Wydział Podstawowych Problemów Techniki
Politechnika Wrocławska

Instalowanie certyfikatów nowym urządzeniom w oparciu o żądania certyfikatów z podpisami typu ID-based

Kajetan Korzycki
Nr indeksu: 236799

Praca inżynierska napisana
pod kierunkiem
dra Przemysława Kubiaka

Politechnika
Wrocławska

Wrocław 2020

Spis treści

Wstęp

Przedmiotem niniejszej pracy inżynierskiej jest system, w którym nowym urządzeniom instalowane są certyfikaty w oparciu o żądania certyfikatów z podpisami typu ID-based. Omówiony zostaje problem związany z wprowadzaniem niezaufanych urządzeń do infastruktury klucza publicznego przedsiębiorstwa produkcyjnego, przedstawiona jest propozycja jego rozwiązania oraz praktyczna implementacja w środowisku demonstracyjnym. Praca zawiera opis architektury systemu wraz z wytycznymi dotyczącymi wdrożenia systemu w rzeczywistych zastosowaniach.

1.1 Zakres pracy

Rozważmy urządzenia, które są w dużej liczbie nabywane przez klientów dysponujących infrastrukturą klucza publicznego (PKI). Załóżmy, że nabywcy chcieliby przed wystawieniem certyfikatów nowym urządzeniom zweryfikować ich autentyczność. W tym celu, producent urządzeń może zaopatrzyć je w asymetryczne klucze do uwierzytelniania pierwszego żądania certyfikatu. Jednocześnie zakładamy, że producent ten nie chce ponosić kosztów związanych z utrzymywaniem PKI - kosztów związanych z zarządzaniem i utrzymywaniem katalogu wydanych kluczy bądź wystawionych certyfikatów. Proponowanym w tej pracy rozwiązaniem jest wykorzystanie narzędzia, jakim są podpisy typu ID-based. Ich cechą charakterystyczną jest to, że klucz weryfikacji podpisu można wyliczyć na podstawie identyfikatora urządzenia, czyli np. jego numeru seryjnego.

W zakres pracy wchodzą:

1. Analiza podpisów ID-based i wybór kandydata do implementacji.
2. Wykonanie projektu systemu oraz opracowanie planu testów.
3. Wykonanie implmenetacji systemu oraz testów.

1.2 Zawartość pracy

Praca składa się z 7 rozdziałów. Rozdział drugi *Architektura i projekt systemu* zawiera szczegółowy opis problemu rozważanego w pracy oraz pokrótce omawia istniejące oraz proponowane w tym dokumencie rozwiązania. Przedstawione są także podstawowe pojęcia związane z kryptografią asymetryczną i infrastrukturą klucza publicznego.

W rozdziale trzecim znajduje się opis komponentów wchodzących w skład proponowanego systemu. Przedstawiona zostaje architektura systemu, szczegóły dotyczące zależności i komunikacji między jego elementami oraz przypadki użycia. Poza tym są tam także uwagi dotyczące założeń implementacyjnych, opisy wybranych protokołów i przedstawienie sposobu realizacji komponentów.

Rozdział czwarty poświęcony jest wybranemu do zaimplementowania algorytmowi ID-based. Zarysowana zostaje teoria dotycząca schematu podpisów Adiego Shamira, a także uwagi dotyczące jego implementacji i bezpieczeństwa.

Rozdział piąty zawiera opis implementacji systemu. Opisane są wykorzystane technologie, przebieg realizacji oraz wybrane kody źródłowe. Przedstawione są także wykonane testy wraz z opisem środowiska testowego.

W rozdziale szóstym umieszczone są wymagania i instrukcje dotyczące instalacji systemu. Zawiera on instrukcje dotyczące przygotowania środowiska oraz uruchomienia samych aplikacji. Poza tym znajdują się tam wskazówki dotyczące wdrożenia systemu w rzeczywistym zastosowaniu.

Rozdział siódmy zawiera wnioski wyciągnięte z pracy oraz opis możliwości rozwoju systemu. Podsumowane zostają efekty pracy, napotkane trudności i kierunki dalszego rozszerzenia implementacji.

Analiza problemu i porównanie z istniejącymi rozwiązaniami

W tym rozdziale przedstawiona jest analiza problemu, będącego istotą niniejszej pracy dyplomowej. Wytłumaczone zostały podstawowe pojęcia związane z omawianym zagadnieniem, zaprezentowani są użytkownicy systemu oraz ich role. Rozważone są czynniki wynikające z rzeczywistych utrudnień oraz ich implikacje. Przytoczone zostały także dotychczas znane rozwiązania podobnych problemów.

2.1 Analiza problemu

Do uwierzytelniania produktów pochodzących od innych przedsiębiorców można posłużyć się kryptografią asymetryczną. Rozwiązaniem pozwalającym na weryfikację są w szczególności tzw. infrastrukury klucza publicznego (PKI) wykorzystujące certyfikaty, za których integralność z kolei odpowiadają podpisy cyfrowe. Powszechnie używane w czasie pisania tej pracy standardy wymagają utworzenia właśnie takiej infrastruktury. W jej skład wchodzi kilka komponentów, a utrzymanie całości wiąże się z kosztami i obowiązkami niekorzystnymi dla producenta.

Szczególnym przypadkiem użycia może być chęć wprowadzenia nowo zakupionego urządzenia do wewnętrznego PKI przedsiębiorstwa. Dzięki istnieniu PKI, urządzenia mogą komunikować się z innymi urządzeniami w przedsiębiorstwie używając zaufanych, szyfrowanych połączeń (np. z użyciem protokołu TLS). Certyfikaty dostępne w standardach (np. X.509), żeby najpierw zweryfikować urządzenie, a następnie wprowadzić je w strukturę PKI w przedsiębiorstwie, wymagają istnienia dwóch infrastruktur klucza publicznego - u producenta i u klienta. Pierwsza z nich służyłaby do weryfikacji urządzenia u klienta po zakupie od producenta, druga natomiast do zapewnienia bezpieczeństwa w komunikacji między urządzeniami w infrastrukturze klienta. Utrzymywanie infrastruktury PKI wiąże się z obowiązkami, a co za tym idzie z kosztami (zob. 2.1.3).

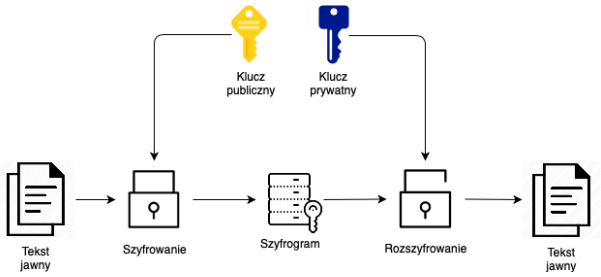
Alternatywnym rozwiązaniem, jest użycie podpisów typu ID-based do weryfikacji produktu przed wprowadzeniem do PKI klienta. Są to podpisy, do których uwierzytelnienia można użyć identyfikatora urządzenia. Takie rozwiązanie znacząco redukuje obowiązki producenta, a jednocześnie nadal pozwala na bezpieczną weryfikację urządzenia.

2.1.1 Kryptografia asymetryczna

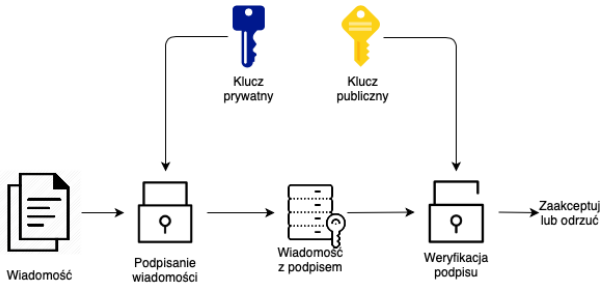
U podstaw omawianych w tej pracy sposobów weryfikacji stoi kryptografia asymetryczna. Jest to idea, której początki sięgają lat 60 XIX wieku. Dotychczas, znane systemy kryptograficzne opierały się na kryptografii symetrycznej. Jej założeniem było współdzielenie jednego klucza przez podmioty szyfrujące i rozszyfrowujące dane. W rewolucyjnym modelu kryptografii asymetrycznej, generowane są dwa osobne, powiązane ze sobą klucze - jeden służący do szyfrowania, drugi do odszyfrowywania wiadomości. Jeden z kluczy zostaje opublikowany, podczas gdy drugi pozostaje sekretem użytkownika systemu. Stąd ten pierwszy nazywany jest *kluczem publicznym*, a drugi *kluczem prywatnym*. Klucz prywatny służy do rozszyfrowywania wiadomości zaszyfrowanej za pomocą klucza publicznego lub do wykonania podpisu wiadomości, który może zostać zweryfikowany również przy użyciu klucza publicznego. Schemat szyfrowania przedstawia rysunek 2.1, natomiast schemat podpisu rysunek 2.2.

Obecnie stosowane schematy szyfrowania w kryptografii asymetrycznej wykorzystują operacje jednokierunkowe. Są to operacje, które stosunkowo łatwo przeprowadzić w jedną stronę, ale w drugą bardzo trudno. Przez łatwość czy trudność rozumiemy tutaj złożoność obliczeniową obliczeń. Przykładem takich operacji jest mnożenie, które jest stosunkowo łatwe i rozkład na czynniki pierwsze, które jest stosunkowo trudne. Na tym opiera się w głównej mierze działanie RSA. Analogicznie, potęgowanie modulo jest

stosunkowo łatwe, a logarytmowanie dyskretne - stosunkowo trudne, na czym zbudowane są schematy ElGamal czy DSA.



Rysunek 2.1: Schemat przedstawiający szyfrowanie z wykorzystaniem kryptografii asymetrycznej.



Rysunek 2.2: Schemat przedstawiający podpisywanie wiadomości z wykorzystaniem kryptografii asymetrycznej.

2.1.2 Infrastruktura Klucza Publicznego

Ze względu na publiczny charakter kluczy szyfrujących lub weryfikujących, ich integralność, a także integralność przypisania klucza do podmiotu zazwyczaj chroniona jest certyfikatami. Infrastruktury Klucza Publicznego (PKI) są zbiorem procedur, systemów, osób i polityk pozwalających na zachowanie tej integralności. Mają one zapewnić zaufane i wydajne zarządzanie kluczami oraz certyfikatami. Głównymi komponentami PKI są:

- Klucze (prywatne i publiczne) oraz certyfikaty,
- *Certificate Authority (CA)* - jednostka generująca certyfikaty i listy przedawnionych certyfikatów (*CRL*).
- *Registration Authority (RA)* - urząd rejetracji, mający za zadanie zapewnić, że użytkownik, który otrzyma certyfikat, jest prawowitym użytkoniwkiem systemu. Weryfikuje on także samą tożsamość użytkownika. Często funkcjonuje razem z *CA* jako jeden podmiot.
- Baza certyfikatów - w większości systemów wystawione (oraz wygaśnięte) certyfikaty przechowywane są w bazie danych po stronie *CA*.
- Polityki i procedury - mają one zapewnić spójność systemu. Wchodzące w ich skład *Certificate Policy (CP)* i *Certification Practice Statements (CPS)* określają jak i komu wydawane, a później w jaki sposób zarządzane są certyfikaty. Określają one także rolę danych certyfikatów w całej infrastrukturze.
- Subskrybent - właściciel certyfikatu.

Przyjęte jest, że klucze dla subskrybenta generowane mogą być zarówno przez *CA*, jak i przez samego subskrybenta, który później dostarczą kopię klucza publicznego do *CA*. Wybór jednego z mechanizmów określa polityka bezpieczeństwa danego systemu. Zależnie od konkretnego zastosowania klucza podpisującego, jedno z rozwiązań może okazać się bardziej korzystne. Jeśli jest użyty do systemów mających

zapewnić niezaprzeczalność, wtedy klucz powinien być generowany po stronie subskrybenta. Kiedy klucz użyty jest do szyfrowania poufnych informacji, lepiej żeby był generowany przez *CA* - umożliwia to awaryjne odtworzenie zaszyfrowanych danych.

Najpopularniejszym obecnie standardem PKI jest opublikowany po raz pierwszy w 1988 roku standard X.509. Został on użyty w wielu znanych protokołach, jak np. protokół HTTPS czy TLS. Certyfikaty zgodne z tym standardem zawierają między innymi klucz publiczny podmiotu, dane pozwalające na jego identyfikację (np. nazwę organizacji, nazwę *hosta*), datę ważności i dane jednostki, która wystawiła certyfikat. Poszczególne techniczne aspekty standardu, omówione są w dalszej części tej pracy.

2.1.3 System ze standardowymi certyfikatami

Przedstawiony problem weryfikacji urządzeń pochodzących od producenta i włączenia do infrastruktury PKI klienta korporacyjnego może być rozwiązany przy użyciu standardowych, szeroko używanych infrastruktur, np. standardu X.509. W tym wypadku rozwiązaniem może być użycie dwóch systemów PKI - jeden po stronie producenta, drugi po stronie klienta.

Ogólny schemat działania przykładowego systemu można opisać w następujących krokach:

1. Producent posiada własną infrastrukturę PKI. W jego skład wchodzi CA, RA z własciwymi dla nich kluczami. Klucz publiczny CA udostępniony jest klientom danego producenta. Podobnie klient posiada własną infrastrukturę PKI, pozwalającą mu na uwierzytelnienie urządzeń działających wewnątrz jego przedsiębiorstwa.

2. Producent wystawia certyfikaty nowo wyprodukowanym urządzeniom. Zostaje na nich osadzony certyfikat wraz z odpowiadającym mu kluczem prywatnym. Alternatywnie, klucz prywatny może być wygenerowany przez urządzenie, a do RA producenta dostarczona jest tylko kopia klucza publicznego urządzenia.

3. Klient po zakupie urządzenia weryfikuje wiarygoność urządzenia, wykorzystując obecny na urządzeniu certyfikat i znany klientowi klucz publiczny producenta. Urządzenie generuje także żądanie podpisania certyfikatu dla infrastruktury PKI klienta.

4. Po weryfikacji, przez CA należące do klienta wystawiony zostaje nowy certyfikat dla urządzenia. Certyfikat ten służy już do uwierzytelnienia wszystkich połączeń wewnątrz przedsiębiorstwa klienta.

5. Klient może bezpiecznie komunikować się z zaufanymi urządzeniami wewnątrz swojego przedsiębiorstwa.

Wykorzystanie w takim schemacie standardowych infrastruktur PKI wiąże się z dodatkowymi obowiązkami po stronie producenta. Oczekuje się, że będzie on utrzymywać listę unieważnionych certyfikatów (*CRL*). Poza tym, każdy z wystawionych już certyfikatów ma swoją datę ważności. Producent musiałby w odpowiedni sposób nimi zarządzać i pilnować, by były aktualne. W praktyce, te problemy związane z utrzymaniem PKI generują dodatkowe koszty.

2.1.4 Kryptogrfia ID-based

Alternatywą dla zastosowania dwóch tradycyjnych systemów PKI jest wykorzystanie kryptografii klucza publicznego opartej na tożsamości (*Identity-based Public Key Cryptography, ID-PKC)*. Jest to rodzaj kryptografii, w której rolę klucza publicznego spełnia znany publicznie ciąg znaków (bądź dane z niego pozyskane) reprezentujący dany podmiot. W praktyce można pomyśleć o tym ciągu, jako o unikalnym identyfikatorze, takim jak np. numer pesel, adres email lub numer IP urządzenia. Koncept ten pojawił się w 1984 roku w pracy *Identity-Based Cryptosystems and Signature Schemes* [24] autorstwa Adiego Shamira, jednego z trzech twórców algorytmu RSA. W pracy została przedstawiona propozycja implementacji podpisów typu ID-based i obserwacje dotyczące hipotetycznej przyszłej konstrukcji analogicznych schematów szyfrowania.

Zasady działania podpisów ID-based można opisać jako zbiór czterech następujących algorytmów:

- **Inicjalizacja** Ten algorytm wykonywany jest przez centralny urząd rejestracji. Odpowiada on za wygenerowanie głównego klucza publicznego (*master public key*), w którego skład wchodzą parametry `params` ustalone dla schematu oraz głównego klucza prywatnego (*master private key*). Klucz publiczny zostaje opublikowany, natomiast prywatny pozostaje sekretem.

- **Generowanie klucza** Mając do dyspozycji tożsamość `id`, główny klucz prywatny oraz parametry `params`, algorytm generuje klucz prywatny d_{id} dla tożsamości `id`. Następnie dystrybuuje klucz do właściciela tożsamości `id` przez bezpieczny kanał komunikacyjny.
- **Podpis** Mając na wejściu wiadomość `m`, tożsamość `id`, klucz prywatny d_{id} i parametry `params`, generowany jest podpis σ dla wiadomości `m`. Algorytm używany jest przez właściciela tożsamości `u` do podpisywania.
- **Weryfikacja** Mając podpis σ, główny klucz publiczny, wiadomość `m` i tożsamość `id`, algorytm zwraca wiadomość o zgodności (lub jej braku) podpisu σ z wiadomością `m` i podpisującym `id`.

W przypadku kryptografii ID-based, zamiast *CA* występuje jej odpowiednik, tzw. *Trust Authority (TA)*, czyli w wolnym tłumaczeniu *Urząd Zaufania*. Schemat działania podpisu jest podobny do tradycyjnych algorytmów asymetrycznych. Najpierw *TA* musi wygenerować swoje klucze do podpisu, czyli tzw. *master public key* oraz *master private key*. Za pomocą swojego *master private key*, *TA* jest w stanie generować klucze prywatne dla danych klientów (dla ich identyfikatorów). Następnie klient będący w posiadaniu wygenerowanego klucza prywatnego, może za jego pomocą dokonywać podpisu. Podpis z kolei może być zweryfikowany za pomocą klucza publicznego klienta, czyli jego identyfikatora oraz klucza publicznego *TA*.

Główną różnicą w stosunku do tradycyjnych algorytmów asymetrycznych jest sposób generowania kluczy. Dzięki publicznie dostępnych informacjom, klucze mogą być generowane bez konieczności wymiany kopii klucza publicznego między podpisującym i klientem, dla którego generowany jest klucz. Warto zauważyć, że *TA* zna wtedy klucze prywatne klientów.

Zmiany te niosą za sobą wiele skutków. W przeciwieństwie do tradycyjnego *CA*, które poświadcza tylko zgodność informacji zawartość w certyfikacie, *TA* odpowiada za dystrybucję kluczy w systemie. Pojawia się także wymóg istnienia bezpiecznego kanału komunikacyjnego między *TA* a klientem w celu przesłania klucza prywatnego. W podstawowym modelu nie ma także możliwości wygaśnięcia ważności kluczy klienta. Rozwiązanie tego problemu wymaga zastosowania dodatkowych składników przy generowaniu klucza, takich jak np. data ważności.

Szczegóły techniczne dotyczące algorytmu zostaną przedstawione w dalszej części tej pracy.

2.1.5 System z certyfikatami typu ID-based

Wprowadzenie podpisów typu ID-based nie wymusza wielu zmian w schemacie postępowania opisanym wcześniej. Zmodyfikowane kroki będą teraz przedstawiać się następująco:

1. Producent posiada własne klucze *master public key* i *msater private key* dla podpisów ID-based. Jest to tzw. *Trust Authority*. Klucz publiczny *TA* udostępniony jest klientom danego producenta. Klient posiada własną infrastrukturę PKI, pozwalającą mu na uwierzytelnienie urządzeń działających wewnątrz jego przedsiębiorstwa.
2. Producent nowo wyprodukowanym urządzeniom generuje klucze prywatne i osadza je (w bezpiecznym środowisku) na urządzeniach.
3. Urządzenie przy weryfikacji przez klienta generuje parę kluczy do zwykłego podpisu (np. RSA) oraz żądanie podpisania certyfikatu (CSR) podpisane jego prywatnym kluczem ID-based. W żądaniu znajduje się wygenerowany klucz publiczny do zwykłego podpisu. Klient może zweryfikować urządzenie, używając podpisu złożonego na CSR i klucza publicznego producenta.
4. Po weryfikacji, przez *CA* należące do klienta wystawiony zostaje nowy certyfikat dla urządzenia. Certyfikat ten służy już do uwierzytelnienia wszystkich połączeń wewnątrz przedsiębiorstwa klienta.
5. Używając kluczy do standardowego podpisu, klient może bezpiecznie komunikować się z zaufanymi urządzeniami wewnątrz swojego przedsiębiorstwa.

Najważniejszą różnicą między tym, a przedstawionym wcześniej rozwiązaniem jest fakt, że w tym przypadku producent nie jest zobowiązany do utrzymywania tradycyjnego PKI. Liczba elementów niezbędnych do funkcjonowania systemu istotnie się zmiejszyła. Nie ma konieczności prowadzenia listy unieważnionych certyfikatów. W prostym modelu, raz wystawione klucze będą też ważne cały czas bez konieczności odnawiania. Oznacza to znaczne uproszczenie dla producenta oraz redukcję kosztów koniecznych do zarządzania taką infrastrukturą. Wymaga to jednak zapewnienia przez producenta bezpiecznego kanału do osadzenia kluczy prywatnych na urządzeniach.

2.2 Istniejące rozwiązania

Biorąc pod uwagę tylko poszczególne części systemu, znaleźć można istniejace implementacje rozwiązujące niektóre podproblemy. U podstaw wielu systemów kryptograficznych leżą często te same, powszechnie znane biblioteki, takie jak `OpenSSL`[1]. Jest to otwarta implementacja wielu algorytmów i protokołów ogólnego przeznaczenia. Może ona służyć zarówno do wykonywania prostych operacji u użytkownika, jak np. generowanie par kluczy, jak i do konstrukcji zaawansowanych systemów, takich jak implementacja protokołu TLS. W bibliotece znaleźć można wsparcie dla standardu certyfikatów X.509. Za jej pomocą można w stosunkowo prosty sposób stworzyć własną infrastrukturę PKI, generować żądania podpisania certyfikatów czy certyfikaty. Dostępnych jest też kilka rodzajów podpisów i algorytmów szyfrowania, takich jak np. RSA czy DSA. Nie jest dostępny jednak żaden podpis typu ID-based. Bez jej modyfikacji nie można zatem wygenerować ani par kluczy ID-based, ani żądania podpisania certyfikatu podpisanego kluczem ID-based. Nie udało się także znaleźć żadnej innej implementacji pozwalającej na zastosowanie tych podpisów przy generowaniu żądania podpisania certyfikatu.

Innym przykładem realizacji infrastruktury PKI w podobnym zastosowaniu jest tzw. system *SCADA*, który odpowiada za nadzorowanie procesu technologicznego lub produkcyjnego. Jego zabezpieczenia opierają się o specjalnie spersonalizowaną strukturę PKI, opisaną szczegółowo w pracy *Customized PKI for SCADA System*[20]. Została ona odpowiednio spersonalizowana pod wymagania i ograniczenia urządzeń wykorzystywanych w produkcji, takich jak np. czujniki, zawory czy pompy. Do czynników branych pod uwagę przy tworzeniu usprawnień w PKI zaliczyć można małą moc obliczeniową, ograniczoną pamięć, wymagany krótki czas wykonywania procedur bezpieczeństwa. Modyfikacji ulegają niektóre procesy uwierzytelniania, tworząc bardziej wydajny, a jednocześnie nadal bezpieczny system. Znalezione materiały nie uwzględniają jednak weryfikacji urządzeń przed włączeniem ich do struktury PKI. Nie są wykorzystane także w żaden sposób podpisy ID-based.

Istnieją jednak systemy wykorzystujące kryptografię opartą na tożsamości. W większości są oparte na odwzorowaniu dwuliniowym na krzywych eliptycznych (*bilinear pairing on elliptic curves*) i oferują nie tyle podpisy, co całe schematy szyfrowania. Wykorzystują one rozwiązania zaproponowane przez Dana Boneh'a i Matthew K. Franklin'a w 2001 roku lub późniejsze. Zostały one także zaimplementowane przez autorów i opublikowane jako biblioteka `C++` pod nazwą *Stanford IBE System*[5]. Jednym z popularniejszych zastosowań jest jednak seria wtyczek do programów pocztowych produkowanych przez firmę *Voltage Security Inc.* Umożliwiają one automatyczne szyfrowanie wychodzących wiadomości email z wykorzystaniem kryptografii ID-based. Zasadniczą różnicą między tymi zastosowaniami, a rozwiązaniem przedstawionym w tej pracy jest to, że w tej pracy wykorzystano tylko schemat podpisów. Poza tym zaimplementowany został oryginalny schemat zaproponowany przez Adiego Shamira, a nie systemy oparte na odzworowaniu duwliniowym.

Warto wspomnieć także o standardzie SM9 wydanym oficjalnie przez narodowy urząd Chińskiej Republiki Ludowej - *Chinese State Cryptographic Authority*. Został on opisany między innymi w pracy *The SM9 Cryptographic Schemes*[21] autorstwa Zhaohui Cheng. Zawiera on opis zarówno podpisów, jak i schematu szyfrowania, protokołu uzgadniania kluczy oraz mechanizmu enkapsulacji kluczy opartych o algorytmy ID-based. Algorytmy te posiadają także identyfikatory OID (*Object Identifier*) w notacji ASN.1. Implementację standardu znaleźć można w blibiotece open-source `GmSSL`[12]. Podobnie jak w wypadku wyżej omawianych, działanie tych algorytmów opiera się na odwzorowaniu dwuliniowym na krzywych eliptycznych, co odróżnia je od tych przedstawionych w tej pracy.

Same podpisy ID-based znalazły swoje zastosowanie w branży *Internet of Things*, czyli internetu rzeczy. W pracy *Authentication in dynamic groups using Identity-based Signatures*[22] z 2018 roku przedstawiony został schemat komunikacji między urządzeniami *IoT* w sieci, gdzie za uwierzytelnienie odpowiadał właśnie ten rodzaj podpisów. Sam scenariusz jest jednak nieco inny niż w problemie omawianym w tej pracy dyplomowej. W przypadku tej pracy, urządzenia weryfikowane są tylko raz przy przejściu do PKI klienta, by następnie uwierzytelniać się tradycyjnymi certyfikatami. W przytoczonym systemie IoT, tradycyjne certyfikaty w ogóle nie występują, a podpisy ID-based odpowiadają za uwierzytelnienie przy każdym połączeniu.

W chwili pisania tej pracy nie udało się znaleźć istniejącego analogicznego systemu, wykorzystującego podpisy ID-based. Nie wyklucza to jednak ich istnienia. Warto mieć na uwadze, że problematyka pracy dotyczy w dużej mierze systemów wysokiego ryzyka, gdzie konieczne jest stosowanie najwyższych standardów bezpieczeństwa. W szczególności oznaczać to może, że stosowane w tym sektorze technologie nie są upubliczniane.

Architektura i projekt systemu

W tym rozdziale przedstawiony jest projekt proponowanego rozwiązania. Opisane zostały poszczególne komponenty, zależności występujące między nimi oraz sposób ich realizacji. Zaprezentowane są także protokoły wykorzystane do implementacji systemu.

3.1 Komponenty systemu

W symulacji systemu wyróżnić można trzy zasadnicze podmioty - *Producenta* produkującego urządzenia, które będą weryfikowane, urządzenie (nazywane dalej *Czujnikiem*) oraz *Klienta*. Zarówno *Producent*, jak i *Czujnik* stanowią pojedyncze urządzenia z aplikacją. W przypadku *Klienta*, wyróżnione zostały 3 niezależne komponenty - *Terminal* sczytujący dane z *Czujnika*, *Urząd rejestracji* (*RA*), odpowiadający za weryfikację żądań podpisania certyfikatów oraz *Urząd certyfikacji* (CA), odpowiadający za wystawianie (w tym podpisywanie) certyfikatów w PKI *Klienta*. Warto zaznaczyć, że urzędy CA i RA zostały zintegrowane w jedną aplikację.

Każde z tych urządzeń udostępnia funkcjonalności pozwalające na prawidłowe działanie całego systemu. Ze względu na poszczególne wykonywane procedury, wyróżnić można 8 komponentów, które osadzone są w poszczególnych aplikacjach.

Lista komponentów oraz podmiot, z którym są związane przedstawia tabela (tab. 3.1).

Lp.	Komponent	Przynależność
1.	Komponent osadzający klucze podpisów typu ID-based na *Czujniku*	*Producent*
2.	Komponent osadzający na *Czujniku* i *Terminalu* tzw. punkt zaufania	*RA*
3.	Komponent generujący żądania certyfikatów oraz weryfikujący certyfikaty wystawione *Czujnikowi*	*Czujnik*
4.	Urząd rejestracji (*RA*) weryfikujący żądania certyfikatów *Czujnika* i *Terminala*	*RA*
5.	Urząd certyfikacji (*CA*) podpisujący certyfikaty dla *Czujnika* i *Terminala*	*CA*
6.	Komponent uwierzytelniający, nawiązujący bezpieczne połączenie z *Terminalem*	*Czujnik*
7.	Komponent uwierzytelniający, nawiązujący bezpieczne połączenie z *Czujnikiem*	*Terminal*
8.	Komponent generujący żądania certyfikatu na *Terminalu* oraz weryfikujący certyfikaty wystawione *Terminalowi*	*Terminal*

Tablica 3.1: Komponenty

3.1.1 Komponent osadzający klucze podpisów typu ID-based na *Czujniku*

Komponent na podstawie posiadanych kluczy *Producenta*, generuje klucz prywatny *Czujnika*, a następnie przy pomocy przenośnej pamięci USB zostaje on przeniesiony na *Czujnik*.

- **Znajduje się w aplikacji:** *Producent*
- **Składniki klucza użyte:** klucz publiczny typu ID-based *Producenta*, klucz prywatny typu ID-based *Producenta*, numer seryjny *Czujnika*

- **Komunikuje się z komponentami:** brak
- **Sposób komunikacji:** nośnik danych
- **Wymienione dane:** klucz prywatny typu ID-based

Przypadki użycia

- Przypadek użycia: Generowanie klucza dla *Czujnika*
  1. *Producent* wprowadza numer seryjny *Czujnika* do komponentu.
  2. Za pomocą odpowiednich funkcji i algorytmów (zob. rozdz. 4), wykorzystując klucz prywatny *master private key Producenta*, wygenerowany i zapisany do pliku zostaje klucz prywatny dla *Czujnika*. Dodatkowo *Producent* zapisuje do bazy wystawionych kluczy numer seryjny *Czujnika*.
  3. Plik z kluczem prywatnym zostaje przeniesiony na zewnętrzny nośnik (np. pamięć USB).
  4. Zewnętrzny nośnik zostaje podłączony do *Czujnika*, a następnie plik z kluczem zostaje przeniesiony do jego pamięci.
- Alternatywa: Producent nie ma wygenerowanego klucza prywatnego:
  - W punkcie 2. zwrócony zostaje błąd. *Producent* musi wygenerować parę kluczy głównych typu ID-based do generowania kluczy dla *Czujników*.

Warto zaznaczyć, że w rzeczywistym systemie odpowiedzialność za zadania z punktu 2. i 3. powinna być rodzielona pomiędzy personel. Za generowanie kluczy powinna odpowiadać inna jednostka niż ta odpowiedzialna za osadzanie kluczy. Takie rozwiązanie daje możliwość wzajemnej kontroli personelu nad osadzanymi i generowanymi kluczami.

3.1.2 Komponent osadzający na *Czujniku* i *Terminalu punkt zaufania*

Komponent transmituje za pomocą sieci WiFi certyfikat `ROOT CA`, który staje się *punktem zaufania* dla *Czujnika* lub *Terminala* względem PKI *Klienta*.

- **Znajduje się w aplikacji:** *RA*
- **Składniki klucza użyte:** brak
- **Komunikuje się z komponentami:** brak
- **Medium transmisyjne:** fale radiowe (sieć Wi-Fi)
- **Wymienione dane:** certyfikat `Root CA`

Przypadki użycia

- Przypadek użycia: Osadzenie *punktu zaufania* na *Czujniku*
  1. Komponent nawiązuje połączenie z *Czujnikiem*.
  2. Komponent wysyła do *Czujnika* plik zawierający certyfikat `Root CA`.
  3. *Czujnik* odbiera reprezentację certyfikatu, zapisuje w odpowiednim miejscu w pamięci urządzenia i zwraca informację o sukcesie operacji.
- Alternatywa: *Czujnik* nie ma klucza prywatnego typu ID-based:
  - *Czujnik* przed wykonaniem punktu 2. zwraca informację o błędzie i kończy procedurę.
- Alternatywa: *Czujnik* ma już poprzednio osadzony *punkt zaufania*:
  - W punkcie 3. *Czujnik* weryfikuje czy nowy punkt zaufania wystawiony jest na ten sam klucz publiczny co poprzedni. Zwraca informację o sukcesie w wypadku pozytywnej weryfikacji, w przeciwnym razie zwraca informcję o błędzie i punktem zaufania pozostaje poprzednio osadzony certyfikat.

W przypadku osadzenia *punktu zaufania* na *Terminalu* schemat jest analogiczny. Jedyną różnicą jest to, że odbiorcą łańcucha certyfikatów jest *Terminal* zamiast *Czujnika*.

3.1.3 Komponent generujący żądania certyfikatów oraz weryfikujący certyfikaty wystawione *Czujnikowi*

Komponent generuje parę kluczy RSA, na podstawie tego klucza publicznego i informacji o *Czujniku* generuje *CSR*, podpisuje go kluczem prywatnym typu ID-based i wysyła do *Terminala*. Następnie czeka na podpisany certyfikat wraz z łańcuchem certyfikatów od `Root CA` do `Signing CA` i weryfikuje jego poprawność.

- **Znajduje się w aplikacji:** *Czujnik*
- **Składniki klucza użyte:** klucz prywatny *Czujnika* typu ID-based, klucze prywatny i publiczny RSA *Czujnika*
- **Komunikuje się z komponentami:** *Urząd rejestracji* (nr 4)
- **Medium transmisyjne:** sieć Wi-Fi
- **Wymienione dane:** *CSR* dla *Czujnika*, łańcuch certyfikatów od `Root CA` do `Signing CA`

Przypadki użycia

Ze względu na silne powiązanie funkcjonalności tego komponentu z komponentami odpowiadającymi *Urzędowi rejestracji* i *Urzędowi certyfikacji*, przedstawiony przypadek użycia dotyczy wszystkich 3 modułów.

- Przypadek użycia: Wystawienie certyfikatu *Czujnikowi* w PKI *Klienta*.
  1. Komponent będący na *Terminalu* nawiązuje połączenie TLS z *Czujnikiem* i wysyła zapytanie o wygenerowanie żądania podpisania certyfikatu. W rzeczywistym systemie takie zapytanie może zawierać dane, które później zawarte będą w certyfikacie urządzenia, takie jak np. numer linii produkcyjnej na której działać będzie czujnik. Przy nawiązaniu połączenia, *Terminal* zostaje zweryfikowany na podstawie przedstawionego przez niego łańcucha certyfikatów, zawierającego certyfikaty początkowy, końcowy i pośrednie od `Root CA` do certyfikatu *Terminala* oraz punktu zaufania znajdującego się na *Czujniku*.
  2. *Czujnik* generuje parę kluczy *RSA*. Klucz prywatny zapisuje na urządzeniu, a klucz publiczny umieszcza w treści żądania podpisania certyfikatu.
  3. *Czujnik* podpisuje *CSR* prywatnym kluczem typu ID-based.
  4. *Czujnik* wysyła do *Terminala* gotowe żądanie podpisania certyfikatu.
  5. *Terminal* odbiera *CSR*. Następnie za pomocą nośnika danych zostaje on przekazany do *RA*.
  6. *RA* weryfikuje podpis zawarty w *CSR*. Wykorzystuje do tego klucz publiczny *Producenta* obecny w pamięci *RA*.
  7. *RA* przekazuje *CSR* do komponentu *CA*.
  8. *CA* na podstawie otrzymanego *CSR* generuje certyfikat dla *Czujnika* podpisany własnym kluczem prywatnym odpowiadającym kluczowi z certyfikatu `Signing CA`. Wygenerowany certyfikat zwracany jest do *RA*, a jego kopia zapisana w bazie wystawionych certyfkatów.
  9. Z komponentu *RA* za pomocą nośnika danych przeniesiony jest na *Terminal* jest certyfikat wraz z łańcuchem certyfikatów (*certificate chain*) zawierającym certyfikaty `Root CA` oraz `Signing CA` (jeśli między nimi istniały inne certyfikaty pośrednie, muszą one być również zawarte w łańcuchu).
  10. *Czujnik* otrzymuje certyfikat i sprawdza zgodność danych z danymi zawartymi w *CSR*, a także poprawność łańcucha certyfikatów względem punktu zaufania obecnego na *Czujniku*.
  11. *Czujnik* zwraca do *Terminala* informację o pomyślnie osadzonym certyfikacie.
- Alternatywnie: *Czujnik* nie posiada klucza prywatnego typu ID-based
  - W punkcie 1. *Czujnik* zwraca wiadomość o błędzie. Procedura kończy się.
- Alternatywnie: Łańcuch przedstawiony przez *Terminal* nie będzie pomyślnie zweryfikowany przez *Czujnik*

  - W punkcie 11. *Czujnik* zwraca wiadomość o błędzie. Procedura kończy się.

- Alternatywnie: *Czujnik* ma już osadzony certyfikat
  - W punkcie 2. *Czujnik* nie generuje nowej pary kluczy. Zamiast tego używa już istniejących - w *CSR* umieszczony zostaje ten sam klucz publiczny, który już znajduje się w certyfikacie.
  - W punkcie 3. *CSR* podpisywany jest standardowym agorytmem RSA, wykorzystując wygenerowany przy pierwszym żądaniu podpisania certyfikatu klucz prywatny.
  - W punkcie 6. *RA* do weryfikacji nie wykorzystuje klucza publicznego *Producenta*. Do weryfikacji podpisu użyty jest klucz publiczny zawarty w żądaniu podpisania certyfikatu.

- Alternatywnie: Weryfikacja podpisu pod *CSR* daje wynik negatywny
  - W punkcie 6. *RA* zwraca błąd i przerywa procedurę.

- Alternatywnie: *Czujnik* stwierdza nieprawdiłowości w certyfikacie
  - W punkcie 11. *Czujnik* zamiast wiadomości o sukcesie, zwraca informację o błędzie i kończy wykonywanie procedury.

3.1.4 Urząd rejestracji weryfikujący żądania certyfikatów *Czujnika* i *Terminala*

Komponent otrzymuje od *Terminala* żądania podpisania certyfikatów (*CSR*) dla *Czujnika* lub *Terminala*, a następnie weryfikuje podpisy znajdujące się na nich (typu ID-based w przypadku pierwszego żądania *Czujnika*, RSA dla *Terminala* i kolejnych żądań *Czujnika*). W przypadku pomyślnej weryfikacji, odsyła podpisane przez siebie certyfikaty urządzeniom ubiegającym się o nie.

- **Znajduje się w aplikacji:** *RA*
- **Składniki klucza użyte:** klucz publiczny *Producenta* typu ID-based (tylko w przypadku weryfikacji pierwszego żądania *Czujnika*), klucze publiczne RSA zawarte w żądaniach podpisania certyfikatu
- **Komunikuje się z komponentami:** *komponent generujący żądania certyfikatów oraz weryfikujący certyfikaty wystawione Czujnikowi* (nr 3), *komponent generujący żądania certyfikatów oraz weryfikujący certyfikaty wystawione Terminalowi* (nr 8), *Urząd certyfikacji* (nr 5)
- **Medium transmisyjne:** sieć Wi-Fi
- **Wymienione dane:** żądania certyfikatów *Czujnika* lub *Terminala*, certyfikaty *Czujnika* lub *Termianala*, łańcuch certyfikatów od `Root CA` do `Signing CA`

Przypadki użycia

Przypadek użycia tego komponentu przedstawiony jest w opisie *aplikacji generującej żądania certyfikatów oraz weryfikującej certyfikaty wystawione Czujnikowi* (nr 3) oraz *aplikacji generującej żądania certyfikatu na Terminalu oraz weryfikującej certyfikaty wystawione Terminalowi* (nr 8).

3.1.5 Urząd certyfikacji (*CA*) podpisujący certyfikaty dla *Czujnika* i *Terminala*

Komponent otrzymuje od *RA* zweryfikowane pomyślnie żądania podpisania certyfikatu (`CSR`) i podpisuje je kluczem prywatnym RSA. Udostępnia także plik z łańcuchem certyfikatów *Root CA*, własnego certyfikatu *Signing CA* i pośrednich pomiędzy nimi (jeśli takowe istnieją).

- **Znajduje się w aplikacji:** *RA*
- **Składniki klucza użyte:** klucz prywatny *CA*
- **Komunikuje się z komponentami:** *Urząd rejestracji* (nr 4)
- **Medium transmisyjne:** brak (zrealizowany w obrębie tej samej aplikacji)
- **Wymienione dane:** żądania certyfikatów *Czujnika* lub *Terminala*, certyfikaty *Czujnika* lub *Termianala*

Przypadki użycia

Przypadek użycia tego komponentu przedstawiony jest w opisie *aplikacji generującej żądania certyfikatów oraz weryfikującej certyfikaty wystawione Czujnikowi* (nr 3) oraz *aplikacji generującej żądania certyfikatu na Terminalu oraz weryfikującej certyfikaty wystawione Terminalowi* (nr 8).

3.1.6 Komponent uwierzytelniający, nawiązujący bezpieczne połączenie z *Terminalem*

Komponent umożliwia nawiazanie bezpiecznego połączenia TLS poprzez standardową procedurę uwierzytelniania TLS. Następnie następuje transmisja sekretów z *Czujnika* do *Terminala*.

- **Znajduje się w aplikacji:** *Czujnik*
- **Składniki klucza użyte:** klucz prywatny RSA *Czujnika*, klucz publiczny RSA *Czujnika*, klucz publiczny *Terminala*
- **Komunikuje się z komponentami:** *komponent uwierzytelniający, nawiązujący bezpieczne połączeniem z Czujnikiem* (nr 7)
- **Medium transmisyjne:** sieć Wi-Fi
- **Wymienione dane:** certyfikaty *Czujnika* i *Terminala*, podpisy wykonane kluczami prywatnymi RSA, sekrety transmitowane przez *Czujnik*

Przypadki użycia

Ze względu na silne powiązanie funkcjonalności tego komponentu z *aplikacją uwierzytelniającą, nawiązującą bezpieczne połaczenie z Czujnikiem*, przedstawiony przypadek użycia dotyczy obu modułów.

- Przypadek użycia: Nawiązanie bezpiecznego połączenia między *Czujnikiem* i *Terminalem*.
  1. *Terminal* nawiązuje połączenie nieszyfrowane z *Czujnikiem* i wysyła zapytanie o nawiązanie bezpiecznego połączenia.
  2. *Czujnik* wysyła zgodę na bezpieczne połączenie.
  3. Następuje obustronne uwierzytelnienie *Czujnika* i *Terminala*, używając protokołu *TLS*. W tym celu użyte zostają certyfikaty obu podmiotów, a także osadzone wcześniej *punkty zaufania*.
  4. Nawiązane zostaje bezpieczne połączenie między *Czujnikiem* i *Terminalem*. Jest gotowe na przesył danych szyfrowanym kanałem.
- Alternatywnie: *Czujnik* nie posiada osadzonego *punktu zaufania* lub certyfikatu z PKI *Klienta*
  - W punkcie 2. *Czujnik* zwraca *Terminalowi* błąd i kończy wykonywanie procedury.
- Alternatywnie: *Terminal* nie posiada osadzonego *punktu zaufania* lub certyfikatu z PKI *Klienta*
  - W punkcie 1. *Terminal* zwraca informację o błędzie i kończy wykonywanie procedury.
- Alternatywnie: Weryfikacja certyfikatu *Terminala* lub *Czujnika* zwraca negatywny wynik
  - W punkcie 3., oba komponenty zwracają błąd i kończona jest procedura.

3.1.7 Komponent uwierzytelniający, nawiązującybezpieczne połączenie z *Czujnikiem*

Komponent umożliwia nawiazanie bezpiecznego połączenia TLS poprzez standardową procedurę uwierzytelniania TLS. Następnie następuje transmisja sekretów z *Czujnika* do *Terminala*.

- **Znajduje się w aplikacji:** *Terminal*
- **Składniki klucza użyte:** klucz prywatny *Terminala*, klucz publiczny *Terminala*, klucz publiczny RSA *Czujnika*

- **Komunikuje się z komponentami:** *komponent uwierzytelniający, nawiązujący bezpieczne połączenie z Terminalem* (nr 6)
- **Medium transmisyjne:** sieć Wi-Fi
- **Wymienione dane:** certyfikaty *Czyjnika* i *Terminala*, podpisy wykonane kluczami prywatnymi RSA, sekrety transmitowane przez *Czujnik*

Przypadki użycia

Przypadek użycia tego komponentu przedstawiony jest w opisie *komponentu uwierzytelniającego, nawiązującego bezpieczne połączenie z Terminalem* (nr 6).

3.1.8 Komponent generujący żądania certyfikatu na *Terminalu* oraz weryfikujący certyfikaty wystawione *Terminalowi*

Komponent generuje parę kluczy RSA, na podstawie tego klucza publicznego i informacji o *Terminalu* generuje `CSR`, podpisuje go kluczem prywatnym RSA i wysyła do *Urzędu rejestracji*. Następnie czeka na podpisany certyfikat i weryfikuje jego poprawność.

- **Znajduje się w aplikacji:** *Terminal*
- **Składniki klucza użyte:** klucz prywatny *Terminala*, klucz publiczny *Terminala*
- **Komunikuje się z komponentami:** *Urząd rejestracji* (nr 4)
- **Medium transmisyjne:** sieć Wi-Fi
- **Wymienione dane:** *CSR* dla *Terminala*, łańcuch certyfikatów od `Root CA` do `Signing CA`

Przypadki użycia

- Przypadek użycia: Wystawienie certyfikatu *terminalowi* w PKI *Klienta*.
  1. Na *Terminalu* zainicjowany zostaje przez użytkownika proces wygenerowania żądania podpisania certyfikatu.
  2. *Terminal* generuje parę kluczy *RSA*. Klucz prywatny zapisuje w pamięci urządzenia, a klucz publiczny umieszcza w treści żądania podpisania certyfikatu.
  3. *Terminal* podpisuje *CSR* prywatnym kluczem *RSA*.
  4. Z *Terminala* za pomocą nośnika danych przeniesione zostaje żądanie podpisania certyfikatu do komponentu *RA*.
  5. *RA* weryfikuje podpis zawarty w *CSR*. Wykorzystuje do tego klucz publiczny zawarty w *CSR*.
  6. *RA* przekazuje *CSR* do komponentu *CA*.
  7. *CA* na podstawie otrzymanego *CSR* generuje certyfikat dla *Terminala* podpisany własnym kluczem prywatnym odpowiadającym kluczowi z certyfikatu *Signing CA*. Wygenerowany certyfikat zwracany jest do *RA*, a jego kopia zapisana w bazie wystawionych certyfkatów.
  8. Z komponentu *RA* na *Terminal* przeniesiony zostaje certyfikat wraz z łańcuchem certyfikatów (*certificate chain*) zawierającym certyfikaty *Root CA* oraz *Signing CA* (jeśli między nimi istniały inne certyfikaty pośrednie, muszą one być również zawarte w łańcuchu).
  9. *Terminal* przy ponownym uruchomieniu weryfikuje obecny na nim nowy certyfikat. Sprawdza zgodność kluczy, datę ważności oraz weryfikuje poprawność dołączonego łańcucha certyfikatów względem punktu zaufania znajdującego się na *Terminalu*.
- Alternatywnie: Weryfikacja podpisu z *CSR* daje wynik negatywny
  - Po punkcie 5. *RA* zwraca błąd i przerywa procedurę.
- Alternatywnie: *Terminal* stwierdza nieprawdiłowości w certyfikacie
  - W punkcie 9. *Terminal* zwraca informację o wykrytych błędach. Niemożliwe jest wtedy nawiązywanie bezpiecznych połączeń z urządzeniami.

3.2 Wymagania wstępne

Dla prawidłowego funkcjonowania opisanych komponentów spełnione muszą zostać założenia wstępne, których realizacja nie wchodzi w zakres tej pracy.

3.2.1 Numer seryjny urządzenia

Czujnik początkowo osadzony ma swój numer seryjny. Jest to jeden, jednoznacznie identyfikujący ciąg znaków. Wykorzystana technologia nie narzuca ograniczenia jego długości, natomiast w systemie posłużono się identyfikatorami długości 10 znaków.

3.2.2 PKI *Klienta*

Po stronie *Klienta* w komponencie *RA* zostały wygenerowane i osadzone w odpowiednich katalogach certyfikat `Root CA` i certyfikat `Signing CA`. Klucze odpowiadające certyfikatowi `Signing CA` użyte są później do podpisywania certyfikatów terminali i czujników u klienta.

3.2.3 Bezpieczne osadzenie punktu zaufania

Klient w momencie osadzana na *Czujniku* certyfikatu będącego punktem zaufania, przeprowadza procedurę w bezpiecznym środowisku. Przez bezpieczne środowisko rozumie się warunki, w których żadne inne, niepożądane urządzenia nie mają dostępu do sieci, w której odbywa się procedura. Zalecanym środowiskiem jest klatka Faradaya.

3.2.4 Znajomość klucza publicznego producenta

Klucz publiczny producenta i funkcje jednokierunkowe, którymi się posłużył, są znane i osadzone na komponencie będącym urzędem rejestracji u *Klienta*. W realnym zastosowaniu, taki klucz mógłby zostać pobrany z zaufanego źródła (np. ze strony internetowej producenta).

3.2.5 Zabezpieczenie klucza prywatnego *Czujnika*

Klucz prywatny wygenerowany przez *Producenta* dla *Czujnika* powinien być odpowiednio zabezpieczony na każdym etapie, co najmniej do osadzenia na *Czujniku* certyfikatu z PKI *Klienta*. W szczególności dotyczy to przeniesienia klucza za pomocą nośnika danych. Pamięć po zgraniu klucza powinna zostać sformatowana lub zniszczona tak, by niemożliwe było odczytanie z niej żadnych fragmentów klucza. Należy także zadbać, aby nie wyciekł od momentu wygenerowania klucza do osadzenia go na urządzeniu. Ponadto klucz obecny na urządzeniu powinien być niemożliwy do odczytania w inny sposób, niż przewidywany przez tą pracę. Szczegóły dotyczące realizacji dystrybucji klucza w tej pracy dostępne są w rozdziale *Algorytm podpisów ID-based wybrany do implementacji*.

3.3 Protokoły wybrane do implementacji

3.3.1 Standard X.509

Do implementacji PKI po stronie *Klienta*, wybrany został standard X.509. Jest to standard definiujący format certyfikatów kluczy publicznych, listy unieważnień i certyfikatów atrybutu. Pierwsza jego wersja powstała w 1988 roku, a obecnie jest to najpopularniejszy standard certyfikatów. Używany jest np. przy uwierzytelnianiu w protokole *TLS* czy opierającym się na nim protokole *HTTPS*. Co za tym idzie, certyfikaty wystawiane stronom internetowym także są zgodne z tym standardem.

Podstawy jego działania są zgodne z opisem przedstawionym w podrozdziałach Infrastruktura Klucza Publicznego oraz System ze standardowymi certyfikatami. X.509 zdefiniowany jest przez Międzynarodowy Związek Telekomunikacyjny (International Telecommunication Union) i opiera się na notacji ASN.1. Szczegóły techniczne, a w szczególności postacie plików konfiguracyjnych, certyfikatów i żądań podpisów, przedstawione są w dokumentach *RFC*.

W pracy standard ten wykorzystywany jest do tworzenia żądań podpisu certyfikatu i generowania certyfikatów. Są one później użyte do nawiązania bezpiecznego połączenia *TLS*. Warto zaznaczyć, że użycie własnego podpisu do żądań podpisania certyfikatu jest zgodne z dokumentami opisującymi *X.509*.

W pracy zostało pominięty element będący listą unieważnień, ponieważ nie jest on istotny dla rozważanej problematyki. W rzeczywistym zastosowaniu, taka lista byłaby prawdopodobnie niezbędna. Jej wdrożenie nie wymagałoby istotnych zmian w zaproponowanym rozwiązaniu. W szczególności zmianie nie uległyby komponenty związane z podpisami typu *ID-based*, będące rdzeniem tej pracy.

Kandydatem alternatywnym dla X.509 były certyfikaty *CVC* (Card Verifiable Certificate), będące schematem dla urządzeń o ograniczonej mocy obliczeniowej, takich jak karty elektroniczne. Przy systemie przedstawionym w tej pracy nie powinno być znaczących różnic w funkcjonalności obu certyfikatów. O wyborze standardu X.509 zadecydowała większa dostępność dokumentacji oraz bibliotek. Także późniejsze nawiązanie połączenia TLS standardowo korzysta z certyfikatów X.509, co jest ułatwieniem przy implementacji i ewentualnych modyfikacjach systemu.

3.3.2 Transport Layer Security (TLS)

Transport Layer Security (TLS) to standard protokołów mających zapewnić bezpieczną komunikację w sieci komputerowej. Jest rozwinięciem protokołu SSL (*Socket Secure Layer*), zaprojektowanego i opublikownego po raz pierwszy w 1994 roku przez firmę *Netscape Communications*. Działa on w warstwie prezentacji modelu OSI, co pozwala na zabezpieczenie warstwy aplikacji. Dlatego też wykorzystywany jest między innymi w protkołach *HTTPS*, *POP3* czy *telnet*.

Sam TSL nie przedstawia żadnych nowych algorytmów, jest natomiast zbiorem tych dotychczas znanych sposobów szyfrowania, technik i schematów kryptograficznych. Do działania wykorzystuje on zarówno szyfrowanie asymetryczne, jak i symetryczne. To pierwsze wykorzystywane jest na początku połączenia do ustalenia wspólnego klucza do dalszej komunikacji za pomocą szyfrów symetrycznych. Od momentu ustalenia klucza, cała komunikacja odbywa się po bezpiecznym, szyfrowanym kanale. Wysyłane mogą być już dane aplikacji pomiędzy klientem i serwerem. Na początku protokołu ustalane są także techniczne parametry komunikacji, takie jak algorytm szyfrowania, algorytm uwierzytelniania i integracji czy długość klucza. Do szyfrowania symetrycznego wykorzystuje się typowo DES lub AES, a do znalezienia wspólnego klucza - protokół Diffiego Hellmana.

Istotnym elementem protokołu jest uwierzytelnienie serwera i klienta. Oba elementy są opcjonalne, natomiast zazwyczaj wymagana jest co najmniej weryfikacja serwera. Odbywa się to z użyciem certyfikatom, domyślnie tych zgodnych ze standardem X.509. Przed ustaleniem wspólnego klucza, klient żąda od serwera jego certyfikat, po czym weryfikuje jego tożsamość. Serwer na tym etapie może też zażądać certyfikatu klienta i następnie go zweryfikować.

W przypadku tej pracy, połączenie TLS zostaje nawiązane między *Czujnikiem* i *Terminalem*. Do jego ustanowienia niezbędne są łańcuchy certyfikatów z PKI *Klienta*, pozwalające na weryfikację zarówno *Czujnika* (pełniącego rolę serwera), jak i *Terminala* (będącego klientem). Pozwala to na transfer np. poufnych wyników pomiarów z *Czujnika* na *Terminal* w sposób bezpieczny, przez szyfrowany kanał.

3.4 Sposób realizacji

W istocie wymienione komponenty zostały pogrupowane i zaimplementowane w 4 niezależnych aplikacjach uruchamianych z poziomu linii poleceń. Każda z nich oferuje poszczególne funkcjonalności z modułów opisanych powyżej. Na zaimplementowany system składają się:

1. Aplikacja `Factory`, odpowiadająca komponentom *Producenta*,
2. Aplikacja `Sensor`, odpowiadająca komponentom *Czujnika*,
3. Aplikacja `Terminal`, odpowiadająca procedurom *Terminala* po stronie *Klienta*,
4. Aplikacja `RA`, odpowiadająca procedurom wykonywanym przez *Urząd rejestracji* i *Urząd certyfikacji* po stronie *Klienta*.

Ponadto każdy z programów do operacji wykorzystujących schemat podpisów ID-based, korzysta z dedykowanej biblioteki. Została ona stworzona jako osobny, niezależny moduł dołączony do każdej aplikacji.

3.4.1 Aplikacja `Factory`

Aplikacja jest symulacją działań, jakie wykonać może *Producent*. Umożliwia ona:

- wygenerowanie pary kluczy typu ID-based służących do generowania kluczy podpisów dla *Czujnika*,
- wygenerowanie klucza prywatnego typu ID-based dla *Czujnika* i zapisania go do pliku.

Program udostępnia menu, z którego wybrać można dostępne funkcjonalnośći.

3.4.2 Aplikacja `Sensor`

Aplikacja jest symulacją *Czujnika*. Umożliwia ona:

- osadzenie klucza prywatnego typu ID-based,
- osadzenie punktu zaufania od *Klienta*,
- wygenerowanie i wysłanie *CSR*, a następnie osadzenie otrzymanego od *Klienta* certyfikatu
- nawiązania bezpiecznego połączenia z *Terminalem*.

Nie wymaga ona bezpośredniej interakcji. Program nasłuchuje na nadchodzące połączenie i zależnie od stanu w którym się znajduje, oferuje odpowiednie funkcjonalności. Aplikacja może znajdować się w jednym z 4 dostępnych stanów. Wszystkie stany oraz dostępne funkcjonalnośći przedstawione zostały w tabeli (tab. 3.2). Diagram stanów przedstawiony jest na rysunku 3.1.

Stan	Opis	Dostępne funkcjonalności
1	aplikacja czeka na osadzenie klucza prywatnego typu ID-based	osadzenie klucza prywatnego typu ID-based
2	aplikacja ma klucz typu ID-based i czeka na osadzenie punktu zaufania	osadzenie punktu zaufania przez *RA*
3	czeka na procedurę wystawienia certyfikatu od *Klienta*	wygenerowanie *CSR*, osadzenie nowego certyfikatu od *RA*
4	aplikacja ma certyfikat z PKI *Klienta* i jest gotowa na nawiazywanie bezpiecznych połączeń.	nawiązanie bezpiecznego połączenia TLS z *Terminalem*, wystawienie nowego certyfikatu

Tablica 3.2: Stany aplikacji `Sensor`

Urządzenie znajdujące się w danym stanie udostępnia tylko procedury odpowiednie dla tego stanu.

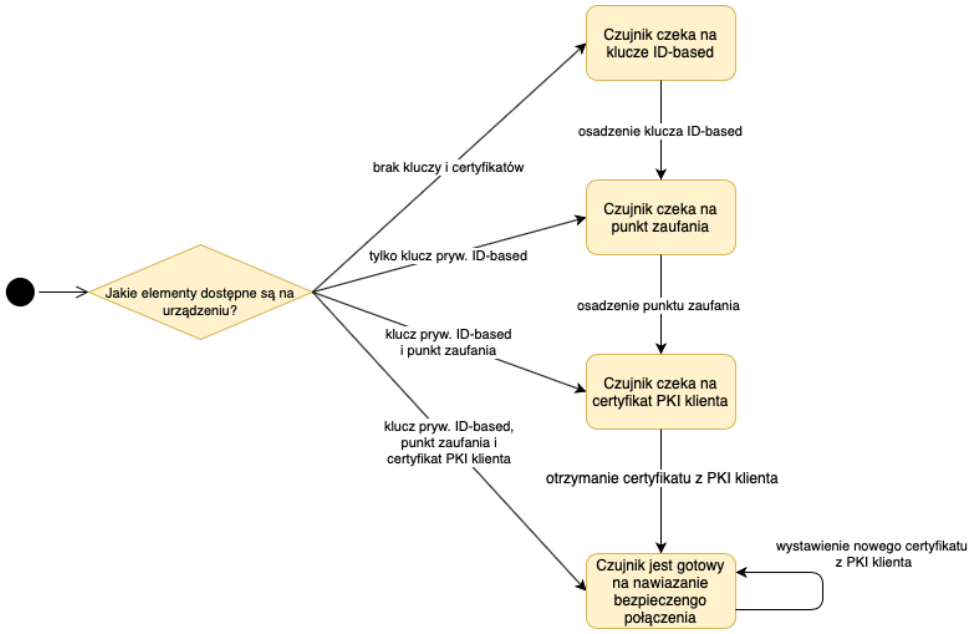
3.4.3 Aplikacja `Terminal`

Aplikacja jest symulacją działań, jakie wykonać może terminal, będący w posiadaniu *Klienta*. Umożliwia ona:

- wygenerowanie pary kluczy RSA, na ich podstawie stworzenie *żądania podpisania certyfikatu (CSR)*, a następnie osadzenie wystawionego przez *CA* certyfikatu,
- odebranie od *Czujnika* żądania podpisania certyfikatu,
- osadzanie na *Czujniku* certyfikatów wystawionych przez CA klienta,
- nawiązanie bezpiecznego połączenia i odbieranie danych od *Czujnika*.

Do nawiązania połączenia z *Czujnikiem* niezbędne jest wcześniejsze osadzenie certyfikatu wraz z łańcuchem certyfikatów dla `Signing CA`. Do uwierzytelnienia *Terminala* względem *Czujnika* niezbędny jest certyfikat *Terminala*.

Program udostępnia menu, z którego wybrać można dostępne funkcjonalnośći. Funkcja generowania żądania certyfikatu dostępna jest w każdym momencie działania, natomiast nawiązywanie bezpiecznego połączenia wymaga istnienia certyfikatu.



Rysunek 3.1: Diagram stanów aplikacji `Sensor`.

3.4.4 Aplikacja `RA`

Aplikacja jest symulacjądziałań, jakie wykonują *Urząd rejestracji* i *Urząd certyfikacji*. Umożliwia ona:

- osadzenie punktu zaufania na *Terminalu* lub *Czujniku*
- weryfikację żądania podpisania certyfikatu z podpisem ID-based lub standardowym i wygenerowanie certyfikatu podpisanego kluczem *Signing CA*,

Program udostępnia menu, z którego wybrać można dostępne funkcjonalnośći. Do weryfikacji pierwszego żądania podpisania certyfikatu od *Czujnika* niezbędne jest umieszczenie klucza publicznego ID-based producenta w pamięci na urządzeniu z aplikacją. Weryfikacja żądań z podpisami RSA i wystawianie certyfikatów odbywa się bez komunikacji z innymi komponentami.

Osadzenie kluczy odbywa się przez nawiązanie połączenia z urządzeniem (*Czujnikiem* lub *Terminalem*), a następnie wysłaniu mu certyfikatu `Root CA`.

Algorytm podpisów ID-based wybrany do implementacji

W rozdziale przedstawiony jest zaproponowany przez Adiego Shamira schemat podpisów ID-based. Omówione jest w jaki sposób tworzone i dystybuowane są klucze, jak odbywa się podpis i weryfikacja oraz w jaki sposób cały schemat zastosowano w systemie certyfikatów X.509.

4.1 Wprowadzenie do schematu podpisów

Schemat wybrany do implementacji to schemat zaprezentowany przez Adiego Shamira w pierwszej pracy na temat kryptografii ID-based z 1984 roku[24]. Przedstawiony tam opis dotyczący idei kryptografii ID-based przedstawiony został w rozdziale 2. Autor podaje również konkretną propozycję implementacji systemu podpisów. Praca zawiera opis niezbędnych algorytmów do działania takiego schematu. Adi Shamir nie przytacza jednak żadnej konkretnej, zrealizowanej implementacji ani detali technicznych. System w dużej mierze składa się z elementów podobnych do tych z RSA, więc można doszukać się tutaj analogii w działaniu obu podpisów, a także przy implementacji skorzystać z dostępnych funkcji kryptograficznych stworzonych dla RSA.

Ogólny schemat podpisu opiera się na następującym warunku pomyślnej weryfikacji podpisu:

$$s^e = i \cdot t^{f(t,m)} \mod n, \tag{4.1}$$

gdzie:

- m - wiadomość podpisywana,
- s, t - podpis,
- i - identyfikator użytkownika,
- n - iloczyn dwóch dużych liczb pierwszych,
- e - duża liczba pierwsza, będąca względnie pierwszą z $\phi(n)$,
- f - funkcja jednokierunkowa.

Poszczególne elementy równania i sposoby ich otrzymywania opisane są w kolejnych podrozdziałach.

4.2 Algorytm generowania kluczy i ich dystrybucja

Do wykonania podpisu przez użytkownika, niezbędne jest wcześniejsze wygenerowanie mu klucza prywatnego, którym wiadomość zostanie podpisana oraz klucza publicznego, służącego później do weryfikacji podpisów. Aby taki klucz otrzymać, najpierw musi istnieć *urząd zaufania* (*Trust Authority*), który taki klucz wygeneruje. Do uzyskania klucza prywatnego dla użytkownika, potrzebny jest klucz prywatny *urzędu zaufania*. Poza tym, do funkcjonowania całego systemu niezbędne jest też opublikowanie przez *urząd zaufania* jego klucza publicznego.

4.2.1 Klucze *Trust Authority*

Pierwszym elementem niezbędnym do stworzenia systemu z podpisami ID-based jest wygenerowanie odpowiednich kluczy dla *urzędu zaufania*. Zgodnie z warunkiem weryfikacji (zob. 4.1), kluczem publicznym *urzędu zaufania* są parametry n, e oraz f. Dla każdego z użytkowników systemu wartości n, e oraz opis funkcji jednokierunkowej f są identyczne. Wartość liczby n może być publiczna, natomiast sekretem pozostaje jej faktoryzacja. Klucze te są niemalze identyczne, jak te generowane w dla algorytmu RSA. Różnicą jest określenie funkcji f dla podpisów ID-based.

Schemat generowania kluczy dla *urzędu zaufania* wygląda następująco:

1. Wybierane są losowo 2 dostatecznie duże liczby pierwsze p i q. Zaleca się, aby miały one zbliżoną długość w bitach, ale były od siebie odległe jeśli chodzi o wartość.
2. Obliczona jest liczba $n = p \cdot q$
3. Obliczona jest wartosć tzw. funkcji Eulera dla n: $\phi(n) = (p-1)(q-1)$
4. Wybrana zostaje liczba e, taka że e jest względnie pierwsze z $\phi(n)$ oraz $1 < e < \phi(n)$
5. Obliczamy liczbę d, taką że $d = e^{-1} \mod \phi(n)$

Wtedy kluczem publicznym jest para (n, e) oraz wybrana przez *urząd zaufania* funkcja f, a kluczem prywatnym para (n, d).

Warto zwrócić uwagę, że równanie z punktu 5. zapisać można w postaci $d \cdot e = 1 \mod \phi(n)$. Następnie do obliczenia liczby d użyć można rozszerzonego algorytmu Euklidesa.

Klucz prywatny pozostaje sekretem *urzędu zaufania*, natomiast klucz publiczny zostaje opublikowany. W szczególności klucz publiczny trafić musi do użytkowników (w przypadku tej pracy - czujników), ponieważ wykorzystywany jest przy wykonywaniu podpisu.

4.2.2 Klucze użytkownika

Klucz publiczny użytkownika ściśle związany jest z jego tożsamością. Tak jak zostało to wcześniej opisane, musi to być unikalna, identyfikująca jednoznacznie użytkownika wartość, jak np. numer pesel czy adres email. W przypadku tej pracy jest to numer seryjny urządzenia.

Teoretycznie do wykonania i weryfikacji podpisu, jako identyfikator mógłby być użyty dosłownie numer seryjny. W celu poprawy bezpieczeństwa, stosuje się jednak rozwinięcie takiego ciągu znaków do długiego ciągu pseudolosowego, przez zastosowanie odpowiedniej funkcji jednokierunkowej. W praktyce może być to bezpieczna funkcja haszująca. Zwiększa to znacznie rozmiar przestrzeni kluczy publicznych, zwiększając tym samym bezpieczeństwo schematu. We wzorze 4.1 symbol i odpowiada rozwinięciu identyfikatora, a nie samemu identyfikatorowi. W wypadku takiego zastosowania, podmiot weryfikujący musi znać nie tyle identyfikator, co jego rozwnięcie - znana musi być zatem publicznie funkcja rozwijająca identyfikator.

Klucz prywatny urządzenia generowany jest przez *urząd zaufania*. Jest on wyrażony jako liczba g spełniająca zależność:

$$g^e = i \mod n \tag{4.2}$$

Liczba ta jest łatwa do obliczenia dla *urzędu zaufania*, natomiast nie powinna być możliwa dla nikogo innego (bez znajomości faktoryzacji liczby n).

Aby obliczyć liczbę g, zauważmy że powyższe równanie 4.2 można zapisać w postaci:

$$g = i^{e^{-1}} \mod n, \tag{4.3}$$

co z kolei sprowadza się do równości:

$$g = i^d \mod n. \tag{4.4}$$

Operacja ta jest już stosunkowo łatwa do wykonania dla *urzędu zaufania*, ponieważ liczby d i n są jej kluczem prywatnym.

Warto zauważyć także, że powyższa operacja generacji klucza odpowiada operacji deszyfrowania szyfrogramu w przypadku algorytmu RSA.

Istotnym problemem w całej procedurze jest dystrybucja klucza. Klucz prywatny użytkownika generowany jest po stronie *urzędu zaufania*. Musi zostać on przeniesiony do użytkownika bezpiecznym kanałem, gdyż jego przechwycenie dawałoby możliwość tworzenia fałszywych podpisów w imieniu użytkownika. W przypadku tej pracy zdecydowano się na przenoszenie klucza za pomocą przenośnej pamięci USB.

4.3 Algorytm podpisu

Do podpisania wiadomości m, użytkownik musi posiadać swój klucz prywatny i klucz publiczny odpowiadającego jego *urzędowi zaufania*. W pierwszej kolejności, podpisujący generuje losową liczbę r i oblicza część podpisu t:

$$t = r^e \mod n. \tag{4.5}$$

Zauważmy, że teraz warunek weryfikacji podpisu można zapisać jako

$$s^e = g^e \cdot r^{ef(t,m)} \mod n. \tag{4.6}$$

Taka postać nie umożliwia jeszcze efektywnego wyliczenia liczby s, gdyż wymagałoby to obliczenia pierwsiatka w grupie $\mod \phi(n)$. Ponieważ e jest względnie pierwsze z $\phi(n)$, możemy wyeliminować e z wykładnika po obu stronach równości. Powstała w ten sposób formuła

$$s = g \cdot r^{f(t,m)} \mod n \tag{4.7}$$

może zostać łatwo obliczona bez wykonywania wykonywania pierwiastkowania. Ostatecznie zwracanym podpisem jest para liczb (s, t).

4.4 Algorytm weryfikacji podpisu

Warunek pomyślnej weryfikacji przedstawiony został już wcześniej w tym rozdziale (zob. podrozdz. 4.1). Do weryfikacji potrzebne są zatem klucz publiczny *urzędu zaufania*, identyfikator użytkownika (opcjonalnie z funkcją skrótu), wiadomość podpisywana i naturalnie sam jej podpis. Niezależnie wykonywane są obliczenia z lewej i prawej strony równości 4.1, a ich porównanie daje wynik weryfikacji. Gdy obie strony są równe, podpis uznany jest za prawdziwy. W przeciwieństwie podpis zweryfikowany jest jako fałszywy.

4.5 Żądanie podpisania certyfikatu

Istotnym elementem tej pracy jest zintegrowanie podpisu ID-based z żądaniem podpisania certyfikatu zgodnego ze standardem X.509. Format żądania umożliwia umieszczenie własnego podpisu w żądaniu. Co istotne, nie ma to wpływu na późniejszą formę certyfikatu - podpis służy tylko do weryfikacji żądania przed wystawieniem certyfikatu. W szczególności oznacza to, że klucz publiczny obecny w żądaniu nie musi być bezpośrednio związany z samym podpisem. Dzięki temu możliwa jest weryfikacja urządzenia na podstawie klucza ID-based, by dalej posługiwać się już standardowym certyfikatem np. przy połączeniu TLS.

Tak jak przypadku certyfikatów w standardzie X.509, tak i również format żądania podpisania certyfikatu wyrażony jest w notacji ASN.1. Jego opis zawarty jest w standardzie *PKCS#10*, w dokumencie *RFC 2986*[23].

W żądaniu znaleźć się może wiele informacji dotyczących urządzenia lub producenta, takich jak dane kontaktowe, linia produkcyjna do której przypisane jest urządzenie i tym podobne. Mogą być to informacje przydatne w rzeczywistej implementacji systemu w przedsiębiorstwie, natomiast na potrzeby tej pracy zdecydowano się na umieszczenie w żądaniu tylko niezbędnych dla weryfikacji informacji. Są to przede wszystkim ID urządzenia, podpis żądania i klucz publiczny RSA. Całe żądanie przedstawione jest jako następująca struktura:

```
CertificationRequest ::= SEQUENCE {
    certificationRequestInfo      CertificationRequestInfo,
    signatureAlgorithm            AlgorithmIdentifier {{SignatureAlgorithms}},
    signature                     BIT STRING
}
```

Pole `certificationRequestInfo` zawiera wszystkie informacje, jakie zostaną umieszczone później w certyfikacie. Z kolei `signature` odpowiada za ciąg bitów będący faktycznym podpisem. W to pole trafi wartość podpisu, gdzie wiadomością podpisywaną jest zawartość obiektu `certificationRequestInfo`.

Aby w żądaniu umieścić własny podpis, należy odpowiednio opisać algorytm w polu `signatureAlgorithm`. Struktura `AlgorithmIdentifier` i `SignatureAlgorithms` w dokumencie[23] wygląda następująco:

```
AlgorithmIdentifier {ALGORITHM:IOSet } ::= SEQUENCE {
    algorithm           ALGORITHM.&id({IOSet}),
    parameters          ALGORITHM.&Type({IOSet}{@algorithm}) OPTIONAL
}

SignatureAlgorithms ALGORITHM ::= {
    ... -- add any locally defined algorithms here -- }
```

Stwierdzenie zawarte wewnątrz `SignatureAlgorithms` w dokumencie w wolnym tłumaczeniu oznacza "tutaj dodaj lokalnie zdefiniowane algorytmy". Wskazuje to zatem na to, że oficjalny format dopuszcza istnienie innych algorytmów podpisu, niż te predefiniowane w popularnych implementacjach.

W polu `algorithm` znajdować się powinno odpowiednie OID (*Object Identifier*) obiektu odpowiadającego kombinacji funkcji haszującej i podpisującej użytej w żądaniu. Stworzenie w pełni prawidłowego żądania podpisania certyfikatu z podpisem typu ID-based wymagałoby umieszczenia OID (object identifier) obiektu, odpowiadającego właśnie algorytmowi ID-based i funkcji haszującej. Nie udało się jednak odnaleźć żadnej pozycji odpowiadającej tym schematom podpisu, stąd w pracy, na potrzeby symulacji wykorzystano przykładowy, nieadekwatny numer OID. Komponent obsługujący werfikację żądań certyfikatu został także zmodyfikowany tak, by to wybrane OID dekodować jako algorytm ID-based.

W przypadku algorytmu podpisu ID-based, opcjonalne pole `parameters` zawierać będzie odpowiednie parametry. Pierwszy z nich - `maskGenAlgorithm` - zawiera OID oraz parametry funkcji jednokierunkowej użytej w podpisie. W przypadku tej implementacji ta sama funkcja wykorzystana jest jako funkcja jednokierunkowa `f` ze schematu podpisu opisanego powyżej oraz do rozwinięcia identyfikatora urządzenia przed wykonaniem na nim operacji. Użyto w tym celu funkcji `SHAKE256`[16], do której opisu nie są wymagane żadne parametry. W polu `idExtLength` znajduje się długość w bitach, do jakiej rozwinięty zostaje za pomocą powyższej funkcji numer ID urządzenia przy korzystaniu z algorytmów podpisu. Ostatni parametr `fOutputLength` określa długość (w bitach) ciągu zwracanego przez funkcję jednokierunkową `f` z algorytmu podpisu. Wartości obu zmiennych w przypadku zaimplementowanej symulacji wynoszą 256 bitów. Struktura parametrów tego algorytmu w notacji ASN.1 wygląda się następująco:

```
Shamir-IDB-sig-params ::= SEQUENCE {
    maskGenAlgorithm     AlgorithmIdentifier {{MaskGenAlgorithms}}
                         DEFAULT shake256,
    idExtLength          INTEGER DEFAULT 256,
    fOutputLength        INTEGER DEFAULT 256
}


MaskGenAlgorithms ALGORITHM ::= {
    ... -- add any locally defined algorithms here -- }
```

W polu `signature` umieszczona zostaje zserializowana para liczb stanowiąca podpis. Numer seryjny urządzenia zostaje umieszczony w polu `Common Name`, będącym jednym z pól wewnątrz elementu `certificationRequestInfo`. Alternatywnie można było utworzyć własne rozszerzenie certyfikatu (tzw. *extension*), w którym umieszczony zostałby właśnie numer seryjny urządzenia. W przypadku rozważanym w tej pracy nie ma jednak takiej konieczności.

4.6 Bezpieczeństwo podpisu

Bezpieczeństwo przedstawionego przed Adiego Shamira schematu podpisów nie zostało do tej pory istotnie podważone. Kluczem do analizy jest fakt, że ich bezpieczeństwo opiera się w głównej mierze na bezpieczeństwie RSA. Tak jak zostało wcześniej przytoczone, algorytmy generowania klucza czy pozyskiwania klucza prywatnego użytkownika są ściśle związane z algorytmami RSA.

Dla każdej wiadomości istnieje duża liczba możliwych podpisów (s, t), ale ich zagęszczenie jest tak niskie, że znalezienie podpisu przez zgadywanie losowych podpisów jest bardzo malo prawdopodobne.

Próby ustalenia jednej z liczb (s, t) i obliczenia drugiej wymaga z kolei obliczenia pierwiastków modulo, co uważa się również za bardzo trudny obliczeniowo problem. Kryptoanalityk nie powinien być także w stanie uzyskać klucza prywatnego g obserwując nawet bardzo dużą liczbę podpisów. Warunkiem tego jest jednak niekorzystanie przez użytkownika wielokrotnie z tej samej wartości losowej r. Istotne jest także użycie bezpiecznej funkcji jednokierunkowej f.

Z punktu widzenia implementacji, odpowiedni dobór liczb pierwszych, wykorzystanych przy tworzeniu klucza jest także decydujący w kwestii bezpieczeństwa. W stworzonej symulacji, za te newralgiczne elementy odpowiadają powszechnie uznane za bezpieczne bibiloteki obsługujące RSA. Więcej szczegółów na ten temat znajduje się w dalszej części pracy.

Implementacja systemu

W rozdziale przedstawione są wykorzystane technologie i narzędzia. Opisane zostały etapy pracy, napotkane trudności implementacyjne oraz wykonane testy.

5.1 Opis technologii

W skład całej symulacji wchodzą 4 aplikacje napisane w języku programowania *Python* w wersji 3[4]. Dodatkowo napisany został zbiór funkcji obsługujących podpisy ID-based dołączony w formie biblioteki.

5.1.1 Wykorzystane biblioteki

W pracy do zaimplementowania większości funkcji kryptograficznych użyto znanych bibliotek dla języka `Python`. Do generowania kluczy RSA, a także kluczy ID-based producenta i klucza prywatnego użytkownika wykorzystano bibliotekę `pycrypto`[15]. Funkcje haszujące używane na różnych etapach w systemie pochodzą z biblioteki `Cryptodome`[10]. Obsługa połączenia TLS zrealizowana została przy użyciu pakietu `ssl`[18].

Części pracy wykorzystujące standardowe certyfikaty X.509 obsługiwane są przez funkcje z biblioteki `OpenSSL`[1], będącej interfejsem dla zaimplementowanej w języku `C` biblioteki o tej samej nazwie. Do tworzenia i weryfikacji żądań podpisania certyfikatu z podpisem ID-based użyto pakietu `asn1crypto`[8] i opartego na niej `csrbuilder`[11]. W celu dodania nowego podpisu konieczne było jednak stworzenie nowych klas obiektów dziedziczących funkcjonalności po tych z biblioteki `asn1crypto`. Tak jak zostało to wcześniej wspomniane, żadna ze znalezionych bibliotek nie zawierała implementacji podpisów ID-based. Do weryfikacji poprawności łańcuchów certyfikatów użyto biblioteki `certvalidator`[9].

Do serializacji obiektów posłużono się pakietem `pickle`[14]. Komunikacja wymagała skorzystania także z biblioteki `socket`[17] do obsługi połączenia i `json`[13] do tworzenia wysyłanych wiadomości. Testy jednostkowe napisane zostały z wykorzystaniem pakietu `unittest`[19]. Wykorzystano także kilka niewspomnianych wyżej bibliotek do typowych dla aplikacji w języku `Python` operacji, takich jak obługa błędów, wyrażenia regularne czy zmiany typu kodowania danych.

5.1.2 Sposoby komunikacji między komponentami

Medium komunikacyjnym w większości interakcji między komponentami jest sieć WiFi. Stąd do nawiązywania takich połączeń między aplikacjami użyto gniazd (ang. *socket*). Zależnie od danej funkcjonalności, komponent może być serwerem lub klientem. Połączenie zostaje nawiązanie przez podanie klientowi adresu IP aplikacji będącej serwerem. Początkowe fazy komunikacji odbywają się przez wysłanie wiadomości w formacie `json`. Serwer po nawiązaniu połączenia wysyła wiadomość powitalną i oczekuje na polecenia od klienta. Następnie, zależnie od stanu aplikacji, serwer odpowiada na żądania. W przypadku połączenia terminala z czujnikiem posiadającym osadzony punkt zaufania, istniejące łącze zostaje zamienione na szyfrowany kanał TLS.

Do osadzania kluczy prywatnych ID-based na urządzeniach wykorzystywany jest zewnętrzny nośnik pamięci, np. pendrive. Po podłączeniu takiego nośnika do urządzenia, skopiowany zostaje z niego klucz prywatny do odpowiedniej lokalizacji na czujniku. Jest to umotywowane względami bezpieczeństwa. Istnieją alternatywy dla takiego rozwiązania, jak chociażby nadanie klucza przez fizyczne łącze sieciowe. Ze względu na dostępność i łatwość implementacji zdecydowano się jednak użyć nośnika pamięci.

5.2 Przebieg implementacji

Ze względu na fakt, że autorem jest jednoosobowy zespół, przy implementacji nie stosowano się bezpośrednio do żadnej znanej metodyki. Przebieg pracy był inkrementacyjny i opierał się na kilkudniowych sprintach z wyznaczonymi kamieniami milowymi. Każda iteracja skutkowała istnieniem funkcjonalnej części systemu. Przed przystąpieniem do samej implementacji dokonano także analizy istniejących rozwiązań i rodzajów podpisów ID-based. Na jej podstawie wybrano odpowiedni algorytm do zaimplementowania.

5.2.1 System kontroli wersji

Do pracy nad projektami, których realizacja przekracza kilka tygodni zaleca się używania systemu kontroli wersji (ang. *Version Control System*, *CVS*). Służy on do zachowywania wszystkich plików projektu wraz z historią zmian. Dzięki niemu w dowolnym momencie pracy można wrócić do wcześniej nadpisanej wersji projektu. Rozwiązanie to pozwala również prześledzić historię zmian czy autorów poszczególnych części kodu.

Do implementacji z tej pracy dyplomowej wykorzystano jeden najpopularniejszych rozproszonych systemów kontroli wersji - Git[6]. Jest to narzędzie stworzone przez Linusa Torwalda, mające służyć pierwotnie jako wsparcie wspomagające rozwój jądra *Linux*. System ten implementuje mechanizm gałęzi, co umożliwia tworzenie tymczasowych odgałęzień na pojedyncze zadania. Poza tym repozytoria dostępne są offline u każdego programisty. Do zapisywania zmian nie jest konieczne połączenie z siecią, a zmiany mogą być później wymieniane między lokalnymi repozytoriami. W tej pracy posłużono się serwisem GitHub[7], wspomagającym użycie tego systemu.

5.2.2 Etapy implementacji

Kamienie milowe zostały określone w następujący sposób:

1. Stworzenie biblioteki obsługującej podpisy ID-based wraz z podstawowymi testami
2. Stworzenie funkcjonalnej aplikacji `Factory`
3. Napisanie szkieletu aplikacji komunikujących się po gnieździe (aplikacji `Terminal`, `Sensor` oraz `RA`)
4. Stworzenie aplikacji `Terminal` generującej i osadzającej klucze ID-based na czujniku
5. Stworzenie plików konfiguracyjnych oraz certyfikatów w standardzie X.509 dla PKI klienta
6. Stworzenie komponentu umieszczającego podpis ID-based w żądaniu podpisania certyfikatu X.509
7. Zaprogramowanie przepływu żądań podpisania certyfikatów oraz certyfikatów między `Terminalem`, `Czujnikiem` i `RA`
8. Implementacja bezpiecznego połączenia między `Terminalem` i `Czujnikiem`
9. Skonfigurowanie środowiska testowego, testy funkcjonalne i jednostkowe powstałych komponentów

5.2.3 Napotkane trudności

Do największych wyzwań należało zaimplementowanie biblioteki obsługującej podpisy ID-based. Adi Shamir w swojej pracy nie przedstawił szczegółów dotyczących konkretnych implementacji. Do rozwiązania były problemy takie jak dobór odpowiednich funkcji jednokierunkowych, generowanie kluczy (w tym także liczb pseudolosowych), dobranie długości poszczególnych zmiennych i konwersja między typami zmiennych. Mając na uwadze bezpieczeństwo, poprawność każdego z tych kroków musiała zostać szczegółowo zweryfikowana.

Drugim największym wyzwaniem było osadzenie podpisu ID-based w żądaniu podpisania certyfikatu w standardzie X.509. Istniejące bilbioteki do obsługi certyfikatów w tym standardzie (np. `OpenSSL`) rzetelnie weryfikują sposób podpisu i nie dopuszczają istnienia innych niż predefiniowane algorytmów podpisu. Jest to problem zarówno przy generowaniu żądania, jak i przy późniejszym ich weryfikowaniu i generowaniu certyfikatu. Z tego względu konieczne było użycie biblioteki do tworzenia struktur w notacji ASN.1. Tutaj dodatkową trudnością była także jej znikoma dokumentacja.

5.3 Omówienie kodów źródłowych

Kluczowym elementem pracy jest implementacja podpisów ID-based. Tak jak wcześniej wspomniano, klucze główne ID-based mogą być takie same jak klucze RSA. Stąd do ich wygenerowania użyto istniejące biblioteki `pycrypto`. Podobnie w przypadku generowania klucza prywatnego dla urządzenia, przy czym w tym przypadku wykonana została operacja deszyfrowania ze schematu RSA. Wszystkie funkcje generujące klucze przedstawione są w kodzie źródłowym 5.1.

Kod źródłowy 5.1: Funkcje generujące klucze ID-based.

```
def generate_RSA_key_pair(bits=2048, e=65537):
    '''
    Generate an RSA keypair with an exponent of 65537 in PEM format.
    Keys can be used as master ID-based key pair.

    Args:
        bits: the key length in bits
        e: part of the public key

    Returns:
        tuple: private key and public key
    '''
    new_key = RSA.generate(bits, e=65537)
    public_key = new_key.publickey()
    private_key = new_key

    return private_key, public_key

def get_device_public_key(i, m_pub_key):
    '''
    Derive device public key

    Args:
        i: user's identity
        m_pub_key: ID-based master public key

    Returns:
        tuple: all three parts of user's public key

    '''
    return expand_identity(i), m_pub_key.n, m_pub_key.e


def get_device_private_key(key, i):
    '''
    Gets user private key. Uses RSA functions.

    Args:
        key: ID-based master private key
        i: user identity

    Returns:
        tuple: device private key
    '''
    g = key.decrypt(expand_identity(i))
    return g, key.n
```

Do efektywnego wykonywania podpisów i ich weryfikacji, do operacji potęgowania wykorzystano funkcję `pow`, umożliwiającą potęgowanie modulo. Poza tym schemat jest implementacją oryginalnej propozycji przedstawionej przez Adiego Shamira.

Kod źródłowy 5.2: Funkcje obsługujące podpisy ID-based.

```
def sign(g, m, n, e):
    '''
    Sign message with device private key

    Args:
        g: private ID-based key
        m: message to sign
        n, e: parts of ID-based public key

    Returns:
        tuple: (s, t) - signature of the message m
    '''
    r = get_random_num()
    t = pow(r, e, n)
    s = g * pow(r, f(t, m), n) % n
    return s, t

def verify(s, i, t, m, n, e=65537, id_ext_len=256, f_out_len=256):
    '''
    Verification of the signature

    Args:
        m: message
        s, t: signature
        i: user identity
        n: product of two large primes
        e: large prime relatively prime to phi(n)
        f: one way function

    Returns:
        bool: True if sign matches message, false otherwise
    '''
    return pow(s, e, n) == expand_identity(i, n=id_ext_len)
            * pow(t,f(t,m, out_len=f_out_len), n) % n
```

Nieprzedstawione tutaj funkcje jednokierunkowe, takie jak funkcja `f` czy `expand_identity`, opierają się na funkcji haszującej `SHAKE256`[16]. Pozwala ona uzyskać wynik o określonej przez użytkownika długości, z maksymalnym poziomem bezpieczeństwa określonym na 256 bitów. Funkcja ta może być zastąpiona inną bezpieczną funkcją haszującą. Zależnie od długości jej wyniku, może to jednak wymagać również zmian w innych elementach systemu.

5.4 Testy

Zarówno podczas, jak i po implementacji przeprowadzano szereg testów funkcjonalnych oraz jednostkowych. Te drugie zostały napisane do krytycznej i najistotniejszej dla całego systemu biblioteki do podpisów ID-based. Testy funkcjonalne przeprowadzone zostały na gotowej implementacji.

5.4.1 Testy jednostkowe

Do zrealizowania testów jednostkowych użyto biblioteki `unittest`. Testy weryfikują poprawne działanie kluczowych funkcji podpisu i generowania klucza oraz pomocniczych funkcji takich jak np. odpowiednie zapisywanie i odczytywanie kluczy z pliku. Sprawdzono także między innymi czy generowane podpisy nie są jednakowe dla różnych wiadomości i czy weryfikacja dla urządzeń z różnym ID będzie miała negatywny rezultat. Każdy z zaimplementowanych testów zwrócił pozytywny wynik.

5.4.2 Testy funkcjonalne

Gotowy system został przetestowany manualnie z wykorzystaniem minikomputera *Raspberry Pi 4b* z systemem *Ubuntu Server 19.10* jako czujnika i laptopa *MacBook Air 7,2* z systemem operacyjnym *macOS Mojave v. 10.14.5* i wersją języka `Python 3.8.0` na obu urządzeniach. Urządzenia znajdowały się w jednej sieci WiFi. Urządzenie *Raspberry Pi* obsługiwane było zdalnie za pomocą połączenia SSH.

W pierwszej kolejności przetestowano scenariusz kończący się sukcesem. Na komputerze, za pomocą aplikacji *Producent* wygenerowano klucz prywatny i otrzymany z ID klucz publiczny dla czujnika. Oba klucze zostały przeniesione na Raspberry Pi za pomocą przenośnej pamięci USB. Następnie, za pomocą połączenia SSH uruchomiono aplikację i testowano po kolei każdą z funkcjonalności. Test zakończył się ustanowieniem bezpiecznego połączenia TLS i odczytaniem transmitowanym z aplikacji `Terminal` danych.

Niezależnie wykonano także wiele prób z założenia kończących się porażką. Testowano między innymi próbe nawiązania połączenia bez osadzonych punktów zaufania, wykorzystanie nieważnych certyfikatów, certyfikatów spoza PKI klienta, próby wielokrotnego osadzania punktów zaufania czy kluczy ID-based. Każda z prób kończyła się zwróceniem spodziewanego komunikatu o błędzie.

Wykonane testy funkcjonalne potwierdziły działanie systemu oraz oczekiwaną obsługę błędów.

Wymagania dotyczące instalacji i wdrożenia systemu

W tym rozdziale omówione są zawartości oprogramowania oraz założenia co do jego instalacji. Przedstawione zostały także możliwe konfiguracje systemu.

6.1 Wymagania instalacji

System przeznaczony jest do użytkowania na dystrybucjach Linuksa oraz systemie *macOS*. Do działania systemu najpierw niezbędne jest wygenerowanie odpowiednich kluczy i certyfikatów, a także spełnienie wymagań systemowych. Dla prawidłowego działania powinny zostać spełnione także warunki opisane w rozdziale 3.2 tej pracy.

6.1.1 Klucze ID-based producenta

W założeniu systemu, producent posiada własny klucz prywatny i publiczny. Są to klucze identyczne jak w systemie RSA, dlatego do ich wygenerowania można użyć bibliotek czy programów odpowiednich dla tego algorytmu, takich jak `OpenSSL`. Klucze te muszą być zakodowane w formacie `PEM` i umieszczone w katalogu `keys` aplikacji `Factory` odpowiednio z nazwami `priv_key.key` i `pub_key.pem`. Skrypt `generate_master_idb_keys.py` generujący przykładowy zestaw takich kluczy znajduje się w katalogu z aplikacją `Factory` na płycie CD.

6.1.2 PKI klienta

Klient, czyli podmiot posługujący się aplikacjami `RA` i `Terminal` powinien posiadać strukturę klucza publicznego w standardzie X.509. W jej skład musi wchodzić co najmniej *Urząd Certyfikacji*. W zalecanym (a także testowym) środowisku proponowana jest hieharchia składająca się z podpisanego przez siebie (*self-signed*) certyfikatu `Root CA` oraz certyfikatu `Signing CA`, który służy do wystawiania wszystkich innych certyfikatów. Do podpisywania certyfikatów przez `Signing CA` niezbędny jest też klucz prywatny odpowiadający temu certyfikatowi. Powinien on być zabezpieczony hasłem, które będzie podawane do aplikacji *RA* każdorazowo przed wystawieniem nowego certyfikatu. Przy domyślnej konfiguracji, certyfikat główny i podpisujący powinny znaleźć się kolejno w plikach `root-ca.crt` i `signing-ca.crt`.

Konieczne jest również utworzenie pliku z łancuchem certyfikatów zawierającym wszystkie certyfikaty od głównego, podpisanego przez siebie `Root CA` do podpisującego inne certyfikaty `Signing CA`. Powinny one zostać umieszczone w pliku o nazwie `signing-ca.ca-bundle`.

Poza tym, do pomyślnej weryfikacji żądań wystawionych przez czujniki niezbędna jest znajomość klucza publicznego producenta. Plik z kluczem `pub_key.pem` również musi znaleźć się w odpowiednim katalogu aplikacji *RA*. W rzeczywistym systemie taki klucz mógłby być dostarczony przez producenta np. za pośrednictwem strony internetowej.

Przykładowe polecenia służące do wygenerowania takich certyfikatów oraz niezbędne dla nich pliki konfiguracyjne znajdują się na płycie CD w katalogu `Sample_PKI`.

Domyślnie katalogi i klucze powinny tworzyć strukturę opisaną następującym drzewem katalogów:

```
certs
├── ca
│   ├── signing-ca.ca-bundle
│   ├── signing-ca.crt
│   ├── root-ca.crt
│   ├── signing-ca
│   │   └── private
│   │       └── signing-ca.key
│   └── root-ca
│       └── private
│           └── root-ca.key
├── producents
│   └── Thesis_Producent
│       └── pub_key.pem
└── signed
```

Certyfikat *Root CA* pod nazwą `root-ca.crt` oraz łańcuch certyfikatów `signign-ca.ca-bundle` muszą trafić także do folderu `certs/anchor/` w aplikacji `Terminal`.

6.1.3 Instalacja środowiska

Do uruchomienia aplikacji niezbędne jest posiadanie w systemie interpretera dla języka `Python` (zalecana jest wersja `3.7.0` lub nowsza). Wiele dystrybucji systemów UNIX i Linux posiada już domyślnie zainstalowany taki interpreter. Instrukcje dotyczące instalacji tego środowiska znaleźć można na oficjalnej stronie języka ([3]).

Do instalacji niezbędnych bibliotek z najlepiej użyć menedżera pakietów `pip`. Standardowo jest on instalowany razem z interpreterem języka `Python`. Instrukcje dotyczące instalacji znaleźć można na stronie twórców ([2]).

Każda aplikacja w katalogu głównym posiada plik `requirements.txt`, zawierający listę niezbędnych do jego uruchomienia bibliotek. Mogą być one zainstalowane poprzez wywołanie polecenia `pip install -r requirements.txt`. Operacja ta powinna być wykonana przed pierwszym uruchomieniem każdej z aplikacji.

6.2 Instrukcja użytkowania

Aby uruchomić dowolną z aplikacji należy uruchomić w interpreterze języka `Python` skrypt `run.py` znajdujący się w katalogu każdej z aplikacji. Może być to wywołane instrukcją `python3 run.py`. Interakcja z programami opiera się na wprowadzaniu znaków na standardowe wejście aplikacji i ich zatwierdzaniu.

6.2.1 Generowanie i osadzenie kluczy ID-based dla czujnika

Aby wygenerować klucze ID-based dla czujnika należy postępować zgodnie z opisanymi krokami:

1. Uruchom aplikację `Factory`
2. Gdy wyświetli się menu, wybierz opcję 1. wpisując liczbę ”1”.
3. Aplikacja spyta o ID urządzenia. Wpisz ID, dla którego ma być wygenerowany klucz prywatny. Domyślnie ID musi mieć dokładnie 10 znaków długości.
4. Aplikacja powinna zwrócić informację o powodzeniu generowania klucza. Wyjdź z aplikacji wpisując liczbę ”2”
5. Klucze prywatny i publiczny urządzenia znajdują się teraz w katalogu `dev_keys/[ID_urządzenia]/` pod nazwami odpowiednio `id_priv.key` i `id_pub.pem`. Klucze te teraz powinny zostać w bezpieczny sposób przeniesione do folderu `certs/ID/` aplikacji `Sensor`.

Przeniesienie klucza prywatnego urządzenia na *Czujnik* w bezpieczny sposób jest kluczowe dla prawidłowego funkcjonowania systemu. W symulacji działania systemu posłużono się nośnikiem pamięci USB.

W pliku `id_list.txt` w katalogu `dev_keys` aplikacji `Factory` znajduje się lista ID urządzeń, dla których wygenerowane zostały klucze ID-based.

6.2.2 Osadzenie punktu zaufania na czujniku

Aby osadzić punkt zaufania na czujniku należy postępować zgodnie z opisanymi krokami:

1. Uruchom aplikację `Sensor` oraz aplikację `RA`
2. Aplikacja `Sensor` czeka na połaczenie. W aplikacji *RA* wybierz opcję "1". Następnie należy wprowadzić kolejno adres IP oraz port czujnika (domyślnie port to `8888`).
3. Obie aplikacje powinny teraz zwrócić informację o pomyślnym wykonaniu operacji osadzenia punktu zaufania. Żeby wyjść z aplikacji `RA`, wybierz teraz opcję "4".

Należy pamiętać, że punkt zaufania może być osadzony na urządzeniu tylko raz. Próba wielokrotnego osadzania skutkuje wypisaniem błędu po stronie *RA*.

Kluczowe dla systemu jest przeprowadzenie tej operacji w bezpiecznym środowisku. Żadne inne urządzenia nie powinny mieć dostępu do tej sieci w chwili wykonywania tych procedur. Zalecanym środowiskiem jest klatka Faradaya.

6.2.3 Wystawienie certyfikatu dla czujnika

Aby wystawić certyfikat z PKI klienta czujnikowi, należy postępować zgodnie z opisanymi krokami:

1. Uruchom aplikację *Sensor* oraz aplikację `Terminal`
2. Aplikacja *Sensor* czeka na połaczenie. W aplikacji *Terminal* wybierz opcję "3". Następnie należy wprowadzić kolejno adres IP oraz port czujnika (domyślnie port to `8888`).
3. Żądanie z czujnika powinno teraz zostać przesłane do aplikacji *Terminal* i zostać zapisane w folderze `csr/[ID_czujnika]/`. Żądanie te należy przenieść teraz do pamięci urządzenia z aplikacją `RA` i umieścić w folderze `csr`.
4. Uruchom aplikację `RA`. Z menu wybierz opcję "3".
5. W aplikacji `RA` wyświetlone zostaną nazwy plików z rozszerzeniem `.csr` znajdujące się w folderze `dev_csr`, które zostały rozpoznane przez aplikację. Wpisz numer odpowiadający nazwie wybranego żądania.
6. Jeśli klucz prywatny `Signing CA` zabezpieczony jest hasłem, należy je podać, gdy aplikacja spyta o jego wprowadzenie.
7. Certyfikat powinien zostać automatycznie wygenerowany i zapisany w folderze `certs/signed` aplikacji *RA*. W folderze z aplikacją `Terminal` utwórz folder `certs/dev_certs/[ID_czujnika]/`, a następnie za pomocą nośnika pamięci, przenieś utworzony certyfikat oraz łańcuch certyfikatów odpowiadający certyfikatom od `Root CA` do `Signing CA` (plik `signing-ca.ca-bundle`) z aplikacji *RA* do tego folderu.
8. W aplikacji `Terminal` wybierz opcję "4".
9. Wpisz nr IP oraz port odpowiadający czujnikowi.
10. Aplikacje powinny się teraz połączyć i automatycznie osadzić wybrany certyfikat na czujniku.

Warunkiem koniecznym do pomyślnego działania powyższych kroków jest pomyślna weryfikacja podpisu pod żądaniem podpisania certyfikatu, wygenerowanym przez *Czujnik*. W razie niepowodzenia tej operacji, obie aplikacje zwrócą informację o błędzie.

6.2.4 Wystawienie certyfikatu dla terminala

Aby wystawić certyfikat z PKI klienta terminalowi, należy postępować zgodnie z opisanymi krokami:

1. Uruchom aplikację `Terminal`.
2. W aplikacji `Terminal` wybierz opcję "1". Wygenerowane zostanie teraz żądanie podpisania certyfiatu, które trafi do folderu `csr`.
3. W aplikacji `RA` wybierz opcję "3".
4. W aplikacji `RA` wyświetlone zostaną nazwy plików z rozszerzeniem `.csr` znajdujące się w folderze `dev_csr`, które zostały rozpoznane przez aplikację. Wpisz numer odpowiadający nazwie wybranego żądania.
5. Jeśli klucz prywatny `Signing CA` zabezpieczony jest hasłem, należy je podać, gdy aplikacja spyta o jego wprowadzenie.
6. Certyfikat powinien zostać automatycznie wygenerowany i zapisany w folderze `certs/signed` aplikacji `RA`. Za pomocą nośnika pamięci, przenieś powstały certyfikat do katalogu `certs` aplikacji `Terminal`
7. Po ponownym uruchomieniu aplikacji *Terminal*, automatycznie załadowany zostanie nowy certyfikat.

6.2.5 Transmisja danych bezpiecznym połączeniem między czujnikiem i terminalem

Aby otrzymać bezpiecznym kanałem dane z aplikacji *Czujnik* na aplikacji *Terminal* należy postępować zgodnie z opisanymi krokami:

1. Uruchom aplikację `Sensor` oraz `Terminal`
2. Aplikacja *Sensor* czeka na połączenie. W aplikacji `Terminal` wybierz opcję "2". Następnie należy wprowadzić kolejno adres IP oraz port czujnika (domyślnie port to `8888`).
3. Następuje teraz nawiązanie połączenia TLS między *Czujnikiem* a *Terminalem*. Jeśli operacja zakończy się sukcesem, na standardowe wyjście aplikacji `Terminal` wypisywane będą dane otrzymane od aplikacji `Sensor`. Aby przerwać otrzymywanie danych, użyj kombinacji klawiszy `CTRL+C`.

Wnioski oraż możliwości rozwoju systemu

W rozdziale został określony stan zakończonych prac projektowych i implementacyjnych. Zaznaczone jest, które z założonych funkcjonalności udało się zrealizować. Opisane są także możliwości rozszerzenia i modyfikacji systemu.

7.1 Wnioski

Celem pracy było przedstawienie systemu, w którym certyfikaty instalowane nowym urządzeniom oparte są o żądania certyfikatów z podpisami typu ID-based. Wiązało się to z koniecznością wyboru odpowiedniego algorytmu i opracowaniem projektu systemu wraz z testami. Cel ten został osiągnięty zgodnie ze wstępnymi założeniami. Oznacza to, że zaproponowane rozwiązanie może być zrealizowane w praktyce. Przedstawiona w tej pracy alternatywa dla systemu certyfikatów X.509 pozwala znacząco zredukować zakres obowiązków, a co za tym idzie koszta, jakie ponieść musiałby producent stosując tradycyjne rozwiązania.

Istotny jest też fakt, że osadzenie podpisu typu ID-based na żądaniu podpisania certyfikatu jest zgodne ze standardem X.509. W szczególności dzięki formatowi żądania, istniejące biblioteki oraz narzędzia są w stanie wygenerować z niego certyfikat, pomimo niezidentyfikowanego przez nie algorytmu podpisu. Oznacza to, że z takich żądań mogą być wystawione certyfikaty bez konieczności modyfikacji istniejących implementacji.

Niestety dzieje się tak tylko jeśli podpis umieszczony na żądaniu nie jest weryfikowany. Do jego weryfikacji konieczne jest użycie algorytmów dla podpisu ID-based. Chociaż dokumentacja standardu X.509 dopuszcza zastosowanie lokalnie zdefiniowanych algorytmów, to żadne ze znalezionych narzędzi i bibliotek nie są otwarte na takie modyfikacje. W szczególności, autorowi nie udało znaleźć się dokumentacji opisującej ewentualnie wdrożenie własnych podpisów w istniejących rozwiązaniach.

7.2 Możliwości rozwoju systemu

Z uwagi na fakt, że opisany system jest tylko symulacją demonstrującą rozwiązanie przedstawionego problemu, rzeczywista implementacja może być dużo bardziej rozbudowana i skomplikowana.

Przede wszystkim infrastruktura klucza publicznego klienta może być mocniej rozwinięta. Prowadzona powinna być lista unieważnionych certyfikatów (CRL), a same certyfikaty mogłyby być rozbudowane o kolejne pola wyznaczające np. nazwę działu czy linii produkcyjnej, na której urządzenie ma pracować. Są to jednak aspekty mocno zależne od wymagań konkretnych przedsiębiorstw i nie mające znaczenia dla prezentowanych rozwiązań, stąd zostały pominięte w tej implementacji. Dodatkowo, urządzenia mogłyby oferować możliwość zmiany punktu zaufania na certyfikat z nową parą kluczy, używając tzw. *cross-certificates*.

Innym pomysłem na rozszczerzenie systemu może być dodanie czasu ważności do podpisów ID-based. W podstawowym, przedstawionym w tej pracy rozwiązaniu, raz wygenerowane klucze prywatne dla urządzeń dają im nieogarniczoną czasowo możliwość tworzenia prawidłowych podpisów. Wymagane mogłoby być ich wygaśnięcie, jeśli urządzenie zmieniłoby ID lub producent ustanowił nowe klucze główne. Można stworzyć system, w którym klucze prywatne urządzeń mają określoną datę ważności. W tym celu, podczas generowania klucza do ID dodawany jest element związany z czasem. Na przykład, dodając do ID kod oznaczający obecny miesiąc, podpisy wystawione po jego upływie nie będą weryfikowane jako prawidłowe.

Bibliografia

[1] Openssl library - biblioteka kryptograficzna. URL: https://www.openssl.org.

[2] Pip - instrukcja instalacji. URL: https://pip.pypa.io/en/stable/installing/.

[3] Python - instrukcja instalacji. URL: https://www.python.org/about/gettingstarted/.

[4] Python - język programowania. URL: https://www.python.org.

[5] Stanford ibe system. URL: https://crypto.stanford.edu/ibe/.

[6] *Git* - rozproszony system kontroli wersji. URL: https://git-scm.com.

[7] *GitHub* - hostingowy serwis internetowy wykorzystujący system git. URL: https://github.com.

[8] `asn1crypto` - biblioteka języka python. URL: https://github.com/wbond/asn1crypto/tree/master/asn1crypto.

[9] `certvalidator` - biblioteka języka python. URL: https://github.com/wbond/certvalidator.

[10] `Cryptodome` - biblioteka kryptograficzna języka python. URL: https://pycryptodome.readthedocs.io.

[11] `csrbuilder` - biblioteka języka python. URL: https://github.com/wbond/csrbuilder.

[12] `GmSSL` - biblioteka kryptograficzna zawierająca standard sm9. URL: http://gmssl.org/english.html.

[13] `json` - biblioteka języka python. URL: https://docs.python.org/3/library/json.html.

[14] `pickle` - biblioteka języka python. URL: https://docs.python.org/3/library/pickle.html.

[15] `pycrypto` - biblioteka kryptograficzna języka python. URL: https://www.dlitz.net/software/pycrypto/.

[16] `SHAKE256` - funkcja kryptograficzna w bibliotece `PyCryptodome`. URL: https://pycryptodome.readthedocs.io/en/latest/src/hash/shake256.html.

[17] `socket` - biblioteka języka python. URL: https://docs.python.org/3/library/socket.html.

[18] `ssl` - biblioteka kryptograficzna języka python. URL: https://docs.python.org/3/library/ssl.html.

[19] `unittest` - biblioteka do testów jednostkowych w języku python. URL: https://docs.python.org/3/library/unittest.html.

[20] Z. S. D. P. Anupam Saxena, Om Pal. Customized pki for scada system. 2010.

[21] Z. Cheng. The sm9 cryptographic schemes. 2017.

[22] T. G. Nils gentschen Felde, Sophia Grundner-Culemann. Authentication in dynamic groups using identity-based signatures. 2018.

[23] M. Nystrom, B. Kaliski. PKCS #10: Certification Request Syntax Specification Version 1.7. RFC 2986, November 2000.

[24] A. Shamir. Identity-based cryptosystems and signature schemes, workshop on the theory and application of cryptographic techniques. strony 47–53, 1984.

Zawartość płyty CD

Płyta CD zawiera kody źródłowe opisanych w tej pracy aplikacji oraz przykładowe pliki konfiguracyjne. W katalogu `components` umieszczone są katalogi z aplikacjami - `Terminal`, `Sensor`, `RA` oraz `Factory`. W katalogu `Sample_PKI` znajduje się plik `commands.sh` zawierający polecenia tworzące przykładowe certyfikaty niezbędne do działania systemu. Zawarte z nim komendy korzystają z przykładowych plików konfiguracyjnych, umieszczonych w folderze `etc` w tym samym katalogu.