, a ten obecnie najczęściej składa się z 2048 bitów modulusa, a więc dokładnie 256 bajtów.

Poza rozwiązaniem opisanego wyżej problemu, projekt API zawiera zestaw komend, który umożliwi skuteczne nawiązanie połączenia TLS, wybór hasła z bazy wewnątrz karty i wymianę zaszyfrowanych danych protokołu HTTP w celu przesłania hasła do serwera.

8.4 Przepływ danych protokołem HTTP

Aplikacje udostępniane przez serwery HTTP pozwalają na zalogowanie się do nich na jeden z dwóch podstawowych sposobów. Pierwszy polega na utworzeniu sesji, a zatem również przysłanie klientowi ciasteczka już w momencie pierwszej wizyty w serwisie. Drugi natomiast tworzy sesję w chwili wpisania przez użytkownika danych logowania. Aby wspierać obie sytuacje, projektowana aplikacja musi udostępniać interfejs pozwalający na przesłanie własnych nagłówków HTTP lub podanie zawartości ciasteczka, tak aby powiązać dane logowania z istniejącą sesją dla

pierwszego przypadku. Dla drugiego musi istnieć interfejs umożliwiający odebranie z karty zawartości ciasteczka przesłanego z odpowiedzią, która trafia na kartę w formie zaszyfrowanej. Same dane logowania login i hasło są prawie zawsze przesyłane za pomocą żądania POST, w formacie zbliżonym do:

login=jan_kowalski&password=tajnehaslo

Ważne jest także umożliwienie zdefiniowania własnych nazw dla parametrów login oraz password z powyższego przykładu. Ponadto, należy zabezpieczyć się na wypadek, gdyby istniała konieczność przesłania dodatkowych danych, np. zawartość obrazka typu CAPTCHA, pojawiającego się po kilku nieudanych próbach logowania (częsta praktyka wśród twórców stron).

Poniżej opisane zostały możliwe sposoby implementacji ze strony klienta.

8.4.1 Wariant 1 – pełna implementacja klienta HTTP

Najbardziej rozbudowanym rozwiązaniem mogłoby być implementacja pełnego klienta HTTP po stronie karty. Klient HTTP byłby odpowiedzialny za wygenerowanie całej zawartości nagłówka żądania HTTP oraz parametrów żądania POST na podstawie specyfikacji otrzymanej od aplikacji klienckiej. Aplikacja po stronie komputera lub też jej użytkownik byłby odpowiedzialny jedynie za podanie kilku wymaganych, niestandardowych wartości. Takich jak:

- URI strony logowania,
- nazwy kluczy parametrów POST,
- pośrednio, przez interfejs menadżera: login oraz nazwa hosta,
- ustawione dotąd ciasteczka.

Za generowanie całej reszty danych byłby odpowiedzialny klient HTTP na karcie. Eliminowałoby to konieczność przesyłania danych typu Cache-Control, User-Agent, a także powodowałoby, że aplikacja kliencka nie musi znać długości hasła.

Przy przetwarzaniu odpowiedzi natomiast kluczowe jest znalezienie wszystkich nagłówków Set-Cookie, tak więc istniałaby konieczność przetworzenia jedynie nagłówków, treść odpowiedzi mogłaby być bezpiecznie

zignorowana. Niemożliwe jest jednak wyeliminowanie konieczności przesyłu dużych bloków danych do karty, gdyż dane te są zaszyfrowane.

Takie podejście umożliwiłoby maksymalne zoptymalizowanie ilości danych wejściowych dla apletu na karcie przy tworzeniu żądania. Nie miałoby to przy tym wpływu na ilość danych wyjściowych żądania, a także na ilość danych przesyłanych przy obsłudze odpowiedzi.

Wadą jest jednak zaprzęgnięcie karty do pracy, której wykonanie można bez problemu oddelegować do warstw wyższych np. klienta na komputerze użytkownika. Może to też spowodować, że przekroczone zostaną ilości dostępnych dla apletu zasobów, w szczególności pamięci RAM. Kolejne trudności mogłyby być wygenerowane, jeśli okazałoby się, że nagłówek nie mieści się w 256 bajtach. Oznaczałoby to, że należy zapamiętać stan generatora, poinformować klienta, że praca nie została zakończona i oczekiwać na następne żądanie.

8.4.2 Wariant 2 – nagłówki dostarczone z zewnątrz

Drugą możliwością byłaby implementacja po stronie karty jedynie minimum niezbędnego do wstawienia hasła we właściwe miejsce. Żądanie HTTP przychodziłoby z zewnątrz. Program na karcie nie musiałby rozumieć jego formatu. Jego jedynym zadaniem, przed zaszyfrowaniem danych, zgodnie z protokołem TLS, byłoby odnalezienie ustalonego znacznika i podmiana danych na prawdziwe hasło. Ponadto, także aplikacja na komputerze użytkownika nie musi zajmować się generowaniem nagłówków, a jedynie je otrzymać.

Jeżeli chodzi zaś o odbiór odpowiedzi, karta powinna już być do pewnego stopnia świadoma formatu nagłówków, ponieważ jej zadaniem jest dostarczenie wartości ciasteczka gdy logowanie powoduje otwarcie sesji. Musi więc zostać zaimplementowana prosta maszyna stanu, parsująca kolejne bajty nagłówka i poszukająca linii zaczynającej się od wartości Set-Cookie. Tylko dane poprzedzone tym kluczem powinny zostać zwrócone. Oczywiście istnieje jeszcze opcja przesłania całej odpowiedzi w formie odszyfrowanej.

Zaletą tego podejścia jest ograniczenie potrzebnych zasobów na karcie do minimum. Stan pomiędzy kolejnymi 256-bajtowymi blokami nie

musi być przechowywany. Jedyną informacją, jaka musi zostać zachowana, jest wartość ciasteczka. Także aplikacja kliencka jest dużo prostsza dzięki takiemu podejściu.

Odbywa się to jednak kosztem ilości danych, jakie muszą zostać przesłane do karty, ponieważ w tym przypadku karta musi dostać pełny nagłówek HTTP. Kolejną wadą jest też konieczność znajomości długości hasła przez klienta, co do pewnego stopnia ułatwia jego potencjalne złamanie. Z drugiej strony nie zdradza żadnych innych informacji na jego temat, takich jak zestaw znaków użytych do jego wygenerowania lub wartości funkcji SHA hasła. Dlatego też dla haseł o odpowiednio dużej długości, nie stanowi to problemu, gdyż jedyną metodą jego odzyskania byłby atak siłowy na stronę logowania.

Implementacja tak prostej aplikacji klienckiej powoduje jednak duże utrudnienie dla potencjalnych jej użytkowników. Muszą oni dostarczyć prawidłowe nagłówki HTTP. Przez to zadania aplikacji ograniczają się do wystawienie prostego interfejsu dla kolejnej warstwy, która potencjalnie powinna być zintegrowana z przeglądarką użytkownika dla łatwości obsługi.

8.4.3 Wariant 3 - nagłówki generowane przez aplikację kliencką

Rozwiązaniem pośrednim w stosunku do poprzednio zaproponowanych jest generowanie nagłówków po stronie aplikacji klienckiej na komputerze użytkownika. Współczesne komputery klasy PC, i urządzenia mobilne dysponują nieograniczoną mocą obliczeniową. Prosta implementacja HTTP po stronie karty i przeniesienie ciężaru obliczeń na klienta oszczędza sporo zasobów.

Efektem zastosowania tego podejścia powinna być wtyczka do przeglądarki, komunikująca się z apletem na karcie pośrednio przez usługę systemową taką jak gniazdo sieciowe lub komunikację bezpośrednią.

Zaletą tego rozwiązania jest dostarczenie pełnego, gotowego do użytku systemu.

Pewną wadą jest natomiast stopień skomplikowania całości. Najbardziej prawdopodobnym efektem byłaby konieczność opracowania trzech elementów systemu:

- aplet JavaCard,
- wtyczka do przeglądarki,
- usługa systemowa pośrednicząca między oboma elementami.

Ponadto wariant ten komplikuje interfejs usługi pośredniej, co może negatywnie wpłynąć na rozszerzalność całości.

8.4.4 Podsumowanie

Z powyższych rozważań wynika, że najmniej wad posiada wariant 3. Jednak posiada on skomplikowany interfejs wejściowy. W porównaniu z nim wariant 2 wymaga jedynie dostarczenia nagłówka oraz nazwy domenowej serwisu docelowego i loginu wybranego użytkownika. Wariant 1 nie jest wskazany ze względu na ograniczenia sprzętowe karty chipowej. Okazuje się, że wariant 2 jest najbardziej optymalny. Istnieje tylko konieczność implementacji wtyczki do przeglądarki (zaproponowanej w rozdziale 8.4.3).

8.5 Interfejs użytkownika

Jedną z najważniejszych kwestii, wpływających na przydatność całego systemu jest interfejs użytkownika. Musi on być na tyle prosty w obsłudze, aby przeciętny użytkownik był w stanie sobie z nim poradzić. Z drugiej strony, system nie może być na tyle zamknięty, aby niemożliwy był rozwój. Interfejsy powinny umożliwiać wejścia do różnych przeglądarek, menadżerów kont wbudowanych w systemy mobilne. Do potwierdzania tożsamości również niezależni deweloperzy powinni mieć możliwość użycia systemu w swoich aplikacjach.

Z tych powodów, najbardziej optymalną docelową architekturą jest opracowanie, oprócz apletu na karcie, pewnego rodzaju usługi udostępniającej interfejs, z którego mogą korzystać inne aplikacje oraz wtyczki dla dowolnej przeglądarki WWW.

Niestety po analizie interfejsów najpopularniejszych przeglądarek okazuje się, że nie udostępniają one funkcji komunikacji z warstwą systemową, a jedynie umożliwiają zastosowanie usług sieciowych. Dlatego też jedynym wyjściem jest zbudowanie lokalnej usługi sieciowej. Ponieważ jednak założeniem projektu jest rozwiązanie typu *Proof of Concept*, która jest prototypem, implementacja wtyczki została pominięta.

Docelowym interfejsem w systemie jest aplikacja konsolowa, działająca zgodnie z normami przyjętymi w systemach uniksowych, a więc

udostępniająca interfejs poprzez funkcję getopt. Pozwala to każdemu potencjalnemu implementującemu jej obsługę i na dostosowanie się do niego niezależnie od wymagań konkretnej implementacji.

Wszystkie parametry są przekazane za pomocą odpowiednich przełączników. Same nagłówki trafiają, jako strumień podany, na standardowe wejście aplikacji. Nagłówki te zawierają maskowane pole o odpowiedniej długości, w które jest wpisywane hasło tuż przed zaszyfrowaniem danych przez aplet.

8.6 Etapy implementacji

Implementacja całego systemu składa się z pięciu etapów. Pierwsze cztery dotyczą implementacji apletu po stronie karty chipowej. Ostatni etap systemu to udostępnienie interfejsu dla użytkowników, którzy chcą użyć system we własnych rozwiązaniach.

8.6.1 Biblioteka kryptograficzna

Pierwszą rzeczą, jaką wykonano jest budowa działającej biblioteki kryptograficznej dostosowanej do użycia w projektowanym systemie. Duża część stanowi jedynie obudowanie istniejących klas w JavaCard. Należy jednak dostosować algorytmy takie jak AES do paddingu używanego w protokole TLS. Dodatkowo należy użyć implementacji funkcji skrótu dostępnych w API do stworzenia brakującej implementacji algorytmów HMAC-SHA-1 oraz HMAC-SHA-256. Ostatnią istotną implementacją jest specyficzna dla TLS funkcja PRF (*PseudoRandom Function*). Bazuje ona na HMAC.

W ramach pracy rozwiązano problem niedostatecznej ilości pamięci operacyjnej. Z tego powodu, powstała obsługa dynamicznego alokatora pamięci. Wchodzi ona de facto w skład biblioteki kryptograficznej, ponieważ część klas jest mocno od tego alokatora zależała.

Efektem tego etapu jest również tymczasowe API zgodne z ISO 7816 dla celów testowych.

8.6.2 Implementacja TLS

Drugim etapem jest implementacja elementów protokołu TLS po stronie karty. Celem jest budowa gotowego fragmentu docelowego API bazującego na ISO 7816, która umożliwia nawiązanie połączenia (handshake TLS) oraz przesłanie danych przez pakiet ApplicationData.

Etap ten wymagał użycia opracowanej w pierwszym etapie biblioteki kryptograficznej. Po jego zakończeniu możliwa jest bezpieczna wymiana danych z użyciem protokołu TLS, uniemożliwiająca podsłuchanie transmisji.

Na tym etapie, obsługa pakietów jest wciąż niezależna od leżącego poniżej protokołu, którym w przypadku projektowanego systemu jest HTTP. Klasa będąca jego efektem traktuje dane otrzymane w pakiecie ApplicationData jako strumień. Drugim dostarczanym interfejsem jest interfejs umożliwiający klientowi HTTP podpięcie się pod ten strumień i ewentualne przekazanie go do kolejnej warstwy.

8.6.3 Prosty klient HTTP

Zadaniem klienta HTTP jest w przypadku żądania HTTP, przekazanie go do menadżera haseł, natomiast dla odpowiedzi, musi przeparsować nagłówki i wyciągnąć zawartość nagłówka Set-Cookie.

8.6.4 Menadżer haseł

Ostatnim etapem przetwarzania po stronie karty jest sam menadżer haseł. Jego zadaniem jest budowa bezpiecznego kontenera na hasła. Hasło jest przekazywane poprzez wklejenie go w odpowiednie maskowane pole w nagłówku żądania HTTP, otrzymane od klienta HTTP. Ponieważ główny ciężar prac programistycznych projektu stanowi implementacja mechanizmów kryptograficznych związanych z protokołem TLS, implementacja samego menadżera jest stosunkowo prosta. Ma on za zadanie, najpierw wygenerować hasło zgodne z wymaganiami użytkownika, następnie wygenerować nagłówek odpowiedzialny za zarejestrowanie się w serwisie lub zmianę hasła na nowe. Dlatego też liczba dostępnych masek w jednym żądaniu musi wynosić minimum 2. Po dokonaniu rejestracji z kolei, ma umożliwić wstawienie już tylko jednego hasła w maskowane pole.

Ze względu na to, że system ma charakter PoC, możliwe jest zapamiętanie jedynie jednego hasła dla każdego z serwisów.

8.6.5 Aplikacja konsolowa

Zadaniem aplikacji konsolowej jest implementacja interfejsu użytkownika. Obudowuje ona też API wystawione przez protokół ISO 7816, którego implementacja jest wysoce złożona. Przy okazji implementacja ta będzie stanowić pewnego rodzaju dokumentację samego protokołu. Ważne jest więc, aby jej kod był jak najbardziej przejrzysty.

Środowiskiem, w którym zostanie zaimplementowana ta aplikacja jest język Python, co pozwala na opracowanie łatwego do zrozumienia kodu, pozbawionego zbędnego narzutu.

9. Implementacja

W rozdziale zostały omówione wybrane szczegóły implementacji, które mają wpływ na funkcjonowanie systemu a szczególnie na jego ograniczenia.

9.1 Aplikacja kliencka

Zadaniami aplikacji klienckiej jest dostarczenie konsolowego interfejsu użytkownika oraz dokumentacja opracowanego interfejsu komunikacyjnego według standardu ISO 7816. Aplikacja została zrealizowana w języku Python. Do zrealizowania założeń konieczne było zastosowanie zewnętrznych bibliotek do:

- obsługi połączenia z kartą chipową po protokole ISO 7816,
- parsowania certyfikatów zserializowanych do formatu ASN.1,
- wygenerowania klas odpowiedzialnych za deserializację struktur certyfikatów.

W tym celu użyte zostały następujące biblioteki: asn1ate (tylko etap kompilacji), pyasn1 oraz pyscard. Wszystkie trzy elementy mogą zostać zainstalowane za pośrednictwem narzędzia pip.

9.1.1 Implementacja TLS

Aplikacja TLS stanowi częściową implementację protokołu TLS. Została ona zaprojektowana w sposób pozwalający na dalszy rozwój. Wszystkie klasy wchodzące w skład pakietu powinny implementować następujące metody:

- init (self, <wszystkie dostępne pola>) – pozwala na opracowanie nowego pakietu o pożądanej zawartości, przeznaczonego do wysłania,
- bytes – pozwala na serializację obiektu do gotowego pakietu TLS,
- str – pozwala na graficzną reprezentację pakietu w formacie pole = wartość, pole2 = wartość2,
- from bytes – pozwala na deserializację pakietu TLS do obiektu docelowej klasy, umożliwiającej dostęp do pól struktury.

Pominięta została natomiast obsługa zawartości klas zapewniających szyfrowanie przechowywanych danych. Powodem jest brak dostępu ze strony aplikacji do kluczy odszyfrowujących transmisję.

Rys. 20. Użyte urządzenia – karta JCOP J2A040 oraz czytnik HID Omnikey 3021 (źródło: własne)

9.1.2 Implementacja API komunikacji z kartą chipową

Drugim istotnym elementem aplikacji jest implementacja klas odpowiedzialnych za obsługę interfejsu komunikacyjnego karty chipowej. Centralną klasą tej implementacji jest klasa pdu, odpowiedzialna za tworzenie APDU, przesyłanych później do karty. Jej architektura umożliwia podanie oddzielnie wszystkich pól PDU (takich jak *CLA* czy P_1) . Udostępnia też, analogicznie do klas implementujących struktury protokołu TLS, metodę bytes, służącą do otrzymywania gotowego do przesłania pakietu. Wszystkie implementacje poszczególnych PDU dziedziczą po tej klasie, dzięki czemu najczęściej wymagana jest jedynie implementacja własnego konstruktora, opakowującego konstruktor klasy pdu, co upraszcza implementację.

9.2 Interfejs komunikacji z kartą chipową

Interfejs komunikacji z kartą chipową jest rozwiązany zgodnie ze standardem ISO 7816. Definiuje on następujące pola, które są do dyspozycji użytkownika: *CLA*, *INS*, P_1, P_2, L_c, *Data* oraz L_e dla żądania. L_c oraz L_e

to długości danych odpowiednio wejściowych i wyjściowych. Odpowiedź zawiera następujące pola: L_e, *Data*, SW_1 oraz SW_2.

Wszystkie zaprojektowane funkcje mają wartość pola *CLA* ustaloną na 0x80. Wartość *INS* służy za oznaczenie żądanej funkcji. Pole *Data*, zarówno w przypadku wejścia jak i wyjścia, upakowuje wszystkie przesyłane dane. Nie ma żadnych oznaczeń, gdzie występuje granica, w przypadku, gdy przesyłany jest więcej niż jeden parametr. W celu przesłania długości parametrów, używane są pola P_1 oraz P_2. Jeżeli funkcja wymaga dwóch parametrów wejściowych, poza polem L_c wypełniane jest pole P_1, przechowujące długość pierwszego z parametrów. Długość drugiego można prosto obliczyć według równania $L = L_c - P_1$. W przypadku trzech parametrów dodatkowo używane jest pole P_2. Maksymalna ilość parametrów wynosi 3. Dodatkowym wariantem funkcji jest przypadek, gdy zachodzi konieczność przesłania więcej niż 254 bajtów do karty lub 255 z karty. W przypadku przesłania do karty, pola P_1 oraz P_2, zamiast służyć jako długości, przechowują dwa pierwsze bajty danych, gdzie P_1 = *Parametr*[1], a P_2 = *Parametr*[0], a pole *Data* = *Parametr*[2 :]. W przypadku dla odpowiedzi z karty została wykorzystana wartość pola SW_2, która w przypadku poprawnego wykonania funkcji, normalnie zawsze zawiera wartość 0x00. W omawianym wariancie komendy przechowuje ostatni bajt odpowiedzi. Dzięki temu rozwiązaniu możliwe jest przesłanie 256 bajtów danych, niezależnie od kierunku przesyłu. Jest to szczególnie przydatne przy przesyłaniu modulusa klucza publicznego, który często zawiera właśnie 256 bajtów.

Ponadto funkcje dzielą się na te dotyczące obsługi protokołu TLS oraz te dotyczące menadżera haseł. Pierwsze mają identyfikatory z zakresu 0x50 - 0x5f, drugie natomiast 0x60 - 0x6f. Wyjątkowe są tu funkcje dotyczące przesyłania danych pakietu ApplicationData (id 0x58 - 0x5b), które trafiają zarówno do części TLS, jak i menadżera. Zostały one jednak zaalokowane do części TLS, gdyż stanowią zewnętrzną warstwę.

9.3 Zarządzanie pamięcią

Jednym z najpoważniejszych problemów, jakie należało rozwiązać w trakcie etapu implementacji apletu JavaCard było ograniczenie zużycia pamięci operacyjnej. Ponieważ żywotność pamięci nieulotnej jest w przy-

padku kart chipowych mocno ograniczona, główną pamięcią musi być pamięć operacyjna. Docelowe urządzenie dysponowało ok. 2 KiB pamięci operacyjnej dostępnej dla użytkownika oraz 36 KiB pamięci EEPROM. Ponieważ brak jest w wbudowanych mechanizmach zarządzanie pamięcią operacyjną, w aplikacji taki mechanizm został zbudowany od podstaw jako klasy alokatora pamięci. Przy instalacji apletu alokuje odpowiednio duży blok pamięci. Doświadczalnie, wielkość wymagana do stabilnego działania aplikacji została ustalona na 1KiB, co daje niecały kilobajt innym klasom, które nie mogłyby zwalniać zajętej przez siebie pamięci.

Interfejsem klasy zarządzającej, nazwanej TemporaryBuffer, są dwie metody: allocate i deallocate, symulujące zachowanie metod malloc oraz free, znanych z biblioteki standardowej C. Wartością zwracaną przez pierwszą z funkcji jest przesunięcie (*offset*) w prealokowanym buforze. Wywołujący musi więc każdorazowo wyciągnąć z klasy referencję do bufora.

Wewnętrznie implementacja zawiera dwie tablice. Pierwsza stanowi prealokowany bufor. Druga opisuje zaalokowane obszary pamięci. Pozwala ona na jednoczesne zaalokowanie do 8 różnych obiektów. Opis każdego z nich składa się z dwóch bajtów offsetu i długości, zapisanych w zmiennych typu short. Całość potrzebuje 32 bajtów. Opisy poszczególnych obiektów są przechowywane w kolejnych wpisach, bez zostawiania pustych wpisów pomiędzy nimi, w kolejności od tego o najniższym offsecie, do najwyższego, co ułatwia szybkie znalezienie pierwszego wolnego obszaru pamięci. W momencie dokonywania alokacji istnieje konieczność przesunięcia wszystkich wpisów z offsetem wyższym niż alokowany w prawo.

Takie rozwiązanie interfejsu zewnętrznego jak i wewnętrznej architektury posiada jedną istotną wadę, którą jest – fragmentacja pamięci. W skrajnym przypadku może to spowodować niemożliwość zaalokowania bloku pamięci o pewnej długości, pomimo iż fizycznie taka ilość pamięci jest dostępna, lecz znajduje się w kliku miejscach. Jest to potencjalnie jedno z miejsc do poprawienia, w przypadku dalszego rozwoju systemu. Rozwiązaniem mogłoby być przekazywanie zamiast offsetu w pamięci, numeru wpisu w tablicy alokacji.

Drugą, wadą jest brak ochrony pamięci pomiędzy poszczególnymi klasami alokującymi, co stwarza niebezpieczeństwo nadpisania pamięci innej klasy. Ponieważ jednak aplet stanowi integralną całość i niemożliwe jest dostanie się do zawartości tymczasowej pamięci z zewnątrz poprzez zainstalowanie drugiego apletu, problem ten nie należy do istotnych. Aby jednak w miarę możliwości ograniczyć skutki potencjalnych błędów, pamięć ta przy zwalnianiu jest czyszczona. Dzięki temu wszelkie wrażliwe dane powinny z niej zniknąć najpóźniej w chwili zakończenia przetwarzania APDU, chyba że z jakiegoś powodu operacja zwolnienia zasobów nie odbyłaby się.

9.4 Implementacja kryptografii protokołu TLS

Rozdzielenie implementacji TLS na dwa urządzenia spowodowało powstanie kilku trudnych do rozwiązania problemów.

Pierwszym z nich jest poprawne generowanie i weryfikacja pola verify data, występującego w komunikacie Finished, przesyłanym przez obie strony komunikacji. W przypadku generowania tego pakietu, rozwiązaniem jest odebranie skrótu wszystkich wiadomości wymienionych między stronami od aplikacji klienckiej. W przeciwną stronę, gdy na karcie ciąży zadanie weryfikacji tego pola, rozwiązaniem tymczasowym jest zaniechanie tej weryfikacji. Skutki takiego podejścia powinny być zbliżone co przeniesienie obowiązku wygenerowania skrótu na aplikację kliencką w poprzednim przypadku. W ostatecznym rozwiązaniu natomiast należy wykonać tą weryfikację. Teoretycznie istnieje możliwość jej dokonania poprzez użycie udostępnianej przez API JavaCard funkcji pozwalającej na zainicjalizowanie funkcji skrótu, wartością części wiadomości, a następnie dodanie skrótu komunikatu ClientFinished i ewentualnie wszystkich komunikatów wysłanych w międzyczasie przez jedną ze stron. Należy jednak zacząć od poznania zasady działania tej klasy, gdyż próba jej użycia zakończyła się niepowodzeniem.

Kolejnym ograniczeniem obecnej implementacji jest długość danych przesłanych protokołem HTTP, a więc opakowanych przez Application-Data od strony aplikacji klienckiej do serwera, a więc nagłówka żądania HTTP. Ponieważ obecnie nie ma możliwości zapamiętania stanu funkcji HMAC danych, w przypadku, gdy miałyby one rozdzielone na dwa PDU,

aby otrzymać faktyczną maksymalną długość danych, należy odjąć od maksymalnej długości PDU wszystkie struktury towarzyszące danym.

Podobne ograniczenie istnieje również w danych przesyłanych z serwera do apletu. Każde PDU musi zawierać IV aktualnie przetwarzanego pakietu. Maksymalna długość jednorazowo przetworzonych danych to więc 240 bajtów. W tym przypadku istnieje możliwość rozwiązania tego problemu w ten sposób, że IV byłby zapamiętany w pamięci operacyjnej przez tymczasowy bufor lub w pamięci EEPROM.

9.5 Parser HTTP

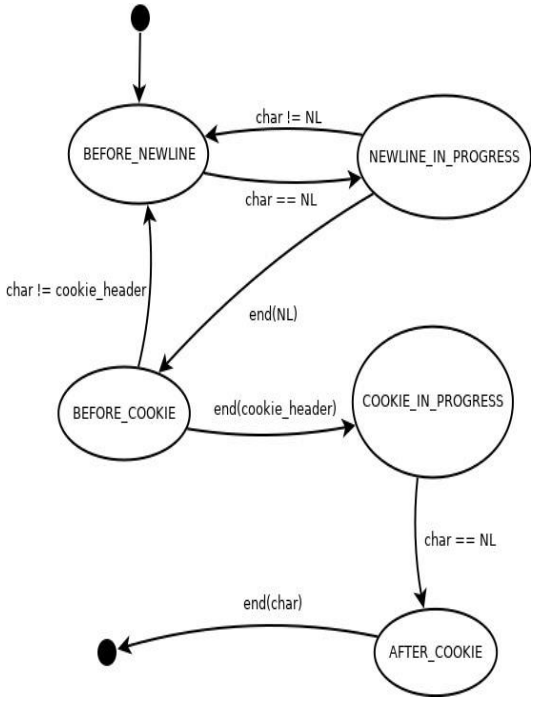
Ostatnim istotnym elementem implementacji jest parser HTTP. Jego zadaniem jest odnalezienie w otrzymanej odpowiedzi ciasteczka sesyjnego i zwrócenie go użytkownikowi, aby mógł go zastosować w swojej aplikacji lub przeglądarce WWW.

Sam parser działa na zasadzie maszyny stanów. Przejścia pomiędzy kolejnymi stanami opisuje rysunek 21. Parser przechodzi przez wszystkie otrzymane bajty odpowiedzi. W momencie dojścia do ostatniego dostępnego bajtu, jego stan jest zapamiętywany, tak aby mógł on kontynuować przy przetwarzaniu przez aplet kolejnego PDU. Pierwszym stanem jaki zostaje ustawiony w momencie otrzymania pierwszej części pakietu HTTP, jest stan oznaczony na schemacie jako BEFORE NEWLINE. Ma to na celu przejście przez pierwszą linię nagłówka, oznaczającą kod błędu HTTP. Jej zawartość zostanie więc zignorowana. Zmiana stanu może wystąpić jedynie, gdy aktualnie przetwarzany bajt będzie pierwszym bajtem znaku nowej linii, który z przypadku HTTP jest dwubajtowy, o wartości 0x0D 0x0A. W tym momencie stan zmienia się na NEWLINE IN PROGRESS.

Stan NEWLINE IN PROGRESS ma na celu sprawdzenie czy kolejny bajt ma wartość 0x0A. Jeżeli tak nie jest, następuje powrót do stanu BEFORE NEWLINE. Jeżeli tak jest, cały ciąg znaku nowej linii zostaje przetworzony i stan zmienia się na BEFORE COOKIE.

Stan BEFORE COOKIE ma na celu sprawdzenie nazwy nagłówka HTTP i porównanie jej z ciągiem ”Set-Cookie: ”. Będzie on więc trwał tak długo aż kolejne bajty nagłówka będą pasować do bajtów tego ciągu. W sytuacji, gdy któryś z bajtów będzie się różnił, następuje powrót do stanu BEFORE NEWLINE, a więc szukany jest koniec obecnego nagłówka,

czyli znak nowej linii. Gdy cały ciąg pasuje do wzorca, następuje zmiana stanu na stan COOKIE IN PROGRESS.



Rys. 21. Diagram stanów parsera HTTP (źródło: własne)

Celem stanu COOKIE IN PROGRESS jest zapamiętanie wartości znalezionego ciasteczka. Zapamiętane zostanie wszystko co występuje po znaku dwukropka i spacji, aż do znaku nowej linii. W momencie napotkania znaku nowej linii, następuje ostatnia w cyklu zmiana stanu, na stan AFTER COOKIE.

Stan AFTER COOKIE jest stanem, który nie wykonuje żadnych działań. Ma on na celu uniemożliwienie powrotu na początek cyklu po znalezieniu ciasteczka i w konsekwencji nadpisania ciasteczka przez kolejne znalezione ciasteczko, umożliwiając klientowi odebranie zapamiętanego ciasteczka.

Podstawową wadą tak zaimplementowanej maszyny stanów jest niemożliwość zapamiętania więcej niż jednego ciasteczka. Stanowi to ograniczenie dla bardziej rozbudowanych serwisów, które mogą potrzebować ustawienia wielu ciasteczek w momencie utworzenia sesji.

10. Analiza systemu

10.1 Rozwiązania projektowe – zagrożenia

Decyzja o implementacji jedynie części protokołu TLS po stronie karty chipowej miała pewne negatywne konsekwencje. Najpoważniejsze problemy objawiły się przy obsłudze komunikatów Finished.

Zaproponowanym rozwiązaniem było przysłanie skrótu wszystkich wymienionych komunikatów z aplikacji klienckiej dla komunikatu kończącego handshake klienta oraz zignorowanie weryfikacji komunikatu przesłanego przez serwer.

Generalnie, ponieważ wszystkie przesłane wcześniej dane są widoczne dla obserwatorów w postaci niezaszyfrowanej (z wyjątkiem pre master secret, szyfrowanego asymentrycznie), nie zawierają one żadnych danych wrażliwych. Najważniejsze z zawartych w nich parametrów to:

- wersja protokołu TLS,
- zestawy algorytmów szyfrujących obsługiwane przez klienta,
- algorytmy kompresji (obecnie nieużywane),
- rozszerzenia protokołu,
- zestaw algorytmów wybrany przez serwer.

Jednym z najpoważniejszym elementem do ataku jest zestaw algorytmów szyfrujących. Jego modyfikacja mogłaby w teorii umożliwić wynegocjowanie przez strony braku szyfrowania (teoretycznie wspierane przez protokół) lub wystarczająco słabego algorytmu, np. uniemożliwiającego weryfikację klucza publicznego (według dokumentacji TLS algorytmy oparte o anonimową wymianę DiffiegoHellmana). Program zaimplementowany w ramach tej pracy zakładał implementację jedynie podstawowego zestawu szyfrowania zdefiniowanego przez standard jako obowiązkowy. Wymuszenie zastosowania innego algorytmu przez atakującego spowodowałoby niemożliwość zestawienia poprawnego połączenia, gdyż aplet menadżera nie zakłada przesłania wybranego algorytmu, zamiast tego zawsze stosuje ten sam zestaw.

Drugim potencjalnym elementem ataku mogłoby być rozszerzenia protokołu. W założeniach projektu systemu nie zakładano implementacji rozszerzeń więc pole na nie powinno być puste.

Należałoby jednak przeprowadzić badania sprawdzające, jaki wpływ mają możliwe do użycia rozszerzenia, które nie powodują zmiany zachowania samego mechanizmu ustanawiania połączenia i wymiany danych. Szczególnie ważne jest zbadanie wpływu rozszerzenia server name, które sygnalizuje serwerowi z jaką witryną inicjujący połączenie chce się połączyć. Potencjalnie pole to mogłoby być również w przyszłości zastosowane do sprawdzania wartości pola Common Name w certyfikacie, co otworzyłoby pole do ataków podmiany certyfikatów.

Ostatnim ważnym polem jest wersja protokołu TLS. Pole to w przeszłości było już stosowane do ataków polegających na wymuszeniu na stronach zejścia do niższej wersji protokołu. Był to najprawdopodobniej jeden z powodów wprowadzenia pola verify data. W teorii atakujący mógłby zmusić obie strony do wynegocjowania komunikacji po protokole SSL 2.0, który jest uznany za złamany i nie powinien być stosowany. Opisywany projekt zakładał implementację wyłącznie protokołu w wersji 1.2, co uniemożliwia ataki downgradujące wersję protokołu. Nawet gdyby udało się przejąć komputer ofiary, to aplet na karcie zawsze generuje wartości HMAC na końcu wiadomości i weryfikuje tą wartość w przypadku komunikatu Finished od serwera, co doprowadziłoby do zerwania połączenia w tym momencie.

Warto również odnotować, że wszelkie ataki próbujące manipulować polami komunikatów TLS musiałyby zakładać pełną kontrolę nad komputerem użytkownika, gdyż to aplikacja działająca na nim odpowiada za wygenerowanie skrótu wszystkich komunikatów, przesłanych później apletowi w celu wygenerowania komunikatu Finished.

Aplikacja opracowana w ramach tej pracy nie powinna pozwolić na jakikolwiek atak, polegający na manipulacji polami pakietów. Nawet gdyby okazało się jednak, że któreś pole może mieć wpływ na bezpieczeństwo protokołu, to warunkiem koniecznym jest pełna kontrola nad komputerem użytkownika.

Należy zauważyć, że w tak skrajnym przypadku istnieje jednak wciąż jedna niezamknięta luka, która dotyczy weryfikacji certyfikatu X.509. Obecnie do karty chipowej przesyłane są jedynie wartości modulus oraz eksponent, co uniemożliwia weryfikację autentyczności całego certyfikatu po stronie karty. Ten problem powoduje, że istnieje możliwość przeprowadzenia ataku man-in-the-middle, polegającego na podstawieniu certyfikatu

serwisu. Atak wygląda w ten sposób, że atakujący staje pomiędzy stronami komunikacji tak, że użytkownik komunikuje się z atakującym nieświadomie zestawiając połączenie TLS z nim, zamiast autentycznego serwera. Atakujący zestawia kolejne połączenie TLS z serwerem. Przy czym użytkownik dostaje certyfikat, którego klucz prywatny zna atakujący. Jedynym sposobem na likwidację tej luki jest przeprowadzenie pełnej weryfikacji certyfikatu po stronie karty chipowej, wraz z weryfikacją poprawności całego łańcucha certyfikacji i porównanie głównego CA z listą znanych centrów certyfikacji, na którą nie ma wpływu użytkownik.

10.2 Wydajność systemu

10.2.1 Czas całego procesu

Do zmierzenia czasu potrzebnego na przeprowadzenie procesu logowania użyte zostały dwa serwisy. Pierwszy z nich (login.localdomain). Jest to specjalnie opracowany do tego celu skrypt PHP, uruchamiany na lokalnej instancji httpd. Celem jego powstania było zbudowanie serwisu wymieniającego jak najmniejsze ilości danych. Zmierzony czas można traktować jako minimalny czas jaki musi zająć proces logowania. Większość tego procesu zajmie nawiązywanie połączenia TLS, w tym generowanie kluczy kryptograficznych.

Od strony interfejsu, przy pierwszym odwiedzeniu serwisu wyświetlana jest informacja o tym, w jaki sposób należy się do niego zalogować. Polega to na przesłaniu parametrów login i pass, przy pomocy metody POST. Dla celów demonstracji każda wartość tych parametrów jest uznawana za prawidłową i login jest później wyświetlany, gdy użytkownik poda właściwy identyfikator sesji. Rejestracja jest symulowana przez dodanie dodatkowego parametru do żądania POST. Skrypt nie wykorzystuje tego parametru w żaden sposób. W momencie przekazania parametrów przez metodę POST, tworzona jest nowa sesja i użytkownikowi zwracany jest nagłówek Set-Cookie, zawierający ciasteczko login, z aktualną wartością loginu, dodatkowo zapamiętaną wewnątrz sesji. Każde kolejne odwołanie do strony przy użyciu tak opracowanej sesji pokaże wartość loginu zapamiętanego w sesji.

Drugim z użytych serwisów jest aplikacja phpMyAdmin, hostowana na zewnętrznym serwerze (s17.hekko.pl), a czas dostępu do niej zależy również od połączenia z Internetem. Ten przypadek jest znacznie bardziej

skomplikowany, gdyż wymaga ustawienia ciasteczka sesyjnego w momencie wysłania żądania logowania, a ponadto także specjalnego tokenu, wysyłanego przez żądanie POST. Ponieważ wybrany serwer nie umożliwia tworzenia nowych kont przez własny interfejs, proces rejestracji zostaje zasymulowany przy użyciu mechanizmu zmiany hasła. W tym przypadku, żądanie komplikuje się jeszcze bardziej, ponieważ wymagane jest ustawienie dwóch kolejnych ciasteczek, autoryzujących żądanie. Łącznie długość żądań wynosi ponad 600 bajtów w przypadku logowania i ponad 800 dla rejestracji. Dokładne długości są zależne od długości wspomnianych ciasteczek. Oznacza to, że przekroczona jest maksymalna długość pojedynczego pakietu ApplicationData, jaki może zostać przesłany do karty chipowej, a więc żądania te zostaną rozdzielone na klika fragmentów.

Stopień skomplikowania tego testu powinien dość realistycznie odzwierciedlić wymagania prawdziwych stron internetowych, a zatem stanowi jednocześnie test przydatności całego systemu do działania. Czas potrzebny na realizację procesu w tym przypadku będzie też przez to ważniejszy w ocenie niż czas osiągnięty w pierwszym badanym przypadku.

Oba testowe serwery miały wystawiony certyfikat SSL z kluczem długości 2048 bitów.

Dla porównania zarówno rejestracja jak i logowanie do obu serwisów zostały przeprowadzone przy pomocy zbudowanego systemu i oprogramowania openssl (polecenie s client).

W przypadku testów aplikacji phpMyAdmin, użyte zostały skrypty bashowe, automatyzujące cały proces (pma.login.pwdman.sh oraz pma.register.pwdman.sh dla badanego systemu oraz działające identycznie pma.login.sh oraz pma.register.sh dla openssl). Poniższa tabela przedstawia wyniki przeprowadzonych badań. Czasy logowania zostały zmierzone przy użyciu linuksowego polecenia time (wartość *real*). Zbudowany system został oznaczony jako pwdman. Pozostałe wyniki pochodzą z narzędzia openssl.

Tabela 3. Rzeczywiste czasy logowania i rejestracji do wybranych serwisów

Serwis	**Czas rejestracji**		**Czas logowania**	
	openssl	**pwdman**	**openssl**	**pwdman**
login.localdomain	0m0,029s	0m4,049s	0m0,025s	0m4,197s
s17.hekko.pl	0m0,249s	0m6,624s	0m0,382s	0m8,241s

Jak można zauważyć czas potrzebny na uzyskanie ciasteczka i tym samym zalogowanie się do serwisu wynosi minimalnie 4 sekundy, wobec czasów nieprzekraczających sekundy przy użyciu openssl. Dla przykładu rzeczywistego serwisu wyniósł natomiast nieco ponad 8 sekund. Można wnioskować, że czas maksymalny nie powinien przekraczać kilkunastu sekund dla najbardziej skomplikowanych przypadków.

10.2.2 Szybkość szyfrowania RSA

Jednym z czynników, które mogą mieć wpływ na czas procesu jest szybkość szyfrowania RSA. Opracowany system wspiera klucze szyfrujące o długości 512, 1024 lub 2048 bitów. Długość szyfrowanych danych nie powinna mieć większego wpływu, gdyż konieczne jest dopełnienie danych paddingiem do długości klucza. Teoretycznie więc szyfrowanie krótszych danych powinno być minimalnie wolniejsze, z powodu konieczności dogenerowania losowych danych jako wypełnienia. Długość danych użytych w teście wynosiła 16 bajtów.

Czas szyfrowania był mierzony przy użyciu narzędzia time (wartość *real*).

Tabela 4. Wydajność szyfrowania algorytmem RSA

Ilość bitów klucza	Czas szyfrowania
512	0m0,231s
1024	0m0,460s
2048	0m1,535s

Jak widać czas potrzebny do zaszyfrowania testowych danych może wynieść do półtorej sekundy. Stanowi to więc prawie 40 procent czasu potrzebnego do przeprowadzenia logowania do testowego serwera.

10.2.3 Szybkość obliczania wartości HMAC

Obliczanie wartości funkcji HMAC jest najczęściej wykonywaną operacją kryptograficzną. Funkcja ta stanowi podstawę funkcji PRF, przy obliczaniu której jej wartość jest liczona więcej razy, w zależności od długości bloku zwracanego przez PRF.

Badanie szybkości obliczania wartości HAMA polegało na zmierzeniu czasu obliczania wartości HMAC dla testów zdefiniowanych w [27] oraz [28] (niektóre zostały pominięte ze względu na długość klucza bądź danych niemieszczących się w pojedynczym APDU). Dla celów zbadania przybliżonego narzutu związanego z implementacją brakujących części protokołu, zbadany został czas potrzebny na obliczenie sumy HMAC danych o długości 3.5 KiB, które mają symulować przybliżoną długość certyfikatu przesyłanego przez serwer.

Badanie zostało przeprowadzone przy użyciu narzędzia time i wartości *real*. Dodatkowo został napisany prosty skrypt w języku Python, obliczający wartość HMAC na komputerze.

Tabela 5. Wydajność obliczania algorytmu HMAC

Test	**Czas obliczania**			
	SHA-1		**SHA-256**	
	Komputer PC	JavaCard	Komputer PC	JavaCard
Test 1	0m0,043s	0m0,204s	0m0,039s	0m0,387s
Test 2	0m0,043s	0m0,361s	0m0,047s	0m0,236s
Test 3	0m0,043s	0m0,240s	0m0,036s	0m0,265s
Test 4	0m0,043s	0m0,244s	0m0,045s	0m0,272s
Test 6	0m0,040s	0m0,279s	0m0,036s	0m0,343s
3.5 KiB	0m0,041s	0m5,085s	0m0,043s	0m5,803s

Powyższa tabela przedstawia wyniki przeprowadzonych badań. Jak widać, czas trwania pojedynczego przejścia algorytmu HMAC nie jest duży i wynosi niewiele ponad ćwierć sekundy dla najwolniejszych przypadków, wobec ok. 40 ms na komputerze PC[1]. Ostatni z przeprowadzonych testów pokazuje, że dla dużych zbiorów danych czas ten już znacznie odbiega od możliwości przeciętnego komputera.

[1] Intel(R) Core(TM) i5-3210M CPU @ 2.50GHz, 12 GiB RAM

10.3 Bezpieczeństwo – generator liczb pseudolosowych

10.3.1 Metoda badań

Jednym z najważniejszych czynników wpływających na bezpieczeństwo całego systemu, a zarazem relatywnie łatwym do zbadania jest jakość generatora liczb pseudolosowych. Na podstawie wartości pre master secret są generowane przez klienta klucze kryptograficzne na bazie modelu deterministycznego. Dla opracowanego systemu na karcie chipowej, jakość samego generatora oraz jego poprawne zastosowanie mają bezpośredni wpływ na bezpieczeństwo systemu.

Aby zbadać wszystkie warianty użycia wbudowanego generatora, testy zostały przeprowadzone trzykrotnie. Pierwszy wariant używał generatora liczb pseudolosowych (RandomData.ALG PSEUDO RANDOM w interfejsie klasy), drugi nie wykonywał inicjalizacji klasy seedem, zamiast tego każde żądanie powodowało budowę nowego obiektu klasy RandomData, aby wymusić powrót obiektu do stanu początkowego. Ostatni test zastosował bezpieczny generator (RandomData.ALG SECURE RANDOM), który jest zalecany do użycia wraz z funkcjami kryptograficznymi.

Dodatkowo, dla porównania z istniejącymi mechanizmami, dostępnymi w systemie Linux, użyte zostały dwie funkcje. Pierwsza to dostępny w bibliotece standardowej języka C funkcja rand(). Aby zaprezentować problemy związane z jej użyciem i wykazać, że wyjście takiego generatora nie przejdzie zdefiniowanych testów, jej wejście zostało niepoprawnie zseedowane przy użyciu funkcji srand(). Drugim blokiem testowym było wyjście pliku /dev/urandom, będącego generatorem liczb pseudolosowych wbudowanym w jądro Linuksa.

Sposób badania generatorów liczb pseudolosowych został zdefiniowany w standardzie FIPS 140 [15], wydanym przez amerykański instytut NIST. Definiuje on zestaw czterech testów, przeprowadzanych na próbce generatora liczb pseudolosowych o wielkości 20000 bitów. Ponieważ nie istnieje możliwość pobrania danych tej wielkości z karty chipowej, pobrane zostały bloki o maksymalnej dostępnej wielkości 255 bajtów, łączone kolejno, aby otrzymać strumień osiągający zadaną długość. Dodatkowo, dla lepszego uwidocznienia błędów, związanych z ustawianiem ziarna (seedu) generatora, zostały pobrane próbki o wielkości 4 bajtów, również złączone w jeden blok o wymaganej wielkości.

Wspomniany standard definiuje testy opisane poniżej.

Monobit Test

Niech X oznacza ilość jedynek w 20000 strumieniu bitów. Test przechodzi, gdy:

$9654 < X < 10346$

Poker Test

1. Podziel 20000 bitowy strumień na 5000 4-bitowych fragmentów
2. Policz liczbę wystąpień wszystkich możliwych 4-bitowych wartości
3. Niech f(x) będzie liczbą wystąpień x, dla $x \in [0;15]$
4. Oblicz wartość wyrażenia:

$$X = \frac{16}{500}\sum_{i=0}^{15}(f(i)^2) - 5000$$

5. Test jest zaliczony, gdy:

$$1{,}03 < X < 57{,}4$$

Runs Test

1. Jako run definiuje się ciąg kolejnych wystąpień tej samej wartości bitu w strumieniu
2. Każdy run powinien zostać zapamiętany wraz z jego długością
3. Run o długości większej niż 6 jest liczony jako posiadający długość równą 6
4. Warunkiem przejścia testu jest, aby ilość wystąpień runu o każdej z długości mieściła się w przedziałach:

- 1: 2267 - 2733
- 2: 1079 - 1421
- 3: 502 - 748
- 4: 223 - 402
- 5: 90 - 223
- 6+: 90 – 223

Long Run Test

Long Run Test jest zdefiniowany jako run o długości co najmniej 34 bitów. Test przechodzi, jeżeli w próbce 20000 bitów nie ma żadnego takiego runu.

10.3.2 Wyniki

W tabeli 6 zebrane zostały wyniki testów zdefiniowanych w [29] i opisanych powyżej. Znajdują się w niej wartości badanych parametrów: *X* dla testów monobit oraz poker i ilości poszczególnych runów dla pozostałych. Dla większej czytelności zamieszczone zostały również dozwolone przedziały wartości dla poszczególnych testów.

Dla wszystkich próbek testowych wyniki dostępne są dla dwóch rozmiarów bloków w jakich zbierane były dane. Wyjątkiem jest urządzenie /dev/urandom, gdzie dane były pobrane w jednym bloku o wymaganym rozmiarze.

Tabela 6. Wyniki przeprowadzonych testów generatora liczb pseudolosowych

Test		**Monobit**	**Poker**			**Runs**				**Long Run**
Typ	**Dł. bloku**			**1**	**2**	**3**	**4**	**5**	**6**	
min		9654	1.03	2267	1079	502	223	90	90	0
max		10346	57.4	2733	1421	748	402	223	223	0
0seed	4	9943	4.96	2517	1248	634	310	128	186	0
	255	10055	3.84	2490	1236	600	298	160	160	0
noseed	4	9905	4.20	2532	1235	615	323	142	155	0
	255	10110	4.06	2460	1295	591	339	148	162	0
secure	4	9954	4.08	2491	1197	643	324	159	180	0
	255	10110	4.47	2547	1232	635	312	165	143	0
rand	4	10625	**42.41**	3749	1875	0	0	0	0	0
	255	10033	4.29	2563	1297	623	269	143	153	0
urandom	-	9967	4.86	2407	1239	671	333	172	138	0

Zgodnie z zamierzeniem użycie bloków o rozmiarze 4 bajtów pozwoliło na zidentyfikowanie błędnego użycia funkcji rand(). Można więc wnioskować, że zdefiniowane testy są w stanie wykryć przynajmniej niektóre problemy z generatorem liczb pseudolosowych.

Warto jednak odnotować, że użycie tego samego generatora z większym rozmiarem bloku nie spowodowało odchylenia żadnego z badanych

parametrów. Gdyby więc testy przeprowadzono jedynie na blokach o rozmiarze 255 bajtów, to nie wykazałyby one żadnych problemów, mimo iż strumień w tym przypadku powinien zawierać kilkukrotnie powtórzony ten sam blok danych. Gdyby w pełni dostosować się do wymagań standardu FIPS, okazałoby się, że wszystkie testy zakończyłyby się pozytywnie, pomimo niebezpiecznego błędu w użyciu.

Testy na blokach czterobajtowych nie wykazały żadnych problemów, można wnioskować, że sam generator wbudowany w kartę chipową jest wystarczająco dobrej jakości pomimo faktu, że również testy przy zerowym seedzie przeszły pomyślnie. W implementacji należy używać jedynie opcji secure do generowania jakichkolwiek danych mających służyć przy operacjach kryptograficznych.

11. Wnioski z implementacji systemu bezpieczeństwa

Celem analizy było zbadanie możliwości urządzeń wbudowanych i wykonalności implementacji protokołu TLS na platformie JavaCard. W wyniku prac powstał prototyp systemu, pozwalający nawiązać połączenie bez ujawniania kluczy kryptograficznych poza serwerem, z którym odbywa się komunikacja. Ponadto stworzony został menadżer haseł, który pozwala przechowywanie haseł bez możliwości ujawnienia ich stronom trzecim, a w szczególności używającemu system użytkownikowi.

Zgodnie z założeniami nie zostało osiągnięte pełne bezpieczeństwo opracowanego systemu. Najważniejszym brakującym fragmentem jest zarządzanie certyfikatami SSL po stronie karty chipowej. Jest planowany temat kolejnych badań.

Kolejnym ograniczeniem były dostępne zasoby. Ich niedobór, w szczególności niska ilość dostępnej pamięci operacyjnej, spowodowała wybór rozwiązań konstrukcyjnych utrudniających dalszy rozwój systemu. Zdecydowano o implementacji jedynie części protokołu TLS po stronie karty.

Zbudowany system charakteryzuje się wydajnością pozwalającą na zastosowanie go przez przeciętnego użytkownika komputera, dzięki temu, że nie wpływa znacząco na czas trwania procesu logowania. Czas ten jest jednak zauważalny, więc potencjalnie system nie będzie w stanie zastąpić tradycyjnego logowania przy pomocy loginu i hasła do czasu dalszego wzrostu wydajności operacji kryptograficznych.

Operacje te zostały w ramach tej pracy zidentyfikowane jako wąskie gardła całego procesu, wpływając na stratę co najmniej 1,5 sekundy (szyfrowanie RSA), wobec 4 sekund jakie zajmuje logowanie do przykładowego systemu.

Podsumowując, w wyniku przeprowadzonych prac udowodniono, że jest możliwe opracowanie menadżera haseł stosującego platformę JavaCard. Zauważyć należy jednak, że opisany w ramach tej pracy system na tym etapie rozwoju (proof of concept) posiada braki, uniemożliwiające jego zastosowanie w praktyce na obecnym etapie rozwoju.

11.1 Dalszy rozwój systemu

Najważniejszym kierunkiem rozwoju jest opracowanie pełnego systemu, możliwego do zastosowania przez przeciętnego użytkownika komputera. Aby to osiągnąć należy przede wszystkim wykonać projekt i implementację infrastruktury klucza publicznego na karcie chipowej. Docelową architekturą mógłby być kolejny aplet JavaCard, do którego trafiałby certyfikat SSL, otrzymany przez protokół TLS, a którego zadaniem byłoby zbadanie jego autentyczności i przesłanie za pomocą metod komunikacji międzyprocesowej klucza publicznego do istniejącego już apletu menadżera haseł.

Kolejnym istotnym kierunkiem badań powinna być kompleksowa analiza opracowanej aplikacji pod kątem bezpieczeństwa implementacji kryptograficznej protokołów, w tym odporności na coraz bardziej wyrafinowane metody ataków, takie jak ataki timingowe. Założeniem takich badań powinna być pełna kontrola atakującego nad systemem użytkownika, aby udowodnić wyższość stworzonego systemu nad istniejącymi rozwiązaniami, gdzie użytkownik ma możliwość podejrzenia przesyłanego hasła.

Wskazana jest również analiza możliwości rozszerzenia systemu o komunikację bezprzewodową przy użyciu technologii takich jak RFID czy NFC. Jest to o tyle ważne, że obecnie coraz więcej użytkowników używa urządzeń mobilnych do łączenia z Internetem. Opracowany system wymaga obecnie podłączenia czytnika kart chipowych przez złącze USB, co dla smartfonów jest teoretycznie możliwe przez złącze USB-OTG. Ponadto implementacja taka może być uproszczona ze względu na fakt, że niektóre karty produkowane przez NXP pod nazwą JCOP (jedna z kart z tej serii była stosowana w systemie) pozwalają jednocześnie na zastosowanie technologii przewodowej ISO 781, jak i bezprzewodowej NFC. Zadanie to wymaga dalszych badań.

Ostatnim celem może być dostosowanie GUI dla ułatwienia jego użycia przez przeciętnego użytkownika. Powinno to zostać osiągnięte przez zbudowanie wtyczki bądź wtyczek do przeglądarek, które umożliwiają przeprowadzenie logowania z zastosowaniem opracowanego systemu za pomocą graficznego interfejsu przeglądarki.

Podsumowanie

Prace nad zastosowaniem systemów wbudowanych pozwoliły zebrać doświadczenie, które potwierdziło tkwiący w nich potencjał i uniwersalność. Ich dodatkowym atutem jest stosowanie ustandaryzowanych modułów co sprawia, że do zaprojektowania części sprzętowej wyposażonej w potrzebne funkcje nie jest już wymagana wiedza elektroniczna. Dzięki czemu możliwe jest skupienie się na części programowej.

Duża liczba przemian technologicznych związana ze zwiększeniem popularności Internetu Rzeczy pozwala sądzić, że systemy wbudowane w ciągu najbliższych lat staną się jedną z wiodących i istotnych technologii.

Wydaje się, że przedstawione rozwiązania okażą się przydatne w projektowaniu własnych systemów wbudowanych.

Bibliografia

[1] Azadeh Kushki. *WLAN positioning systems : principles and applications in location-based services*. Cambridge University Press, 2012.

[2] Janusz Narkiewicz. *GPS i inne satelitarne systemy nawigacyjne*. WKŁ, 2007.

[3] Calamp lmu-900 | flexible, economical gps tracking unit. http://www.calamp.com/products/tracking-and-telemetry-devices/fleet-tracking-units/lmu-900-series

[4] Lokalizator gps dt-gps-tracker mini. http://www.podsluchy.com.pl /lokalizatory-gps/lokalizator-gps-dt-gps-tracker-mini.html

[5] Tracker cat 5 plus. http://www.trackerdirect.co.uk/tracker-cat-5-plus

[6] Tracking device teardown. https://www.ifixit.com/Teardown/Tracking+Device+Teardown/5250

[7] European Telecommunications Standards Institute. *Digital cellular telecommunications system (Phase 2+); Technical realization of the Short Message Service (SMS) Point-to-Point (PP) (GSM 03.40)*, 1996.

[8] European Telecommunications Standards Institute. *Digital cellular telecommunications system (Phase 2+); Alphabets and language-specific information (GSM 03.38)*, 1995.

[9] SIMCom Wireless Solutions Ltd. *SIM908_AT Command Manual_V1.02*, 2011.

[10] ecall: Time saved = lives saved. http://ec.europa. eu/digital-agenda/ecall-time-saved-lives-saved

[11] STMicroelectronics. *Description of STM32F2xx Standard Peripheral Library*, 2011.

[12] Marek Adam Galewski. *STM32 : aplikacje i ćwiczenia w języku C*. Wydawnictwo BTC, 2011.

[13] Krzysztof Paprocki. *Mikrokontrolery STM32 w praktyce*. Wydawnictwo BTC, 2009.

[14] Rafael Saraiva Campos. *RF positioning : fundamentals, applications, and tools*. Artech House, 2015.
[15] Security: Detailed information on the security of keepass. http://keepass.info/help/base/security.html
[16] How it works - lastpass. https://www.lastpass.com/how-it-works
[17] More details - the mooltipass hardware password keeper. https://www.themooltipass.com/#section5
[18] T. Dierks, E. Rescorla; RTFM Inc. *RFC 5246: The Transport Layer Security (TLS) Protocol Version 1.2*, 2008.
[19] Three-factor authentication: Something you know, something you have, something you are. https://blog.gemalto.com/ security/2011/09/05/three-factor-authentication-something-youknow-something-you-have-something-you-are/
[20] Rozporządzenie parlamentu europejskiego i rady (uew sprawie ochrony osób fizycznych w związku z przetwarzaniem danych osobowych i w sprawie swobodnego przepływu takich danych oraz uchylenia dyrektywy 95/46/we (ogólne rozporządzenie o ochronie danych), art. 32 ust. 1a. http://eur-lex.europa.eu/legal-content/PL/TXT/?uri=uriserv:OJ.L_.2016.119.01.0001.01.POL&toc=OJ:L:2016:119:TOC
[21] The transport layer security (tls) protocol version 1.3. https://tools.ietf.org/html/draft-ietf-tls-tls13-21
[22] Authentication cheat sheet. https://www.owasp.org/index.php/Authentication_Cheat_Sheet
[23] Rozporządzenie ministra spraw wewnętrznych i administracji z dnia 29 kwietnia 2004 r. w sprawie dokumentacji przetwarzania danych osobowych oraz warunków technicznych i organizacyjnych, jakim powinny odpowiadać urządzenia i systemy informatyczne służące do przetwarzania danych osobowych. http://isap.sejm.gov.pl/DetailsServlet?id=WDU20041001024
[24] Bleichenbacher's rsa signature forgery based on implementation error. https://web.archive.org/web/20070206184043/ https://www.imc.org/ietf-openpgp/mail-archive/msg14307.html

[25] Security of cbc ciphersuites in ssl/tls: Problems and countermeasures. https://www.openssl.org/~bodo/tls-cbc.txt

[26] Release notes java card tm specifications version 2.2.2 changes to virtual machine specification, version 2.2.2. http://www.oracle.com/technetwork/java/embedded/javacard/documentation/releasenotes-jcspecspw-2-2-2-142671.html#Virtual_Machine_Specification_Changes

[27] M. Nystrom, RSA Security. *RFC 4231: Identifiers and Test Vectors for HMACSHA-224, HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512*, 2005.

[28] P. Cheng, IBM; R. Glenn, NIST. *RFC 2202: Test Cases for HMAC-MD5 and HMAC-SHA-1*, 1997.

[29] Security requirements for cryptographic modules. https://web.archive.org/web/20030203024720/http://www.itl.nist.gov:80/lab/fips/fip140-1.txt

[30] Alg rsa no pad uses wrong plain text length - github.: https://github.com/licel/jcardsim/issues/18

[31] International Organization for Standardization. *ISO 7816-4 Identification cards - Integrated circuit cards Part 4: Organization, security and commands for interchange*, 2005.

[32] Java card v2.2.2 api. http://download.oracle.com/otndocs/jcp/java_card_kit-2.2.2-fr-oth-JSpec/

[33] Joshua Davies. *Implementing SSL/TLS Using Cryptography and PKI; Chapter 6: A Usable, Secure Communications Protocol: Client-Side TLS*. Wiley Publishing, Inc., 2011.

[34] Bruce Schneier. *Applied Cryptography: Protocols, Algorithms and Source Code in C*. Wiley, 2015.

Spis rysunków

Spis tabel

ISBN 978-83-7283-884-1