WOJSKOWA AKADEMIA TECHNICZNA
WYDZIAŁ CYBERNETYKI

Uwierzytelnianie i autoryzacja urządzeń Internetu rzeczy w środowisku federacyjnym z wykorzystaniem technologii rejestrów rozproszonych

mgr inż. Michał Jarosz

Promotor:
dr hab. inż. Zbigniew Zieliński

Promotor pomocniczy:
dr inż. Konrad Wrona

Warszawa 2024

WOJSKOWA AKADEMIA TECHNICZNA
WYDZIAŁ CYBERNETYKI

Streszczenie

mgr inż. Michał Jarosz

Realizacja procesu uwierzytelniania i autoryzacji urządzeń Internetu rzeczy jest szczególnie trudna w środowiskach federacyjnych, w których nie ma lub jest ograniczone zaufanie pomiędzy organizacjami. Jednym z podstawowych warunków funkcjonowania federacji jest integracja informacyjna organizacji, która umożliwia wymianę informacji oraz korzystanie z zasobów, należących do poszczególnych podmiotów tworzących federację. Kluczowym wyzwaniem podczas wdrażania i wykorzystywania środowiska federacyjnego jest zapewnienie bezpieczeństwa wymiany informacji pomiędzy różnymi podmiotami. Natomiast same operacje uwierzytelniania i autoryzacji w takim środowisku muszą być realizowane według wspólnie określonych reguł. Reguły te nie mogą być zmienione bez wiedzy i braku akceptacji któregokolwiek uczestnika federacji. Jeżeli procesy uwierzytelniania i autoryzacji dotyczą środowiska Internetu rzeczy, w których to liczba urządzeń jest ogromna, to procesy te muszą być realizowane w sposób wydajny, skalowalny oraz bez udziału operatora. Problemy uwierzytelnienia i autoryzacji urządzeń IoT działających w środowiskach federacyjnych są nielicznie poruszane w literaturze. A same rozwiązania prezentowane w znalezionych pracach badawczych nie mogą być uznane za w pełni satysfakcjonujące.

W pracy zaproponowano rozwiązanie bazujące na wykorzystaniu rejestru rozproszonego, który umożliwia ustalenie wspólnych reguł wymiany informacji pomiędzy organizacjami tworzącymi rejestr. Poprzez wykorzystanie kodów łańcuchowych możliwa jest automatyzacja procesu uwierzytelniania i autoryzacji przy jednoczesnym przestrzeganiu ustalonych zasad. Zaproponowane rozwiązanie cechuje się także wykorzystaniem kryptografii symetrycznej, co jest unikalne w porównaniu do znalezionych w literaturze propozycji uwierzytelniania urządzeń w środowiskach federacyjnych.W pracy przedstawiono wyniki oceny bezpieczeństwa, wydaj-

ności i skalowalności zaproponowanego rozwiązania. Ocena bezpieczeństwa obejmuje analizę jakościową, a także formalną z wykorzystaniem odpowiednich narzędzi. Badanie wydajności obejmuje różne konfiguracje opracowanego rozwiązania. W niniejszej pracy przedstawiono także model sieci Petriego, który umożliwia przeprowadzenie symulacji badań wydajności rejestru rozproszonego dla różnych jego konfiguracji.

Uzyskane wyniki potwierdzają skuteczność zaproponowanego rozwiązania.

Składam serdeczne podziękowania moim promotorom:

Panu dr. hab. inż. Zbigniewowi Zielińskiemu *oraz*

Panu dr. inż. Konradowi Wronie

za opiekę merytoryczną, cenne uwagi i sugestie, przekazaną wiedzę oraz okazaną mi cierpliwość i wyrozumiałość.

Spis treści

Spis rysunków

Spis tabel

Spis akronimów

AA Authentication and Authorization

ABAC Attribute-based Access Control

ACL Access Control List

AEAD Authenticated Encryption with Associated Data

AES Advanced Encryption Standard

API Application Programming Interface

ASN.1 Abstract Syntax Notation One

AWS Amazon Web Services

CA Certification Authority

CapBAC Capabillity Based Access Control

CBOR Concise Binary Object Representation

CIMIC Civil-Military Cooperation

CoAP Constrained Application Protocol

CRL Certificate Revocation List

CSPRNG Cryptographically-Secure Pseudo-Random Number Generator

DAC Discretionary Access Control

DDoS Distributed Denial of Service

DoS Denial of Service

DRAM Dynamic Random-Access Memory

DTLS Datagram Transport Layer Security

GCM Galois/Counter Mode

EAP Extensible Authentication Protocol

EC2 Elastic Compute Cloud

FPGA Field-Programmable Gate Array

gRPC Remote Procedure Call

HADR Humanitarian Assistance and Disaster Relief

HLF HyperLedger Fabric

HMAC Hash-based Message Authentication Code

HSM Hardware Security Module

HTTP Hypertext Transfer Protocol

I2C Inter-Integrated Circuit

IaC Infrastructure as Code

IAM Identity and Access Management

IBC Identity Based Cryptography

IdP Identity Provider

IEEE The Institute of Electrical and Electronics Engineers

IETF Internet Engineering Task Force

IMU Inertial Measurement Unit

IoT Internet of Things

IT Information Technology

JSON JavaScript Object Notation

KDC Key Distribution Centre

KDF Key Derivation Function

KGC Key Generation Centre

LAAFFI Lightweight Authentication and Authorization Framework for Federated IoT

M2M Machine-to-Machine

MAC Message Authentication Code

MAC Mandatory Access Control

MITM Man in the Middle

MSP Membership Service Provider

MS-KILE Microsof Kerberos Network Authentication Service V5 Extensions

NIST National Institute of Standards and Technology

NFT Non-Fungible Token

NATO North Atlantic Treaty Organization

NSA National Security Agency

OAuth Open Authorization

OPS Operations Per Second

PCG Phonocardiogram

PDP Policy Decision Point

PKI Public Key Infrastructure

PoC Proof of Concept

PPG Photoplethysmography

PRF Pseudo Random Function

PSK Pre-shared key

PUF Physical Unclonable Functions

RADIUS Remote Authentication Dial-In User Service

RBAC Role-based Access Control

REST Representational State Transfer

SAML Security Assertion Markup Language

SDK Software Development Kit

SDN Software-Defined Network

SIEM Security Information and Event Management

SP Service Provider

SPI Serial Peripheral Interface

SRAM Static Random Access Memory

SSO Single Sign-On

TCP Transmission Control Protocol

TLS Transport Layer Security

TPM Trusted Platform Module

UART Universal Asynchronous Receiver-Transmitter

UDP User Datagram Protocol

USB Universal Serial Bus

UUID Universally Unique Identifier

WAF Web Application Firewall

Wi-Fi Wireless Fidelity

Rozdział 1

Wprowadzenie

1.1 Motywacja oraz aktualny stan wiedzy

Aktualnie Internet rzeczy jest jedną z najbardziej dynamicznie rozwijających się gałęzi sektora Information Technology (IT), co przejawia się zastosowaniem tej technologii w wielu dziedzinach. Do licznych przykładów zastosowań na skalę przemysłową można zaliczyć inteligentne miasta (ang. smart city), inteligentne sieci przesyłowe (ang. smart grid), transport i logistykę, telemedycynę itp. Termin Internet rzeczy, ang. Internet of Things (IoT), odnosi się do rozproszonej sieci łączącej różne obiekty fizyczne zdolne do zbierania danych z otoczenia (przy pomocy sensorów), oddziaływania na otoczenie (przy pomocy efektorów), przetwarzania zebranych danych oraz komunikowania się między innymi urządzeniami oraz systemami komputerowymi. Proces zbierania danych oraz ich przetwarzania ma na celu wypracowanie działań kontekstowo zależnych, które mogą być podejmowane bez udziału człowieka, przynoszą oszczędności, zwiększają wydajność oraz poprawiają produkty i usługi. Szacuje się, że obecnie na świecie jest aktywnych ponad 29,3 mld urządzeń IoT, gdzie jeszcze pięć lat temu było ich 18,4 mld [36]. Ten szybki rozwój powoduje, że urządzenia IoT są coraz częściej spotykane w różnego rodzaju sytuacjach. Odmieniają nasze życie domowe poprzez automatyzację podstawowych czynności jak koszenie trawnika, odkurzanie mieszkania, ale też monitorują stan naszego zdrowia. Poza wykorzystaniem urządzeń IoT przez nas w codziennym życiu, są one także chętnie wykorzystywane m.in. w przemyśle i przez służby mundurowe oraz paramundurowe. Taki szybki rozwój systemów IoT wprowadza też wiele nowych problemów bezpieczeństwa oraz uwypukla już istniejące, które wcześniej były ignorowane.

Bezpieczeństwo systemów IoT jest przedmiotem wielu prac badawczych i jednym z kluczowych wyzwań warunkującym wdrożenie IoT w zastosowaniach krytycznych, tj. takich, w których naruszenie bezpieczeństwa może prowadzić do narażenia na szwank życia i zdrowia

ludzi, wyrządzenia znacznych strat materialnych czy też spadku zaufania do takiego systemu IoT. Należy zwrócić uwagę, że zbiór zagrożeń w sieciach IoT jest bardziej liczny niż w systemach IT. Jest to spowodowane zwiększoną powierzchnią ataku ze względu na wykorzystanie głównie komunikacji bezprzewodowej oraz lokalizacji urządzeń IoT w miejscach łatwiej dostępnych (np. w przestrzeni publicznej), jak również słabszą konfigurację sprzętową urządzeń czy też bardzo ograniczone możliwości modyfikacji konfiguracji sprzętowej urządzeń IoT. Przez powierzchnię ataku [60] rozumie się najczęściej zbiór wszystkich punktów, poprzez które atakujący może próbować uzyskać dostęp do systemu, komponentu lub środowiska, wykonać na nim akcję lub wyodrębnić z niego dane.

Podstawowe wyzwania związane z bezpieczeństwem IoT wiążą się z zapewnieniem poufności, integralności, dostępności usług i danych, uwierzytelniania i kontroli dostępu oraz wiarygodności i zaufania [167]. W wielu zastosowaniach IoT umożliwia ciągłe przesyłanie danych między urządzeniami i użytkownikami w celu osiągnięcia określonych celów. W takim środowisku udostępnianie, uwierzytelnianie, autoryzacja, kontrola dostępu i niezaprzeczalność są ważne dla zapewnienia bezpieczeństwa usług dostarczanych przez system IoT. W tym kontekście ograniczone zasoby obliczeniowe (tj. niska moc obliczeniowa, mała wielkość pamięci masowej), a także budowanie sieci IoT w sposób doraźny wymaga dostosowania istniejących technik do tego nowego środowiska lub także zaproponowania nowych rozwiązań.

Ze względu na specyfikę systemów IoT urządzenia najczęściej komunikują się z pomocą łączności bezprzewodowej. Wykorzystanie tego typu komunikacji wiąże się z możliwością łatwiejszego podsłuchania transmisji, zagłuszania komunikacji, ataku *Man in the Middle (MITM)* czy też fałszowania wiadomości. Aby przeciwdziałać takim atakom, należy wykorzystywać kryptografię w celu zabezpieczenia komunikacji. Jednak ze względu na ograniczoną moc obliczeniową urządzeń IoT, wykorzystane protokoły kryptograficzne muszą mieć cechy, które determinują je jako „lekkie" [155]. Poprzez lekkie protokoły należy rozumieć protokoły, które używają mniej zasobów sprzętowych, co daje możliwości wykorzystania ich w urządzeniach IoT. Wykorzystanie kryptografii umożliwia zachowanie poufności oraz integralności przesyłanych danych. W środowisku IoT występuje także problem wstępnej konfiguracji urządzeń IoT, w ramach której umieszczany jest na urządzeniu IoT materiał kryptograficzny, który później będzie wykorzystywany przez urządzenie do zapewnienia poufności, integralności i uwierzytelniania komunikacji [178]. Problemem jest także zastosowanie lekkiego systemu zarządzania zaufaniem, który umożliwi zapewnienie wiarygodności w relacjach pomiędzy urządzeniami IoT umieszczonymi w otwartych i dynamicznych środowiskach. Zaufanie jest koncepcją ściśle

powiązaną z bezpieczeństwem systemów IoT, ponieważ każdy uczestnik musi być pewnym prawdziwości otrzymanych danych. Aby zbudować skuteczny, efektywny system zarządzania zaufaniem przystosowany do systemów IoT, trzeba rozwiązać wiele problemów, do których możemy zaliczyć: lekki protokół uwierzytelniania i autoryzacji urządzeń IoT, który umożliwi zweryfikowanie tożsamości stron komunikacji oraz zapewni, że obydwie strony mogą wymienić się wiadomościami; lekki system rozliczalności [168], aby móc zbierać informacje o tym komu, kto, jakie dane udostępnił; a także lekki protokół routingu [133, 140, 161], który zapewni możliwość komunikacji pomiędzy oddzielnymi sieciami, w których znajdują się urządzenia IoT przy zachowaniu odporności na różne ataki. Możliwość zapewnienia komunikacji jest niezmiernie ważna w systemach IoT, gdzie urządzenia przemieszczają się [64].

Problem uwierzytelniania i autoryzacji urządzeń występuje w przypadku sieci IoT będących pod kontrolą jednej organizacji, ale jest bardziej istotny w sytuacji, gdy wiele organizacji musi ze sobą współpracować z wykorzystaniem swoich sieci IoT. Przykładem takiego scenariusza współpracy mogą być operacje wykonywane przez wiele organizacji przy likwidacji skutków katastrof – ang. Humanitarian Assistance and Disaster Relief (HADR) czy też sytuacje dotyczące współpracy organizacji cywilnych i wojskowych – ang. Civil-Military Cooperation (CIMIC). W takich przypadkach wymagane jest ustanowienie bezpiecznego, federacyjnego środowiska IoT, gdzie podmioty mogą mieć ograniczone relacje zaufania, a także wykorzystywane są zasoby już należące do poszczególnych organizacji. Takie wykorzystanie już istniejących zasobów wymaga rozwiązania problemu braku interoperacyjności w wymianie danych. Budowanie federacji w przypadku HADR i CIMIC może być doraźne, ale i tak wymaga zbudowania kompleksowego systemu, który musi powstać w krótkim czasie i spełniać wszystkie wymagania bezpieczeństwa postawione przez zaangażowane podmioty. W kontekście CIMIC kompleksowe podejście do ochrony wrażliwych danych musi spełniać zarówno rygorystyczne wymagania wojskowe, jak i cywilne wymagania prawne m.in. wprowadzone przez europejskie ogólne rozporządzenie o ochronie danych osobowych (RODO) [54].

W charakterze przykładu rozważmy sytuację klęski żywiołowej, takiej jak trzęsienie ziemi lub powódź, która dotyka inteligentne miasto. W takim scenariuszu siły wojskowe mogą zostać poproszone o pomoc lokalnym jednostkom cywilnym. Ich zadaniem może być zapewnienie logistyki, wsparcia medycznego oraz przeprowadzenie akcji ratowniczo-poszukiwawczych, wymagających koordynacji działań z jednostkami cywilnymi. Aby skutecznie koordynować działania, należy posiadać możliwie pełny oraz aktualny obraz sytuacyjny. Powstaje więc potrzeba uzyskania danych z infrastruktury inteligentnego miasta. Do takiej infrastruktury

mogą należeć: czujniki, efektory, elementy komunikacyjne (np. łączność bezprzewodowa), kamery monitoringu, stacje monitorujące pogodę i zanieczyszczenie, drony i inne elementy (np. monitorujące ruch samochodów). Aby polepszyć obraz sytuacyjny, można skorzystać z urządzeń oferowanych przez mieszkańców, jak również wojsko może rozstawić swoje własne elementy infrastruktury. Wykorzystując dane z urządzeń inteligentnego miasta, mieszkańców oraz wojska, można zbudować system, który zapewni aktualny i pełny obraz sytuacyjny. Zbiór urządzeń IoT zarządzanych przez daną organizację (właściciela) będziemy rozpatrywać jako oddzielną domenę bezpieczeństwa. W każdej domenie mogą funkcjonować inne zasady udostępniania i ochrony zasobów. Poszczególne urządzenia mogą przesyłać dane bezpośrednio do systemu, ale mogą także korzystać z węzłów brzegowych (bram), które będą pośredniczyć w komunikacji, w tym zapewniać bezpieczeństwo przesyłania danych, jeżeli urządzenie ma niewystarczającą wydajność. Można oczekiwać, że w przyszłości operacje reagowania kryzysowego i usuwania skutków katastrof będą w coraz większym stopniu opierać się na obrazie sytuacyjnym zbudowanym w oparciu o dane z różnych źródeł [146].

Innym przykładem może być tworzenie federacji państw NATO i podmiotów spoza NATO uczestniczących w misjach (ang. Federated Mission Networking), z których każdy zachowuje kontrolę nad własnymi zdolnościami i działaniem, jednocześnie akceptując i spełniając wymagania określone we wstępnie wynegocjowanych i uzgodnionych ustaleniach, na przykład w zakresie polityki bezpieczeństwa. Główną ideą jest połączenie sił w sfederowanym środowisku misji w dowolnym momencie, w krótkim czasie i przy optymalizacji zaangażowanych zasobów. Oczekiwanym efektem jest lepsze dowodzenie i pełna kontrola działań oraz podejmowanie decyzji dzięki usprawnionej wymianie informacji.

Jak pokazuje doświadczenie wojny w Ukrainie, podczas działań wojennych często dochodzi do ataków na obiekty infrastruktury cywilnej z dala od linii frontu, które znajdują się na obszarach silnie zurbanizowanych. Ataki te mogą być przeprowadzane na przykład za pomocą dronów, pocisków rakietowych lub innych środków napadu powietrznego, a ich wynikiem jest często zniszczenie budynków mieszkalnych lub budynków instytucji cywilnych. W wyniku takich ataków dochodzi do licznych ofiar w ludziach, których życie często zależy od szybkości ich zlokalizowania i udzielonej im pomocy. Również doświadczenia wojny w Ukrainie pokazują, że wykorzystanie cywilnej infrastruktury komunikacyjnej ma sens i nawet w przypadku zniszczenia obiektów na dużą skalę na danym obszarze, sieć bezprzewodowa może funkcjonować. Dzięki temu, w przypadku sfery cywilnej, można skorzystać z infrastruktury inteligentnego miasta. Do zachowanej infrastruktury można również podłączyć osobiste urządzenia mobilne,

takie jak smartfony czy smartwatche, wyposażone w specjalną aplikację pomocową, którą można uruchomić w nagłych przypadkach. Ich zadaniem jest np. przekazywanie informacji do sieci w przypadku wykrycia stanu zagrożenia życia, takiego jak poważne obrażenia ciała. Aby zapewnić wiarygodność takich źródeł i bezpieczeństwo transmisji informacji, urządzenia obsługiwane przez użytkowników prywatnych i organizacje cywilne powinny być rejestrowane i uwierzytelniane przez zaufaną stronę taką jak operator komórkowy. Urządzenia mobilne, w momencie wykrycia problemu ze zdrowiem użytkownika, powinny wysyłać do systemu informację alarmową wraz z dokładną lokalizacją. Jeżeli w określonym promieniu takich zgłoszeń będzie wiele, to operator systemu będzie mógł wcześniej wysłać pomoc lub drona w celu określenia aktualnej sytuacji. Taki dron również mógłby komunikować się z urządzeniami mobilnymi osób poszkodowanych, przekazywać obraz z kamer video lub analizować zagrożenia np. skład powietrza pod kątem obecności gazów bojowych. Operator systemu na podstawie danych z drona i innych sensorów może uzyskać aktualny obraz sytuacji, aby móc lepiej określić przyczynę wielu alarmów. Może na przykład skorzystać z czujników dźwięku i kamer w celu określenia, czy nie doszło do wybuchu rakiety w danym miejscu. Takie czujniki mogą także posłużyć do wykrycia i określenia odległości, prędkości, kierunku nadlatującego pocisku rakietowego czy drona. Takie informacje powinny być automatycznie, bez zbędnej zwłoki, przekazywane do sieci wojskowej, aby obrona powietrzna mogła z wyprzedzeniem przygotować się do zestrzelenia takiego obiektu.

Proces rejestracji urządzeń w takim scenariuszu powinien być przeprowadzany przez upoważnione podmioty. Przykładowo, w przypadku urządzeń wojskowych za rejestrację odpowiedzialny jest wojskowy personel IT, w przypadku kamer miejskich – pracownicy miejscy, a w przypadku prywatnych telefonów i smartwatchy – operatorzy komórkowi. Oczywiście w takim systemie musi istnieć zaufanie do danych, a tym samym do źródeł, z których one pochodzą. Jak już wspomniano budowa zaufania, w szczególności w środowisku federacyjnym IoT (obejmujących wiele domen organizacyjnych), jest jednym z aktualnych problemów badawczych. Zaufanie w środowisku federacyjnym jest poważniejszym problemem, ponieważ poza zaufaniem do urządzeń, które są pod naszą kontrolą trzeba wziąć pod uwagę urządzenia należące do innych organizacji oraz osób prywatnych. Aby zbudować zaufanie w środowisku federacyjnym, trzeba stworzyć skuteczny mechanizm uwierzytelniania i autoryzacji urządzeń w takim środowisku. System uwierzytelniania i autoryzacji IoT w środowiskach federacyjnych musi spełniać kilka warunków:

1. Zapewnienie niezwłocznej interoperacyjności (ang. zero day interoperability), czyli gotowość do wdrożenia w sytuacji wystąpienia potrzeby zbudowania federacyjnego środowiska IoT.

2. Wsparcie dla urządzeń IoT, czyli protokoły wykorzystane w systemie powinny być wystarczająco wydajne na urządzeniach o ograniczonych zasobach. Narzut komunikacyjny protokołu uwierzytelniania powinien być zminimalizowany pod względem liczby wiadomości wymienianych między stronami uwierzytelniania, a także rozmiaru wysyłanych wiadomości.

3. Odseparowanie organizacji, czyli aby żadna z organizacji nie mogła uzyskać klucza do komunikacji z urządzeniami innej organizacji bez jej zgody oraz po spełnieniu warunków zapisanych w polityce bezpieczeństwa federacji.

4. Decentralizacja, aby utworzyć zaufanie organizacji między sobą. Nie może istnieć tylko jedna organizacja kontrolująca cały system. Urządzenie lub użytkownik, należący do jednej organizacji, powinien być w stanie uwierzytelnić się na serwerze należącym do innej organizacji i korzystać z jej usług bez tworzenia tożsamości lub rejestrowania danych uwierzytelniających w tej organizacji. Decentralizacja jest również sposobem na osiągnięcie wysokiej niezawodności.

5. Przechowywanie tożsamości urządzeń w bezpiecznym magazynie danych.

6. Zapewnienie w miarę możliwości liniowej skalowalności względem liczby urządzeń.

7. Zagwarantowanie bezpiecznej komunikacji pomiędzy urządzeniami IoT, w szczególności należącymi do różnych organizacji.

8. Zapewnienie rozliczalności, czyli system powinien umożliwić monitorowanie komunikujących się urządzeń IoT pochodzących z różnych domen.

9. Zapewnienie wysokiej dostępności, aby urządzenia IoT miały możliwość wykonania powierzonych im zadań w każdym momencie. Właściwość tę można zrealizować poprzez zwiększenie liczby instancji usług, aby można było obsłużyć więcej żądań lub w celu utrzymania ciągłości biznesowej, gdy niektóre instancje są nieosiągalne.

W literaturze można znaleźć różne metody uwierzytelniania urządzeń w środowiskach federacyjnych, jednak na podstawie wykonanego przeglądu literatury [8, 35, 115, 156, 164, 170,

184, 192, 196, 200, 202] można stwierdzić, że nie spełniają one wymienionych wymagań bezpieczeństwa, niezawodności i wydajności. W wielu pracach [35, 115, 164, 184, 192, 196, 200, 202] autorzy opisują wykorzystanie rejestru rozproszonego. Rejestr rozproszony to zdecentralizowana, odporna na manipulacje baza danych. Każda z organizacji uczestniczącej w budowie rejestru rozproszonego utrzymuje swoją własną kopię rejestru w węzłach. Węzłem może być dowolny komputer z odpowiednim oprogramowaniem umożliwiającym utworzenie, zarządzanie i świadczenie usługi rejestru rozproszonego. Węzły rejestru rozproszonego nie muszą być identyczne pod względem konfiguracji sprzętowej i programowej. Umożliwia to zwiększenia bezpieczeństwa, ponieważ węzły są niezależne od błędów bezpieczeństwa wynikających z podatności znalezionych w sprzęcie, sterownikach lub oprogramowaniu. Wykorzystanie luki bezpieczeństwa w jednym węźle nie będzie oznaczało, że każdy węzeł jest podatny. Same węzły mogą znajdować się także w różnej lokalizacji geograficznej, co prowadzi do minimalizacji skutków ataków na pojedyncze węzły, takich jak ataki Denial of Service (DoS). Cały rejestr rozproszony jest odporny na awarię pojedynczych węzłów, ponieważ każda organizacja może posiadać kilka węzłów oraz same węzły działają w sieci peer–to–peer. W rejestrze rozproszonym dane zapisywane są w formie transakcji, które są zgłaszane przez użytkowników należących do organizacji [31]. Proces dodawania transakcji do rejestru może różnić się w zależności od implementacji rejestru rozproszonego. Jeżeli węzeł zbuduje blok, zawierający transakcje, jest on przesyłany do weryfikacji przez inne węzły. Dodanie bloku z transakcjami do rejestru wymaga weryfikacji wszystkich transakcji znajdujących się w bloku. Po sprawdzeniu bloku następuje głosowanie nad dodaniem bloku do rejestru, jeżeli węzły zagłosują za dodaniem (zostanie osiągnięty konsensus), to blok jest dodany do rejestru rozproszonego na każdym z węzłów. Nie istnieje żadna zaufana trzecia strona, która może wpłynąć na wynik konsensusu. Nie istnieje także jedna centralna organizacja decydująca o modyfikacji rejestru rozproszonego. Same bloki z transakcjami nie są usuwane oraz nie mogą być nadpisywane, więc można prześledzić wszystkie zmiany od momentu utworzenia rejestru.

Pierwszą implementacją rejestru rozproszonego był Bitcoin zaprezentowany w 2008 roku przez Satoshi Nakamoto [137] w ramach kryptowaluty o tej samej nazwie. W Bitcoin i innych kryptowalutach to użytkownicy sieci zlecają transfer tokenów mających symbolizować kryptowalutę. Informacje o transferze zapisywane są w formie publicznie dostępnych transakcji, których weryfikację może przeprowadzić każda osoba. Bitcoin jest utrzymywany i zarządzany przez właścicieli węzłów, do których może dołączyć każdy – wystarczy tylko posiadać odpowiednie oprogramowanie. Ponieważ powszechne stało się wykorzystywanie tej technologii

w kryptowalutach (Bitcoin [137], Ethereum [53], i inne), utożsamia się ją z kryptowalutami. Jednak można ją wykorzystać także w innych celach. Do przykładowych scenariuszy wykorzystania rejestrów rozproszonych możemy zaliczyć sytuacje, gdy zachodzi potrzeba:

1. wymiany danych pomiędzy minimum dwoma organizacjami, a wymiana danych ma charakter transakcyjny,
2. utworzenia globalnej cyfrowej unikalnej tożsamości np. dla sztuki cyfrowej – Non-Fungible Token (NFT),
3. utworzenia bezpiecznego rejestru uniemożliwiającego modyfikacje np. księgi wieczyste,
4. zautomatyzowania działań pomiędzy podmiotami wraz z monitorowaniem zmian np. akredytacja systemów [194],
5. udostępnienia podmiotom pełnego dowodu pochodzenia wraz z historią zmian np. w łańcuchu dostaw [15].

W przypadku systemów IoT rejestr rozproszony jest wykorzystywany przy: bezpiecznej wymianie wiadomości pomiędzy urządzeniami IoT [98, 109], łańcuchach dostaw [149], zapewnieniu integralności i uwierzytelnianiu danych [116, 204], przechowywaniu danych [114], wykrywaniu intruzów w sieci IoT [14] czy też jako element systemu uwierzytelniania i autoryzacji, tak jak zaproponowano w niniejszej pracy.

Wykorzystanie rejestrów rozproszonych ma sens, gdy spełnione są następujące warunki [2, 195]:

1. występuje potrzeba współpracy minimum 2 podmiotów,
2. mamy do czynienia z brakiem zaufania pomiędzy podmiotami,
3. konieczne jest posiadania spójnej, współdzielonej bazy danych,
4. dane będą tylko dodawane, nie mogą być modyfikowane i usuwane,
5. zachodzi potrzeba przechowywania historii zmian.

Rejestry rozproszone ze względu na ich właściwości mogą z powodzeniem pełnić kluczową rolę w środowiskach federacyjnych. Do kluczowych właściwości rejestrów rozproszonych można zaliczyć możliwość osiągnięcia konsensusu bez zaufanej strony trzeciej, dzięki temu organizacje budujące federację nie muszą ufać innej, ale weryfikować poprawność zgłoszonych przez

nie transakcji. Innymi kluczowymi cechami jest brak jednej głównej organizacji zarządzającej całym rejestrem (decentralizacja), a także brak pojedynczego punktu awarii. W niektórych implementacjach rejestru rozproszonego możliwe jest wykorzystanie kodów łańcuchowych, czyli programów weryfikujących transakcje. Umożliwia to utworzenie programu, który da możliwość automatycznego egzekwowania zasad bezpieczeństwa każdej organizacji, a tym samym zapewni spełnienie wymagań bezpieczeństwa związanych z wymianą informacji pomiędzy różnymi członkami federacji.

Niestety, badacze w opracowywanych rozwiązaniach korzystają z rejestru rozproszonego bez wykorzystania ich pełnego potencjału. W artykułach [115, 164, 184, 192, 196, 200, 202] brak jest wykorzystania kodów łańcuchowych czy też rozważań o wdrażaniu polityk dostępu do danych.

Autorzy artykułu [202] wykorzystali rejestr rozproszony, który posłużył im do utworzenia bazy skrótów certyfikatów. Sam proces uwierzytelniania przebiega w sposób podobny do tego stosowanego w Public Key Infrastructure (PKI) z tym wyjątkiem, że urządzenia odpytują rejestr o skrót certyfikatu, aby potwierdzić, że jest autentyczny i nie został odwołany. Klucze prywatne przechowywane są na urządzeniach IoT. Opisany system wykorzystuje rejestr rozproszony tylko jako bazę danych, w której przechowywane są skróty certyfikatów oraz lista Certificate Revocation List (CRL) wszystkich organizacji. System nie wspiera autoryzacji urządzeń IoT. Skalowalność takiego rozwiązania nie została zbadana przez autorów, nie ma także informacji o rozliczalności. Ponieważ wspomniane wcześniej rozwiązanie bazuje na Bitcoin, który jest publicznym rejestrem rozproszonym, to można założyć, że skalowalność jest niska oraz że nie ma rozliczalności w przypadku odczytu danych. W innej publikacji [200] autorzy także wykorzystali rejestr rozproszony jako bazę danych, z tym wyjątkiem, że są tam przechowywane dane uwierzytelniające dla urządów certyfikacyjnych należących do różnych organizacji. W tej publikacji również brakuje opisu rozliczalności i skalowalności. Zaprezentowany system nie wspiera autoryzacji urządzeń IoT. Zastosowanie rejestru rozproszonego w formie bazy danych zostało także opisane w artykule [196]. W tym przypadku autorzy wprowadzili rozwiązanie, które polega na przechowywaniu certyfikatów dla urządzeń IoT w rejestrze rozproszonym. Uwierzytelnienie jest podobne do PKI. Opracowany system również nie wspiera autoryzacji urządzeń IoT, autorzy nie opisali także skalowalności oraz rozliczalności. Certyfikaty publikowane są w rejestrze rozproszonym IOTA, który został zaprojektowany do obsługi systemów IoT, dlatego możemy założyć, że rozwiązanie jest wydajniejsze niż zaprezentowane w [202]. W rejestrze dane należące do różnych organizacji nie są odseparowane, dlatego dostęp

do certyfikatów jest osiągalny dla każdego członka federacji. Poza wykorzystaniem schematów bazujących na PKI, badacze rozwijają także schematy wykorzystujące kryptografię bazującą na identyfikatorach (Identity Based Cryptography (IBC)). W tym schemacie każda organizacja ma swoje własne centrum generowania kluczy (Key Generation Centre (KGC)), który jest odpowiedzialny za generowanie kluczy prywatnych dla kluczy publicznych będących identyfikatorami urządzeń. Schemat wykorzystujący IBC, został przedstawiony w artykułach [164, 184]. Autorzy tych artykułów również wykorzystują rejestr rozproszony jako bazę danych. W obydwu artykułach kwestia skalowalności jest tylko wspomniana, ale nie została ona zbadana. Rozliczalność nie została przez autorów poruszona. Autoryzacja urządzeń IoT została tylko wspomniana w publikacji [184]. Klucze publiczne przechowywane w rejestrze są dostępne dla każdego członka federacji, więc własność separacji nie została spełniona. Schemat IBC został także wykorzystany w systemie przedstawionym w [35]. W tym przypadku autorzy nie wykorzystali rejestru rozproszonego tylko jako bazy danych, ale skorzystali z kodów łańcuchowych do wykonywania uproszczonego zarządzania kluczami (m.in. inicjalizację KGC, rejestrację urządzenia IoT). Właściwość skalowalności została tylko wspomniana w pracy, nie poruszono w niej rozliczalności. System nie wspiera autoryzacji urządzeń IoT. Inne podejście zostało przedstawione w artykułach [115, 192]. Jednak w tych przypadkach rejestr służy jako baza danych, właściwości skalowalności i rozliczalności nie zostały uwzględnione oraz klucze publiczne urządzeń IoT są przechowywane bezpośrednio w rejestrze, co umożliwia ich odczyt przez członków federacji. Zaprezentowane w artykułach [115, 192] rozwiązania nie umożliwiają autoryzacji urządzeń. Ze względu na cechy rejestrów rozproszonych założono, że wszystkie schematy, bazujące na rejestrach, oferują decentralizację oraz wysoką dostępność. Rozwiązań wykorzystujących kryptografię asymetryczną do uwierzytelniania urządzeń IoT w środowiskach federacyjnych jest bardzo dużo, w pracy przedstawiono tylko prace reprezentatywne dla przedmiotu rozprawy.

Poza wykorzystaniem wyłącznie kryptografii asymetrycznej można znaleźć podejście hybrydowe. Takie podejście przedstawili autorzy w artykule [156]. Do uwierzytelniania urządzeń IoT należących do tej samej domeny co dostawca tożsamości, wykorzystują kryptografię symetryczną, natomiast do komunikacji z urządzeniami należącymi do innej domeny wykorzystują kryptografię asymetryczną. Dane uwierzytelniające przechowywane są u dostawcy tożsamości, który pełni także rolę urzędu certyfikacji (ang. Certification Authority (CA)). Rozwiązanie zaproponowane w [156] wspiera autoryzację urządzeń. Autorzy pomyśleli także o skalowalności. Ponieważ dostawca tożsamości jest w posiadaniu jednej organizacji, to nie jest zapewniona

decentralizacja.

Z przeprowadzonej analizy prac [8, 170] wynika, że stosowane są również rozwiązania bazujące na kryptografii symetrycznej, jednak znane przykłady ich wykorzystania w środowiskach federacyjnych IoT są nieliczne. Przykładowo w pracy [170] autorzy przedstawili koncept wykorzystania Remote Authentication Dial-In User Service (RADIUS) z Extensible Authentication Protocol (EAP) do uwierzytelniania urządzeń IoT w środowiskach federacyjnych. W artykule nie jest poruszona kwestia skalowalności, ale serwer RADIUS może być skalowalny oraz zapewnia rozliczalność [141]. Serwer jednak jest pod kontrolą jednej organizacji, dlatego też własność decentralizacji nie została spełniona. W artykule nie wspomniano o autoryzacji urządzeń. Inny schemat został przedstawiony w pracy [8]. Autorzy przedstawili schemat uwierzytelniania urządzeń IoT należących do różnych domen z wykorzystaniem współdzielonego serwera. Serwer ten może występować w wielu instancjach i współdzielić bazę danych zawierającą informacje o przynależności urządzenia IoT do kontrolera domeny. Każda organizacja może posiadać swój własny kontroler domeny, w związku z czym własność separacji została spełniona. Urządzenia, które będą komunikować się ze sobą, muszą być umieszczone w tej samej domenie, przy czym urządzenia mogą należeć do wielu domen. Autorzy tego rozwiązania również nie poruszyli kwestii skalowalności, rozliczalności oraz wysokiej dostępności swojego rozwiązania. Zaproponowany schemat nie wspiera autoryzacji urządzeń IoT. Od protokołu uwierzytelniania urządzeń IoT wymaga się, aby był protokołem lekkim.

W przeanalizowanych pozycjach literatury przeważają protokoły wykorzystujące kryptografię asymetryczną, która nie posiada cechy „lekkości". W przypadku rozwiązań bazujących na kryptografii symetrycznej znalezione metody uwierzytelniania nie zapewniają skalowalności oraz nie są zdecentralizowane, co jest przeszkodą w uzyskaniu zaufania do pozostałych uczestników. W przypadku wykorzystania kryptografii symetrycznej pozostaje problem wymiany klucza uwierzytelniającego przed rozpoczęciem komunikacji. W środowiskach IoT, gdzie urządzeń może być dużo, a pojedyncze urządzenie może komunikować się z wieloma z nich, jest to problem bardzo istotny. Ze względu na ograniczenia zasobowe urządzeń IoT oraz zachowanie rozliczalności nie można także przechowywać wszystkich kluczy na każdym urządzeniu. Dlatego warto sprawdzić, czy przy użyciu dostępnych informacji o urządzeniu można tworzyć klucze umożliwiające bezpieczną komunikację przy niskim wykorzystaniu zasobów urządzenia IoT. Takie rozwiązanie może pozwolić na tworzenie dużej liczby kluczy w sposób deterministyczny, bez konieczności przechowywania wszystkich z nich w pamięci urządzenia. Sam proces wymiany kluczy pomiędzy urządzeniami może obejmować wykorzystanie rejestru

rozproszonego, którego zadaniem będzie weryfikacja, czy oba urządzenia mogą się ze sobą komunikować. Dodatkowym atutem wykorzystania rejestru rozproszonego będzie zapewnienie decentralizacji oraz rozliczalności w środowisku federacyjnym.

W żadnym z przeanalizowanych schematów nie udało się ustalić, czy zapewni on niezwłoczną interoperacyjność. Własność ta jest głównie wymagana w sytuacji, gdy organizacje muszą utworzyć federację możliwie jak najszybciej. Porównanie omówionych schematów zostało przedstawione w tabeli 1.1.

Analizując przedstawione przykłady mechanizmów uwierzytelniania dla urządzeń IoT w środowiskach federacyjnych, można dojść do kilku kluczowych wniosków. Pierwszym z nich jest przeważająca liczba mechanizmów bazujących na kryptografii asymetrycznej, która jest mniej wydajna niż kryptografia symetryczna. Niektóre rozwiązania, np. opisane w pracy [170], korzystają z innych protokołów, które nie są przystosowane do słabszych urządzeń IoT. W przypadku rozwiązań niekorzystających z kryptografii asymetrycznej, których nie ma wiele, nie wykorzystują one rejestrów rozproszonych. Rozwiązania te nie posiadają innych mechanizmów, które mogłyby oferować decentralizację. W przypadku rozwiązań bazujących na rejestrze rozproszonym to poza artykułem [35], rozwiązania traktują rejestr jak współdzieloną bazę danych. Propozycje te korzystają z opcji dodania, pobrania danych bez ich dodatkowego przetwarzania w ramach rejestru rozproszonego. Autorzy artykułów również nie skupili się na separacji danych i problemie autoryzacji dostępu do danych. Trudno jest także wywnioskować, czy zaproponowane schematy wspierają niezwłoczną interoperacyjność, która jest kluczowa w środowiskach federacyjnych budowanych wokół sytuacji nagłych. Większość sprawdzonych rozwiązań nie ma także analizy formalnej bezpieczeństwa. Mało którzy autorzy wspomnianych artykułów analizują bezpieczeństwo opracowanego rozwiązania w sposób formalny.

1.2 Teza i cele rozprawy

Analiza literatury, dotyczącej tematyki poruszanej w ramach prowadzonych badań, pozwoliła na sformułowanie następującej tezy pracy:

Umiejętne wykorzystanie unikatowych parametrów urządzeń IoT wraz z zastosowaniem technologii rejestrów rozproszonych zapewnia efektywne i bezpieczne uwierzytelnianie i autoryzację urządzeń IoT, a także umożliwia bezpieczną komunikację urządzeń IoT w środowisku federacyjnym.

Aby potwierdzić prawdziwość postawionej tezy, w rozprawie postawiono następujące cele:

Tabela 1.1: Porównanie metod uwierzytelniania i autoryzacji ze spełnieniem warunków wymaganych do wykorzystania w środowisku federacyjnym. ? = brak danych, IBC – Kryptografia oparta na tożsamościach, MAC – Message Authentication Code, EAP – Extensible Authentication Protocol, RR – Rejestr rozproszony

Metoda uwierzytelniania	**Kryptograficzny mechanizm**	**Artykuł**	**Zapewnia autoryzację**	**Zastosowanie RR**	**Skalowalność**	**Przystosowanie do IoT**	**Separacja**	**Rozliczalność**	**Decentralizacja**	**Wysoka dostępność**	**Zapewnienie niezwłocznej interoperacyjności**	**Analiza bezpieczeństwa**
Asymetryczna kryptografia	PKI	[202]	Nie	Baza danych	Nie	Nie	Nie	?	Tak	Tak	?	Tak (opis)
		[196]	Nie	Baza danych	?	Nie	Nie	?	Tak	Tak	?	Nie
		[200]	Nie	Baza danych	?	Nie	Nie	?	Tak	Tak	?	Tak (opis)
	IBC	[184]	Tylko wspomniano	Baza danych	Tylko wspomniano	Nie	Nie	?	Tak	Tak	?	Tak (opis)
		[164]	Nie	Baza danych	Tylko wspomniano	Nie	Nie	?	Tak	Tak	?	Tak (opis)
		[35]	Nie	Baza danych Zarządzanie kluczami	Tylko wspomniano	Nie	Nie	?	Tak	Tak	?	Tak (opis)
	Bezcertyfikatowe	[192]	Nie	Baza danych	Nie	Nie	Nie	?	Tak	Tak	?	Tak (opis + Tamarin)
		[115]	Nie	Baza danych	?	Nie	Nie	?	Tak	Tak	?	Tak (opis)
Hybrydowa	MAC + PKI	[156]	Tak	Nie	Tak	Nie	Nie	?	Nie	Tak	?	Tak (opis + Scyther)
Symetryczna kryptografia	EAP	[170]	Nie	Nie	Nie	Nie	?	Tak	Nie	?	?	Nie
	wyzwanie–odpowiedź	[8]	Nie	Nie	?	Tak	Nie	?	Nie	?	?	Tak (opis + AVISPA)

1. Opracowanie protokołu umożliwiającego uwierzytelnianie i autoryzację urządzeń IoT w środowisku federacyjnym z wykorzystaniem rejestru rozproszonego. Protokół powinien dostarczać mechanizmu ustalania kluczy kryptograficznych dla komunikujących się urządzeń IoT.

2. Opracowanie prototypowego rozwiązania z wykorzystaniem Hyperledger Fabric.

3. Ocena skalowalności i wydajności opracowanego rozwiązania w środowisku rzeczywistym.

4. Jakościowa, jak i formalna ocena bezpieczeństwa opracowanego protokołu z wykorzystaniem odpowiednio wybranych narzędzi weryfikacji protokołu.

5. Modelowanie i ocena wydajności protokołu z wykorzystaniem hierarchicznych kolorowych sieci Petriego. Po wykonaniu badań w środowisku rzeczywistym wykorzystać wyniki do skalibrowania modelu oceny wydajności wykorzystującego model sieci Petriego.

1.3 Struktura rozprawy

Rozprawa składa się z sześciu rozdziałów. Rozdział pierwszy stanowi wprowadzenie do rozprawy. Przedstawiono w nim motywację, postawiono tezę pracy oraz sformułowano cele rozprawy. W drugim rozdziale zamieszczono przegląd technologii i zagadnień związanych z problematyką rozprawy, która stanowi rozszerzenie tego rozdziału o zagadnienia poruszane w dalszej części pracy. Opisano w nim, między innymi, problemy, jakie wiążą się ze środowiskami federacyjnymi, wymagania względem wyboru rejestru rozproszonego, problemy związane z zarządzaniem tożsamościami i dostępem do usług, typy urządzeń IoT i ich ograniczenia oraz na końcu rozdziału opisano wykorzystanie układów Physical Unclonable Functions (PUF) do identyfikacji urządzeń. W rozdziale trzecim przedstawiono autorskie rozwiązanie problemu uwierzytelniania i autoryzacji urządzeń w środowiskach federacyjnych, umożliwiających wykorzystanie trzech różnych źródeł, na podstawie których można utworzyć klucz uwierzytelniający urządzenie. Jednym z nich jest wykorzystanie unikatowych parametrów (cech) urządzenia do jego uwierzytelniania. Parametry te przesyłane są w procesie rejestracji do rejestru rozproszonego. Podczas komunikacji urządzenia z inną usługą, przesyłanie wiadomości są szyfrowane z wykorzystaniem klucza symetrycznego powstałego z tych parametrów. Klucz ten może być utworzony tylko przez urządzenie IoT i węzły rejestru rozproszonego, ponieważ

tylko te elementy środowiska IoT mają dostęp do parametrów. Z tego też powodu każda ze stron komunikacji pewna jest tożsamości drugiej. W rejestrze rozproszonym przechowywane są także reguły kontroli dostępu, na podstawie których możliwe jest m.in. określenie, czy dwa urządzenia IoT mogą się ze sobą komunikować. W rozdziale czwartym przedstawiono ocenę bezpieczeństwa opracowanego protokołu. Na tę ocenę składa się ocena opisowa oraz weryfikacja formalna z wykorzystaniem narzędzi weryfikacji protokołów. W rozdziale tym opisano też entropię wybranych parametrów, które posłużyły do utworzenia kluczy symetrycznych. Rozdział piąty zawiera ocenę skalowalności i wydajności implementacji opracowanego protokołu. W tym rozdziale zawarto także opis i wyniki badań wydajnościowych protokołu z wykorzystaniem opracowanego modelu w formie czasowej hierarchicznej kolorowanej sieci Petriego. Rozdział szósty stanowi podsumowanie rozprawy.

Kody programów, skrypty, modele oraz wyniki badań wykorzystane w rozprawie są dostępne pod adresem: `https://gitlab.com/MichalJarosz/doktorat`.

Rozdział 2

Przegląd technologii i zagadnień związanych z problematyką rozprawy

W tym rozdziale scharakteryzowano zagadnienia związane z problematyką pracy. Na początku przedstawiono ideę federacji, problemy występujące w środowisku federacyjnym Internetu rzeczy, wykorzystywane metody uwierzytelniania i autoryzacji stosowane w takim środowisku. Opisano także, czym jest rejestr rozproszony, jakie są podtypy rejestrów rozproszonych, w tym szczegółowo przedstawiono budowę łańcucha bloków. Na podstawie klasyfikacji rejestrów rozproszonych opisanej w tym rozdziale wybrano rodzaj rejestru rozproszonego, który jest odpowiedni do budowy systemu uwierzytelniania i autoryzacji w środowisku federacyjnym. Rozdział ten zawiera także porównanie kilku wybranych implementacji rejestru rozproszonego i uzasadnienie wyboru odpowiedniej implementacji do zrealizowania systemu uwierzytelniania i autoryzacji urządzeń IoT w środowisku federacyjnym. W rozdziale tym opisano także metody zarządzania tożsamościami w środowiskach IoT, a także przedstawiono klasyfikację urządzeń IoT i dla każdej klasy scharakteryzowano możliwości urządzeń IoT. W końcowej części rozdziału scharakteryzowano układy PUF i możliwości ich wykorzystania do uwierzytelniania urządzeń IoT.

2.1 Środowisko federacyjne IoT

Federacja to zbiór domen, które ustanowiły między sobą zaufanie [151]. Poziom zaufania może być różny, ale zazwyczaj obejmuje takie procesy jak uwierzytelnianie oraz autoryzację. Architektura federacyjna ma wiele istotnych zalet, do których można zaliczyć uproszczenie procesów, np. uwierzytelnianie jednokrotne, redukcję kosztów poprzez uproszczenie infrastruktury informatycznej, minimalizację ilości przechowywanych danych, ponieważ organizacje budujące federacje nie muszą przechowywać, przetwarzać, weryfikować zgodności z przepisami

danych, które do nich nie należą, oszczędność czasu oraz skoncentrowanie się na realizacji misji, a nie na procesie zarządzania tożsamościami. Zbudowanie federacji wymaga jednak ustalenia pewnych rozwiązań oraz zasad, które będą obowiązywały w tej federacji. Różni właściciele systemów, chcący skorzystać z usług oferowanych przez federację, będą musieli się dostosować do tych ustaleń [151]. Członkowie federacji mają również różne podejścia do tożsamości federacyjnej (zwłaszcza w zakresie prywatności i udostępniania informacji), które sprawiają, że konieczne jest ustalenie wiarygodnego sposobu zarządzania danymi członków federacji [69]. W federacyjnym środowisku występują także problemy, które można spotkać w niesfederowanych środowiskach, ale w tym przypadku mogą mieć bardziej rozległe konsekwencje. Takim problemem jest uwierzytelnianie atakującego, korzystającego z wykradzionej tożsamości, skompromitowanie lub nieuczciwe działanie członków federacji. Konsekwencje takich problemów mogą wpłynąć na zdolności do realizacji misji w założonym zakresie oraz czasie, ale także mogą mieć negatywny wpływ na operacje lub aktywa organizacji należących do tej federacji poprzez np. ujawnienie poufnych informacji. Dla uczestników systemu federacyjnego korzyści z udziału w federacji muszą przewyższać postrzegane ryzyko. Z formalnego punktu widzenia istotna jest ocena ryzyka, która musi nastąpić przed założeniem federacji. Na podstawie tej oceny przyszli członkowie federacji mogą określić korzyści udziału w federacji oraz metody minimalizowania wpływu potencjalnych zagrożeń. Przykładem takiego procesu może być ograniczenie zbieranych informacji tylko do tych niezbędnych w realizacji misji federacji. Między innymi takie podejście umożliwia zapewnienie niezwłocznej interoperacyjności (ang. zero day interoperability), czyli gotowość do wdrożenia w sytuacji wystąpienia potrzeby zbudowania federacyjnego środowiska IoT. Jawność zasad, użytych technologii, reguł, które panują w federacji wpływa także pozytywnie na budowanie zaufania pomiędzy członkami tej federacji. Budowa zaufania w federacji jest konieczna do osiągnięcia celów misji, do której ta federacja została powołana.

Zaufanie w rozproszonych środowiskach jest zwykle realizowane za pomocą infrastruktury klucza publicznego (ang. PKI) albo poprzez zaufaną trzecią stronę, jak ma to miejsce w protokole Kerberos. Zbudowanie zaufania w rozproszonych środowiskach IoT z wykorzystaniem PKI lub Kerberos nie sprawdza się. W przypadku infrastruktury klucza publicznego konieczne jest przechowywanie certyfikatów dla kilku urzędów certyfikacji należących do innych członków federacji oraz wykorzystywana jest kryptografia asymetryczna, co w przypadku urządzeń IoT ograniczonych zasobowo (procesor, pamięć RAM) uniemożliwia ich wdrożenie.Problemem jest także rozpropagowanie informacji o certyfikatach na urządzeniach IoT. Protokół Kerbe-

ros może wykorzystywać kryptografię symetryczną, jednak wymaga dokładnej synchronizacji czasu pomiędzy urządzeniem IoT a serwerem Key Distribution Centre (KDC) oraz serwerami usług. Poza tym budowanie zaufania pomiędzy domenami w oparciu o protokół Kerberos jest bardzo skomplikowane. Sam protokół nie jest przystosowany do budowy zaufania pomiędzy domenami w sposób hierarchiczny [176], a sam proces budowy zaufania pomiędzy domenami wymaga utworzenia dodatkowych powiązań, które opierają się na współdzielonym haśle. W Kerberos z góry narzucone są protokoły kryptograficzne, z których wszyscy muszą korzystać. Wiadomości mogą mieć wiele formatów serializacji, ale wybór formatu serializacji jest narzucony przez autora implementacji usługi. Utrudnia to wdrożenie na urządzeniach IoT, ponieważ urządzenia muszą w takim przypadku potrafić komunikować się z usługami obsługującymi Microsof Kerberos Network Authentication Service V5 Extensions (MS-KILE) lub Abstract Syntax Notation One (ASN.1), co dla niektórych z urządzeń może być niewykonalne z uwagi na zasoby pamięciowe [175]. Należy także wspomnieć o protokołach opracowanych w ramach WS-Federation oraz Liberty Alliance, które z powodzeniem stosuje się w przypadku rozwiązań webowych, ale nie są one przystosowane do uwierzytelniania i autoryzacji urządzeń IoT.

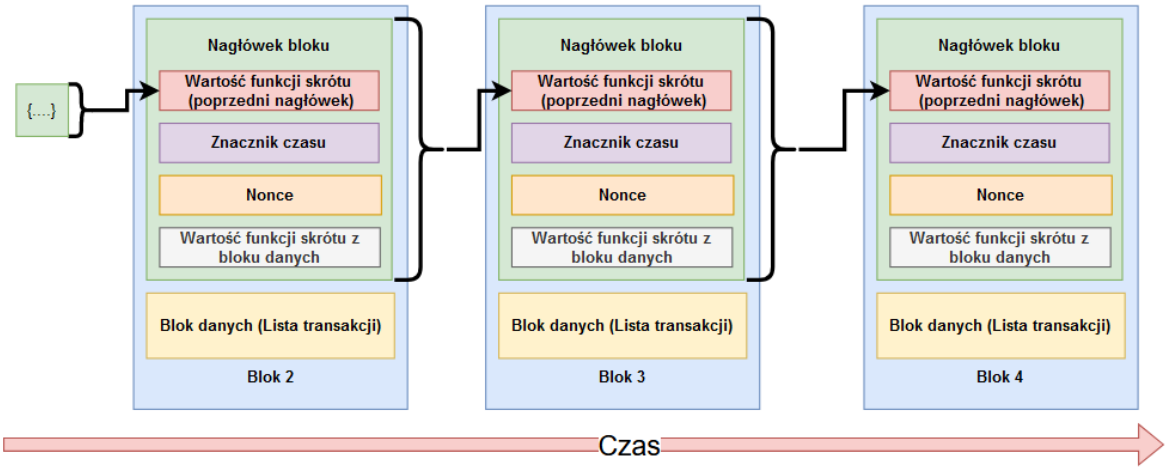
Istnieją już zalecenia budowania federacji dla usług webowych [169] oraz wytyczne do zarządzania tożsamościami użytkowników w środowiskach federacyjnych [69]. Jednak podczas analizy literatury nie udało się odnaleźć wytycznych dotyczących budowy środowiska federacyjnego dla systemów IoT. Sama budowa środowisk federacyjnych jest powszechna. Dobrym przykładem budowy środowiska federacyjnego opartego na interoperacyjności w celu osiągnięcia wspólnego celu jest koncepcja Federated Mission Networking (FMN) opracowana w North Atlantic Treaty Organization (NATO). W ramach tej koncepcji przygotowane są różne rozwiązania umożliwiające tworzenie federacji pomiędzy partnerami NATO oraz zewnętrznymi organizacjami. Wśród tych koncepcji rozwijane są również elementy związane z IoT dla między innymi inteligentnych miast [147] czy też operacji militarnych [127].

2.2 Technologia blockchain

Termin blockchain jest tłumaczony z języka angielskiego na język polski jako łańcuch bloków [47, 108]. Wykorzystanie łańcuchów bloków ma zastosowanie wszędzie tam, gdzie ważna jest decentralizacja, niezmienialność zapisanych danych, weryfikowalność z regułami zapisanych danych czy też jasne zasady obowiązujące członków budujących łańcuch bloków. Pier-

wotnym zastosowaniem tego typu struktur były kryptowaluty, jednak w późniejszym czasie łańcuch bloków jest proponowany do wykorzystywania także w służbie zdrowia [77, 153], systemach Internetu rzeczy [5, 44], usługach rządowych [20, 83], zastosowaniach militarnych [185] czy też wymiany danych [120].

Najpopularniejszą implementacją łańcucha bloków jest Bitcoin, dlatego też zawarty w pracy opis samej technologii jest oparty na strukturze tego łańcucha bloków. W innych implementacjach niektóre elementy struktury mogą się różnić. Łańcuch bloków składa się z bloków, które są połączone z wykorzystaniem wartości funkcji skrótu. Każdy blok składa się z dwóch części [124]: nagłówka oraz danych. Nagłówek bloku w Bitcoin zawiera numer wersji, wartość funkcji skrótu poprzedniego bloku, wartość nonce, której rolą jest uzyskanie wartości skrótu o pewnych właściwościach, np. sześciu początkowych zerach, znacznik czasu, indeks trudności oraz wartość korzenia drzewa Merkle dla danych przechowywanych w tym bloku. W polu danych zawarte są transakcje zgłoszone przez użytkowników sieci. Jakakolwiek zmiana transakcji w polu danych wpływa na zmianę wartości korzenia drzewa Merkle dla całego pola danych. Dzięki temu mamy możliwość sprawdzenia, czy ktoś nie zmanipulował transakcji. Ponieważ każdy blok zawiera wartość funkcji skrótu z poprzedniego bloku i jest ona wymagana do obliczenia wartości funkcji skrótu następnego bloku, to jakakolwiek zmiana w poprzednikach bloku też może być wykryta, gdyż wartość funkcji skrótu będzie inna. Na rysunku 2.1 przedstawiono przykładowy schemat łańcucha bloków.



Rysunek 2.1: Przykładowy schemat budowy bloku. Opracowanie własne na podstawie: [198]

Transakcja umieszczona w polu danych bloku reprezentuje interakcję między stronami. W przypadku kryptowalut będzie oznaczała transfer tokenów pomiędzy użytkownikami (tak

jak w Bitcoin), w innych przypadkach będzie natomiast rejestracją zachodzących działań na zasobach. Transakcja jest zgłaszana przez użytkownika sieci. Budowa transakcji jest zależna od implementacji rejestru rozproszonego. W przypadku Bitcoin transakcja dotycząca transferu tokenów zawiera kilka informacji [198]. Na początku znajduje się wersja transakcji, potem liczba danych wejściowych, identyfikator transakcji oraz dane wejściowe. Dane te zawierają listę adresów, z których mają zostać pobrane tokeny oraz liczbę tokenów dla każdego adresu. Po danych wejściowych znajduje się informacja o liczbie danych wyjściowych. Następne pola określają, gdzie i ile tokenów ma zostać przypisane oraz jakie warunki muszą być spełnione, aby można było tę operację wykonać. Ostatni polem jest czas, kiedy blok zostaje uznany za prawidłowy.

Transakcje są głównie wykorzystywane do przekazywania tokenów pomiędzy kontami użytkowników, ale mogą być wykorzystane do przekazywania danych. W przypadku łańcuchów bloków, wykorzystujących kody łańcuchowe, transakcje są wykorzystywane do wysyłania danych, przetwarzania ich oraz przechowywania wyników w łańcuchu bloków. Najważniejszymi cechami transakcji są jej autentyczność oraz ważność. Ważność transakcji oznacza, że transakcja spełnia wymagania protokołu oraz wszystkie warunki kodów łańcuchowych wdrożonych w łańcuchu bloków. Ważność jest weryfikowana w procesie weryfikacji transakcji podczas ustalania konsensusu. Autentyczność natomiast oznacza, że nadawca miał odpowiednie uprawnienia do dodawanych lub modyfikowanych zasobów łańcucha bloków. Użytkownik, podpisując zgłaszaną transakcję, przekazuje informacje o swojej tożsamości. Natomiast proces autoryzacji transakcji zostaje wykonany również podczas fazy weryfikacji transakcji. Transakcje w przypadku kryptowalut wymagają podania adresów kont, pomiędzy którymi będą transmitowane środki (tokeny). Adresy te najczęściej powstają jako wynik funkcji skrótu dla kluczy publicznych właścicieli kont. Klucze prywatne, które są wymagane do dostępu do kont, są przechowywane w programie nazywanym portfelem. Portfel, poza funkcją przechowywania kluczy prywatnych, przechowuje także klucze publiczne i powiązane adresy kont. Może także wykonywać operacje, takie jak wysłanie transakcji czy też obliczanie całkowitej sumy tokenów przechowywanych na kontach. Utrata klucza prywatnego oznacza, że dostęp do kont zostanie utracony. Jeżeli strata będzie polegała na wykradzeniu, to przestępca uzyska dostęp do konta i może dokonać transferu tokenów, którego nie będzie można już cofnąć.

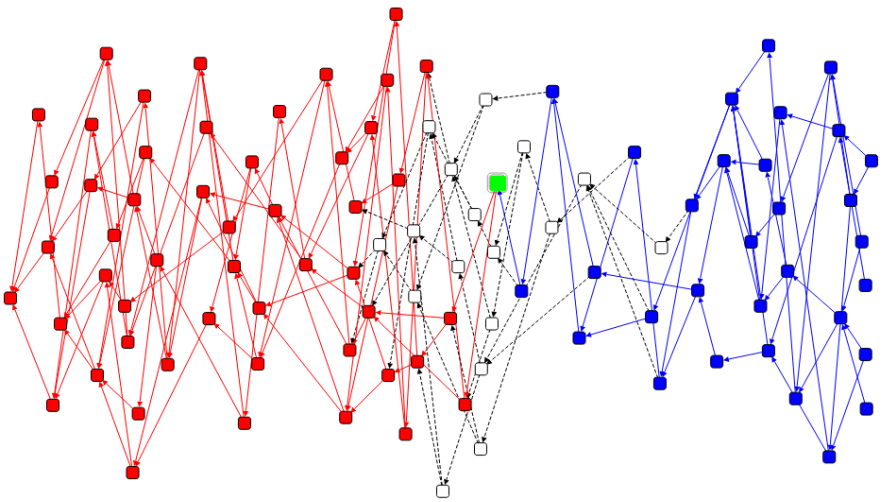
Kluczowym zadaniem w łańcuchu bloków jest osiągnięcie konsensusu przy ustalaniu, jakie dane mogą do niego zostać dodane. W Bitcoin, który jest publicznym łańcuchem, jest to szczególnie istotne ze względu na anonimowość oraz liczebność węzłów, które mogą zgłaszać

swoje propozycje bloków. Autorzy tej kryptowaluty wykorzystali algorytm nazwany dowodem pracy (ang. Proof-of-Work). W tym algorytmie prawo do publikacji ma węzeł, który pierwszy rozwiąże zagadkę, której rozwiązanie wymaga znacznych nakładów obliczeniowych. Rozwiązanie tej zagadki jest dowodem wykonania pracy. Sama zagadka jest tak zaprojektowana, że jej rozwiązanie jest bardzo wymagające pod względem obliczeniowym, ale sprawdzenie rozwiązania jest już bardzo proste. Dzięki temu wszystkie węzły będą mogły przy niskim nakładzie mocy obliczeniowej zweryfikować to rozwiązanie. Najczęściej zagadka polega na znalezieniu takiej wartości *nonce*, że wartość funkcji skrótu dla konkatenacji wartości *nonce* i innych pól z nagłówka bloku (wybór pól jest zależny od implementacji mechanizmu konsensusu) będzie zaczynała się od określonego ciągu cyfr np. zer. Liczba cyfr jest zwiększana wraz ze zwiększaniem się mocy węzłów podłączonych do sieci rejestru rozproszonego. Dostosowanie trudności zagadki ma na celu zapewnienie, że żaden podmiot nie będzie mógł przejąć kontroli nad procesem zgłaszania nowych bloków. Rozwiązanie jednej zagadki nie wpływa na rozwiązanie kolejnych, ponieważ zagadki są niezależne. Jeżeli węzeł zgłosi blok, który jest poprawny, to otrzymuje nagrodę w postaci tokenów. W przypadku sieci składających się z bardzo wielu węzłów, węzły mają tendencję do organizowania się w grupy w celu szybszego znalezienia odpowiedniej wartości *nonce*. Jeżeli któryś z bloków znajdujący się w grupie znajdzie odpowiednią wartość *nonce*, to nagroda jest przyznawana po równo wszystkim węzłom w grupie. Ten algorytm jest wykorzystywany również w innych implementacjach np. Litecoin, Dogecoin.

2.3 Rejestry rozproszone

Pierwszą implementacją rejestrów rozproszonych był Bitcoin upubliczniony w 2009 roku [198]. Bitcoin jest oczywiście łańcuchem bloków, który jest jednym z podtypów rejestrów rozproszonych. Poza łańcuchem bloków istnieje także Tangle [171]. Tangle jest acyklicznym grafem skierowanym, w którym role wierzchołków pełnią transakcje. Aby użytkownik mógł dodać swoją transakcję, musi zatwierdzić 2 inne i ich wartości funkcji skrótu umieścić w swojej transakcji. Jeżeli okaże się, że zatwierdzone transakcje są nieważne, to i transakcja zgłoszona przez użytkownika będzie automatycznie uznana za nieważną. Ten typ rejestru rozproszonego został zaprojektowany specjalnie do wykorzystania na urządzeniach IoT [23]. Tangle został zastosowany jako podstawa kryptowaluty IOTA, przeznaczonej do wykorzystania w środowisku IoT. Do kluczowych cech IOTA należy zaliczyć: dużą skalowalność, ponieważ każdy użytkownik sieci może dodać transakcję, a im więcej użytkowników, tym więcej transakcji jest

dodawanych; brak opłat za dodawanie transakcji oraz lekki protokół zatwierdzania transakcji. Ostatnią zaletą jest to, że użytkownicy nie muszą przechowywać całego rejestru, a tylko fragment. Do wad tego rozwiązania należy zaliczyć jego niezdecentralizowanie — istnieje organ, zwany koordynatorem, który weryfikuje dodawane transakcje. Jeżeli są one poprawne, to co 10 sekund generuje kamień milowy, czyli transakcję potwierdzającą wszystkie poprzednie transakcje wysłane przez użytkowników sieci. Jest to rozwiązanie tymczasowe i w przyszłości zostanie zmienione[90]. Rozwiązanie to również nie wspiera kodów łańcuchowych. Na rysunku 2.2 przedstawiono wizualizację Tangle. Zielony kwadrat oznacza pojedynczą transakcję, kolorem czerwonym zaznaczono wszystkie transakcje, które zostały pośrednio i bezpośrednio potwierdzone przez tę transakcję, natomiast kolorem niebieskim, transakcje, które potwierdzają pośrednio lub bezpośrednio dodaną transakcję.



Rysunek 2.2: Przykładowy schemat Tangle

Innym typem rejestru rozproszonego, który jest zaimplementowany i wykorzystywany, jest Hashgraph [171]. Jest to rozwiązanie opatentowane, ale od 2022 roku zostało udostępnione na licencji Apache 2.0. Z tego też względu przez ten czas zostało wykorzystane tylko w jednym rozwiązaniu, którym jest kryptowaluta Hadera. Hashgraph opiera się na seriach czasowych zdarzeń dla każdego węzła w sieci. Ponieważ to rozwiązanie umożliwia tworzenie węzłów tylko uprawnionym podmiotom, to liczba węzłów jest ograniczona. Według dokumentacji węzły usuwają z pamięci informacje o starych transakcjach ze względu na ograniczenie wykorzystania przestrzeni dyskowej [80], ale można skorzystać z innych usług, aby sprawdzić wszystkie historyczne transakcje [78]. Twórcy argumentują tę decyzję tym, że algorytm zatwierdzania transakcji zapewnia ostateczność oraz jest wykonywany przez zweryfikowane i uprawnione węzły,

więc nie może dojść do szkodliwego działania. Użytkownicy wysyłają transakcje do węzłów, węzeł, który otrzymał transakcje, grupuje je i tworzy zdarzenie. Następnie są one przesyłane do pozostałych węzłów, korzystając z protokołu plotki (ang. gossip protocol). Poza tym węzły informują się nawzajem, kiedy otrzymały poszczególne zdarzenia (*gossip about gossip*). Na podstawie informacji o czasach zdarzeń odebranych przez wszystkie węzły wyznaczana jest kolejność weryfikacji. W tym celu każdy węzeł wyznacza medianę z czasów odebrania zdarzenia przez wszystkie węzły. Do zatwierdzenia transakcji Hashgraph korzysta z wirtualnego głosowania. Polega ono na tym, że wynik głosowania, dla tych samych danych zapisanych w rejestrze, tych samych danych w zdarzeniu, tego samego programu i czasu wyznaczonego przez węzły będzie taki sam na każdym węźle. W takiej sytuacji węzły nie muszą wymieniać się informacjami o wynikach głosowania, a tylko weryfikują zdarzenie, bazując na informacjach, które posiadają. Do zalet tego rozwiązania należy bardzo krótki czas zapewniania pewności zgłoszonych transakcji, brak konieczności wykonywania skomplikowanych obliczeń oraz w przyszłości wsparcie dla kodów łańcuchowych. Do wad należy zaliczyć konieczność posiadania uprawnionych węzłów, ponieważ musimy mieć do nich zaufanie, że przesyłają poprawne informacje o zdarzeniach.

Implementacje rejestru rozproszonego możemy podzielić według dwóch kryteriów [59, 66]. Pierwszym kryterium jest podział ze względu na to, kto może mieć dostęp do danych zapisanych w rejestrze. Jeżeli nie ma ograniczeń w dostępie do danych, to mówimy o rejestrze publicznym. W takim rejestrze użytkownicy mogą przeglądać transakcje zapisane w rejestrze, dodawać swój węzeł do sieci, reguły są niezmienne (z wyjątkiem rozwidleń opisanych później), nie ma możliwości zmiany danych zapisanych w rejestrze, a samą decentralizację ze względu na to, że każdy może utworzyć węzeł można określić jako wysoką. Użytkownicy takiego rejestru rozproszonego są anonimowi i nie można mieć do nich zaufania. Przeciwieństwem rejestrów publicznych są rejestry prywatne. W takim rejestrze tylko uwierzytelniani użytkownicy z nadanymi uprawnieniami mają dostęp do danych zapisanych w rejestrze rozproszonym oraz mogą dołączyć węzły do takiej sieci. Taki rodzaj rejestru rozproszonego ma z tego powodu mniej węzłów, a tym samym łatwiej jest zmienić zasady panujące w takiej sieci (w tym uznać, że np. x-ostatnich bloków jest nieważne i wycofać je). Mniejsza liczba węzłów wpływa także na mniejszą decentralizację. Porównanie cech publicznych i prywatnych rejestrów rozproszonych zostało przedstawione w tabeli 2.1.

Inne kryterium klasyfikacji stanowi podział ze względu na to, jakie węzły mogą zatwierdzać transakcje. Możemy wyróżnić implementacje rejestrów rozproszonych z węzłami upraw-

Tabela 2.1: Porównanie cech publicznych oraz prywatnych rejestrów rozproszonych

Cecha	**Publiczny rejestr rozproszony**	**Prywatny rejestr rozproszony**
Dostęp do danych	Każdy	Tylko określeni użytkownicy
Tworzenie węzłów	Każdy	Tylko określeni użytkownicy
Zmiana zasad	Nie zmienne	Zmienne
Usunięcie bloków	Bardzo trudno możliwe	Możliwa
Decentralizacja	Wysoka	Niska
Zaufanie do użytkowników	Nie	Tak
Anonimowość	Tak	Nie

nionymi (ang. permissioned) – w takich implementacjach tylko niektóre węzły będą miały uprawnienia do zatwierdzania transakcji, a tym samym dodawania ich do rejestru rozproszonego. W takich rejestrach rozproszonych najczęściej nie są potrzebne żadne opłaty, a samo zatwierdzenie transakcji jest szybkie. Przeciwieństwem są rejestry rozproszone z węzłami bez uprawnień (ang. permisionless). W takim przypadku każdy węzeł rejestru rozproszonego może weryfikować i zgłaszać do dodania transakcje. Niestety w tym przypadku czas oczekiwania na zatwierdzenie transakcji jest dłuższy, a samo zatwierdzenie może wymagać uiszczenia dodatkowej opłaty. Różnice między rejestrami rozproszonymi z węzłami uprawnionymi i bez węzłów uprawnionych przedstawiono w tabeli 2.2.

Tabela 2.2: Porównanie rejestrów rozproszonych z węzłami uprawnionymi i bez węzłów uprawnionych

Cecha	**Rejestr rozproszony z węzłami uprawnionymi**	**Rejestr rozproszony bez węzłów uprawnionych**
Weryfikacja transakcji	Ograniczona	Każdy węzeł
Potwierdzenie transakcji	Szybkie	Wolne
Zarządzanie uprawnieniami	Tak	Nie
Koszt transakcji	Bez opłat	Zależy od implementacji

Na podstawie tych dwóch podziałów można wyróżnić 4 kategorie:

- Rejestr rozproszony publiczny z węzłami uprawnionymi (czasem nazywany podejściem hybrydowym), np. Hedera, Corda,

- Rejestr rozproszony prywatny z węzłami uprawnionymi, np. Hyperledger Fabric [85], Multichain [136],
- Rejestr rozproszony publiczny z węzłami bezuprawnień, np. Bitcoin [137], Ethereum [53],
- Rejestr rozproszony prywatny z węzłami bezuprawnień, np. Holochain [82], LTO Network [123].

O ile w przypadku rejestrów rozproszonych z węzłami uprawnionymi uzyskanie konsensusu w sprawie dodania nowych bloków nie stanowi większego problemu, to w przypadku rejestrów rozproszonych z węzłami bez uprawnień, gdzie z reguły jest znacznie więcej użytkowników i każdy z nich może zgłosić nową porcję danych przy jednoczesnym braku zaufania do tych danych, jest to już trudniejsze. W takim przypadku konsensus musi umożliwić ustalenie, czy zgłoszone dane są poprawne oraz który węzeł dostanie możliwość zgłoszenia swoich danych (bloków, transakcji). Metoda konsensusu jest ustalana na początku powstania rejestru rozproszonego i zapisywana w pierwszym bloku. Metoda osiągnięcia konsensusu może się zmienić podczas działania rejestru rozproszonego, ale wiąże się to ze zmianą, na którą muszą zgodzić się wszystkie organizacje tworzące rejestr rozproszony. Obecnie istnieje wiele algorytmów osiągnięcia konsensusu. Algorytmy stosowane w rejestrach rozproszonych, wykorzystujących węzły uprawnione i algorytmy stosowane w przypadku wykorzystania rejestrów rozproszonych z węzłami bezuprawnień, są różne. W przypadku rejestrów rozproszonych z węzłami bez uprawnień, węzły nie ufają sobie. Dlatego potrzebna jest pewnego rodzaju nagroda za zgłaszanie bloków zawierających poprawne transakcje. Konieczne jest także ograniczenie możliwości oszukiwania lub ukarania użytkowników próbujących oszukiwać. Stąd też algorytmy konsensusu są zasobochłonne lub wymagają zaliczki, która jest tracona w momencie oszustwa. W przypadku rejestrów rozproszonych z węzłami uprawnionymi może istnieć pewien poziom zaufania. W takim przypadku nie ma potrzeby stosowania zasobochłonnego algorytmu konsensusu albo wykorzystania tokenów symbolizujących walutę.

W wielu publikacjach powtarza się, że zaletą rejestrów rozproszonych jest niezmienność, co nie jest prawdą [198]. Przykładem sytuacji, gdy dodane dane do rejestru rozproszonego mogą być zmienione jest porzucenie alternatywnych wersji łańcucha bloków w sytuacji dodania w podobnym czasie bloków przez różne węzły, czy też sytuacja twardego rozwidlenia, w której można wycofać ostatnio dodane dane. Do twardego rozwidlenia może dojść w sytuacji, gdy wprowadzona nowa wersja protokołu nie jest kompatybilna ze starszą wersją. Nowa wersja jest

najczęściej wprowadzana w sytuacji wykrycia luki w obecnej wersji protokołu lub też zmiany mechanizmu konsensusu.

Oczywiście rejestr rozproszony również może być podatny na wiele ataków. Jednym z nich jest atak 51%, który może wystąpić wtedy, gdy grupa węzłów, będąca w posiadaniu jednej osoby lub organizacji, ma ponad 50% mocy obliczeniowej sieci lub też posiada ponad 50% tokenów. Może dzięki temu przejąć sieć, ponieważ to węzły tej organizacji będą odpowiedzialne za dodawanie nowych bloków. Atak ten jest bardzo kosztowny do przeprowadzenia oraz nie koniecznie musi przynieść zysk. Innym atakiem jest atak podwójnego wydatku (ang. double spending). W tym ataku możliwe jest wydanie tego samego tokenu więcej niż raz.

Rejestry rozproszone są określane systemami bez zaufanej trzeciej strony. Co jest prawdą tylko w rejestrach rozproszonych publicznych z węzłami bez uprawnień [198]. W innych kategoriach rejestrów rozproszonych musi istnieć grupa administratorów, która zarządza całym rejestrem rozproszonym. Przy korzystaniu z rejestrów rozproszonych musi istnieć również zaufanie do wykorzystanych technologii kryptograficznych, implementacji protokołu i kodów łańcuchowych, które mogą mieć błędy. Użytkownicy, korzystający z rejestru rozproszonego, także muszą mieć zaufanie, że ich transakcja będzie przetworzona w sposób uczciwy. W rejestrze rozproszonym powinno być również zaufanie do innych użytkowników, że nie zmówią się w grupę mogącą wykonać atak 51%.

Niektóre implementacje rejestrów rozproszonych wspierają programy zwane *smart contract*. Tłumaczenie tego terminu jako "kontrakt inteligentny"nie oddaje istoty tego pojęcia, dlatego w rozprawie użyto określenia "kod łańcuchowy". Sam termin *smart contract* został wprowadzony przez Nicka Szabo w 1994 roku. Kodem łańcuchowym nazywamy program, który weryfikuje warunki kontraktu zapisanego w formie zrozumiałej przez komputery [131]. Kody łańcuchowe rozszerzają możliwości rejestrów rozproszonych, dając możliwość stworzenia własnych warunków dodania oraz odczytania danych do/z rejestru. Kody łańcuchowe są realizowane przez węzły rejestru rozproszonego, wszystkie węzły muszą uzyskać ten sam wynik, aby transakcja była uznana za poprawną. Dlatego też kody łańcuchowe, jeżeli chcemy dodać dane do rejestru, muszą być deterministyczne. Kod łańcuchowy jest wykonywany z parametrami dostarczonymi przez użytkownika przy każdej próbie dodania transakcji do rejestru rozproszonego. Jeżeli kod łańcuchowy musi korzystać z innych danych niż dane wejściowe oraz dane zapisane w rejestrze rozproszonym, to konieczne jest skorzystanie z usług zwanymi wyroczniami (ang. oracles). Kod łańcuchowy zapisany jest w rejestrze rozproszonym, widoczny dla każdego użytkownika sieci, a jakakolwiek zmiana kodu łańcuchowego wymaga osiągnięcia

konsensusu przez węzły. Do utworzenia kodów łańcuchowych należy użyć języka wspieranego przez implementację rejestru rozproszonego. Niektóre implementacje umożliwiają pisanie kodów łańcuchowych w językach powszechnie wykorzystywanych takich jak Node.js [55], Golang [55], Web Assembly [189], Rust [188] natomiast inne wymagają języków opracowanych specjalnie do tego celu np. Solidity [52], Vyper [190].

2.4 Wybór implementacji rejestru rozproszonego

Ze względu na wiele implementacji rejestru rozproszonego ważne jest wybranie odpowiedniej, którą można wykorzystać w opracowywanym protokole uwierzytelniania urządzeń. Ze względu na to, że niektóre implementacje rejestrów rozproszonych są tworzone z myślą o wykorzystaniu w konkretnych rozwiązaniach, np. kryptowalutach, to wybrane rozwiązanie musi się cechować brakiem specjalizowania lub też być przygotowane do wykorzystania w systemach uwierzytelniania i autoryzacji. Innym wymogiem jest zapewnienie odpowiedniej skalowalności i przepustowości. Istotne jest także, aby implementacja obsługiwała kody łańcuchowe oraz można było utworzyć kilka oddzielnych rejestrów. Da to większą kontrolę nad dostępem do danych oraz pomoże zapewnić ich izolację. Bezpieczeństwo przechowywanych danych może być także zwiększone, jeżeli implementacja będzie obsługiwać dane prywatne, tj. w rejestrze będzie przechowywana tylko wartość skrótu danych, a same dane będą poza rejestrem bądź też dane wrażliwe będą mogły być przechowywane w postaci zaszyfrowanej. Ważne jest także, aby wybrana implementacja była dopracowana, miała społeczność, która wykorzystuje ją w swoich projektach lub też miała wsparcie dużych firm. Takie cechy dają większe prawdopodobieństwo, że będzie ona dalej rozwijana oraz będą wydawane aktualizacje bezpieczeństwa. Cechą pomocną jest modularność implementacji, dzięki czemu np. można dobrać odpowiedni algorytm konsensusu do konkretnego rozwiązania.

Wybór implementacji rejestru rozproszonego należy rozpocząć od określenia, jaki typ rejestru będzie odpowiedni do zdefiniowanych wymagań. Ponieważ system uwierzytelniania i autoryzacji musi mieć cechę prywatności oraz kontroli danych, to najlepszym rozwiązaniem będzie wykorzystanie prywatnego rejestru rozproszonego z uprawnionymi węzłami. Skorzystanie z takiego rejestru poza wymienionymi cechami będzie charakteryzować się większym zaufaniem do użytkowników, szybszym zatwierdzaniem transakcji czy też brakiem opłat.

Na podstawie wyników porównania kilku implementacji rejestrów rozproszonych przedstawionych w tabeli 2.3 wybrano Hyperledger Fabric jako implementację do przygotowania

wersji Proof of Concept (PoC) protokołu uwierzytelniania i autoryzacji urządzeń IoT. Hyperledger Fabric jest implementacją rejestru prywatnego z węzłami uprawnionymi. W związku z tym wymagane jest uwierzytelnianie wszystkich członków sieci (użytkowników i węzłów), a w przypadku konieczności np. weryfikacji transakcji także posiadanie odpowiednich uprawnień. Hyperledger Fabric (HLF) umożliwia wykorzystanie kodów łańcuchowych. Uzyskano dzięki temu możliwość pisania własnych reguł dodawania danych oraz odczytywania danych zapisanych w rejestrze rozproszonym. Sam język pisania kodu łańcuchowego (Golang) jest językiem powszechnie wykorzystywanym (nie został opracowany specjalnie do wykorzystania w Hyperledger Fabric), co ułatwia przewidywaną implementację protokołu. Wykonanie dowolnej operacji na rejestrze rozproszonym jest możliwe tylko przez wykonanie kodu łańcuchowego. Hyperledger Fabric wspiera możliwość tworzenia wielu rejestrów rozproszonych (wielu kanałów) w jednej sieci. Każdy kanał może przechowywać innego rodzaju dane, a sam dostęp do konkretnego kanału może być ograniczony dla poszczególnych członków sieci. Poza tym zmiana kodu łańcuchowego wymaga zatwierdzenia tylko przez członków tego kanału, a nie wszystkich użytkowników sieci. HLF nie jest specjalizowany do wykorzystania tylko w konkretnym przypadku np. tylko do implementacji kryptowaluty. Hyperledger Fabric jest rozwiązaniem modułowym. W starszej wersji można było wybrać algorytm konsensusu – dla usługi Ordering obecnie dostępny jest tylko Raft. Można natomiast wybrać typ wykorzystywanej bazy danych. Hyperledger Fabric ma także możliwość wykorzystania Hardware Security Module (HSM) oraz umożliwia przechowywanie danych prywatnych (dane te nie są przechowywane w rejestrze rozproszonym, a tylko ich wartość skrótu, ale muszą spełniać reguły zapisane w kodzie łańcuchowym). Kod Hyperledger Fabric jest otwarty, a samo rozwiązanie jest dojrzałe oraz dopracowane. Hyperledger Fabric jest rozwijany przez wiele dużych firm, a także istnieje możliwość szybkiego utworzenia sieci, wykorzystując dostępne usługi chmurowe.

Każdy z węzłów Hyperledger Fabric składa się z dwóch usług: *rówieśnika* (ang. peer) oraz zamawiania (ang. ordering). Nie jest konieczne, aby każdy węzeł Hyperledger Fabric miał uruchomione te obie usługi. Usługa *peer* jest odpowiedzialna za wykonywanie kodów łańcuchowych, przechowywanie kodów łańcuchowych, przechowywanie oraz aktualizację rejestru. Każda usługa *peer* może pełnić kilka ról [85]. Pierwszą jest *lider*, który w sytuacji gdy organizacja ma wiele węzłów w kanale, to węzeł z tą rolą jest odpowiedzialny za odebranie bloku od lidera z usługą *ordering* i przekazanie go do węzłów wewnątrz organizacji z rolą *committing*. Węzły z rolą *zobowiązującą* (ang. committing) otrzymują bloki do zatwierdzenia, weryfikują je i dodają do przechowywanego rejestru rozproszonego. Węzeł Hyperledger Fabric może mieć

Tabela 2.3: Porównanie wybranych implementacji prywatnych rejestrów rozproszonych z węzłami uprawnionymi

Właściwość	**Hyperledger Fabric**	**Hyperledger Besu**	**IOTA**	**Quorum**
Ograniczony dostęp do danych	Tak	Tak	Tak	Tak
Modularność	Tak	Nie	Nie	Nie
Prywatne dane	Tak	Tak	Tak	Tak
Skalowalność i przepustowość	Wysoka	Wysoka	Wysoka	Wysoka
Oddzielne rejestry	Tak	Nie	Nie	Nie
Wspieranie kodów łańcuchowych	Tak	Tak	Tak	Tak

także rolę *zatwierdzający* (ang. endorsing). Węzeł z taką rolą ma zainstalowany kod łańcuchowy, odbiera żądania od aplikacji i może generować propozycje transakcji. Ostatnią rolą jest rola *zakotwiczony* (ang. anchor). Rola ta jest przydzielana węzłom wykorzystywanym do komunikacji z innymi węzłami Hyperledger Fabric należącymi do innych organizacji. Każda organizacja musi mieć minimum jeden węzeł z tą rolą.

Węzeł w usłudze ordering może pełnić jedną z dwóch ról. Rolę *lidera* (ang. leader) może mieć tylko 1 węzeł w całej sieci. Węzeł z tą rolą jest odpowiedzialny za tworzenie bloku z otrzymywanych transakcji. Usługa *ordering* w Hyperledger Fabric korzysta z algorytmu konsensusu nazwanego Raft. *Lider* spośród wszystkich węzłów wybierany jest w sposób automatyczny. Nikt nie ma wpływu na sposób wyboru lidera w ramach tego protokołu. Pozostałe węzły mają rolę *śledzący* (ang. follower). Jeżeli lider nie jest osiągalny, to ze zbioru pozostałych węzłów wybierany jest nowy lider. Węzły z tą rolą odbierają wiadomości od lidera, aby posiadać identyczny stan wiedzy co lider i móc go zastąpić w razie jego niedostępności.

Poza rejestrem rozproszonym Hyperledger Fabric posiada także dodatkową bazę danych. Baza ta przechowuje ostatnie wartości dla każdego identyfikatora. Jej zadaniem jest skrócenie czasu odpowiedzi poprzez odpytanie bazy danych o wartość dla identyfikatora, a nie przejrzenie całego rejestru rozproszonego. Domyślnie wykorzystywana jest tu baza LevelDB [67], która jest bazą typu NoSQL. Zamiast niej można wykorzystać CouchDB [61], która przechowuje dane w postaci JSON jest mniej wydajna od LevelDB, ale obsługuje proste zapytania.

2.5 Zarządzanie tożsamościami urządzeń w systemach IoT

Dużym problemem w sieciach IoT jest identyfikacja urządzeń. Jest ona konieczna, jeżeli chcemy zapewnić bezpieczne połączenie z urządzeniem IoT. Sam proces identyfikacji musi nastąpić przed faktycznym uwierzytelnianiem urządzenia IoT. Aby móc przeprowadzić proces identyfikacji urządzenia, trzeba nadać mu tożsamość. Tożsamość urządzenia IoT jest cyfrową reprezentacją podmiotu stworzoną przez niego samego lub przez inny podmiot [41]. Pojedyncza tożsamość jest powiązana wyłącznie z jednym urządzeniem IoT, jednak urządzenie może posiadać kilka tożsamości. Każda tożsamość obejmuje istotne atrybuty w kontekście np. różnych organizacji czy też aplikacji. Zgodnie z zaleceniami ITU-T [91] tożsamość składa się z trzech komponentów: identyfikatora, atrybutów oraz danych uwierzytelniających. Identyfikator to seria cyfr, liter i symboli używana do utworzenia niepowtarzalnego ciągu. Według zaleceń Internet Engineering Task Force (IETF) [87] można w tej roli wykorzystać Universally Unique Identifier (UUID) [112]. Atrybutami są fragmenty danych, które określają charakterystykę podmiotu będącego właścicielem tożsamości. Natomiast poświadczenia są danymi, które mogą być wykorzystane przez podmiot do potwierdzenia swojej tożsamości. Samo utworzenie i nadanie tożsamości jest tylko fragmentem całego cyklu życia urządzenia IoT. Aby zarządzać i zapewnić bezpieczeństwo dostępu do zasobów danych, a także w celu ochrony samych urządzeń, stosuje się szereg technologii i procesów zwanych procesem zarządzania tożsamością i dostępem (ang. Identity and Access Management (IAM)). IAM obejmuje takie procesy jak nadanie, usuwanie tożsamości, uwierzytelnianie tożsamości, autoryzację urządzenia posiadającego nadaną tożsamość w celu umożliwienia skorzystania z usługi. To właśnie IAM zapewnia, że tylko uwierzytelnione urządzenia mają dostęp do określonych usług. Istnieje wiele modeli procesu IAM [41], jednak na potrzeby tej rozprawy istotne są tylko dwa. Pierwszym z nich jest model federacji. Model ten wymaga trzech elementów: użytkownika, dostawcy usług Service Provider (SP) oraz dostawcy tożsamości Identity Provider (IdP). Użytkownikiem w tych rozważeniach będzie urządzenie IoT, które zamierza uzyskać dostęp do jakiejś usługi. Usługa ta może być oferowana w chmurze, ale równie dobrze może być oferowana przez inne urządzenie IoT. Dostawca usług SP to element, który oferuje usługę, ale zazwyczaj nie oferuje żadnej metody potwierdzenia tożsamości i dostępu, lecz polega na dostawcy tożsamości IdP. Natomiast dostawca tożsamości IdP jest elementem, który przechowuje i potwierdza tożsamość użytkowników. W modelu federacyjnym dostawcy tożsamości IdP najczęściej replikują między sobą dane o tożsamościach, natomiast dostawcy usług SP współdzieleni są pomiędzy

organizacjami. Rezultatem takiego podejścia jest możliwość korzystania z usług oferowanych przez różne organizacje bez konieczności posiadania wielu tożsamości. Tożsamość urządzenia jest zarządzana przez organizację, do której należy urządzenie IoT. Poza zaletami takimi jak większa wydajność, zmniejszenia skutków awarii pojedynczych dostawców tożsamości IdP czy też możliwości korzystania z Single Sign-On (SSO) takie podejście ma też swoje wady. Tożsamości mogą zawierać cenne informacje, które powinny być chronione tak, aby nie były udostępnione podczas replikacji pomiędzy dostawcami tożsamości IdP, pomimo że pomiędzy organizacjami istnieje umowa określająca zasady współpracy.

Podejściem pozwalającym na spełnienie wymogu zachowania prywatności tożsamości jest wykorzystanie pseudonimów dla tożsamości [4]. W takim podejściu tożsamość jest replikowana w zakresie tylko niezbędnych atrybutów oryginalnej tożsamości, natomiast identyfikatory zastępowane są pseudonimami. Pozwala to uzyskać coś zbliżonego do anonimowości pod warunkiem, że zostanie zachowana prywatność powiązania oryginalnej tożsamości z pseudonimem. Autorzy publikacji [41] zwracają uwagę również na problem utrzymania spójności informacji o tożsamościach podczas modyfikacji lub odwoływania. Problemem może być także to, że istnieje wiele standardów umożliwiających wymianę informacji pomiędzy dostawcami tożsamości np. Security Assertion Markup Language (SAML), Open Authorization (OAuth) czy też OpenID, gdzie każda z nich ma swoje zalety i wady.

Drugim modelem jest model suwerenny, którego głównym elementem jest rejestr rozproszony. Organizacje, posiadające węzły rejestru rozproszonego, rejestrują tożsamości urządzeń IoT w rejestrze rozproszonym. Wykorzystanie rejestru rozproszonego wprowadza wszystkie implikacje związane z tą technologią, m.in. brak koordynatora (decentralizacja), niezmienność danych zapisanych w rejestrze czy też dostępność danych dla wszystkich organizacji tworzących ten rejestr. W tym modelu to rejestr rozproszony pełni rolę dostawcy tożsamości IdP i każdy, kto ma dostęp oraz odpowiednie uprawnienia może skorzystać z tego dostawcy. Wadą takiego podejścia jest problem zaufania do organizacji, które będą miały dostęp danych przechowywanych w rejestrze rozproszonym. Ze względów bezpieczeństwa oraz prywatności przechowywanie danych wrażliwych nie jest zalecane. Aby przeciwdziałać ujawnianiu danych wrażliwych, zaproponowano przez badaczy podejście wykorzystujące dowód z wiedzą zerową (ang. zero-knowledge proof) [41]. W tym podejściu wadą jest wprowadzenie dodatkowych elementów, którym trzeba zaufać, ponieważ to one poświadczają tożsamość urządzenia IoT, a rejestr rozproszony służy tylko jako miejsce przechowywania tych potwierdzeń. Natomiast zaletą tego podejścia jest to, że tylko właściciel tożsamości jest odpowiedzialny za jej ochronę

i bez jego działania nie zostanie ona upubliczniona.

Proces zarządzania tożsamością i dostępem dla środowisk IoT musi być skalowalny, aby móc obsłużyć ogromną liczbę urządzeń. Ważne jest także zapewnienie dostępności oraz wydajności, aby dostarczenie tożsamości, zmiana tożsamości (w tym zmiana praw dostępu) była możliwie jak najszybsza, aby atakujący nie mógł wykorzystać ujawnionych poświadczeń. Ważne jest także dostosowanie procesu uwierzytelniania i autoryzacji do możliwości urządzeń IoT, m.in. takie procesy muszą być zautomatyzowane tak, aby niepotrzebna była ingerencja właściciela urządzenia IoT, czy też uwzględniały możliwość przemieszczania się urządzeń IoT. Obecnie wiele prac badawczych poświęconych jest opracowywaniu procesów zarządzania tożsamością i dostępem IAM tak, aby były one dostosowane do środowiska IoT [63, 203]. Jednym z aspektów poruszanych w publikacjach jest poprawa relacji pomiędzy użytkownikami IoT i rozumienia tożsamości, tak aby procesy IAM były lepiej dostosowane do heterogenicznego środowiska IoT [41]. Badacze zwracają uwagę [41], że w wielu pracach dostawcy tożsamości są przedstawiani jako płaska baza danych z prostymi relacjami i nieuwzględniana jest dynamiczność środowiska IoT. Innym z aspektów jest uwzględnienie kontekstu przy procesach uwierzytelniania i autoryzacji. Obecnie proponowane mechanizmy uwierzytelniania i autoryzacji mają statycznie zdefiniowane reguły i nie są przystosowane do dynamiczności środowiska IoT. Kolejną kwestą jest pytanie, czy istnieje zapotrzebowanie na wspólny, znormalizowany organ wydający standaryzowane identyfikatory urządzeń IoT o zasięgu globalnym [106]. Pewne prace w tym kontekście są już prowadzone przez badaczy. W artykule [154] zaproponowano wykorzystanie sieci Software-Defined Network (SDN) oraz usług chmurowych do nadawania identyfikatorów w sposób globalny. Istnieje jeszcze problem sposobu identyfikowania urządzeń IoT. Obecnie jest możliwe skorzystanie z wielu standardów, które nie są ze sobą kompatybilne. Niektóre, tak jak UUID, nie mają zawartej w sobie żadnej informacji o urządzeniu, natomiast przykładowo identyfikator w standardzie oneM2M zawiera informację o producencie i typie urządzenia [107]. W wielu artykułach porusza się temat wykorzystania sztucznej inteligencji do identyfikacji urządzeń IoT [103, 121]. Z jednej strony takie podejście, bazujące na analizie zachowania urządzeń (m.in. częstotliwości wysyłania wiadomości, typu wiadomości) do identyfikacji urządzeń, może pozwolić na identyfikację urządzeń przejętych przez atakujących, ponieważ ich profil zachowania będzie różnił się od poprzedniego. Z drugiej strony, gdy atakujący wie, jakie cechy są wykorzystywane do identyfikacji urządzenia, może próbować dostosować komunikację tak, aby utrudnić wykrycie przejęcia urządzenia. Takie metody identyfikacji nie zapewniają poziomu pewności jak uwierzytelnianie przez urządzenie IoT z wykorzystaniem

identyfikatora i poświadczeń, ale mogą stanowić dodatkowy czynnik zwiększający zaufanie do urządzenia.

Uwierzytelnianie to proces weryfikacji tożsamości podanej przez urządzenie IoT. Sam wybór sposobu uwierzytelniania jest bardzo istotny, ponieważ w większości przypadków raz wybrany sposób jest później trudny do zmiany. Sposobów uwierzytelniania urządzeń IoT jest bardzo dużo, dlatego też próbuje się wprowadzić różne taksonomie stosowanych metod uwierzytelniania przy użyciu kryteriów wybranych na podstawie podobieństw i głównych cech tych metod. Bazując na znalezionych taksonomiach [6, 7, 76], można wyróżnić wiele kryteriów. Pierwszym z kryteriów jest warstwa, w której zachodzi proces uwierzytelniania. W literaturze można spotkać się z podziałem środowiska IoT na trzy do siedmiu warstw [110]. Dodatkowe warstwy w architekturach więcej niż trójwarstwowych wiążą się z podziałem warstw architektury trójwarstwowych na bardziej szczegółowe lub też wprowadzają warstwy abstrakcyjne, takiej jak: warstwa biznesowa (wprowadzająca pojęcie zarządzania całym systemem IoT w szczególności modelami biznesowymi) czy też warstwa środowiska (dodająca świadomość istnienia innych obiektów wokół urządzeń IoT). Ponieważ trójwarstwowa architektura środowiska IoT jest podstawową architekturą, to właśnie ona została wykorzystana jako opis tego kryterium. Uwierzytelnianie może nastąpić w: warstwie percepcji, czyli w warstwie zawierającej elementy odpowiedzialne za wykrywanie, zbieranie i przechowywanie danych z czujników; warstwie sieci, która zawiera elementy odpowiedzialne za odbiór danych z warstwy percepcji i przesłanie ich do warstwy aplikacji; lub też w warstwie aplikacji, w której znajdują się elementy odpowiedzialne za odbieranie danych z sieci i dostarczenie ich użytkownikowi w sposób zrozumiały dla niego.

Innym kryterium jest czynnik uwierzytelniający, na podstawie którego weryfikowana jest tożsamość. Czynnikiem może być [60]: coś, co urządzenie posiada; coś, czym urządzenie jest; coś, co urządzenie ma wbudowane (np. PUF) lub też kontekst, w którym funkcjonuje urządzenie. Następnym kryterium jest sposób uwierzytelniania, tj. wykorzystanie tokenów, utworzonych przez dostawcę tożsamości IdP lub uwierzytelnianie bez tokenów. W procesie uwierzytelniania może brać udział różna liczba stron, które muszą się uwierzytelnić. Takie kryterium uwzględni uwierzytelnianie jednostronne, gdy tylko jedna strona uwierzytelnia się; dwustronne, gdy obie strony uwierzytelniają się wzajemnie; lub też trójstronne, gdy wykorzystywana jest zaufana trzecia strona.

Architektura procesu uwierzytelniania także może stanowić kryterium klasyfikacji metod uwierzytelniania. W ramach architektury uwierzytelniania można wyróżnić uwierzytelnianie

z wykorzystaniem centralnego serwera uwierzytelniania. W takim przypadku mówimy o architekturze scentralizowanej. Jeżeli proces uwierzytelniania przebiega bez centralnego serwera pomiędzy rozproszonymi węzłami, to jest to architektura rozproszona. Rozwiązania wykorzystujące rejestry rozproszone należą do tej kategorii.

Niektóre metody uwierzytelniania mogą wykorzystywać dodatkowe komponenty sprzętowe. Jeżeli wykorzystywany komponent przechowuje poświadczenia oraz wykonuje na nich operacje, to jest to jawny sposób wykorzystania. Przykładem takiego komponentu jest Trusted Platform Module (TPM). Przeciwieństwem jawnego sposobu wykorzystania jest niejawny sposób wykorzystania, w którym wykorzystywane są cechy fizyczne komponentu. Wykorzystanie komponentu do uwierzytelniania w sposób niejawny ma miejsce w przypadku wykorzystania układów PUF. Kryterium klasyfikacji jest też sesyjność procesu uwierzytelniania. Po wykonaniu procesu uwierzytelniania może zostać zestawiona sesja, która będzie trwała, dopóki będzie zestawione połączenie sieciowe między stronami. Można także przeprowadzać proces uwierzytelniania przy wysłaniu każdej wiadomości. Powoduje to wysłanie większej liczby bajtów i wiadomości, ale nie jest tworzona sesja pomiędzy stronami komunikacji.

Ostatnim kryterium, jakie można wyróżnić, jest rodzaj wykorzystanej kryptografii w procesie uwierzytelniania. Wykorzystywana jest kryptografia asymetryczna, w ramach której można wyróżnić wykorzystanie certyfikatów jawnych, niejawnych, jak i podejście bez certyfikatów. Wykorzystanie certyfikatów pozwala drugiej stronie zweryfikować, czy klucz publiczny użyty do uwierzytelniania jest autentyczny. Sam certyfikat może być również przechowywany na karcie chipowej (certyfikat sprzętowy) lub w pamięci urządzenia (certyfikat programowy). Poza kryptografią asymetryczną wykorzystywana jest również kryptografia symetryczna, funkcje skrótu lub uwierzytelnianie może nastąpić bez wykorzystania metod kryptograficznych. Do metod uwierzytelniania bez wykorzystania kryptografii można zaliczyć np. uwierzytelnianie podstawowe (ang. basic auth), w którym porównuje się dane identyfikujące podane przez urządzenie z zapisanymi u drugiej strony komunikacji. Inną metodą uwierzytelniania bez wykorzystania kryptografii jest uwierzytelnianie typu wyzwanie-odpowiedź (ang. challenge—response), w którym urządzenie otrzymuje wyzwanie, które przetwarza wykorzystując operacje XOR lub przesunięć bitowych oraz sprzętowych układów takich jak PUF. Wynik operacji jest zwracany w celu uwierzytelniania urządzenia. Przykład wykorzystania takiego schematu przedstawili autorzy [57, 71]. W literaturze [7] można jeszcze spotkać uwierzytelnianie z uzgadnianiem klucza, jednak w tym schemacie uwierzytelnianie jest tylko fragmentem całego procesu, w którym można uwierzytelnić urządzenie lub całą grupę urządzeń z wyko-

rzystaniem kryptografii symetrycznej bądź asymetrycznej.

W procesie zarządzania tożsamością i dostępem ważne jest także odpowiednie zarządzanie uprawnieniami do usług oferowanych przez organizację. Proces sprawdzenia, czy podmiot może wykonać operację na obiekcie nazywany jest autoryzacją [172]. Poprzez podmiot w środowiskach IoT należy rozumieć urządzenie IoT lub usługę [84]. Obiekt to zasób systemowy, do którego dostęp jest zarządzany przez system kontroli dostępu [84]. Obiektem może być plik, katalog, usługa, proces. Operacja jest wykonywana na obiekcie na żądanie podmiotu [84]. Do operacji można zaliczyć m.in.: odczyt, modyfikację, usunięcie, zapis, kopiowanie. Aby proces autoryzacji mógł zostać przeprowadzony, podmiot musi zostać najpierw uwierzytelniony, chociaż istnieją wyjątki. Przykładowo, projektując aplikację wizualizującą dane z sensorów IoT, dostęp do danych może być przydzielony tylko uwierzytelnionym użytkownikom posiadającym określone prawa dostępu. Jednak dostęp do danych z sensorów publicznych nie musi wymagać żadnych praw, a tym samym podmiot nie musi się uwierzytelnić. Autoryzacja zależy od polityki dostępu konfigurowanych przez administratorów lub właścicieli obiektów. Polityka to reprezentacja reguł lub relacji, która umożliwia określenie, czy żądany dostęp jest dozwolony [84]. Sam proces autoryzacji jest bardzo ważnym elementem systemu dostępu do obiektów. Konieczne jest zapewnienie poufności danych, a tym samym ograniczenie dostępu do nich tylko określonym podmiotom.

W procesie zarządzania prawami dostępu ważne jest, aby przestrzegać trzech zaleceń [142]. Pierwszym z nich jest nadawanie jak najmniejszych uprawnień do zasobu. Podmiot powinien posiadać minimalne uprawnienia do zasobu, ale wystarczające do wykonania przez niego wymaganego zadania. Kolejnym zaleceniem jest stosowanie się do zasady *Domyślnie zabronione*. Jeżeli podmiot nie ma przydzielonych żadnych praw lub przydzielone prawa nie pasują do wymaganych przez obiekt, to podmiot nie powinien mieć dostępu do zasobu. Natomiast ostatnim zaleceniem jest logowanie wszystkich prób dostępu. Wszystkie próby dostępu do zasobu powinny być zapisywane.

W przypadku budowania systemu zarządzania tożsamością i dostępem IAM należy wybrać odpowiedni model kontroli dostępu. Różne modele kontroli dostępu starają się rozwiązać problem *większe bezpieczeństwo czy łatwiejsza administracja*. Z jednej strony dla większego bezpieczeństwa lepiej jest, aby każde prawo do obiektu było bardziej granularne, a tym samym podmiot miał zdefiniowane więcej szczegółowych praw. Z drugiej strony dla łatwiejszego administrowania lepiej jest mieć mniej praw do zarządzania [99]. W związku z tym nie każdy z modeli będzie odpowiedni do zastosowania w środowisku IoT. Każdy model kontroli dostępu

musi spełnić wymogi dotyczące prywatności danych oraz poprawić egzekwowanie zasad bezpieczeństwa przy jednoczesnym obniżeniu ryzyka ujawnienia nieuprawnionemu podmiotowi danych oraz kosztów zarządzania kontrolą dostępu.

Uznaniowa kontrola dostępu (Discretionary Access Control (DAC)) oparta jest na tożsamości podmiotu i regułach dostępu określających, co podmiotowi wolno (lub czego nie wolno) robić. W przypadku uznaniowej kontroli dostępu właściciel obiektu ma pełne prawa do tego obiektu i może delegować uprawnienia do innych podmiotów. Właściciel może też przenieść własność obiektu na inny podmiot. Jest to jedna z podstawowych metod kontroli dostępu, używana powszechnie nie tylko w środowiskach IoT. Najczęściej uprawnienia są zapisywane w formie list kontroli dostępu (Access Control List (ACL)). Brak centralnego zarządzania sprawia, że ten model jest łatwy do wdrożenia w rozproszonych aplikacjach. Do wad tego modelu trzeba zaliczyć brak wiedzy użytkowników o stanie systemu, przez co można nieświadomie nadać uprawnienia nieautoryzowanym podmiotom. W przypadku sieci IoT wadą jest też duża liczebność urządzeń, każdy właściciel urządzenia IoT musi nadać indywidualnie uprawnienia do każdego urządzenia IoT. Przykład wykorzystania tego modelu w sieci IoT opisano w [97].

Obowiązkowa kontrola dostępu (Mandatory Access Control (MAC)) to wymuszona przez system metoda ograniczenia dostępu do obiektów, bazująca na porównaniu etykiet bezpieczeństwa (określające jak wrażliwe, krytyczne są obiekty) z poświadczeniami bezpieczeństwa przypisanymi podmiotom [173]. Obowiązkowa kontrola dostępu zapewnia scentralizowany sposób wprowadzania reguł kontroli dostępu, natomiast sam proces egzekwowania może być lokalny. Tylko administrator systemu może zdefiniować klasyfikację obiektu. Obiekty tworzone przez podmiot dziedziczą posiadane przez niego prawa. Zaletą takiego podejścia jest ograniczenie możliwości ujawnienia informacji poprzez większą czytelność przepływu informacji pomiędzy obiektami z określonymi poziomami wrażliwości zdefiniowanymi w etykietach tych obiektów. Wadą tego modelu w przypadku środowisk IoT jest utrudnione zarządzenie etykietami podmiotów oraz obiektów ze względu na ich liczebność, a także zmienność etykiet obiektów. Przykład użycia tego modelu w sieciach IoT przedstawiono w [102].

W modelu kontroli dostępu opartej na rolach (Role-based Access Control (RBAC)), rola jest utożsamiana z funkcją, jaką pełni podmiot w organizacji [173]. Model kontroli opartej na rolach jest bardziej elastyczny niż model obowiązkowej czy uznaniowej kontroli dostępu, ponieważ podmiotom można przypisać kilka ról, a rola może być związana z wieloma podmiotami. Poszczególnym rolom są przypisywane stosowne uprawnienia w sposób centralny.

W odróżnieniu od DAC uprawnienia nie są przypisywane użytkownikom bezpośrednio. Wadą tego modelu jest to, że role mogą być przypisane w sposób, który będą tworzyć luki w polityce kontroli dostępu. Luki takie mogą na przykład wynikać z sytuacji dodania nowych uprawnień do roli, która jest przypisana także innym podmiotom, które nie powinny posiadać tych uprawnień. W przypadku IoT ten model może znaleźć zastosowanie. Każdemu urządzeniu możemy przypisać zbiór ról, w których urządzenie może funkcjonować. Ten model kontroli dostępu jest powszechnie wykorzystywany w środowisku IoT [16, 74, 93, 119]. Jest on także zaimplementowany w formie rozszerzonej o domeny. W tym rozszerzonym przypadku urządzenie IoT może mieć przypisane role należące do różnych domen, a tym samym inne uprawnienia w innych domenach. Proces przypisania roli w innej domenie wymaga zgody administratora drugiej domeny.

Tradycyjna kontrola dostępu opiera się na tożsamości podmiotu żądającego możliwości wykonania operacji na obiekcie bezpośrednio lub poprzez predefiniowane role przypisane do podmiotu. Jednak takie podejście jest kłopotliwe w zarządzaniu, biorąc pod uwagę potrzebę bezpośredniego powiązania podmiotu z prawami lub rolą. Dodatkowo zauważono, że same uprawnienia są często niewystarczające do wyrażenia rzeczywistych zasad kontroli dostępu [84]. Model kontroli dostępu oparty na atrybutach (Attribute-based Access Control (ABAC)) wykorzystuje atrybuty podmiotu, obiektu oraz aktualne warunki kontekstowe do podjęcia decyzji o wykonaniu (lub nie) operacji na obiekcie. Poprzez atrybut rozumie się cechę podmiotu, obiektu lub otoczenia. Atrybut jest reprezentowany poprzez parę *nazwa:wartość*. Cechy otoczenia to kontekst operacyjny lub sytuacyjny, który występuje podczas próby wykonania operacji na obiekcie. Do cech otoczenia można zaliczyć m.in.: czas, datę, lokalizację, aktualną sytuację (np. poziom zagrożenia). System kontroli dostępu po odebraniu żądania wykonania operacji pobiera atrybuty podmiotu oraz obiektu i na podstawie określonej polityki dokonuje decyzji, czy podmiot może wykonać żądaną operację. W tym modelu ważne jest, aby nadać odpowiednie atrybuty podmiotom i obiektom. Atrybuty podmiotu są przypisywane i zarządzane przez administratora systemu. Atrybuty obiektów są nadawane wraz z utworzeniem obiektu przez właściciela obiektu lub administratora. Przykład wykorzystania tego modelu autoryzacji w środowisku IoT przedstawiono w artykułach [49, 191]. Wykorzystanie ABAC w środowisku federacyjnym przedstawiono w artykule [158].

Kontrola dostępu oparta na możliwościach (Capabillity Based Access Control (CapBAC)) opiera się na posiadaniu pewnego elementu (najczęściej tokenu), który daje podmiotowi uprawnienia dostępu do obiektu [75]. Token ten jest generowany przez usługę Policy Decision

Point (PDP) po wcześniejszym uwierzytelnianiu podmiotu, gdy podmiot ma uprawnienia do tego obiektu. Token poza informacją o wystawcy zawiera informacje o obiekcie oraz uprawnienia do tego obiektu. Token jest podpisywany przez wystawcę, co oznacza, że jest odporny na manipulacje i jednoznacznie identyfikowalny. Podpis jest weryfikowany w momencie próby zrealizowania operacji na obiekcie. Podmiot po otrzymaniu tokenu od usługi PDP może wysłać żądanie wykonania operacji na obiekcie wraz z otrzymanym tokenem. Jeżeli token jest poprawny i ważny (token zawiera informacje o czasie wystawienia i ważności), to operacja może być wykonana. Problemem w tym modelu jest zaufanie do wystawcy tokenu. W tabeli 2.4


```
{
"id": "edere0129",
"issuer": "-----BEGIN PUBLIC KEY-----\n
MIIBtjCCASsGByqGSM44BAEwggEeAoGBAMSU+10Bg6UzPclx/473cFrZTvbyaLS2\n
xmT6iojhYBM2lnXrBoErbrD5/x2CIFS8TgndpTW+fLXQfzsvFu/mtizn1Oe1TFZt\n
/i57LnULXXcgMYumrDfYXnKiTZEDtrMDCrk9G6b+1mIsVNCGVdjhr+Xuhw0kvsat\n
0536Hg3Fsca8xU9nVRfkmO1bscc26hOAog8RDWeFQdAHSIzKwg9B4o5Nn4qXlFcI\n
VvxIVU1x5wTYWw==\n
-----END PUBLIC KEY-----\n",
"issue_time": "2017-11-10 20:12:32",
"subject": "Samuel:128.226.76.37",
"resource": "http://128.226.78.217",
"starttime": "2017-11-10 18:12:32",
"endtime": "2017-11-13 16:12:32",
"issuer_sign": "302c0b84251618qsed276c7fe55befd09e4414d3fb81e5d34b",
"access_right": [
    {"resource": "/test/api/v1.0/dt/project",
        "action": "GET",
        "conditions": {"value": {"start": "14:12:32", "end": "19:32:32"},
        "type": "Timespan"}},
    {"resource": "/test/api/v1.0/dt/create",
        "action": "POST",
        "conditions": {"value": {"start": "17:12:32", "end": "19:32:32"},
        "type": "Timespan"}}
    ]
}
```


Rysunek 2.3: Przykład tokenu. Źródło: [197]

przedstawiono porównanie różnych modeli kontroli dostępu. Granularność, czyli możliwość stworzenia bardzo szczegółowych polityk, została oznaczona na wysoką dla modelu ABAC. ABAC przy sprawdzeniu uprawnień bierze pod uwagę aktualny kontekst operacyjny. Zarządzanie kontrolą dostępu jest trudne w przypadku modelu RBAC ze względu na konieczność analizy często rozbudowanych uprawnień ról, aby nie nadać podmiotowi nieuprawnionemu dostępu do obiektu. Zarządzanie uprawnieniami w modelu MAC jest czasochłonne ze względu na konieczność zaangażowania administratora systemu w proces nadawania uprawnień. Tylko model CapBAC ma możliwość delegowania uprawnień poprzez możliwość przekazania otrzymanego tokenu. Modele CapBAC oraz ABAC mogą być stosowane w bardzo rozbudowanych środowiskach IoT, ponieważ są łatwo skalowalne.

Modele MAC i DAC nie są odpowiednie dla systemów IoT ze względu na sposób nada-

Tabela 2.4: Porównanie różnych modeli kontroli dostępu

Cecha	**DAC**	**MAC**	**RBAC**	**ABAC**	**CapBAC**
Granularność	Zależy od wdrożenia	Niska	Niska	Wysoka	Niska
Kontekst	Nie	Nie	Nie	Tak	Nie
Łatwość zarządzania	Nie	Nie	Średnia	Tak	Tak
Delegowanie uprawnień	Tak	Nie	Nie	Nie	Tak
Skalowalność	Średnia	Nie	Średnia	Nie	Tak

wania uprawnień i zarządzania nimi, które przy dużych środowiskach są bardzo uciążliwe. Dla systemów IoT lepszym wydaje się model RBAC, który jest prostszy w zarządzaniu, ale nie oferuje precyzyjnej kontroli dostępu do zasobów. Model ABAC jest precyzyjny i odpowiedni dla systemów IoT, ale jest trudny w skalowaniu. Natomiast model CapBAC może nie być przystosowany do nadawania uprawnień w środowiskach federacyjnych. Wybór jednego z modeli RBAC, ABAC czy też CapBAC jest uzależniony od oczekiwań organizacji, która jest odpowiedzialna za wdrożenie systemu.

Poza wymienionymi modelami kontroli dostępu, które są znane i wykorzystywane istnieją także modele badawcze. Przykładem może być wykorzystanie rejestru rozproszonego do przechowywania uprawnień [165]. Model ten jest podobny do DAC, ponieważ to właściciele obiektów definiują uprawnienia. Zaproponowane podejście zapisuje w rejestrze informacje o dostępie, co z jednej strony zapewnia rozliczalność, ale z drugiej nie jest wydajne przy dużej liczbie urządzeń IoT.

2.6 Charakterystyka współczesnych urządzeń IoT

W środowiskach IoT można spotkać różne rodzaje urządzeń – od urządzeń małych, o niskim zapotrzebowaniu na energię elektryczną, po minikomputery. Ze względu na wydajność urządzenia IoT mają różne możliwości, a tym samym inne problemy bezpieczeństwa. Aby usystematyzować urządzenia IoT, wprowadzono podział na klasy [33]. Ponieważ procesy wytwarzania układów elektronicznych cały czas się rozwijają i tworzone są układy mniejsze o niższym zapotrzebowaniu na energię, a przy tym wydajniejsze, możliwe jest wykorzystanie urządzeń IoT do nowych zastosowań wykorzystujących np. uczenie maszynowe. Z tego też po-

wodu zostały zaproponowane inne klasyfikacje urządzeń IoT [201]. Nie wszystkie urządzenia mogą wspierać dowolną metodę uwierzytelniania, a niektóre nawet nie stosują żadnej.

Klasa 0 – sensory

Urządzenia w klasie 0 są bardzo ograniczone pod względem pamięci i możliwości przetwarzania danych. Posiadają mniej niż 10kB pamięci RAM oraz przestrzeń na dane mniejszą niż 100 kB. Nie mają zasobów obliczeniowych wymaganych do bezpiecznej komunikacji z wykorzystaniem Internetu, do komunikacji wykorzystują inne urządzenia w roli bramy. Urządzenia tej klasy zwykle posiadają wgrane oprogramowanie oraz mają wstępną konfigurację. Możliwość zarządzania tymi urządzeniami ogranicza się do otrzymywania danych diagnostycznych np. sygnał "keep alive", informacje o włączeniu/wyłączeniu urządzenia, podstawowe informacje o stanie sensora. Przykładem urządzenia tej klasy mogą być: sensory stanu zdrowia (Photoplethysmography (PPG), Phonocardiogram (PCG), Inertial Measurement Unit (IMU)), sensory wilgotności, temperatury. Urządzenia w tej klasie najczęściej wykorzystują komunikację opartą na: Universal Serial Bus (USB), Universal Asynchronous Receiver-Transmitter (UART), Inter-Integrated Circuit (I2C), Serial Peripheral Interface (SPI). Zarządzanie tożsamością urządzeń w tej klasie najczęściej nie jest możliwe.

Klasa 1

Urządzenia należące do klasy pierwszej nie mają możliwości korzystania ze standardowych protokołów wykorzystywanych w sieci Internet (nie mogą korzystać z Transport Layer Security (TLS), Hypertext Transfer Protocol (HTTP)). Muszą wykorzystywać lekkie protokoły takie jak Constrained Application Protocol (CoAP). Mogą też szyfrować dane oraz weryfikować poprawność przesłanych danych. Wymagają bramy do komunikacji w celu realizacji bardziej skomplikowanych zadań. Do tej klasy urządzeń należą mikrokontrolery posiadające około 10kB pamięci RAM oraz przestrzeń na dane około 100kB. Przykładem urządzenia IoT należącego do tej klasy jest *Arduino Uno* czy też *Adafruit Feather 32u4*. Do urządzeń należących do tej klasy można już implementować proste metody uwierzytelniania. Jednak ze względu na niewielki rozmiar pamięci na kod aplikacji, wykorzystywane schematy muszą być proste, a wykorzystany materiał kryptograficzny zazwyczaj jest zapisany podczas programowania urządzenia. Jeżeli urządzenia tej klasy oferują jakieś usługi, to proces autoryzacji opiera się na sztywno zdefiniowanych zasadach, które są ograniczone pod względem liczebności oraz

możliwości modyfikacji. Lepszym rozwiązaniem jest wykorzystanie zewnętrznych usług, które będą realizowały proces autoryzacji klientów.

Klasa 2

Do klasy drugiej należy zaliczyć urządzenia posiadające około 50kB pamięci RAM oraz 250kB pamięci na dane. Urządzenia tej klasy mogą korzystać z protokołów wykorzystywanych do komunikacji z dowolną usługą w sieci Internet. Można przy pomocy takiego urządzenia zbudować prosty serwer WWW. Do tej klasy należą m.in. *Arduino Mega 2560*, *WeMos D1*. Urządzenia te pod względem stosowanych metod uwierzytelniania nie różnią się od urządzeń zaliczanych do klasy pierwszej.

Urządzenia klasyfikowane powyżej klasy 2

Klasa ta jest bardzo obszerna, jej głównym założeniem jest wykorzystanie przez urządzenie IoT systemu operacyjnego, nawet w bardzo prostej formie. Przykładem urządzeń, które mogą mieć prosty system operacyjny, pomimo niedużej pamięci RAM i flash, są urządzenia: *ESP32*, *Raspberry Pi Pico*. Urządzenia należące do tej klasy są już na tyle wydajne, że mogą wykorzystywać dowolne metody uwierzytelniania i autoryzacji.

Parametry urządzeń IoT zaliczanych do klasy pierwszej, drugiej oraz urządzeń z prostym systemem operacyjnym przedstawiono w tabeli 2.5.

Do tej klasy należą też urządzenia, które posiadają rozbudowany system operacyjny z graficznym interfejsem i mogą przechowywać wiele gigabajtów danych. Do takich urządzeń IoT można zaliczyć: *Raspberry Pi Zero*, *Beaglebone Black*, których specyfikację przedstawiono w tabeli 2.6.

Każde z urządzeń IoT oferuje różnego rodzaju interfejsy do komunikacji. W przypadku urządzeń należących do klasy 0 zazwyczaj służą one do komunikacji między sensorem a bramą, natomiast w klasach wyższych można przy ich pomocy do urządzenia podłączyć różne moduły. W przypadku modułów bezpieczeństwa najpopularniejszym jest TPM - standard układu scalonego opracowany przez Trusted Computing Group. Poza tym modułem można jeszcze podłączyć moduł PUF, jeżeli nie został on wbudowany w urządzenie.

Tabela 2.5: Porównanie konfiguracji sprzętowej urządzeń należących do klas 1,2 oraz urządzeń IoT z prostym systemem operacyjnym

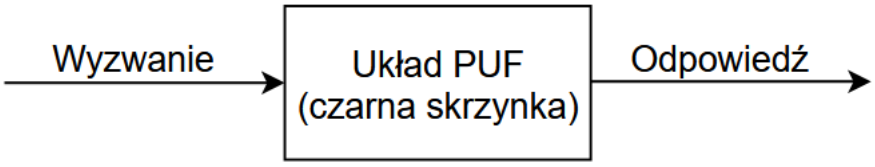
Parametr	**Arduino Uno**	**Adafruit Feather 32u4**	**Arduino Mega 2560**	**WeMos D1**	**ESP32**	**Raspberry Pi Pico**
Mikrokontroler	ATmega328	ATmega32u4	Atmega2560	ESP8266	Tensilica LX6	RP2040
Taktowanie	16 MHz	8 MHz	16MHz	80MHz	240MHz	133 MHz
Pamięć Flash	32 kB	32 kB	256 kB	4 MB	4MB	2 MB
Pamięć SRAM	2 kB	2 kB	8 kB	80 kB	520 kB	264 kB
Wyjścia PWM	6	7	15	10	16	16
Wejścia analogowe	6	10	16	1	18	3
Komunikacja szeregowa	USB, UART, I2C, SPI					

Tabela 2.6: Porównanie konfiguracji sprzętowej *Beaglebone Black* z *Raspberry Pi Zero*

Parametr	**Beaglebone Black**	**Raspberry Pi Zero**
Procesor	AM335x (1 rdzeń)	Broadcom BCM2835(1 rdzeń)
Taktowanie	1 GHz	1 GHz
Pamięć Flash	4GB eMMC + karta microSD	karta microSD
Pamięć RAM	512 MB	512 MB
Wyjścia wideo	HDMI	miniHDMI
Wejścia cyfrowe	65	16
Wejścia analogowe	7	Brak
Komunikacja bezprzewodowa	Brak	802.11 b/g/n; BLE 4.1
Komunikacja przewodowa	Ethernet 10/100 Mbps	brak
Komunikacja szeregowa	USB, UART, I2C, SPI	USB, UART, I2C, SPI

2.7 Wykorzystanie PUF do identyfikacji urządzeń IoT

Fizyczne nieklonowalne funkcje (ang. PUF) to układ fizyczny realizujący „niepodrabialną" funkcję typu wyzwanie-odpowiedź [72]. Odpowiedź wygenerowana przez układ na otrzymane wyzwanie musi być: niezmienna w czasie, unikatowa (aby żadne dwa układy PUF nie były takie same), łatwa do wykorzystania, trudna do replikacji (układ nie może być skopiowany), jednokierunkowa (na podstawie odpowiedzi nie można uzyskać wyzwania) i bardzo trudna lub niemożliwa do przewidzenia (aby odpowiedzi nie można było odgadnąć). Schemat działania układu PUF przedstawiono na rysunku 2.4.



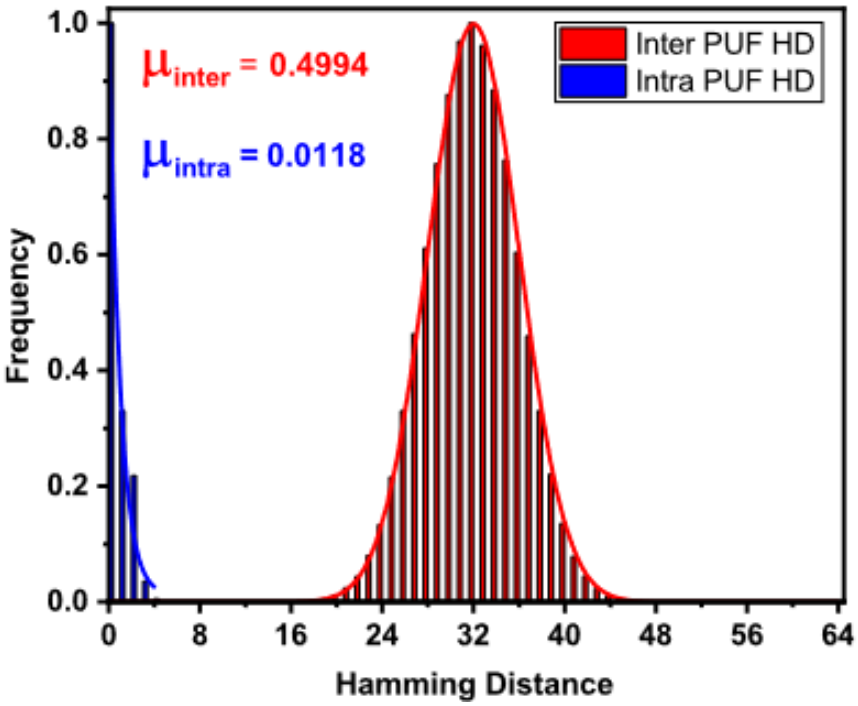
Rysunek 2.4: Schemat układu PUF

Aby uzyskać unikalność oraz trudność do przewidzenia, układ musi posiadać cechę losowości. Losowość w układzie można uzyskać na dwa sposoby. Pierwszym z nich jest ingerencja w proces powstawania układu i dodanie elementu losowego (proces randomizacji). W ten sposób powstają jawne układy PUF (ang. explicit). Drugim sposobem jest uzyskanie w procesie produkcji losowości występującej w sposób naturalny poprzez wykorzystanie odchyłek parametrów wytwarzanego układu scalonego. Układ z takim sposobem uzyskania losowości

nazywany jest niejawnym PUF (ang. implicit). Z praktycznego punktu widzenia niejawne PUF nie wiążą się z dodatkowymi kosztami, ponieważ losowość w układzie powstaje w sposób nieunikniony podczas tworzenia układu. Powstawanie unikalności w układzie nie jest pożądane przez producentów, ponieważ mogą mieć negatywny wpływ na wydajność oraz powtarzalność pracy, do której układ został zaprojektowany. Z tego też powodu producenci próbują uniknąć powstawania losowego efektu podczas produkcji układu, jednak nie jest to technicznie możliwe. W rezultacie niejawne PUF mają tę zaletę, że nawet producent układu, mający pełną kontrolę nad procesem produkcji układu, nie może usunąć ani kontrolować nieuniknionego efektu losowości — co stanowi podstawę funkcjonalności PUF. Z tego też powodu nie ma możliwości analizy układów PUF bez posiadania układu fizycznie. Każdy atak fizyczny, próbujący wyodrębnić informacje cyfrowe z układu scalonego, musi zostać przeprowadzony, gdy układ jest włączony. Ataki inwazyjne są trudniejsze do przeprowadzenia bez modyfikowania cech fizycznych, z których wywodzi się sekret. Dlatego do zabezpieczenia PUF nie są wymagane stale zasilane aktywne mechanizmy antysabotażowe. Można natomiast próbować wpłynąć na warunki środowiskowe, w których pracuje układ PUF. Przykładowo, jeśli ktoś mógłby zmodyfikować temperaturę środowiska, gdzie pracuje PUF, mógłby potencjalnie spowodować niepoprawne działanie tego układu.

Użyteczność PUF można określić za pomocą dwóch parametrów [125]. Pierwszym z nich jest odległość wewnętrzna (ang. intra-distance), czyli odległość Hamminga lub ułamkowa odległość Hamminga między dwiema różnymi odpowiedziami tego samego układu PUF na to samo wyzwanie. Odpowiedzi są zbierane przy różnych warunkach (m.in. temperaturze, napięciu). Natomiast drugim parametrem jest odległość pośrednia (ang. inter-distance), czyli odległość Hamminga lub ułamkowa odległość Hamminga między dwiema odpowiedziami dwóch różnych PUF na to samo wyzwanie. Charakterystyki obydwu parametrów są prezentowane w postaci histogramu pokazującego występowanie obu odległości. Przykład takiego histogramu został przedstawiony na rysunku 2.5.

W literaturze oba histogramy są przybliżane rozkładem Gausa i są podsumowywane poprzez podanie ich średnich [126]: μ_{intra}, μ_{inter} oraz odchyleń standardowych: $\sigma_{intra}, \sigma_{inter}$. Należy zauważyć, że μ_{intra} określa średni szum odpowiedzi, tj. mierzy średnią odtwarzalność odpowiedzi w odniesieniu do wcześniejszej obserwacji tej samej odpowiedzi (cecha niezmienności w czasie). Natomiast μ_{inter} wyraża unikalność układu. Jeżeli odpowiedzi układu są ciągami bitów, najlepszą rozróżnialnością, jaką można osiągnąć jest to, że połowa bitów różni się. W przypadku, gdy μ_{inter} jest wyrażony jako odległość Hamminga, wynik powinien być



Rysunek 2.5: Przykład histogramu z wynikami badania układu PUF. Źródło: [101]

jak najbliższy 0,5.

Układy PUF można sklasyfikować na podstawie ich bezpieczeństwa [125]. Silnym układem PUF nazywamy układ, zawierający dużo par wyzwanie-odpowiedź, który nawet udostępniony przeciwnikowi przez dłuższy czas umożliwia znalezienie innej pary wyzwanie-odpowiedź, która nie jest znana przeciwnikowi. Taki układ musi obsługiwać duży zestaw par wyzwanie-odpowiedź, ponieważ w przeciwnym razie przeciwnik może zapytać o wszystkie wyzwania i nie pozostaną żadne nieznane wyzwania oraz nie jest możliwe zbudowanie dokładnego modelu PUF na podstawie zaobserwowanych par wyzwanie-odpowiedź, lub innymi słowy, PUF jest nieprzewidywalny. Układ PUF, który nie spełnia tych wymagań, w szczególności ma mały zestaw par wyzwanie-odpowiedź nazywany jest słabym PUF. W skrajnym przypadku taki układ może mieć tylko jedną parę wyzwanie-odpowiedź. Ponieważ słabe PUF obsługują na ogół niewielką liczbę par wyzwanie-odpowiedź, pary te muszą być utrzymywane w tajemnicy. Jeżeli para zostanie ujawniona, wówczas każde inne urządzenie będzie w stanie emulować PUF. Wszystkie pary wyzwanie-odpowiedź są niezależne od siebie, dzięki temu, znając jedną parę wyzwanie-odpowiedź, atakujący nie może wygenerować innych bez posiadania dokładnie tego układu PUF. Utworzenie i obsługa silnego PUF jest bardzo trudna ze względu na to, że odpowiedzi uzyskane z silnych PUF są wrażliwe na warunki środowiskowe (np. tempe-

ratura, napięcie), co skutkuje dużym szumem odpowiedzi. Dodatkowo w przypadku silnych PUF występują ataki wykorzystujące uczenie maszynowe, które utrudniają zbudowanie silnego PUF [135].

Słabe PUF są wykorzystywane do identyfikacji urządzeń. Za wykorzystaniem słabych PUF do identyfikacji przemawia niezawodność odpowiedzi, jaką ten typ PUF generuje. Ponieważ zestaw par wyzwanie-odpowiedź może być niewystarczający, można wykorzystać większą liczbę układów PUF. Słabe PUF wykorzystywane są również w procesie generowania kluczy, ponieważ proces ten ma zerową tolerancję na błędy. Ponieważ generowanie klucza wykorzystuje fizyczną losowość wprowadzoną na etapie produkcji, nie ma potrzeby dodatkowej dystrybucji klucza w momencie rozpoczęcia działania urządzenia IoT. Co więcej, z uwagi na fakt, że losowość jest utrwalona w układzie PUF, nie jest wymagana konwencjonalna nieulotna pamięć do przechowywania klucza. Zapewnia to dodatkowe zabezpieczenie przed atakami, ponieważ klucz nie jest przechowywany w formacie cyfrowym, ale pojawia się w pamięci nieulotnej tylko wtedy, gdy jest to konieczne do działania. Dodatkowo klucze takie są odporne na manipulację. Natomiast silne PUF wykorzystywane są do uwierzytelniania ze względu na dopuszczalny próg błędu w tym procesie [81]. Opracowane zostały też metody generowania kluczy z wykorzystaniem silnych PUF, jednak wymagają one dodatkowych algorytmów korekcji błędów. Wybór metody korekcji błędów zależy od typu wykorzystanego układu PUF [81, 143]. Wykazano również, że wiele technik korekcji błędów powoduje wyciek bitów tworzonego klucza, np. z powodu konieczności przechowywana bitów parzystości [81].

W typowym scenariuszu wykorzystującym PUF, układ jest używany w dwóch fazach [126]. W pierwszej fazie, zwanej rejestracją, zbierane są odpowiedzi dla wyzwań, które będą wysyłane do układu PUF podczas fazy drugiej. Wszystkie pary wyzwanie-odpowiedź są przechowywane w zewnętrznej bazie. Jeżeli scenariusz wykorzystania zakłada, że na odpowiedziach uzyskanych z układu będą wykonywane jakieś operacje, to w bazie przechowywany jest wynik tych operacji. Przykładem takiej operacji może być kwantyzacja odpowiedzi, jeżeli układ PUF zwraca wynik w postaci analogowej. W drugiej fazie do układu PUF wysyłane są te same wyzwania, które były wykorzystane w fazie rejestracji. Układ generuje odpowiedź, która następnie jest poddawana różnym operacjom, a wynik jest porównywany z tym zapisanym w bazie danych lub też jest wykorzystywany w ten sam sposób jak wynik zapisany w bazie danych. Przykładem drugiego przypadku jest proces szyfrowania wiadomości, podczas którego do urządzenia wyposażonego w układ PUF wysyłana jest wiadomość zabezpieczona kluczem wygenerowanym na podstawie odpowiedzi zapisanych w bazie danych w procesie rejestracji,

oraz wyzwania, które są wymagane do uzyskania tych odpowiedzi. Układ PUF generuje odpowiedzi dla otrzymanych wyzwań, a następnie odpowiedzi są zamieniane na klucz. Klucz jest wykorzystywany do odszyfrowania wiadomości. Jeżeli wiadomość nie została zmodyfikowana, wykorzystano ten sam układ PUF oraz układ wygenerował poprawne odpowiedzi to operacja odszyfrowania zakończy się sukcesem.

Jeżeli układ PUF wykorzystywany jest do identyfikacji, to zalecane jest wykorzystanie każdej pary wyzwanie-odpowiedź tylko raz. Po wykorzystaniu każda para powinna zostać usunięta z bazy danych [126].

Obecnie istnieje bardzo wiele układów PUF. Można wyróżnić układy PUF, które są w pełni elektryczne lub też hybrydowe. Do układów hybrydowych zalicza się rodziny [128]: magnetyczną, optyczną i wykorzystującą częstotliwości radiowe.

W każdym z hybrydowych układów PUF można wyodrębnić kilka elementów, np. układ pomiarowy, układ posiadający unikalną właściwość, które są niezbędne, aby uzyskać działający układ PUF, co jest wadą. W układach w pełni elektronicznych wszystkie wymagane elementy znajdują się najczęściej w jednym układzie scalonym, przez co proces wdrożenia takiego układu jest łatwiejszy, a sam układ PUF zajmuje mniej miejsca. Taki układ PUF, w którym element pomiarowy jest zintegrowany z elementem posiadającym unikalną właściwość, nazywa się układem wewnętrznym PUF.

W przypadku urządzeń IoT wykorzystywane są układy PUF wykorzystujące elektroniczne cechy układu. Najczęściej są to układy wykorzystujące [128]:

1. Bezpośrednie charakterystyki — w tej rodzinie PUF mierzy się bezpośrednie charakterystyki układu takiej jak napięcie, natężenie, pojemność. Przykładem jest tutaj badanie napięcia progowego tranzystora. Napięcie progowe tranzystora odpowiada minimalnej różnicy napięcia między bramką a źródłem tranzystora, która jest potrzebna do umożliwienia przepływu prądu między źródłem a drenem. Wyzwaniem PUF jest w tym przypadku numer/pozycja tranzystora, a odpowiedzią jego wartość napięcia progowego.

2. Pamięć ulotną – rodzina układów PUF, która wykorzystuje pamięć ulotną, a dokładniej początkowy stan komórek w pamięci ulotnej. Układy PUF wykorzystują pamięć Static Random Access Memory (SRAM) lub Dynamic Random-Access Memory (DRAM). Dla pamięci SRAM istotny jest moment włączenia pamięci. Ze względu na różnice w procesie produkcyjnym każdy tranzystor ma nieco inne właściwości fizyczne, a to prowadzi do różnic we właściwościach elektrycznych. Podczas włączenia pamięci każda komórka

ma swój własny preferowany stan wynikający z napięć progowych tranzystorów. Stany komórek w pamięci SRAM dają unikalny, losowy ciąg zer i jedynek. Ponieważ jedna komórka zwraca zawsze 1 bit losowości, należy wybrać ciąg adresów komórek o określonej długości, aby uzyskać odpowiedź o odpowiedniej długości. Ciąg adresów komórek w tym przypadku jest wyzwaniem, a stan komórek odpowiedzią. Ważne jest, żeby nie wystąpiła żadna operacja zapisu w pamięci SRAM, aby komórka pamięci SRAM istniała w stanie metastabilnym. W tym stanie występujące sprzężenie zwrotne popychające komórkę w kierunku stanu "1"będzie równe sprzężeniu zwrotnemu popychającemu komórkę w kierunku stanu "0", utrzymując w ten sposób komórkę w tym stanie metastabilnym w nieskończoność [81]. Należy zauważyć, że stan końcowy zależy od różnicy między dwiema pętlami sprzężenia zwrotnego, więc pomiar jest różnicowy. W związku z tym szumy, na które wpływ ma temperatura środowiska, wahania napięcia zasilania nie powinny mieć silnego wpływu na przejście komórki w inny stan. Nie wszystkie komórki w pamięci SRAM są stabilne. Stosując powtarzane pomiary, można ocenić stabilność bitu wyjściowego SRAM PUF i selektywnie wykorzystać najbardziej stabilne bity jako wyjście PUF. W artykule [73] PUF SRAM został przetestowany na układach Field-Programmable Gate Array (FPGA). Okazuje się, że nie jest to możliwe wykorzystanie komórek SRAM układu FPGA, ponieważ w najpopularniejszych układach FPGA wszystkie komórki SRAM są twardo resetowane do zera bezpośrednio po włączeniu zasilania, a zatem cała losowość jest tracona [126]. Można temu zaradzić, budując PUF o strukturze motyla (ang. Butterffly).

3. Pamięć nieulotną – rodzina PUF wykorzystująca komórki pamięci, które mogą przechowywać dane wtedy, gdy nie są zasilane oraz mają możliwość resetowania komórki. Układy PUF z tej rodziny zazwyczaj działają poprzez zastosowanie zmodyfikowanego sygnału zapisu do komórki pamięci, który powoduje, że komórka pamięci ma 50% szans na zapisanie w stanie 0 lub 1, tworząc losowy wzór. Ze względu na nieulotność tego typu pamięci, wzorzec pozostaje zapisany do momentu celowego zresetowania. Przykładem tego typu układu PUF jest układ wykorzystujący memrystor. Memrystory to elementy elektroniczne, które przełączają się między stanem wysokiej i niskiej rezystancji po osiągnięciu napięcia progowego. Po przekroczeniu napięcia progowego przewodzenia memrystor przechodzi w stan niskiej rezystancji, a przy prądzie ujemnym przechodzi do stanu wysokiej rezystancji. W najbardziej prostym przypadku układu PUF impuls

jest przykładany do szeregu memrystorów o wartości napięcia progowego, tak że istnieje prawdopodobieństwo 50% wystąpienia przejścia pomiędzy stanami rezystancji. Efektem tego procesu jest to, że każdy memrystor ma stan wysokiej lub niskiej rezystancji w nieprzewidywalny, ale powtarzalny sposób w oparciu o zmiany powstałe na etapie produkcji. Na podstawie odczytu rezystancji pojedynczych memrystorów uzyskiwana jest odpowiedź. Wyzwaniem w tym przypadku są numery memrystorów w macierzy, których stan rezystancji będzie odczytywany.

Istotnym problemem, na jaki wskazują autorzy artykułów [126, 166], jest wyliczenie entropii dla układów PUF. Do testowania entropii wykorzystywane są testy Diehard [32] oraz NIST [152, 186]. Jednakże z powodu ograniczonej długości odpowiedzi generowanych przez PUF oraz ograniczonej liczby odpowiedzi, oba zestawy testów oferują jedynie niski poziom pewności co do ich wyniku. W szczególności w przypadku testów losowości, które w rzeczywistości mają na celu przetestowanie pozornej losowości na wyjściu generatorów liczb pseudolosowych, nie jest jasne, czy przejście tych testów ma jakiekolwiek znaczenie dla PUF. Dodatkowo testy te szacują jedynie niezależną entropię w ramach jednego układu PUF. Aby układ PUF uznać za bezpieczny, musi on być również nieprzewidywalny, biorąc pod uwagę odpowiedzi z innych układów PUF. A to podejście nie jest oceniane przez wykorzystanie tych zestawów testów. Dodatkowo autorzy pracy [166] wykazali, że wyniki estymacji entropii różnią się znacznie w zależności od kolejności odpowiedzi układów PUF oraz że występuje przeszacowanie entropii układu PUF przy użyciu testu NIST 800-90B. Oba te problemy według autorów wynikają z różnic w produkcji obwodów i tranzystorów.

2.8 Podsumowanie

W rozdziale tym scharakteryzowano współczesne urządzenia IoT oraz przedstawiono ich klasyfikację. Wynika z niej, że dopiero urządzenia IoT należące do klasy 1 mogą być uwierzytelniane bez pomocy innych urządzeń, ale wykorzystane metody uwierzytelniania muszą być dostosowane do małej pamięci RAM oraz przestrzeni na dane. W rozdziale opisano układy PUF, które mogą być wykorzystane do uwierzytelniania urządzeń IoT, zamiast wykorzystywania przechowywanych w pamięci urządzenia kluczy kryptograficznych.

Na podstawie analizy dostępnej literatury można stwierdzić, że brakuje prac, które zawierają zalecenia do budowy federacji wokół środowisk IoT, dlatego istotne wydaje się, że opracowanie takich zaleceń jest potrzebne w szybko rosnącym segmencie sieci IoT. Analizując

problemy występujące w środowiskach federacyjnych oraz zarządzaniu tożsamościami, m.in. problem ustalenia ogólnoobowiązujących zasad czy też oczekiwań, można dojść do wniosku, że wykorzystanie rejestrów rozproszonych może je rozwiązać. Trzeba jednak wziąć pod uwagę, że bardzo istotne jest zbudowanie zaufania do poszczególnych członków federacji, a tym samym do urządzeń do nich należących. Biorąc pod uwagę ograniczenia urządzeń IoT, wydaje się, że najlepszym sposobem będzie wykorzystanie kryptografii symetrycznej połączonej z lekkimi protokołami wymiany danych np. CoAP. Wykorzystanie układów PUF jest korzystne z takiego punktu widzenia, że urządzenie jest wyposażane w układ mogący generować klucze jednoznacznie potwierdzające tożsamość urządzenia. Powstaje pytanie, czy można skorzystać z już obecnych, zainstalowanych komponentów sprzętowych oraz zainstalowanego oprogramowania, aby móc wygenerować klucz mogący potwierdzić tożsamość urządzenia? W takiej sytuacji klucz nie będzie przechowywany w sposób jawny na urządzeniu, a generowany na potrzeby operacji wymagającej uwierzytelniania urządzenia.

Rozdział 3

Protokół uwierzytelniania, autoryzacji i bezpiecznej komunikacji urządzeń IoT w środowiskach federacyjnych

W niniejszym rozdziale przedstawiono opis opracowanego protokołu nazwanego Lightweight Authentication and Authorization Framework for Federated IoT (LAAFFI) [92]. Nazwa nawiązuje do najważniejszych cech tego rozwiązania, które uznano za wyróżniające względem innych protokołów. Na początku rozdziału omówiono ogólne założenia opracowanego protokołu. W dalszej części przedstawiono dokładny opis operacji wraz ze schematami komunikacji, które są przewidziane w ramach tego protokołu. Opisano także sposób autoryzacji urządzeń IoT w protokole LAAFFI. Na końcu rozdziału znajdują się uwagi implementacyjne dotyczące przygotowanego Proof of Concept (PoC) protokołu.

3.1 Założenia dotyczące konstrukcji protokołu LAAFFI

Protokół uwierzytelniania i autoryzacji urządzeń IoT w środowiskach federacyjnych LAAFFI wykorzystuje unikatowe cechy urządzenia IoT, które służą do utworzenia klucza umożliwiającego uwierzytelnianie urządzenia. Klucz jest wykorzystywany do zaszyfrowania/odszyfrowania przesyłanych danych. Proces uwierzytelniania opiera się na znajomości klucza, który został wykorzystany do zaszyfrowania/odszyfrowania przesyłanych danych. Jeżeli obie strony potrafią uzgodnić wspólny klucz kryptograficzny na podstawie znajomości cech dystynkcyjnych urządzeń IoT, to można przyjąć, że obie strony komunikacji są uwierzytelnione. Założenie takie nie ma sensu w momencie wykorzystania jednego klucza przez grupę urządzeń lub też w komunikacji multicast, ponieważ nie ma wtedy możliwości jednoznacznego wskazania urządzenia, które wysłało komunikat. Taki problem nie wystąpi w sytuacji, gdy tylko dwa urządzenia komunikują się ze sobą. Przy założeniu, że tylko te dwa urządzenia znają klucz, każde

z nich może mieć pewność co do tożsamości drugiego. W przypadku LAAFFI klucz może być utworzony przez urządzenie IoT oraz węzły rejestru rozproszonego. Ważne jest tutaj zaufanie do węzłów i odpowiednia implementacja mechanizmów bezpieczeństwa. Klucz ten może być tworzony na podstawie:

1. Unikatowych danych – w typ przypadku wykorzystywane są unikatowe dane o urządzeniu. Mogą to być dane sprzętowe takie jak: numer seryjny urządzenia, numer seryjny karty pamięci; programowe takie jak: identyfikatory partycji, identyfikatory systemu plików, klucze przechowywane na urządzeniu.

2. Odpowiedzi pochodzących z elektronicznych modułów PUF umieszczonych w urządzeniu IoT. W pamięci urządzenia muszą być przechowywane wyzwania, które posłużą do uzyskania odpowiedzi z układu PUF. Zaleca się wykorzystaniem PUF opartego o układy pamięci SRAM ponieważ układ pamięci jest łatwo wdrażalny, zajmuje niewiele miejsca oraz jest energooszczędny. Układ PUF oparty o pamięć SRAM oferuje mniejszą, ale stabilną liczbę par wyzwanie-odpowiedź. W celu zwiększenia liczby par można skorzystać z układu, który będzie miał więcej komórek pamięci lub też z kilku układów pamięci. Stabilność układu jest istotna, jeżeli ma on posłużyć do tworzenia kluczy.

3. Losowych ciągów znaków – w tym przypadku urządzenie generuje losowy ciąg znaków, który musi mieć wymaganą entropię. Entropia jest miarą losowości [186]. Ciąg znaków posiadający entropię nie oznacza tym samym, że jest losowy. Do określenia losowości muszą być przeprowadzone testy [152]. Ten przypadek powinien mieć zastosowanie tylko wtedy, gdy nie istnieje możliwość wykorzystania elementów urządzenia IoT określonych w punktach 1 i 2.

W przypadku 1 i 2 urządzenie nie przechowuje żadnych danych w pamięci trwałej poza programem odpowiedzialnym za realizację funkcji opisywanego protokołu. Program ten zawiera zbiór poleceń (dla unikatowych danych konfiguracyjnych) i wyzwań (dla układów PUF) stanowiących wymuszenia do uzyskania wartości. Odpowiedzialny on jest także za komunikację z innymi urządzeniami IoT należącymi do federacji, węzłami rejestru rozproszonego oraz innymi zewnętrznymi usługami dostępnymi dla urządzenia. W programie zapisana jest lista poleceń i wyzwań, które umożliwiają uzyskanie danych, z których będzie tworzony klucz. Wszystkie dane wymagane do utworzenia klucza muszą być zawsze dostępne dla urządzenia oraz są one niezmienne. Każde z poleceń i wyzwań umożliwia uzyskanie jednej wartości. Ta

wartość w dalszej części rozprawy będzie nazywana parametrem, a cały zbiór parametrów dla urządzenia - tablicą parametrów. Tablica parametrów nie może być upubliczniona, ponieważ umożliwiłoby to odtworzenie klucza i możliwość czytania wiadomości przesyłanych do tego urządzenia oraz podszywania się pod to urządzenie. Tablica parametrów jest unikatowym cyfrowym „odciskiem palca" (ang. fingerprint) urządzenia, ponieważ do jej utworzenia wykorzystane- są unikatowe parametry urządzenia IoT.

W przypadku losowych ciągów znaków (przypadek określony w pkt. 3) zamiast wykonywania poleceń generowane są losowe ciągi znaków, które są zapisane w pamięci urządzenia IoT. Losowe ciągi znaków są potem wykorzystywane jako parametry. Każdy z parametrów musi mieć odpowiednią entropię.

Przykładowa lista poleceń, które umożliwiają uzyskanie unikatowych parametrów dla Raspberry Pi z zainstalowanym systemem operacyjnym Raspbian została przedstawiona w tabeli 3.1.

Tabela 3.1: Przykładowa lista poleceń wraz z wyliczoną entropią

Lp	**Polecenie**	**Entropia w bitach**
1	/opt/vc/bin/vcgencmd otp_dump | grep "^ 29"| cut -c 4-	32
2	cat /proc/device-tree/serial-number	32
3	sudo udevadm info -a -n /dev/mmcblk0 | grep serial | awk -F '" '{print $2}'	29
4	sudo udevadm info -a -n /dev/mmcblk0 | grep cid | awk -F '" '{print $2}'	33
5	sudo blkid | grep PTUUID | awk 'print $2' | awk -F '" 'print $2'	19
6	sudo blkid | grep root | awk 'print $3' | awk -F '" 'print $2'	128
7	sudo blkid | grep boot | awk 'print $4' | awk -F '" 'print $2'	32
8	sudo blkid | grep SETTINGS | awk 'print $3' | awk -F '" 'print $2'	128
9	sudo blkid | grep RECOVERY | awk 'print $4' | awk -F '" 'print $2'	32
10	sudo cat /etc/shadow | grep pi | awk -F '$' 'print $3'	93
11	sudo cat /etc/shadow | grep pi | awk -F '$' 'print $4' | awk -F ':' 'print $1' | cut -c 44-	260
12	sudo cat /etc/shadow | grep pi | awk -F '$' 'print $4' | awk -F ':' 'print $1' | cut -c -44	260
13	sudo dumpe2fs /dev/mmcblk0p5 2> /dev/null | grep Hash | awk 'print $4'	129
14	sudo dumpe2fs /dev/mmcblk0p7 2> /dev/null | grep Hash | awk 'print $4'	129
15	sudo cat /etc/machine-id	129

Pierwsze cztery polecenia w tabeli 3.1 odnoszą się do danych sprzętowych pochodzących z urządzenia oraz karty SD. Dobór odpowiednich parametrów nie zawsze jest prosty. Przykładowo numer seryjny Raspberry Pi nie powinien być używany, jeżeli wykorzystywane są wbudowane interfejsy sieciowe do komunikacji, ponieważ ich adresy MAC oraz adres sprzętowy Bluetooth są tworzone na podstawie numeru seryjnego urządzenia. Powoduje to, że znając adres MAC można w prosty sposób uzyskać numer seryjny urządzenia. Pozostałe polecenia przedstawione w tabeli 3.1 umożliwiają uzyskanie programowo-konfiguracyjnych parametrów. Bazując na entropii parametrów z tabeli 3.1 można wywnioskować, że do uzyskania klucza o entropii 128 bitów należy użyć kilku parametrów, ponieważ nie w każdym przypadku jeden, lub dwa wystarczą. Zagadnienie dotyczące oceny poziomu entropii zostało przedstawione w podrozdziale 4.1 rozprawy. Klucz o entropii 128 bitów jest wystarczający do wykorzystania w operacji szyfrowania symetrycznego — co jest podstawą opisywanego protokołu. W przypadku klonowania karty SD wraz z zainstalowanym systemem trzeba wymusić wygenerowanie nowych parametrów programowych podczas pierwszego uruchamiania systemu.

Zgodnie z wymaganiem wsparcia dla urządzeń IoT (rozdział 1.1, pkt. 2) LAAFFI jest przystosowany do obsługi małowydajnych urządzeń IoT poprzez wykorzystanie kryptografii symetrycznej do uwierzytelniania oraz zabezpieczenia komunikacji oraz poprzez możliwość wykorzystania protokołów uznawanych za lekkie do komunikacji pomiędzy urządzeniem a innymi elementami środowiska IoT. Wykorzystanie kryptografii symetrycznej powinno zagwarantować bezpieczeństwo komunikacji, które również było jednym z postawionych wymagań (rozdział 1.1, pkt. 7). Kolejnym wymaganiem jest decentralizacja (rozdział 1.1, pkt. 4), która została osiągnięta poprzez wykorzystanie rejestru rozproszonego. Jeżeli została wykorzystana funkcja danych prywatnych dostępna w Hyperledger Fabric lub dane będą szyfrowane przez organizacje, możliwe jest także uzyskanie odseparowania danych należących do różnych organizacji, co również było wymaganiem postawionym przed opracowanym protokołem (rozdział 1.1, pkt. 3). Wykorzystanie funkcji danych prywatnych pozwoli także na przechowywanie danych w odseparowanym, bezpiecznym magazynie danych (rozdział 1.1, pkt. 5). Ponieważ do każdej operacji wymagane jest wykonanie autoryzacji przez węzeł rejestru rozproszonego, to możliwe jest osiągnięcie rozliczalności pomiędzy organizacjami (rozdział 1.1, pkt. 8). Wykorzystanie w rejestrze rozproszonym wielu węzłów daje możliwość osiągnięcia wysokiej dostępności usług operacji wykonywanych w ramach opracowanego protokołu (rozdział 1.1, pkt. 9), a także zwiększenie wydajności (rozdział 1.1, pkt. 6). Identyfikator urządzenia wykorzystany w protokole jest generowany w sposób losowy, przez co nie zawiera żadnych dodatkowych informacji

o urządzeniu. Wykorzystanie losowego identyfikatora nie wymaga dodatkowej pracy administratora/operatora systemu. Opracowane rozwiązanie jest możliwe do wdrożenia w środowisku lokalnym, ale także w chmurze z wykorzystaniem podejścia Infrastructure as Code (IaC), co umożliwia szybkie przygotowanie kompletnego środowiska (wymagany czas przygotowania rzędu kilkudziesięciu minut). Co spełnia wymaganie „niezwłocznej interoperacyjności" (rozdział 1.1, pkt. 1).

3.2 Architektura protokołu LAAFFI

Opracowany protokół wymaga do działania poza urządzeniem IoT jeszcze dwóch komponentów:

1. bramy aplikacyjnej,
2. rejestru rozproszonego.

Każda organizacja wchodząca w skład federacji powinna posiadać minimum dwie aktywne bramy aplikacyjne oraz minimum dwa węzły rejestru rozproszonego. Taka liczba bram aplikacyjnych oraz węzłów rejestru rozproszonego ma na celu minimalizację sytuacji, gdy organizacja nie będzie mogła świadczyć usług związanych z protokołem LAAFFI. Jeżeli urządzenie będzie chciało wykonać jakąś operację, to zawsze na początku komunikuje się z bramą aplikacyjną, która pośredniczy w komunikacji z węzłami rejestru rozproszonego.

Wykorzystanie bramy aplikacyjnej jest wymagane z kilku powodów:

1. Urządzenie nie musi mieć informacji o całej architekturze rejestru rozproszonego. Węzły rejestru rozproszonego mogą być dodawane, usuwane, wyłączane w dowolnym momencie i nie powinno to zaburzyć pracy całego systemu. Zadaniem bramy aplikacyjnej jest wysłanie żądań do aktywnych węzłów rejestru rozproszonego.
2. Ponieważ urządzenie nie ma informacji o aktualnie dostępnych węzłach rejestru rozproszonego, to brama aplikacyjna może służyć jako punkt kontaktu.
3. Brama aplikacyjna pełni rolę elementu równoważenia obciążenia pomiędzy węzłami rejestru rozproszonego.
4. W rejestrze rozproszonym węzły mogą pełnić określone role np. niektóre węzły tylko przechowują dane i nie mogą brać udziału w procesie weryfikacji i dodawania transakcji.

Brama aplikacyjna powinna kierować żądania do odpowiednich węzłów, które mogą zrealizować operację.

5. Z reguły różne usługi oraz implementacje rejestru rozproszonego wykorzystują do komunikacji TLS wraz z PKI. Ze względu na ograniczenia urządzeń IoT powinno się wykorzystywać lekkie protokoły, dlatego brama aplikacyjna może dokonywać konwersji pomiędzy protokołem używanym do komunikacji z urządzeniem IoT a protokołem wykorzystywanym do komunikacji z węzłami rejestru rozproszonego lub innych usług.

6. Podczas komunikacji z urządzeniami IoT może dojść do modyfikacji przesyłanej wiadomości. Brama aplikacyjna może wykrywać niepoprawne wiadomości i je odrzucać. Dzięki temu węzły rejestru rozproszonego nie będą musiały obsłużyć błędnych wiadomości. Takie podejście pozwoli na odciążenie węzłów rejestru rozproszonego, a tym samym na zwiększenie wydajności całej sieci rejestru rozproszonego.

7. Brama aplikacyjna dodatkowo może służyć jako firewall dla wiadomości wysyłanych przez urządzenia IoT i odrzucać wiadomości np. od urządzeń należących do innych organizacji, jeżeli organizacja chce, aby ich bramy aplikacyjne obsługiwały tylko urządzenia z tej samej organizacji.

Operacje uwierzytelniania i autoryzacji przebiegają w węźle rejestru rozproszonego. Aby węzeł mógł uwierzytelnić urządzenie, musi mieć możliwość odtworzenia klucza, który został wykorzystany przez urządzenie do zabezpieczenia przesyłanych danych. W tym celu w rejestrze rozproszonym muszą być przechowywane wszystkie parametry urządzeń IoT. Proces wysłania tablicy parametrów do rejestru następuje w fazie rejestracji urządzenia.

Rejestr rozproszony poza tablicą parametrów przechowuje także:

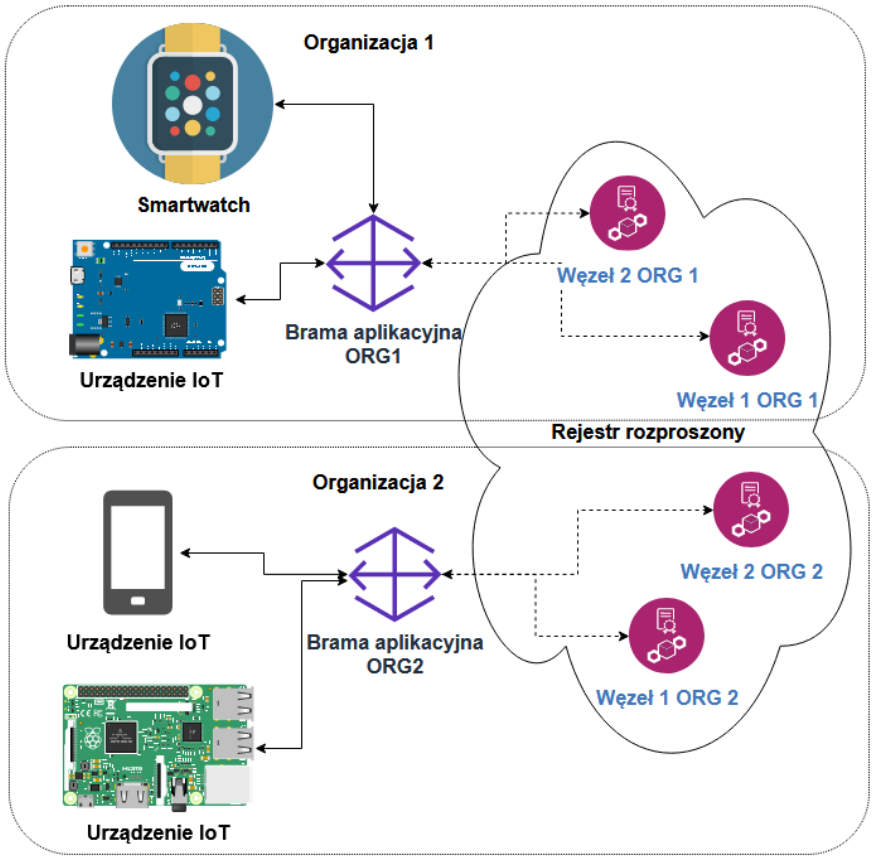
1. Jednorazowe dane uwierzytelniające, które mogą być wykorzystywane przez urządzenie IoT w procesie rejestracji.

2. Reguły autoryzacji, na podstawie których podejmowana jest decyzja, czy urządzenie ma prawo wykonać żądaną operację.

3. Dodatkowe informacje o urządzeniu m.in. kontakt do właściciela urządzenia, typ urządzenia itd.

Ogólny schemat architektury został przedstawiony na rysunku 3.1. Taki ogólny schemat zakłada istnienie kilku organizacji (na rysunku pokazano tylko dwie). Każda organizacja po-

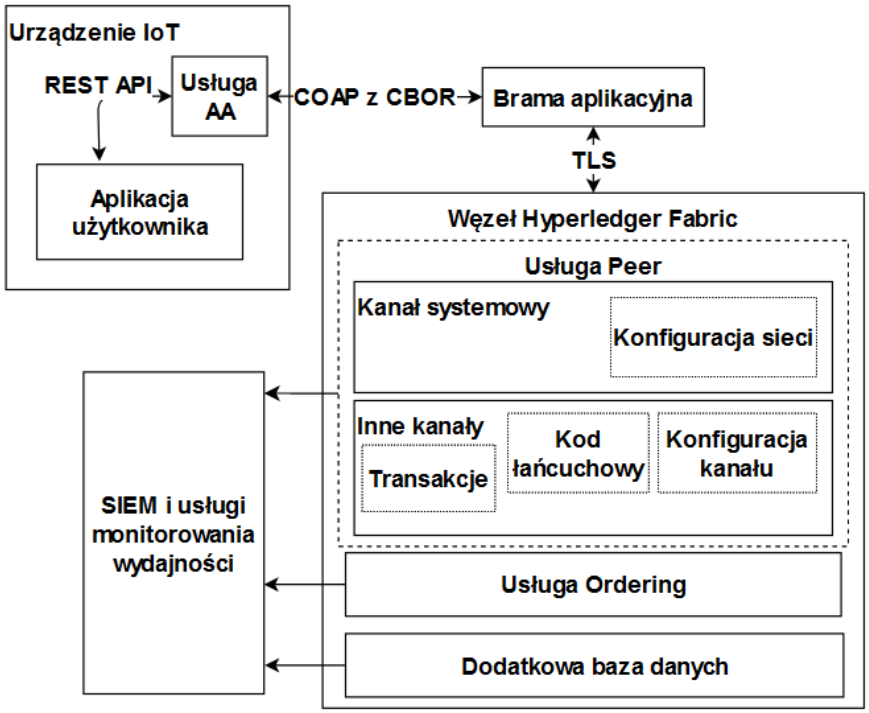
siada urządzenia IoT, które do niej należą. Urządzenia IoT korzystają z bramy aplikacyjnej do komunikacji z węzłami rejestru rozproszonego. Na rysunku 3.1 pokazano, że każda organizacja składa się z dwóch węzłów oraz jednej bramy aplikacyjnej, ale może ich być więcej.



Rysunek 3.1: Przykład architektury federacyjnej sieci IoT składającej się z dwóch organizacji

Na rysunku 3.2 przestawiono szczegółowy schemat komponentów wymaganych i zalecanych we wdrożeniu LAAFFI. Zgodnie z rysunkiem 3.2 urządzenie IoT składa się z dwóch elementów. Pierwszym z nich jest aplikacja użytkownika. Aplikacja ta jest uruchomiona na urządzeniu IoT i wykonuje zadanie zlecone przez użytkownika. Język oraz wykorzystane technologie w tej aplikacji muszą być przystosowane do pracy na konkretnym urządzeniu IoT. Aplikacja komunikuje się z usługą Authentication and Authorization (AA). Komunikacja pomiędzy aplikacją użytkownika a usługą AA odbywa się z wykorzystaniem mechanizmu, który jest odpowiedni dla danego urządzenia. Wdrożenie usługi AA jako oddzielnej aplikacji upraszcza tworzenie aplikacji użytkownika, ponieważ tworzone aplikacje nie muszą implementować opisanego protokołu, a jedynie mechanizm komunikacji z usługą AA.

Komunikacja pomiędzy urządzeniem IoT a bramą aplikacyjną wykorzystuje protokół CoAP



Rysunek 3.2: Szczegółowy schemat komponentów składowych protokołu LAAFFI

[163] oraz sposób zapisu danych Concise Binary Object Representation (CBOR) [29]. CoAP jest protokołem zaprojektowanym z myślą o wykorzystaniu w urządzeniach z ograniczonymi zasobami. Umożliwia komunikację Machine-to-Machine (M2M) bez konieczności wykorzystania scentralizowanego urządzenia pełniącego rolę brokera. Zapewnia to możliwość łączenia się czujników i innych węzłów IoT w celu publikacji danych. Wszystkie zasoby udostępnione przez serwer są identyfikowane za pomocą adresów URL, a klient ma do nich dostęp wykorzystując metody podobne do wykorzystywanych w protokole HTTP: DELETE, GET, POST, PUT. Protokół CoAP korzystać z User Datagram Protocol (UDP) lub Transmission Control Protocol (TCP) oraz obsługuje transmisje multisesyjną, co sprawia, że wykrywanie zasobów jest bardziej wydajne. Format danych CBOR został zaprojektowany z myślą o uzyskaniu jak najmniejszego rozmiaru wiadomości poprzez wykorzystanie kodu programu o minimalnej zajętości pamięci, ograniczonej liczbie instrukcji procesora oraz zużycia prądu. Podobnie jak JavaScript Object Notation (JSON) umożliwia on przesyłanie obiektów danych w formacie nazwa-wartość, ale w sposób zwięzły.

Brama aplikacyjna łączy się z węzłami rejestru rozproszonego wykorzystując TLS. Protokół ten jest wymagany przez wykorzystaną implementację rejestru rozproszonego - Hyperledger Fabric. Aby zapewnić uwierzytelnianie obu stron oraz autoryzację bramy aplikacyjnej

wykorzystywana jest infrastruktura klucza publicznego, która w Hyperledger Fabric nazywana jest Membership Service Provider (MSP).

Każdy węzeł rejestru rozproszonego w ramach usługi Peer może być częścią wielu kanałów (może obsługiwać wiele rejestrów rozproszonych). W przypadku przygotowanej implementacji wydzielono 4 kanały:

1. Kanał rejestracji — kanał przechowuje dane potrzebne w procesie rejestracji urządzenia IoT, a także zawiera zainstalowany kod łańcuchowy umożliwiający obsługę tych danych (dodanie nowych poświadczeń tymczasowych, usunięcie, weryfikację istnienia poświadczeń tymczasowych oraz odszyfrowanie wiadomości odebranej od urządzenia IoT w procesie rejestracji.

2. Kanał uwierzytelniania — kanał przechowuje tablicę parametrów wraz z identyfikatorem urządzenia oraz zmienną określającą liczbę parametrów wymaganych do utworzenia bezpiecznego klucza. Kod łańcuchowy zainstalowany w tym kanale umożliwia dodanie urządzenia, aktualizację zapisanych danych o urządzeniu (np. w przypadku ponownej rejestracji), wygenerowanie klucza do komunikacji pomiędzy urządzeniami, zaszyfrowanie oraz odszyfrowanie wiadomości.

3. Kanał autoryzacji — kanał przechowuje uprawnienia do każdego urządzenia oraz ma zainstalowany kod łańcuchowy odpowiedzialny za dodawanie, usuwanie oraz weryfikację uprawnień.

4. Kanał informacji o urządzeniu — kanał przechowuje informacje o urządzeniu oraz ma zainstalowany kod łańcuchowy odpowiedzialny za dodawanie, modyfikację oraz usunięcie informacji.

Dzięki takiemu podejściu można tworzyć oddzielne zasady dostępu do poszczególnych kanałów, a tym samym do danych w nich przechowywanych. Istotnym elementem całej infrastruktury jest system monitoringu. W ramach tego systemu zaleca się wdrożenie systemu Security Information and Event Management (SIEM) do przechowywania, korelowania oraz analizy zdarzeń z węzłów rejestru rozproszonego. Wykorzystanie tego systemu umożliwi wykrywanie naruszeń bezpieczeństwa w ramach każdej organizacji. Dodatkowo każdy węzeł powinien przesyłać swoje dane o wydajności, nie tylko samych usług, ale także pamięci, procesora, dysku oraz sieci. Umożliwi to wczesne wykrywanie problemów wydajnościowych oraz przyspieszy reakcję administratorów, aby nie doszło do wyłączenia węzła rejestru rozproszonego.

3.3 Protokół uwierzytelniania urządzeń

Protokół uwierzytelniania urządzeń IoT LAAFFI umożliwia wykonanie podstawowych operacji takich jak: rejestracja urządzenia, wykonanie żądanej operacji na rejestrze rozproszonym lub innej usłudze oraz bezpiecznej komunikacji pomiędzy urządzeniami IoT. W każdej operacji istotną rolę odgrywa zabezpieczenie komunikacji pomiędzy stronami przy jak najmniejszym nadmiarze przesyłanych danych. W wyniku operacji ustanowienia bezpiecznej komunikacji pomiędzy urządzeniem IoT oraz węzłem rejestru rozproszonego, a także pomiędzy urządzeniami IoT można wykonywać inne operacje, które są przewidziane w ramach działań zaplanowanych dla urządzeń IoT. Uwierzytelnianie w opracowanym protokole bazuje na zdolności do odszyfrowania przesłanego szyfrogramu przy zaufaniu, że klucz wykorzystany do zaszyfrowania tej wiadomości jest w posiadaniu tylko obu stron komunikacji. Zasadniczą rolę w procesie uwierzytelniania pełni rejestr rozproszony, który przechowuje tablicę parametrów, na podstawie której generowane są klucze.

Wszystkie operacje wykonywane na rejestrze rozproszonym są realizowane za pomocą kodów łańcuchowych. Oznacza to, że wszystkie organizacje wchodzące w skład federacji budującej rejestr rozproszony mają identyczne kody łańcuchowe. W związku z tym każda operacja zlecona przez dowolnego użytkownika bądź urządzenie IoT musi spełniać identyczne wymagania zapisane w kodzie łańcuchowym.

3.3.1 Rejestracja urządzenia

Zgodnie z przyjętymi założeniami przedstawionymi w podrozdziale 3.1 rozprawy, ustalony dla danego urządzenia IoT A zbiór parametrów, reprezentujący jego unikatową tożsamość, będzie reprezentowany w formie tablicy TP_A, która zawiera wartości parametrów $\{p_1, p_2, ..., p_n\}$. Bezpośrednim celem procesu rejestracji jest zapisanie tablicy parametrów TP_A w wiarygodnym i niezmiennym magazynie danych, jakim jest rejestr rozproszony RR. Przyjmijmy, że federacja F składa się z organizacji $\{O_1, O_2, .., O_i, .., O_l\}$, gdzie i to numer organizacji, a l liczba wszystkich organizacji. Proces rejestracji urządzeń IoT, należących do danej organizacji przeprowadzany jest w każdej z organizacji oddzielnie z wykorzystaniem bramy aplikacyjnej danej organizacji BA_i, co może być realizowane na dwa sposoby:

1. Urządzenie IoT jest podłączone bezpośrednio do bramy aplikacyjnej BA_i. Oba urządzenia znajdują się w bezpiecznym środowisku. W ten sposób tablica parametrów TP_A nie

musi być zabezpieczana w momencie przesyłania do bramy aplikacyjnej BA_i, ponieważ nikt nie jest w stanie jej podsłuchać.

2. Urządzenie IoT posiada zapisane jednorazowe dane do zabezpieczenia przesyłanej tablicy parametrów TP_A. Jednorazowe dane to identyfikator $Temp_{ID}$ i hasło/klucz $Temp_{Key}$. Odpowiedź od bramy aplikacyjnej BA_i jest już zabezpieczona w sposób opisany w tym podrozdziale.

Zgodnie z przyjętymi założeniami oraz ustaloną architekturą protokołu (podrozdział 3.2) w urządzeniu IoT (oznaczmy go jako urządzenie A) znajduje się program, który będzie uruchomiony jako usługa AA (rysunek 3.2). Program ten realizuje proces rejestracji w sposób automatyczny po pierwszym uruchomieniu urządzenia IoT pod warunkiem utworzenia bezpiecznego kanału komunikacji. Usługa AA po uruchomieniu generuje tablicę parametrów TP_A. Jeżeli parametry te są unikatowymi danymi konfiguracyjnymi (sekcja 3.1 punkt 1) lub PUF (sekcja 3.1 punkt 2), to program posiada odpowiednie polecenia, aby te parametry otrzymać. Parametry nie są zapisywane w sposób jawny w rejestrze rozproszonym, a w postaci wyniku funkcji skrótu: $TP = \{G(p_1), G(p_2), ..., G(p_n)\}$, gdzie $G(p_i) = HMAC(U, p_i)$. Klucz U może być jednym z parametrów programowych lub losową wartością zapisaną na urządzeniu. Nie zdecydowano się wykorzystać Key Derivation Function (KDF) [96] ze względu na słabą wydajność KDF na urządzeniach IoT. Wykorzystanie Hash-based Message Authentication Code (HMAC) [46] z kluczem U umożliwia wygenerowanie nowej tablicy parametrów TP dla urządzenia IoT, jeżeli poprzednia tablica parametrów zostałaby upubliczniona. Jeżeli tablica parametrów TP zostanie ujawniona, to urządzenie musi zarejestrować się jeszcze raz, wykorzystując inny klucz U.

Tablica parametrów TP_A urządzenia A jest przesyłana do bramy aplikacyjnej BA_i na jeden z dwóch sposobów opisanych na początku sekcji 3.3.1. Jeżeli w procesie rejestracji są wykorzystywane dane jednorazowe. to brama aplikacyjna BA_i po otrzymaniu wiadomości zawierającej tablicę parametrów TP_A musi odszyfrować te dane, co nie jest konieczne, jeżeli urządzenie jest podłączone bezpośrednio to bramy aplikacyjnej BA_i. W związku z koniecznością odszyfrowania wiadomości jest ona przesyłana do węzła rejestru rozproszonego RR_i, który ją odszyfrowuje przy użyciu klucza jednorazowego $Temp_{Key}$ zapisanego w rejestrze rozproszonym RR. Jeżeli operacja się powiedzie, to tablica parametrów TP_A jest zwracana do bramy aplikacyjnej BA_i, która generuje identyfikator ID_A oraz wysyła żądanie zapisania go wraz z tablicą parametrów TP_A w rejestrze rozproszonym RR do węzła RR_i.

Po zapisaniu identyfikatora urządzenia ID_A i tablicy parametrów TP_A w rejestrze rozproszonym RR, brama aplikacyjna BA_i musi wysłać identyfikator ID_A do urządzenia A. W tym celu wysyła żądanie przygotowania wiadomości M do węzła RR_i rejestru rozproszonego. Wiadomość M będzie zawierała zaszyfrowany identyfikator ID_A. Węzeł rejestru rozproszonego RR_i generuje k losowych numerów ze zbioru $1, 2, ..., n$, gdzie n jest liczbą elementów w tablicy TP_A. Numery te tworzą listę indeksów $LI = \{n_1, n_2, ..., n_k\}$ parametrów w tablicy parametrów TP_A, z których będzie tworzony klucz K do zaszyfrowania wiadomości M. Wartości parametrów z tablicy parametrów TP_A o indeksach odpowiadających wartościom listy indeksów LI są ze sobą konkatenowane. Następnie jest obliczana funkcja skrótu $H()$ dla wyniku konkatenacji, wynik ten jest kluczem służącym do zaszyfrowania danych: $K = H(G(p_{n_1})|G(p_{n_2})|....|G(p_{n_k}))$. Aby zmniejszyć prawdopodobieństwo otrzymania dwóch identycznych szyfrogramów, które powstały jako wynik szyfrowania danych przy użyciu tego samego klucza, należy w procesie szyfrowania wykorzystać również wartość *nonce*. W opisywanym rozwiązaniu wartość nonce jest generowana losowo oraz przesyłana w sposób jawny.

LAAFFI wykorzystuje algorytm uwierzytelnionego szyfrowania z powiązanymi danymi (Authenticated Encryption with Associated Data (AEAD)) [187] zamiast algorytmu szyfrującego, ponieważ algorytm szyfrujący nie zapewnia integralności danych. W dalszej części pracy poprzez szyfrowanie należy rozumieć wykorzystanie algorytmu szyfrującego z powiązanymi danymi. Szyfrowaniu z wykorzystaniem klucza K oraz wartości *nonce*, poza identyfikatorem ID_A, który musimy przesłać do urządzenia A podlega także znacznik czasu t. Znacznik czasu t pozwala na odrzucenie wiadomości przeterminowanych otrzymywanych przez odbiorcę. Zmniejsza to istotnie ryzyko ataków powtórzeniowych. W wyniku operacji szyfrowania uzyskiwany jest szyfrogram $E_A = E[K; nonce; (ID_A, t)]$, gdzie $E[]$ jest funkcją szyfrującą identyfikator urządzenia i znacznik czasu: (ID_A, t) wykorzystującą klucz K i wartość *nonce*. Do bramy aplikacyjnej BA_i przesyłana jest wiadomość:

- $M =< null, nonce, LI, E_A >$, jeżeli urządzenie było bezpośrednio podłączone do bramy aplikacyjnej BA_i,
- $M =< Temp_{ID}, nonce, LI, E_A >$, jeżeli urządzenie nie było bezpośrednio podłączone do bramy aplikacyjnej BA_i.

Brama aplikacyjna BA_i przesyła wiadomość M do urządzenia A. Urządzenie A po odebraniu wiadomości M odczytuje z niego listę indeksów parametrów LI. Na podstawie wartości indeksów z listy LI wybierane są odpowiadające tym indeksom polecenia do uzyskania parametrów:

$p_{n_1}, p_{n_2}, ..., p_{n_k}$. Następnie w urządzeniu tworzony jest klucz $K = H(G(p_{n_1})|G(p_{n_2})|....|G(p_{n_k}))$, który będzie służył do odszyfrowania szyfrogramu E_A. Z użyciem klucza K i wartości *nonce* dokonywane jest odszyfrowanie szyfrogramu E_A. W wyniku tej operacji urządzenie A uzyskuje identyfikator urządzenia ID_A i znacznik czasu t. Wartość znacznika czasu t jest porównywana z czasem aktualnym. Jeżeli różnica pomiędzy aktualnym znacznikiem czasu i znacznikiem odszyfrowanym t jest zbyt duża (5 sekund dla przygotowanej implementacji), to wiadomość M jest uważana za nieważną. W przeciwnym przypadku urządzenie A zapisuje identyfikator ID_A i uznawane jest za zarejestrowane. Diagram komunikacji, która jest widoczna dla osób postronnych, przedstawiono na rysunkach 3.3 i 3.4. Na diagramach zaznaczono, że urządzenie A komunikuje się z BA_i/RR ponieważ wiadomość jest przesyłana do bramy aplikacyjnej BA_i, która pośredniczy w komunikacji z węzłem rejestru rozproszonego RR.

$$A \xrightarrow{TP_A} BA_i/RR$$
$$A \xleftarrow{null, LI, nonce, E_A} BA_i/RR$$

Rysunek 3.3: Diagram komunikacji pomiędzy urządzeniem A oraz bramą aplikacyjną BA_i w procesie rejestracji — podłączenie urządzenia bezpośrednio do bramy aplikacyjnej

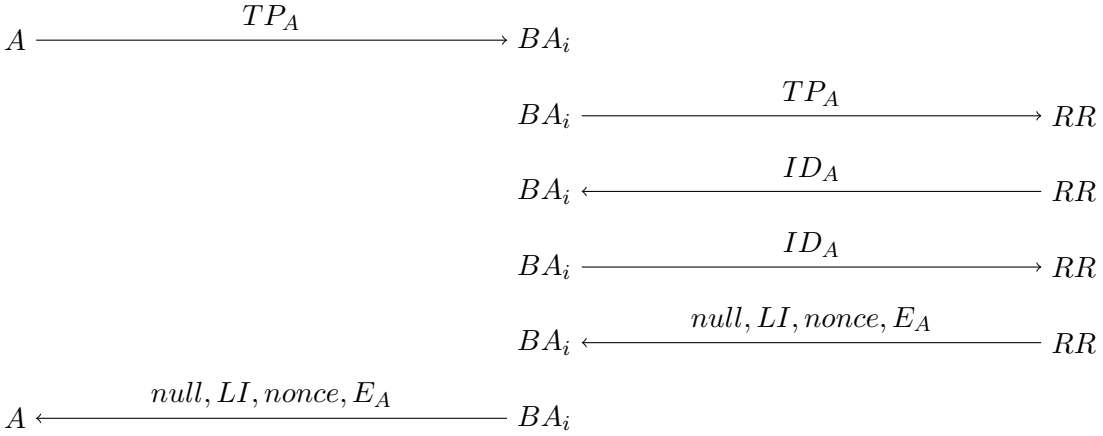
$$A \xrightarrow{Temp_{ID}, null, nonce, E_A} BA_i/RR$$
$$A \xleftarrow{Temp_{ID}, LI, nonce, E_A} BA_i/RR$$

Rysunek 3.4: Diagram komunikacji pomiędzy urządzeniem A oraz bramą aplikacyjną BA_i w procesie rejestracji — wykorzystanie jednorazowego identyfikatora i klucza

Na diagramach 3.5, 3.6 przedstawiono cały proces komunikacji z uwzględnieniem wiadomości przesyłanych między BA_i i RR. Komunikacja pomiędzy BA_i i RR jest zabezpieczona z wykorzystaniem TLS.

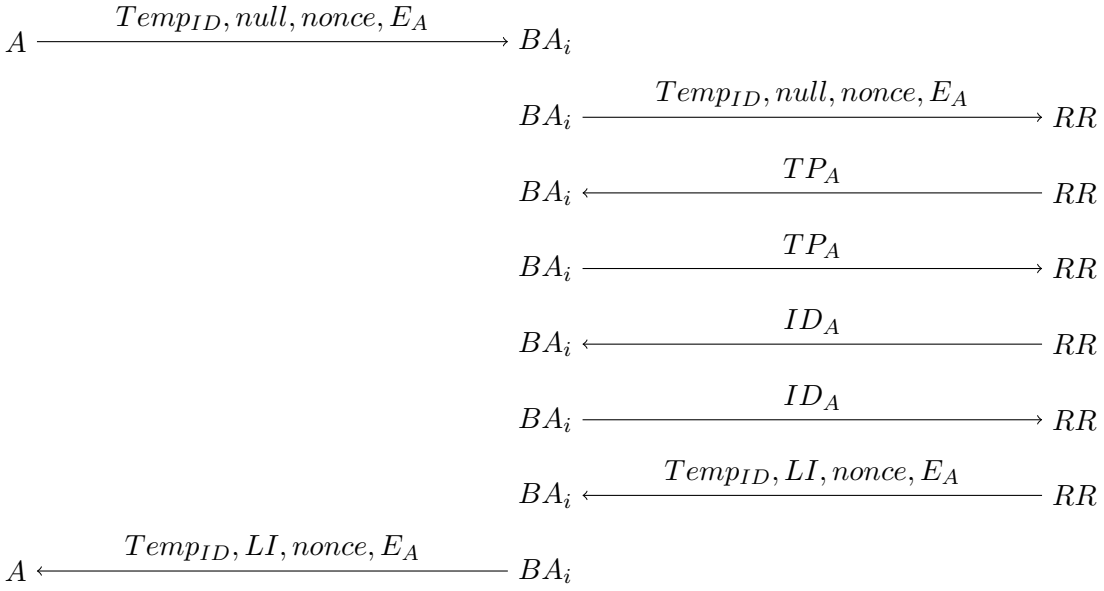
3.3.2 Wykonanie żądanej operacji na rejestrze rozproszonym lub innej usłudze przez urządzenie IoT

W sytuacji, gdy urządzenie A musi wykonać jakąś operację na rejestrze rozproszonym lub innej usłudze musi wysłać żądanie wykonania operacji do bramy aplikacyjnej BA_i. Operacją taką może być odczyt danych z innego kanału lub uzyskanie danych z usługi, która nie ob-



Rysunek 3.5: Diagram pełnej komunikacji pomiędzy urządzeniem A oraz bramą aplikacyjną BA_i i węzłem rejestru rozproszonego RR w procesie rejestracji — podłączenie urządzenia bezpośrednio do bramy aplikacyjnej

sługuje protokołu LAAFFI. Dane potrzebne do wykonania operacji są oznaczone jako *dane*. W pierwszym kroku urządzenie generuje k-elementową listę indeksów ze zbioru $\{1, 2, ..., n\}$: $LI_{A \to RR} = \{n_1, n_2, ..., n_k\}$. Następnie urządzenie A wykonuje polecenia, które umożliwiają uzyskanie parametrów o indeksach $LI_{A \to RR}$ w tablicy parametrów TP_A. Poszczególne parametry, tak jak w przypadku rejestracji urządzenia, są wynikiem funkcji HMAC dla wartości uzyskiwanych w rezultacie zastosowania kolejnych poleceń: $G(p_i) = HMAC(U, p_i)$. Urządzenie dokonuje konkatenacji parametrów, a wartość funkcji skrótu z konkatenacji jest kluczem wykorzystywanym do zabezpieczenia danych *dane*: $K_{A \to RR} = H(G(p_{n_1})|G(p_{n_2})|....|G(p_{n_k}))$. Urządzenie generuje losową wartość $nonce_{A \to RR}$ oraz aktualny znacznik czasu $t_{A \to RR}$. Dane do operacji *dane* oraz znacznik czasu $t_{A \to RR}$ są szyfrowane z użyciem klucza $K_{A \to RR}$ i wartości $nonce_{A \to RR}$ tj. $E_{A \to RR} = E[K_{A \to RR}; nonce_{A \to RR}; (dane, t_{A \to RR})]$. Urządzenie A wysyła wiadomość $M_{A \to RR}$ składającą się z identyfikatora ID_A, listy indeksów $LI_{A \to RR} = \{n_1, n_2, ..., n_k\}$, wartości $nonce_{A \to RR}$, szyfrogramu $E_{A \to RR}$ do dowolnej bramy aplikacyjnej BA_i – wysłane dane pokazano na rysunku 3.7. Brama aplikacyjna BA_i po odebraniu wiadomości $M_{A \to RR}$ przesyła ją do węzła rejestru rozproszonego RR_i w celu odszyfrowania. Węzeł RR_i na podstawie identyfikatora urządzenia ID_A odczytuje tablicę parametrów TP_A zapisaną w procesie rejestracji. Z tablicy parametrów TP_A wybiera parametry wskazywane przez listę indeksów $LI_{A \to RR} = \{n_1, n_2, ..., n_k\}$ i na ich podstawie tworzy klucz $K_{A \to RR} = H(G(p_{n_1})|G(p_{n_2})|....|G(p_{n_k}))$. Następnie na podstawie klucza $K_{A \to RR}$ i wartości $nonce_{A \to RR}$ odszyfrowuje szyfrogram $E_{A \to RR}$. Wynikiem są dane do operacji *dane* oraz



Rysunek 3.6: Diagram pełnej komunikacji pomiędzy urządzeniem A oraz bramą aplikacyjną BA_i i węzłem rejestru rozproszonego RR w procesie rejestracji — wykorzystanie jednorazowego identyfikatora i klucza

znacznik czasu $t_{A \to RR}$. Jeżeli różnica między aktualnym czasem a przesłanym znacznikiem $t_{A \to RR}$ jest akceptowalna oraz większa od poprzednio przesłanej to dane operacyjne *dane* są zwracane do bramy aplikacyjnej BA_i. W przeciwnym przypadku zwracany jest komunikat o błędzie. Brama aplikacyjna BA_i po odebraniu danych *dane* od węzła rejestru rozproszonego RR_i wykonuje operację zapisaną w *dane*. Żądanie do wykonania operacji może wymagać wykonania innej operacji z użyciem innej usługi. Rolą BA_i jest ustanowienie bezpiecznego połączanie z usługą i wykonanie żądania zleconego przez urządzenie IoT. Wymagane jest również sprawdzenie, czy urządzenie IoT może taką operację wykonać. Autoryzacja przebiega z wykorzystaniem rejestru rozproszonego, co zostało opisane w 3.4. Po wykonaniu operacji brama aplikacyjna BA_i wysyła odpowiedź *response* do węzła rejestru rozproszonego RR_i w celu jej zaszyfrowania. Węzeł RR_i generuje k-elementową listę indeksów ze zbioru $\{1, 2, ..., n\}$: $LI_{RR \to A} = \{m_1, m_2, ..., m_k\}$. Z tablicy parametrów TP_A urządzenia A zapisanej w rejestrze, kod łańcuchowy wybiera w sposób losowy parametry o indeksach $LI_{RR \to A}$, a na ich podstawie tworzy klucz $K_{RR \to A} = H(G(p_{m_1})|G(p_{m_2})|....|G(p_{m_k}))$. Generowana jest także wartość $nonce_{RR \to A}$ oraz aktualny znacznik czasu $t_{RR \to A}$. Odpowiedź *response* oraz znacznik czasu $t_{RR \to A}$ jest szyfrowany z użyciem $K_{RR \to A}$ oraz $nonce_{RR \to A}$: $E_{RR \to A} = E[K_{R \to AA}; nonce_{RR \to A}; (response, t_{RR \to A})]$. Wiadomość $M_{RR \to A}$ zawierająca identyfikator urządzenia

ID_A, listę indeksów $LI_{RR\to A}$, wartość $nonce_{RR\to A}$, szyfrogram $E_{RR\to A}$ jest zwracana do bramy aplikacyjnej BA_i, która przesyła ją do urządzenia A. Urządzenie A po odebraniu wiadomości $M_{RR\to A}$ odczytuje listę indeksów $LI_{RR\to A}$. Na podstawie listy indeksów $LI_{RR\to A}$ urządzenie odtwarza wartości parametrów o podanych indeksach, które zostały użyte do utworzenia klucza $K_{RR\to A}$. Urządzenie odtwarza klucz $K_{RR\to A} = H(G(p_{m_1})|G(p_{m_2})|....|G(p_{m_k}))$ i dokonuje odszyfrowania $E_{RR\to A}$. Wynikiem odszyfrowania jest *response* oraz znacznik czasu $t_{RR\to A}$. Jeżeli znacznik czasu $t_{RR\to A}$ jest nieaktualny to wiadomość $M_{RR\to A}$ jest odrzucana. W przeciwnym przypadku wynik *response* jest wynikiem zwróconym przez rejestr rozproszony RR. Proces komunikacji został przestawiony na rysunku 3.7. Pełny proces komunikacji pomiędzy urządzeniem A, BA_i i RR został przedstawiony na rysunku 3.8. Pomiędzy BA_i i RR ustanowione jest połączenie zabezpieczone z wykorzystaniem TLS z obopólnym uwierzytelnianiem.

$$A \xrightarrow{ID_A, LI_{A\to RR}, nonce_{A\to RR}, E_{A\to RR}} BA_i/RR$$
$$A \xleftarrow{ID_A, LI_{RR\to A}, nonce_{RR\to A}, E_{RR\to A}} BA_i/RR$$

Rysunek 3.7: Diagram komunikacji wykorzystującej LAAFFI pomiędzy urządzeniem A a bramą aplikacyjną BA_i w procesie wykonywania operacji przez BA_i

$$\begin{aligned}
A \xrightarrow{ID_A, LI_{A\to RR}, nonce_{A\to RR}, E_{A\to RR}} & BA_i \\
& BA_i \xrightarrow{ID_A, LI_{A\to RR}, nonce_{A\to RR}, E_{A\to RR}} RR \\
& BA_i \xleftarrow{dane} RR \\
& BA_i \xrightarrow{response} RR \\
& BA_i \xleftarrow{ID_A, LI_{RR\to A}, nonce_{RR\to A}, E_{RR\to A}} RR \\
A \xleftarrow{ID_A, LI_{RR\to A}, nonce_{RR\to A}, E_{RR\to A}} & BA_i
\end{aligned}$$

Rysunek 3.8: Diagram pełnej komunikacji pomiędzy urządzeniem A a rejestrem rozproszonym RR w procesie wykonywania operacji zleconej przez urządzenie A

3.3.3 Bezpieczna komunikacja urządzenia IoT z innym urządzeniem IoT

Można wskazać wiele sytuacji, gdy urządzenia IoT należące do różnych organizacji muszą się ze sobą komunikować w sposób bezpieczny. Załóżmy, że urządzenie IoT musi komunikować się z innym urządzeniem, które należy do innej organizacji w celu pobrania od niego aktualnych danych z jego sensorów. Konieczne jest ustanowienie bezpiecznego kanału, aby zapewnić bezpieczną komunikację między urządzeniami. Można to osiągnąć, szyfrując przesyłane wiadomości za pomocą wspólnego klucza. Taki klucz może być wygenerowany przez rejestr rozproszony na podstawie tablicy parametrów TP urządzenia, z którym ma być nawiązana komunikacja. Aby możliwe było nawiązanie komunikacji pomiędzy dwoma urządzeniami muszą być one zarejestrowane w rejestrze rozproszonym RR. Załóżmy, że po wykonaniu rejestracji urządzenie A będzie miało identyfikator ID_A, natomiast urządzenie B identyfikator ID_B. Urządzenie A może uzyskać identyfikator urządzenia B na wiele sposobów m.in. może go otrzymać od innego urządzenia lub rejestru rozproszonego, urządzenie B może rozgłaszać swój identyfikator lub też urządzenie A może mieć skonfigurowane w programie, że ma się z tym urządzeniem komunikować. Jeżeli zachodzi potrzeba przesłania danych z urządzenia A do urządzenia B, procedura komunikacji zaczyna się podobnie jak w przypadku 3.3.2. Jedyną różnicą jest to, że zamiast danych operacyjnych *dane* do bramy aplikacyjnej BA_i wysyłany jest zaszyfrowany identyfikator ID_B urządzenia B. Wiadomość po odszyfrowywana przez węzeł rejestru rozproszonego RR_i wraz z identyfikatorem ID_B jest przekazywana do bramy aplikacyjnej BA_i.

Kolejnym krokiem jest wygenerowanie klucza do komunikacji pomiędzy urządzeniami A i B. W tym celu brama aplikacyjna BA_i wysyła żądanie wygenerowania klucza $K_{A\leftrightarrow B}$ do węzła rejestru rozproszonego RR_i. Węzeł RR_i w pierwszym kroku sprawdza, czy urządzenie A może komunikować się z urządzeniem B. Proces autoryzacji przebiega z wykorzystaniem rejestru rozproszonego, co zostało opisane w 3.4. Jeżeli urządzenia nie mogą się ze sobą komunikować, to klucz nie zostanie wygenerowany, a urządzenie A otrzyma odpowiedni komunikat. Natomiast w przypadku, gdy urządzenia mogą się ze sobą komunikować, to węzeł rejestru rozproszonego RR_i przystępuje do tworzenia klucza $K_{A\leftrightarrow B}$. Węzeł RR_i generuje k-elementową listę indeksów ze zbioru $\{1, 2, ..., l\}$: $LI_{A\rightarrow B} = \{n_1, n_2, ..., n_k\}$. Generowany jest także aktualny znacznik czasu $t_{A\leftrightarrow B}$.

Klucz do komunikacji pomiędzy urządzeniami jest wynikiem funkcji skrótu dla konkatenacji parametrów z tablicy parametrów TP_B o indeksach $LI_{A\rightarrow B}$ oraz znacznika czasu $t_{A\leftrightarrow B}$.

Ustalony klucz ma postać $K_{A\leftrightarrow B} = H(G(p_{n_1})|G(p_{n_2})|....|G(p_{n_k})|t_{A\leftrightarrow B})$. Następnym krokiem jest wysłanie klucza $K_{A\leftrightarrow B}$ wraz z listą indeksów $LI_{A\leftrightarrow B}$ i znacznikiem czasu $t_{A\leftrightarrow B}$ do urządzenia A. Aby to zrobić węzeł rejestru rozproszonego RR_i generuje k-elementową listę indeksów $\{1, 2, ..., l\}$: $LI_{RR\rightarrow A} = \{n_1, n_2, ..., n_k\}$. Wybiera z tablicy parametrów TP_A parametry o indeksach zapisanych w liście indeksów $LI_{RR\rightarrow A}$ i na ich podstawie tworzy klucz: $K_{RR\rightarrow A} = H(G(p_{n_1})|G(p_{n_2})|....|G(p_{n_k}))$. Generowany jest także aktualny znacznik czasu $t_{RR\rightarrow A}$ oraz losowa wartość nonce $nonce_{RR\rightarrow A}$. Wykorzystując klucz $K_{RR\rightarrow A}$ oraz wartość $nonce_{RR\rightarrow A}$, węzeł rejestru rozproszonego RR_i szyfruje klucz $K_{A\leftrightarrow B}$ wraz z listą indeksów $LI_{A\leftrightarrow B}$ oraz znacznikiem czasu $t_{A\leftrightarrow B}$, rezultatem jest szyfrogram: $E_{RR\rightarrow A} = E[K_{RR\rightarrow A}; nonce_{RR\rightarrow A}; (K_{A\leftrightarrow B}, LI_{A\leftrightarrow B}, t_{A\leftrightarrow B}, t_{RR\rightarrow A})]$. Wiadomość $M_{RR\rightarrow A}$ zawierająca: identyfikator urządzenia ID_A, listę indeksów $LI_{RR\rightarrow A}$, wartość $nonce_{RR\rightarrow A}$ oraz szyfrogram $E_{RR\rightarrow A}$ jest przesyłana do bramy aplikacyjnej BA_i, która wysyła ją to urządzenia A. Po odebraniu wiadomości $M_{RR\rightarrow A}$ urządzenie A odszyfrowuje ją tak jak zostało opisane to w sekcji 3.3.2, weryfikując znacznik czasu $t_{RR\rightarrow A}$ z aktualnym. Jeżeli jest poprawny, to urządzenie A może przystąpić do wysłania danych *dane* do urządzenia B.

Aby wysłać dane do *dane* do urządzenia B, urządzenie A generuje losową wartość $nonce_{A\rightarrow B}$ oraz nowy znacznik czasu $t_{A\rightarrow B}$. Wykorzystując klucz $K_{A\leftrightarrow B}$, który został odszyfrowany z wiadomości $M_{RR\rightarrow A}$ oraz wygenerowaną wartość $nonce_{A\rightarrow B}$ dokonuje szyfrowania danych *dane* oraz znacznika czasu $t_{A\rightarrow B}$, wskutek czego powstaje szyfrogram $E_{A\rightarrow B} = E[K_{A\leftrightarrow B}; nonce_{A\rightarrow B}; (dane, t_{A\rightarrow B})]$. Do urządzenia B wysyła wiadomość: $M_{A\rightarrow B}$ zawierającą identyfikator urządzenia ID_A, listę indeksów $LI_{A\leftrightarrow B}$, wartość $nonce_{A\rightarrow B}$, znacznik czasu $t_{A\leftrightarrow B}$ oraz szyfrogram $E_{A\rightarrow B}$.

Urządzenie B po odebraniu wiadomości $M_{A\rightarrow B}$ porównuje znacznik czasu $t_{A\leftrightarrow B}$ z aktualnym. Jeżeli różnica jest akceptowalna, to urządzenie B wykonuje polecenia wymagane do uzyskania parametrów o indeksach zapisanych w liście indeksów $LI_{A\leftrightarrow B}$. Na podstawie parametrów oraz znacznika czasu $t_{A\leftrightarrow B}$ tworzony jest klucz $K_{A\leftrightarrow B} = H(G(p_{n_1})|G(p_{n_2})|....|G(p_{n_k})|t_{A\leftrightarrow B})$. Klucz $K_{A\leftrightarrow B}$ wraz z wartością $nonce_{A\rightarrow B}$ wykorzystywany jest do odszyfrowania $E_{A\rightarrow B}$. Urządzenie B uzyskuje przesłane dane *dane* oraz znacznik czasu $t_{A\rightarrow B}$. Znacznik $t_{A\rightarrow B}$ porównywany jest z aktualnym i podobnie jak w poprzednich sekcjach 3.3.1 i 3.3.2 wiadomość $M_{A\rightarrow B}$ jest akceptowana lub odrzucana. Jeżeli wszystkie kroki się powiodą, to urządzenie B może wykonać operacje związane z przesłanymi informacjami zawartymi w polu *dane*.

W celu wysłania odpowiedzi *response*, urządzenie B generuje losową wartość $nonce_{B\rightarrow A}$

oraz aktualny znacznik $t_{B\to A}$. Odpowiedź $response$ jest szyfrowana z użyciem klucza $K_{A\leftrightarrow B}$, oraz wartości $nonce_{B\to A}$ w następujący sposób: $E_{B\to A} = E[K_{A\leftrightarrow B}; nonce_{B\to A}; (response, t_{B\to A})]$. Urządzenie B przesyła do urządzenia A: identyfikator ID_B, listę indeksów $LI_{A\leftrightarrow B}$, wartość $nonce_{B\to A}$, znacznik czasu $t_{A\leftrightarrow B}$ oraz szyfrogram $E_{B\to A}$. Urządzenie A bazując na zapisanym kluczu $K_{A\leftrightarrow B}$ oraz $nonce_{B\to A}$ odszyfrowuje szyfrogram $E_{B\to A}$. Wynikiem odszyfrowania $E_{B\to A}$ jest odpowiedź $response$ oraz znacznik czasu $t_{B\to A}$. Znacznik ten jest porównywalny z aktualnym. Jeżeli różnica pomiędzy znacznikami jest akceptowalna, oraz znacznik czasu jest większy od wcześniejszego to wiadomość $M_{B\to A}$ nie jest odrzucana. Klucz $K_{A\leftrightarrow B}$ jest przechowywany przez obydwa urządzenia do końca prowadzenia komunikacji lub dopóki znacznik czasu $t_{A\leftrightarrow B}$ jest uznawany za akceptowalny. Diagram komunikacji wykorzystującej LAAFFI dla tego przypadku został przedstawiony na rysunku 3.9. Na rysunku 3.10 przedstawiono diagram pełnej komunikacji wymaganej do uzyskania bezpiecznego połączenia pomiędzy urządzeniami IoT.

$$A \xrightarrow{ID_A, LI_{A\to RR}, nonce_{A\to RR}, E_{A\to RR}} BA_i/RR$$

$$A \xleftarrow{ID_A, LI_{RR\to A}, nonce_{RR\to A}, E_{RR\to A}} BA_i/RR$$

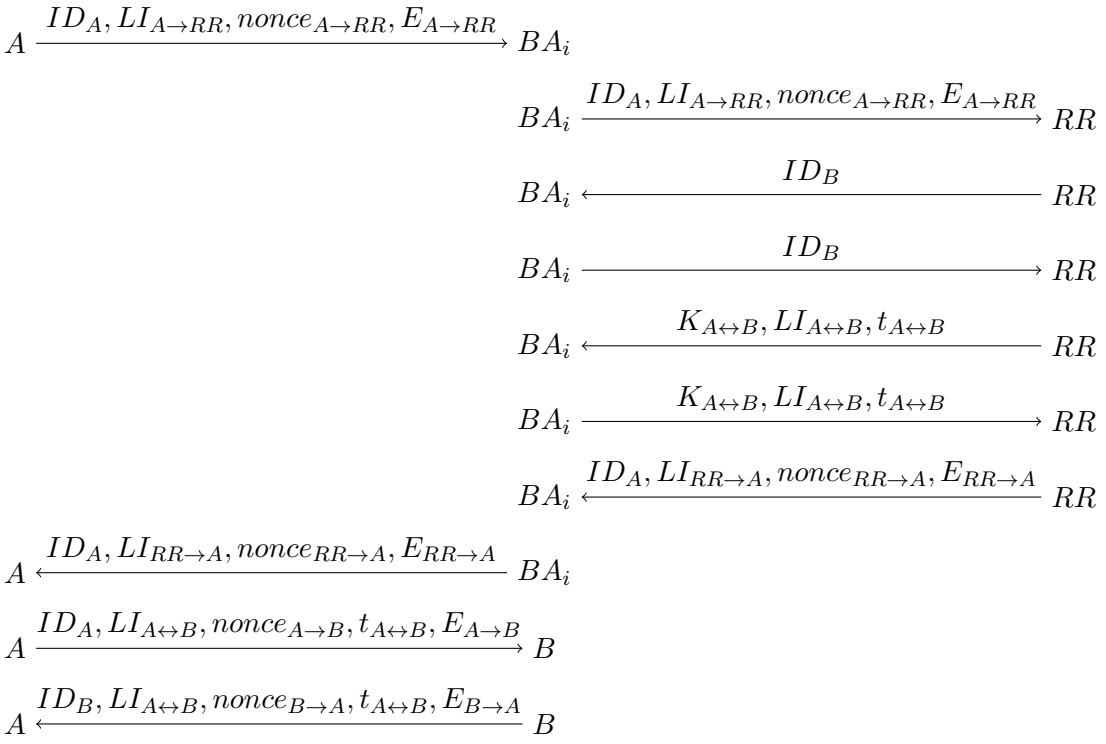
$$A \xrightarrow{ID_A, LI_{A\leftrightarrow B}, nonce_{A\to B}, t_{A\leftrightarrow B}, E_{A\to B}} B$$

$$A \xleftarrow{ID_B, LI_{A\leftrightarrow B}, nonce_{B\to A}, t_{A\leftrightarrow B}, E_{B\to A}} B$$

Rysunek 3.9: Diagram nawiązania bezpiecznej komunikacji pomiędzy urządzeniami A i B przedstawiający wiadomości zabezpieczone z wykorzystaniem LAAFFI

3.4 Proces autoryzacji

Autoryzacja w środowisku federacyjnym jest niezbędna, aby zapewnić kontrolę dostępu wszystkim elementom środowiska IoT. Do elementów można zaliczyć m.in. użytkowników, pracowników, urządzenia IoT wraz z udostępnianymi przez nich usługami, usługi oferowane przez organizacje (w tym rejestr rozproszony). Model autoryzacji musi zostać zdefiniowany na początku tworzenia federacji, tak aby od początku wszyscy uczestnicy federacji korzystali z tego samego modelu autoryzacji. Weryfikacja uprawnień jest bardzo istotnym procesem ze względu na konieczność zachowania poufności danych. W LAAFFI weryfikacja uprawnień jest realizowana jako element kodu łańcuchowego obligatoryjnie przy wykonywaniu innych operacji



Rysunek 3.10: Diagram pełnej komunikacji potrzebnej do nawiązania bezpiecznego połączenia pomiędzy urządzeniami A i B

np. udostępnienia klucza do komunikacji między urządzeniami, czy też wykonywania żądań dla innych usług. Realizacja procesu autoryzacji jako kod łańcuchowy gwarantuje, że reguły autoryzacji są identyczne dla wszystkich organizacji. Autoryzacja może być także wykonywana niezależnie jako oddzielny proces poprzez wysłanie żądanie weryfikacji uprawnień przez bramę aplikacyjną BA_i. Ponieważ proces autoryzacji jest realizowany przez kod łańcuchowy, to wszystkie operacje związane z weryfikacją uprawnień muszą spełniać określone wymagania. Do takich wymagań należy zaliczyć brak możliwości modyfikacji uprawnień urządzeń, które należą do innej organizacji, możliwość nadawania uprawnień tylko zarejestrowanym urządzeniom, modyfikacja uprawnień tylko przez uprawnione osoby, czy też ograniczenie możliwości wyboru uprawnień dla konkretnego typu urządzenia. Wszystkie te wymagania są weryfikowane podczas procesu modyfikacji uprawnień. Ponieważ wszystkie uprawnienia są przechowywane w rejestrze rozproszonym w oddzielnym kanale, to każdy z członków federacji należący do tego kanału ma możliwość sprawdzenia uprawnień każdego urządzenia IoT.

3.5 Uwagi implementacyjne

Wykorzystanie protokołu TLS do komunikacji pomiędzy węzłami rejestru rozproszonego a bramą aplikacyjną w protokole LAAFFI jest obligatoryjne, ponieważ tylko ten protokół jest wspierany przez Hyperledger Fabric. Wykorzystanie protokołu CoAP oraz formatu danych CBOR do komunikacji urządzenia IoT z bramą aplikacyjną jest zalecane ze względu na przystosowanie tych rozwiązań do wykorzystania w środowiskach IoT. Komunikacja pomiędzy usługą AA a aplikacją użytkownika została zrealizowana z wykorzystaniem Representational State Transfer (REST) Application Programming Interface (API). Jednak nic nie stoi na przeszkodzie, aby wykorzystać inne mechanizmy np. Remote Procedure Call (gRPC), pamięć współdzieloną.

Brama aplikacyjna została zaimplementowana przy użyciu języka Golang. Zamiast tego języka można wykorzystać dwa inne: Node.js oraz Java. Ograniczenie to wynika z faktu, że tylko dla tych trzech języków istnieje Software Development Kit (SDK) [86], które umożliwia połączenie się z węzłami Hyperledger Fabric [62], a który został wykorzystany do implementacji rejestru rozproszonego. Uzasadnienie wyboru Hyperledger Fabric przedstawiono w sekcji 2.4. Golang został wybrany ze względu na większą wydajność tego języka w porównaniu z innymi dostępnymi.

Do zarządzania wdrożoną metodą autoryzacji zaimplementowano następujące operacje:

1. dodanie uprawnienia,
2. usunięcie uprawnienia,
3. wyświetlenie uprawnień urządzenia IoT,
4. sprawdzenie uprawnień.

W ramach opracowanej implementacji zastosowano metodą autoryzacji opartą o listę kontroli dostępu (ang. ACL). Zastosowana metoda autoryzacji obejmuje tylko weryfikację uprawnień do urządzeń IoT oraz usług przez nie udostępnianych. Z tego względu podmiotem może być tylko urządzenie IoT. Obiektami, do którego mogą być nadane uprawnienia są usługi oferowane przez urządzenie IoT, natomiast zbiór operacji nie został jawnie określony. Właściciel urządzenia IoT sam może zdefiniować rodzaje operacji, jakie mogą być wykonane.

Implementacja uwzględnia wymagania oraz operacje opisane na początku tej sekcji. Operacje mogą być wywołane z innego kodu łańcuchowego — tak się dzieje w przypadku procesu

generowania klucza do komunikacji pomiędzy urządzeniami. Kod łańcuchowy z kanału odpowiedzialnego za uwierzytelnianie sprawdza, czy urządzenia mogą się komunikować ze sobą poprzez wywołanie innego kodu łańcuchowego. Operacja sprawdzenia uprawnień może też być wywoływana przez bramę aplikacyjną — taka sytuacja ma miejsce, gdy urządzenie musi np. wykonać operację na innej usłudze. Dodawanie lub usuwanie uprawnień jest możliwe tylko przez osobę, która należy do tej samej organizacji co urządzenie IoT. W operacji nadawania uprawnienia do każdego urządzenia dodawany jest wpis zawierający dwie informacje:

1. jaki operacje może wykonać dane urządzenie,
2. jaką operację, może wykonać na tym urządzeniu inne urządzenie.

Sprawdzenie uprawnień polega na zweryfikowaniu czy urządzenie, na którym ma być wykonana operacja ma wpis zawierający nazwę operacji, która ma być wykonana oraz nazwa urządzania przy nazwie tej operacji zgadza się z nazwą urządzenia, które chce uzyskać dostęp. Jeżeli tak jest, to urządzenie żądające ma prawo wykonać żądaną operację.

Można także zaimplementować inne metody autoryzacji korzystające z Hyperledger Fabric jako systemu przechowującego oraz sprawdzającego uprawnienia. Opracowany protokół nie posiada ograniczenia co do metody autoryzacji.

Niezależnie od wykorzystywanej metody autoryzacji powstaje problem: skąd organizacja zarządzająca urządzeniem IoT lub też właściciel urządzenia ma wiedzieć, że inne urządzenie należące do tej samej lub innej organizacji chce mieć nadane uprawnienie do wykonania jakiejś operacji. Takie żądanie może być wysłane innym kanałem komunikacji np. e-mailem, lub też można skorzystać z rejestru rozproszonego. W drugim przypadku organizacje będą zgłaszały swoje żądania jako transakcje w ramach rejestru rozproszonego. Właściciel urządzenia będzie mógł zaakceptować takie żądanie lub też je odrzucić. Jeżeli zaakceptuje to uprawnienie zostanie zmienione. W takim scenariuszu rejestr rozproszony będzie przechowywał całą historię żądań i ich rezultatów.

3.6 Podsumowanie

W tym rozdziale opisano założenia, architekturę, informacje o implementacji, a także opisano dokładnie operacje, jakie są przewidziane w ramach opracowanego protokołu LAAFFI. Protokół ten jest modularny, ponieważ nie został zaprojektowany z myślą o wykorzystaniu konkretnych protokołów komunikacyjnych i kryptograficznych, technologii i rozwiązań. Z tego

też powodu można zastosować różne protokoły kryptograficzne, czy też modele autoryzacji. LAAFFI ze względu na wykorzystanie rejestru rozproszonego, a także lekkich protokołów spełnia założenia postawione w rozdziale pierwszym. Ocena bezpieczeństwa protokołu oraz jego wydajność jest przedmiotem rozważań przedstawionych w kolejnych rozdziałach.

Rozdział 4

Ocena bezpieczeństwa opracowanego protokołu

W tym rozdziale przedstawiono ocenę bezpieczeństwa opracowanego protokołu. Najpierw opisano analizę entropii parametrów urządzeń oraz wyznaczono minimalną odległość Hamminga dla parametrów. Następnie przedstawiono opis i analizę powodzenia możliwych ataków na opracowany protokół. Na końcu przedstawiono analizę formalną bezpieczeństwa LAAFFI z wykorzystaniem narzędzi Verifpal i Tamarin.

4.1 Analiza losowości parametrów

Jednym z aspektów bezpieczeństwa protokołu LAAFFI, który należy ocenić jest zachowanie poufności przesyłanych danych. Osiągnięcie tego jest możliwe w sytuacji, gdy tylko strony komunikacji mają dostęp do przesyłanej informacji. W rozdziale 3.1 zaproponowano utworzenie bezpiecznego klucza szyfrowania symetrycznego na podstawie jednego z trzech różnych źródeł:

1. PUF,
2. losowo wygenerowanego zestawu parametrów dla urządzenia,
3. zestawu dobranych unikatowych danych konfiguracyjnych urządzenia.

Aby ocenić przydatność zaproponowanego rozwiązania, w sytuacji 1) zostanie oceniona entropia na podstawie danych literaturowych. Dla źródła 2) potrzebne jest użycie odpowiedniego generatora, którego działanie zostanie przedstawione. Natomiast dla sytuacji 3) należy ocenić entropię parametrów sprzętowo-konfiguracyjnych uzyskiwanych z urządzenia IoT.

Według National Institute of Standards and Technology (NIST) [152] losowość sekwencji jest określana jako sekwencja bitów, w których prawdopodobieństwo wystąpienia wartości "0"i "1"jest równe 0,5, a wartość każdego bitu jest niezależna od pozostałych tzn. każdy bit jest

nieprzewidywalny. Aby uzyskać sekwencję bitów uznawanych za losowe, korzysta się z generatorów liczb losowych i pseudolosowych. Natomiast jeżeli sekwencja bitów ma zdefiniowaną długość, można wykorzystać funkcje pseudolosowe (ang. Pseudo Random Function (PRF)) [172]. Ponieważ w LAAFFI klucz do zabezpieczenia komunikacji tworzony jest w sposób deterministyczny (poprzez wykorzystanie funkcji skrótu i HMAC), to w dalszej części rozważań skupiono się tylko na funkcjach pseudolosowych, ponieważ wykorzystanie funkcji pseudolosowych pozwala na dla ciągu wejściowego uzyskanie zawsze tego samego ciągu wyjściowego o określonej długości. Dane wyjściowe PRF są obliczeniowo nierozróżnialne od prawdziwie losowych danych wyjściowych[45]. NIST zdefiniował [186] pseudolosowość, jako proces deterministyczny, którego wartości wyjściowe są skutecznie nierozróżnialne od wartości procesu losowego, o ile wewnętrzne stany i działania są nieznane. Dla celów kryptograficznych "skutecznie nieodróżnialny"oznacza "niemieszczący się w granicach obliczeniowych ustalonych przez zamierzoną siłę zabezpieczeń" [186]. Aby taka funkcja wygenerowała pseudolosowy ciąg, który nie będzie możliwy do odgadnięcia przez atakującego, należy podać na jej wejściu ciąg bitów (ziarno), który będzie miał odpowiedni poziom entropii tzn. największą liczbę bitów, których nie można przewidzieć. Entropia jest miarą niepewności lub nieporządku systemu [12] i im większa entropia, tym mniejsza pewność wyników. Zatem, podając miarę entropii, można wywnioskować, że entropia jest maksymalna, gdy rozkład jest jednorodny, ponieważ przy takim rozkładzie niepewność będzie największa tzn. żaden wynik nie będzie bardziej prawdopodobny od innych. Obecnie za akceptowalną wartość entropii dla kluczy symetrycznych uznaje się 128 bitów [42], natomiast National Security Agency (NSA) zaleca 256 bitów [10]. Oczywiście zbadanie samej entropii dla ciągu jest niewystarczające, aby móc stwierdzić, że dany ciąg jest losowy. Do tego potrzebne są testy statystyczne takie jak przestawione w: [32, 152].

Uzyskanie odpowiednio wysokiej entropii źródła jest najważniejszym elementem, aby móc utworzyć klucz mogący zagwarantować (przy wykorzystaniu bezpiecznego algorytmu szyfrowania oraz procesu zarządzania kluczami kryptograficznymi) poufność przesyłanych danych. W przypadku protokołu LAAFFI należy ocenić entropię źródła dla trzech sytuacji, w których można wykorzystać trzy różne źródła, z których można utworzyć klucz.

4.1.1 Układy PUF

Do obliczenia entropii układów PUF najczęściej wykorzystuje się min-entropię opisaną w [186]. Min-entropia opisuje nieprzewidywalność wyniku określonego wyłącznie przez prawdopodo-

bieństwo najbardziej prawdopodobnego wyniku. Określana jest wzorem: $H = log_2 \frac{1}{p_{max}}$ gdzie: H – min-entropia, p_{max} – prawdopodobieństwo poprawnego odgadnięcia najbardziej prawdopodobnego wyniku w pierwszej próbie.

Ze względu na budowę niektórych układów PUF, okazuje się, że oszacowanie min-entropii może dać błędne wyniki [166]. Oszacowanie entropii zależy od uporządkowania odpowiedzi układów PUF, a w niektórych przypadkach entropia jest przeszacowana. Dlatego też można spotkać wykorzystanie metody ważenia drzewa kontekstu (ang. Context tree weighting) [126, 144], lub wykorzystuje się testy opisane w NIST 800-22 [152]. W przypadku testów 800-22, ich wykorzystanie jest mocno dyskusyjne, ponieważ są one przygotowane do badania losowości generatorów liczb pseudolosowych [126]. Proponowane są też inne metody szacowania entropii [22] dla niektórych rodzajów PUF.

W tabeli 4.1 przedstawiono wartości entropii oszacowanej dla różnych rodzajów PUF.

Tabela 4.1: Wartość entropii dla różnych rodzajów układów PUF

Rodzaj PUF	**Entropia [bit]**	**Artykuł**
Wykorzystywane bezpośrednie charakterystyki	6,6 bit z każdego sensora	[126]
Pamięć ulotna SRAM	0,76 bit na komórkę	[126]
	0,797 – 0,799 bit na komórkę	[37]
Pamięć ulotna DRAM	0,95 bit na komórkę	[1]

Poza rodzajem układu PUF wartość entropii dla układów PUF jest zależna także od innych czynników m.in. temperatury [100], wieku układu [182] a także liczby błędnych bitów generowanych przez układ. Im więcej błędów występuje w odpowiedzi PUF tym mniejszą entropię można uzyskać [193], dlatego też trzeba zwracać uwagę na odległość wewnętrzną pomiędzy odpowiedziami układu PUF.

Ogólnie rzecz biorąc dokładne oszacowanie entropii odpowiedzi z układów PUF jest bardzo trudne, a sam wynik nie jest pewny. Wynika to z istoty układów PUF, dla których odpowiedzi są zależne od procesów fizycznych. Pomimo tego wykorzystanie układów PUF w protokole LAAFFI jest możliwe. Należy jednak wykorzystać więcej odpowiedzi układu do utworzenia klucza uwierzytelniającego, tak aby wielkość oczekiwanej entropii była uznana za wystarczającą.

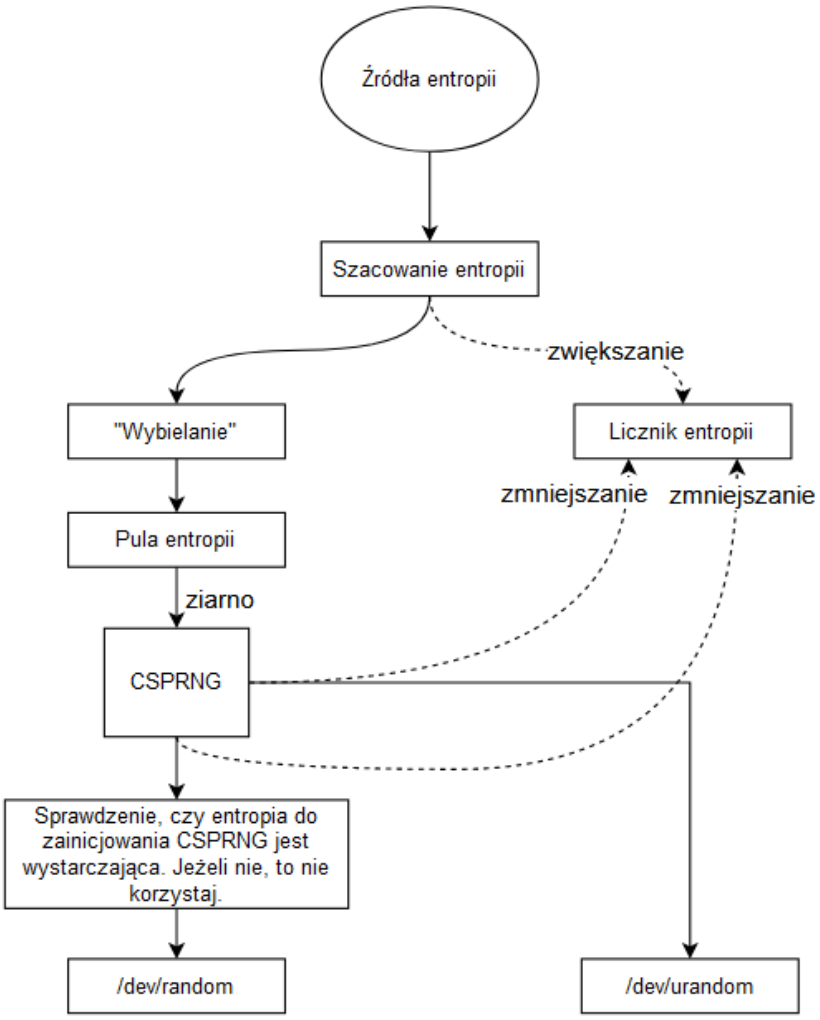
W publikacji [126] zwraca się także uwagę na kwestię różnic pomiędzy różnymi układami PUF. Jeżeli różne układy PUF mają niską wartość odległości pośredniej Hamminga (ang.

inter-distance), to znając odpowiedzi jednego układu można próbować odgadnąć odpowiedzi innego, bliźniaczego układu PUF.

4.1.2 Losowe ciągi bitów

W tym przypadku urządzenie podczas procesu rejestracji generuje losowe ciągi bitów, które są zapisywane w jego pamięci. Uzyskanie losowości w generowaniu tych ciągów jest możliwe dzięki źródłu entropii dostarczanemu przez system operacyjny, w badanym przypadku jest to system operacyjny Linux. Ze względu na trudności uzyskiwania losowych wartości w systemie operacyjnym, wykorzystywany w systemie Linux generator jest pseudolosowy. Do generowania wartości pseudolosowych wykorzystywany jest *szum środowiskowy* z systemu komputerowego, który jest trudny do odtworzenia przez atakującego. Źródła losowości obejmują czasy między naciśnięciami klawiszy, przerwy między niektórymi przerwaniami systemowymi itp., które są niedeterministyczne i tym samym trudne do zbadania przez zewnętrznego atakującego, który nie ma dostępu do urządzenia. Bity pochodzące z tych źródeł są poddawane operacji wybielania (ang. whitening lub de-biasing). Operacja ta ma na celu korekcję błędów powstałych w źródle entropii. Jednocześnie szacowana jest wielkość entropii. Szacowanie entropii jest wykonywane przy użyciu interpolacji wielomianowej dla czasów wystąpienia zdarzeń otrzymania losowych bitów [65, 145]. Bity ze źródeł losowości są następnie przekazywane do *puli entropii*. *Pula entropii* jest zbiorem losowych bitów przechowywanych w pamięci systemu operacyjnego. Bity z puli entropii są wykorzystywane jako ziarno w kryptograficznie bezpiecznym generatorze pseudolosowym (ang. Cryptographically-Secure Pseudo-Random Number Generator (CSPRNG)). Od wersji 5.6 jądra systemu operacyjnego Linux, */dev/random* i */dev/urandom* pobierają dane z CSPRNG z tą różnicą, że */dev/random* nie będzie udostępniał danych, jeżeli CSPRNG nie zostanie zainicjowany ziarnem o odpowiednio dużej entropii.

Proces generowania wartości pseudolosowych został przedstawiony na rysunku 4.1. W przypadku urządzeń IoT wiele źródeł entropii nie może być wykorzystanych, ponieważ one nie istnieją, np. przerwania z klawiatury, dysku twardego. Dlatego urządzenia IoT mogą być wyposażone w sprzętowe generatory liczb losowych. Niektóre z nich mają wbudowany sprzętowy generator liczb losowych, np. Raspberry Pi 3.



Rysunek 4.1: Metoda generowania liczb pseudolosowych w systemie Linux od wersji 5.6. Źródło: Opracowanie własne na podstawie [183]

4.1.3 Wykorzystanie parametrów konfiguracyjnych

Jeżeli chcemy zweryfikować entropię źródła należy wyznaczyć min-entropię dla każdego parametru. Ze względu na to, że kolejne wartości następujących bajtów parametrów mają jednakowe prawdopodobieństwo wystąpienia można skorzystać ze wzoru [186]:

$H = log_2 \frac{1}{p_{max}}$

gdzie: H – min-entropia parametru, p_{max} – prawdopodobieństwo poprawnego odgadnięcia najbardziej prawdopodobnego wyniku w pierwszej próbie.

Ze względu na to, że prawdopodobieństwo wystąpienia każdej wartości dla parametru jest identyczne, można skorzystać również ze wzoru: $H = L * log_2 b$

gdzie: H – min-entropia parametru, L – liczba bajtów parametru (jego długość), b – liczba

możliwych znaków.

Jednakowe prawdopodobieństwo dla parametrów programowych wynika z tego, że parametry programowe są wygenerowane w sposób pseudolosowy. Powstały one na urządzeniu podczas instalacji i konfiguracji systemu operacyjnego oraz oprogramowania.W przypadku opracowanej metody samo urządzenie nie musi posiadać generatora liczb pseudolosowych, który służy do wygenerowania parametrów programowych, ponieważ można wygenerować ich wartości podczas instalowania systemu (np. z użyciem zewnętrznego generatora liczb pseudolosowych) poza samym urządzeniem IoT, a potem kartę z zainstalowanym i skonfigurowanym systemem umieścić w docelowym urządzeniu IoT. W przypadku parametrów sprzętowych generowanie numeru seryjnego lub innej cechy odbywa się na etapie produkcji komponentu. Założono, że jest to robione w sposób pseudolosowy. Zbadanie losowości tych parametrów jest dyskusyjne, ponieważ ewentualnemu badaniu poddawane byłyby kilkuznakowe ciągi pochodzące z różnych urządzeń, a nie jeden ciąg znaków pochodzący z jednego źródła, kłóci się z założeniami opracowanymi przez NIST [152]. Testy te wymagają uzyskania minimum 1 miliona bitów, a w przypadku parametru, jakim jest numer seryjny, z każdego urządzenia można uzyskać tylko 64 bity (numer seryjny ma 8 bajtów). Można zatem wskazać trzy podstawowe powody, dla których zrezygnowano z przeprowadzenia testów losowości. Pierwszym z nich jest wymóg wykorzystania sekwencji minimum kilku tysięcy bitów, których nie można uzyskać z pojedynczego urządzania. Należałoby dokonać konkatenacji z wielu urządzeń, co oznacza, że wynik takich testów będzie mocno dyskusyjny. Drugim z nich jest koszt pozyskania urządzeń. Aby przeprowadzić testy losowości NIST [152] wymagane byłoby posiadanie dostępu do minimum 15 625 urządzeń (ponieważ: 1 000 000 bitów wymaganych / 64 bity dostarczone przez 1 urządzenie), gdzie koszt jednego urządzenia wynosi ok. 90 złotych. Trzecim powodem jest to, że badanie odnosiłoby się do oceny losowości dla pojedynczego modelu lub modeli producenta. Z tych samych powodów nie zbadano losowości parametrów sprzętowych dla karty pamięci.

W rozprawie ograniczono się do przebadania entropii dla wybranych parametrów, które uznano za obiecujące do wykorzystania w procesie tworzenia klucza. W pierwszym kroku zebrano wartości parametrów z 20 urządzeń Raspberry Pi, 20 kart pamięci oraz 20 zainstalowanych systemów operacyjnych. Następnie obliczono entropię dla każdego parametru przy użyciu narzędzia ent [1]. Wyniki zostały przedstawione w tabeli 4.2.

Na podstawie wyników przedstawionych w tabeli 4.2, można wyliczyć, że biorąc pod uwagę

[1] `http://www.linuxcertif.com/man/1/ent/`

Tabela 4.2: Lista poleceń wraz z wyliczoną entropią

Lp.	**Parametr**	**Entropia [bit]**	**Liczba znaków**
1	Numer seryjny urządzenia Raspberry pi	32	8
2	Identyfikator sprzętowy Raspberry pi	32	8
3	Numer seryjny karty pamięci	29	8
4	Wartość CID	33	9
5	PTUUID dla partycji	19	5
6	UUID systemu plików	128	32
7	UUID systemu plików dla FATBOOT	32	8
8	Ziarno hasła z /etc/shadow	93	16
9	Wartość funkcji skrótu z /etc/shadow	Zależna od hasła	88
10	Ziarno dla funkcji skrótu katalogów (ext4)	129	32
11	Identyfikator systemu operacyjnego	129	32

siedem parametrów o najniższej entropii (numer seryjny urządzenia Raspberry Pi, identyfikator sprzętowy Raspberry Pi, numer seryjny karty pamięci, wartości CID, PTUUID dla partycji, UUID systemu plików dla FATBOOT oraz ziarno hasła) można otrzymać 274 bity entropii wejściowej. Jest to wartość wystarczająca do utworzenia klucza o entropii 128 bitów [186]. Wartość entropii wynika z sumowania entropii dla wybranych parametrów. Wartości entropii można sumować ze sobą pod warunkiem, że parametry są niezależne. Należy zachować ostrożność przy wyborze parametrów do identyfikacji urządzenia IoT, ze względu na pewne zależności pomiędzy parametrami wprowadzane przez producentów. Przykładowo w Raspberry Pi wartości adresów MAC i adres sprzętowy Bluetooth zależą od numeru seryjnego urządzenia. Adres fizyczny interfejsu przewodowego (adres MAC) to wartość B827EB oraz 48 dolnych bitów z numeru seryjnego, natomiast adres fizyczny interfejsu bezprzewodowego to B827EB oraz 48 dolnych bitów z numeru seryjnego XOR 0x555555. Istotne jest również, aby podczas wykorzystywania numeru seryjnego jako jednego z parametrów nie wykorzystywać interfejsów urządzenia do komunikacji ponieważ doprowadzi to do ujawnienia informacji o numerze seryjnym urządzenia. Niektóre urządzenia mogą również pochodzić z firm, które nie przywiązują wagi do unikatowości generowanych numerów seryjnych i innych parametrów. Spotkano się z tym problemem podczas wykonywania badań dla kart pamięci, pochodzących z popularnego chińskiego sklepu internetowego. Karty te miały identyczne numery seryjne

i numery CID. Dlatego badanie zostało powtórzone dla kart firmy Samsung.

Ponieważ entropia dla niektórych parametrów jest kilkakrotnie wyższa niż uznawana za wystarczająca do utworzenia trudno zgadywalnego klucza dlatego, niektóre parametry można podzielić na co najmniej dwie części. Przykładem może tu być wartość funkcji skrótu hasła, która może mieć nawet 512 bitów entropii. Liczba bitów entropii dla tego parametru zależy od wykorzystanego algorytmu do utworzenia skrótu hasła, ale także od entropii wykorzystywanego hasła. Operację podzielenia skrótu hasła można przeprowadzić poprzez wykorzystanie PRF, a następnie podzielenie wyniku na części o odpowiedniej długości analogicznie jak tworzony klucz sesji w protokole 802.11i [34].

Na podstawie otrzymanych parametrów w zaproponowanym protokole LAAFFI tworzone są klucze kryptograficzne. Minimalna liczba parametrów wykorzystywanych do utworzenia klucza zależy od entropii poszczególnych parametrów. Ważne jest, aby suma entropii parametrów wykorzystanych do utworzenia klucza była dwukrotnie większa niż oczekiwana entropia klucza kryptograficznego [186]. W proponowanym w rozprawie rozwiązaniu ostateczna wartość klucza uzyskiwana jest w wyniku wyznaczenia wartości funkcji skrótu dla konkatenacji wartości funkcji HMAC poszczególnych parametrów. Dowód na to, że HMAC jest funkcją PRF przedstawiono w artykule [21]. Niezdecydowano się na wykorzystanie funkcji KDF ponieważ jej wykorzystanie nie zwiększa entropii. Uznano też, że zwiększenie trudności przeprowadzenia ataków typu brute-force jest w przypadku urządzeń IoT dyskusyjne. Urządzenia IoT mają słabsze procesory, przez co wykonanie KDF jest czasochłonne, poza tym urządzenia muszą być energooszczędne, więc i same operacje wykonywane na urządzeniu nie mogą być energochłonne. Przy zabezpieczeniu komunikacji skupiono się na wykorzystaniu algorytmów uznanych za bezpieczne, a także na trudności "odgadnięcia"danych wejściowych, z których powstaje klucz (zwiększenie przestrzeni klucza). Kombinacja bezpiecznego algorytmu szyfrowania, wykorzystanie klucza o wystarczającej entropii, a także odpowiednie zarządzanie dostępem do parametrów przełoży się na zapewnienie poufności komunikacji pomiędzy urządzeniami IoT.

Przygotowane parametry konfiguracyjne mogą zależeć od urządzeń lub ich komponentów, które zostały zbudowane w tym samym okresie czasu i mają podobne numery seryjne, a także oprogramowania, które zostało zainstalowane w tym samym czasie na różnych urządzeniach. Dlatego należy zbadać, jak wartości parametrów pochodzących z różnych urządzeń, komponentów i oprogramowania różnią się między sobą. Podobny problem występuje w układach PUF, gdzie dla tych układów przyjęto wyliczać dwa parametry określające powtarzalność

i unikatowość odpowiedzi. Ze względu na podobne podejście w opracowanym protokole, postanowiono wyznaczyć unikatowość wartości parametrów w sposób zbliżony do oceny wykonanej dla układów PUF [125]. Badanie powtarzalności wartości parametrów uznano za zbędne, ponieważ parametry są niezmienne w czasie, a także czynniki środowiska nie są w stanie zmienić wartości parametrów. W celu zbadania unikatowości na podstawie zebranych wartości dla poszczególnych parametrów obliczono odległość Hamminga oraz wyznaczono wartość średnią μ_{inter} wartości parametrów. Wartości parametrów zostały odczytane z 20 urządzeń Raspberry Pi 4 i kart pamięci Samsung, które uzyskano z tej samej serii produkcyjnej. System operacyjny na każdej karcie pamięci był instalowany w ciągu kilku godzin, nazwa oraz hasło dla użytkownika, nazwa systemu była identyczna dla każdej instalacji. Aby obliczyć odległość Hamminga, każdą z wartości parametrów zamieniono na ciąg bitowy i dla ciągów bitowych została obliczona odległość Hamminga. Wartość μ_{inter} w najlepszym przypadku powinna być zbliżona do 0,5. Wartość średniej μ_{inter} została wyliczona korzystając ze wzoru [58],:

$$\mu_{inter} = \frac{1}{NL} * \sum_{i=1}^{M-1} \sum_{j=i+1}^{M} HD(x_i, x_j)$$

gdzie N – liczba wszystkich par wartości badanego parametrów, dla których obliczono odległość Hamminga, L – długość wartości parametru, M – liczba wartości parametrów, natomiast $\sum_{i=1}^{M-1} \sum_{j=i+1}^{M} HD(x_i, x_j)$ – suma wartości odległości Hamminga dla wszystkich par wartości badanego parametru. Liczba wszystkich par może być obliczona ze wzoru: $N = \frac{M*(M-1)}{2}$. Wartości μ_{inter} zostały przedstawione w tabeli 4.3.

Wartości μ_{inter} uzyskane podczas tego badania wskazują, które parametry najlepiej jest wykorzystać do tworzenia klucza. Należy zwrócić uwagę na niską wartość μ_{inter} numeru seryjnego karty pamięci oraz jej numeru CID, co spowodowane jest tym, że karty pochodziły z jednego zestawu i zdarzały się karty o numerach seryjnych następujących po sobie. Taka sytuacja nie była zaobserwowana dla numerów seryjnych urządzeń Raspberry Pi. Wartość μ_{inter} m.in. dla ziarna hasła jest zbliżona do 0,41, pomimo że wartość tego parametru jest generowana w sposób pseudolosowy. Spowodowane jest to prawdopodobnie tym, że ziarno hasła jest zapisywane przy użyciu zbioru symboli, a tym samym nie wszystkie bity reprezentujące symbol są wykorzystywane.

Tabela 4.3: Lista poleceń wraz z wyliczonym μ_{inter}

Lp.	**Parametr**	μ_{inter}
1	Numer seryjny urządzenia Raspberry pi	0,339
2	Identyfikator sprzętowy Raspberry pi	0,328
3	Numer seryjny karty pamięci	0,229
4	Wartość CID	0,235
5	PTUUID dla partycji	0,319
6	UUID systemu plików	0,309
7	UUID systemu plików dla FATBOOT	0,382
8	Ziarno hasła z /etc/shadow	0,403
9	Wartość funkcji skrótu z /etc/shadow	0,410
10	Ziarno dla funkcji skrótu katalogów (ext4)	0,320
11	Identyfikator systemu operacyjnego	0,323

4.2 Analiza możliwości przeprowadzenia ataków w środowisku federacyjnym IoT

Tworzenie nowego protokołu jest procesem podatnym na błędy. Dlatego niezbędne jest zweryfikowanie jego właściwości bezpieczeństwa przy użyciu różnych metod. Proces weryfikacji właściwości bezpieczeństwa protokołów jest powszechną praktyką i każdy protokół używany do komunikacji z urządzeniami powinien przejść przez ten proces. W związku z tym, że ataków, które można przeprowadzić jest bardzo dużo, powstają różne klasyfikacje, które starają się grupować ataki według różnych własności. Analizując literaturę dostrzeżono, że najczęściej ataki są klasyfikowane według warstw modelu IoT, w której dany atak można przeprowadzić [79, 134], a także według elementów, których te ataki dotyczą [159, 177]. Przykładowo w publikacji [177] autorzy wydzielili następujące elementy: urządzenie IoT, oprogramowanie, w ramach którego zalicza się sterowniki, system operacyjny oraz aplikacje, dane pochodzące z urządzeń IoT, protokoły, w których wydzielono protokoły komunikacyjne, protokoły połączeniowe i protokoły sieciowe. Dodatkowo oddzielną grupą są ataki kanałem bocznym i kryptoanaliza.

Ataki, które można przeprowadzić w sieciach federacyjnych IoT są tożsame z atakami, które można przeprowadzić w sieciach niefederacyjnych. Jednak przy analizie ataków w sieciach federacyjnych należy rozważyć atakującego, który należy do innej organizacji niż urzą-

dzenia lub usługa, która jest celem ataku. Atakujący nienależący do organizacji będącej właścicielem urządzenia lub usługi ma najprawdopodobniej większe uprawnienia niż podmiot spoza federacji, ale mniejsze niż członek organizacji będącej właścicielem urządzeń i usług. W tabeli 4.4 przedstawiono listę ataków potencjalnie możliwych do przeprowadzenia wobec opracowanego protokołu LAAFFI. Kryterium doboru ataków była możliwość naruszenia właściwości bezpieczeństwa informacji. Lista ataków powstała na podstawie analizy artykułów [79, 134, 157, 159, 177]. Niektóre z wymienionych w tabeli 4.4 ataków są uogólnionym zbiorem różnych ataków. Przykładem takiego ataku jest atak typu DoS, do którego można zaliczyć między innymi: ataki typu Flood, ataki fragmentacyjne, ataki odbite (ang. reflected attack).

Tabela 4.4: Lista ataków, które atakujący może próbować przeprowadzić przeciwko LAAFFI

<table>
<tr><th>Warstwa percepcji</th><th>Warstwa sieciowa</th><th>Warstwa aplikacyjna</th></tr>
<tr><td>Klonowanie urządzenia</td><td>Zagłuszanie urządzenia</td><td>Utrata danych</td></tr>
<tr><td>Wstrzyknięcie złośliwego kodu</td><td>Przejęcie sesji</td><td>Wstrzyknięcie złośliwego kodu</td></tr>
<tr><td colspan="3">DoS / Distributed Denial of Service (DDoS)</td></tr>
<tr><td>Atak powtórzenia</td><td colspan="2">Kryptoanaliza</td></tr>
<tr><td>Dodanie złośliwego urządzenia</td><td colspan="2">Atak typu man-in-the-middle</td></tr>
<tr><td>Fałszowanie tożsamości</td><td colspan="2">Ataki na wykorzystane protokoły i technologie</td></tr>
<tr><td>Podmiana urządzenia
lub jego komponentów</td><td>Wstrzykiwanie pakietów</td><td>Modyfikacja uprawnień
przez osoby nieupoważnione</td></tr>
<tr><td>Podszywanie się za urządzenie</td><td>Podsłuchiwanie transmisji</td><td>Atak polegający
na nadużywaniu władzy</td></tr>
<tr><td>Atak typu Sybil</td><td></td><td>Nieautoryzowany dostęp</td></tr>
<tr><td>Uszkodzenie urządzenia</td><td></td><td>Dodanie złośliwego węzła
rejestru rozproszonego
lub bramy aplikacyjnej</td></tr>
<tr><td>Atak kanałem bocznym</td><td></td><td>Ochrona danych przed
niepowołanym dostępem</td></tr>
</table>

Opracowany protokół działa na poziomie każdej z trzech warstw modelu systemu IoT: percepcji, sieciowej i aplikacyjnej.

W warstwie percepcji, która obejmuje urządzenia IoT, atakujący może próbować przepro-

wadzić następujące ataki na opracowany protokół:

- Atak powtórzenia (ang. replay attack) – Atak, który polega na przechwyceniu wiadomości i jej późniejszym wysłaniu. W przypadku opracowanego rozwiązania wykonanie tego ataku jest niemożliwe ze względu na to, że każda wiadomość zawiera znacznik czasu utworzenia, który jest weryfikowany. Urządzenie IoT bądź węzeł rejestru rozproszonego w momencie odszyfrowania wiadomości porównuje ten znacznik czasu z aktualnym czasem. Wadą tego rozwiązania jest posiadanie poprawnie zsynchronizowanego czasu na każdym urządzeniu IoT. Czas na urządzeniach IoT powinien być synchronizowany z serwerem czasu organizacji w sposób zapewniający uwierzytelnianie serwera czasu oraz poufność komunikacji. W przygotowanej implementacji czas, po jakim wiadomość zostaje odrzucona ze względu na to, że jest uznana za nieaktualną wynosi 5 sekund. Znacznik jest także porównywany ze znacznikiem otrzymanym w poprzedniej wiadomości. Jeżeli jest on identyczny lub mniejszy to wiadomość zostaje odrzucona. Znacznik czasu jest przesyłany wraz z danymi w postaci zaszyfrowanej z wykorzystaniem algorytmu AEAD.

- Atak kanałem bocznym (ang. side channel attack) – To atak, w którym obserwowana jest charakterystyka urządzenia [68] w celu uzyskania informacji o stosowanym algorytmie, kluczu lub danych niezaszyfrowanych. Atakujący analizuje czas wykonywania operacji, wykorzystanie energii czy też emisje elektromagnetyczne [95]. Należy podkreślić, że możliwość wykonania tego ataku zależy od budowy urządzenia IoT oraz wykorzystanych technologii. Z tego powodu zrezygnowano z badania możliwości przeprowadzenia ataku kanałem bocznym. Należy założyć, że jeżeli atakujący ma dostęp do urządzenia IoT to należy uznać, że urządzenie zostało skompromitowane i powinno być uznane za niezaufane.

- Podszywanie się za urządzenie (ang. spoofing attack) – Szeroki zbiór ataków, których celem jest podszycie się pod urządzenie, bramę aplikacyjną, węzeł rejestru rozproszonego. Ponieważ w przygotowanym protokole, każda wiadomość jest uwierzytelniona, nie ma możliwości przeprowadzenia ataków tego typu, dopóki atakujący nie uzyska dostępu do parametrów, z których może być utworzony klucz. Wzajemne uwierzytelnianie bramy aplikacyjnej z węzłem rejestru rozproszonego następuje z wykorzystaniem TLS w oparciu o certyfikaty obu stron. Dlatego ważne jest zabezpieczenie klucza prywatnego każdej bramy aplikacyjnej oraz węzła rejestru rozproszonego, aby atakujący nie uzyskał możliwości podszycia się za bramę aplikacyjną lub węzeł rejestru rozproszonego.

- Podmiana urządzenia lub jego komponentów – Atak polegający na podmianie urządzenia lub komponentów urządzenia. Klucz generowany w protokole LAAFFI do zaszyfrowania i uwierzytelnienia wiadomości jest generowany na podstawie odpowiedzi pozyskanych z układów PUF albo na bazie parametrów reprezentujących wybrane cechy sprzętowo-programowe urządzenia IoT. W obydwu przypadkach atak jest możliwy do wykrycia na podstawie sprawdzania liczby odrzuconych wiadomości na podstawie liczby niepowodzeń odszyfrowania wiadomości utworzonych przez podmienione urządzenie. Wykrycie tego ataku nie jest możliwe w przypadku gdy klucz powstaje z parametrów zapisanych na karcie pamięci.

- Dodanie złośliwego węzła – W przypadku tego ataku, atakujący dodaje swoje urządzenie do środowiska federacyjnego IoT. W przypadku opracowanego protokołu, bez rejestracji urządzenia nie istnieje możliwość komunikacji z innymi komponentami systemu. Natomiast, w sytuacji, gdy atakującemu udało się zarejestrować swoje złośliwe urządzenie, to urządzenie to może komunikować się i wykonywać operacje tylko z urządzeniami i usługami, do których będzie miało nadane uprawnienia.

- Atak typu Sybil – Atak polega na utworzeniu wielu tożsamości dla jednego węzła IoT. Dla opracowanego protokołu możliwe jest zarejestrowanie urządzenia IoT więcej niż raz. Urządzenie jest rejestrowane przez jedną organizację, dlatego ważne jest, aby każda organizacja dokładnie nadzorowała proces rejestracji. Podczas procesu rejestracji rekomendowana jest weryfikacja, czy dane urządzenie nie zostało już wcześniej zarejestrowane.

- Klonowanie urządzenia – Atak na urządzenie IoT polegający na sklonowaniu urządzenia. Sklonowane urządzenie zachowuje się jak prawdziwy węzeł, ale angażuje się w działania złośliwe i przeprowadza ataki typu wormhole, blackhole, ponieważ ma dostęp do poufnych danych. Aby przeprowadzić atak klonowania urządzenia wymagane jest uzyskanie danych uwierzytelniających innego urządzenia IoT. W przypadku LAAFFI atak jest możliwy jeżeli atakujący przejmie kontrolę nad innym urządzeniem z uprawnieniami umożliwiającymi na uzyskanie wszystkich wartości parametrów potrzebnych do utworzenia kluczy. Jeżeli LAAFFI wykorzystuje parametry sprzętowo–programowy urządzenia lub PUF, sklonowanie karty pamięci jest niewystarczające do przeprowadzenia tego ataku.

- Fałszowanie tożsamości – Atak polega na wygenerowaniu fałszywej tożsamości i wyko-

rzystaniu jej w środowisku IoT. Protokół LAAFFI do uwierzytelniania wymaga rejestracji urządzenia w rejestrze rozproszonym, a także rejestr rozproszony jest wymagany do uwierzytelniania we wszystkich przewidzianych operacjach, więc nikt nie może wykorzystać fałszywej tożsamości. Rejestracja urządzenia też następuje w określonych sytuacjach i jest dokonywana przez uprawnione osoby.

- Wstrzyknięcie złośliwego kodu – Powodzenie przeprowadzenia tego ataku w przypadku LAAFFI wymaga, aby aplikacja użytkownika była podatna na tego typu atak. Jeżeli aplikacja użytkownika ma poprawnie zaimplementowany mechanizm walidacji otrzymywanych danych, to przeprowadzenie tego ataku nie powinno być możliwe.

- Atak typu DoS/DDoS (ang. denial-of-service attack/distributed denial-of-service attack) – Celem ataku jet wykorzystanie dostępnego zasobu, najczęściej tym zasobem jest przepustowość sieci, wydajność procesora, rzadziej miejsca na dysku. Urządzenia IoT są podatne na tego typu ataki, ze względu na charakterystykę sprzętową. Aby przeprowadzić skutecznie atak DoS/DDoS wymagane jest wysyłanie do urządzenia odpowiednio dużej liczby wiadomości. Liczba wiadomości jest zależna od konfiguracji sprzętowej urządzenia IoT.

- Uszkodzenie urządzenia – Atak polega na uszkodzeniu urządzenia IoT i tym samym uniemożliwieniu lub ograniczeniu możliwości świadczenia usług przez to urządzenie. Opracowany protokół umożliwia szybkie wdrożenie większej liczby urządzeń IoT, co przełoży się na ograniczenie skutków uszkodzenia pojedynczych urządzeń IoT.

W warstwie sieciowej atakujący może spróbować wykonać ataki:

- Podsłuchiwanie transmisji – W przypadku LAAFFI przesyłane dane są zaszyfrowane. Jedynymi danymi, które są przesyłane w formie jawnej są: identyfikator urządzenia, wykorzystane numery parametrów oraz wartość nonce. Dane te nie dają możliwości utworzenia klucza wykorzystanego do zabezpieczenia przesyłanych danych.

- Atak typu man-in-the-middle – W tym ataku atakujący podsłuchuje i modyfikuje wiadomość przesyłaną pomiędzy stronami. Wiadomość w opracowanym protokole składa się z elementów, których modyfikacja jest łatwa do wykrycia. Identyfikator, numery parametrów oraz wartość nonce są niezbędne do utworzenia klucza i odszyfrowania wiadomości. Zmiana dowolnego elementu uniemożliwi utworzenie poprawnego klucza

i odszyfrowanie przesyłanego szyfrogramu. Modyfikacja szyfrogramu uniemożliwi poprawne odszyfrowanie, co może być łatwo wykryte poprzez wykorzystanie algorytmu szyfrującego wykorzystującego AEAD.

- Wstrzykiwanie pakietów – Atak polega na wstrzykiwaniu pakietów w celu zakłócenia komunikacji. W przypadku LAAFFI atak może polegać na duplikowaniu przesyłanych pakietów lub tworzeniu niepoprawnych pakietów. W obydwu przypadkach zostaną one zignorowane przez odbiorcę.

- Kryptoanaliza – Zbiór ataków, których celem jest uzyskanie tekstu jawnego przesyłanej wiadomości. Przykładem tego ataku są: atak brute-force, atak słownikowy, atak statystyczny. Aby utrudnić uzyskanie jawnego tekstu wiadomości przesyłanej pomiędzy urządzeniem IoT a bramą aplikacyjną lub innym urządzeniem IoT zaleca się wykorzystać algorytmy szyfrujące i HMAC uważane za bezpieczne. Parametry wykorzystane do utworzenia klucza muszą też mieć odpowiednią ilość entropii, tak aby atakujący nie mógł odgadnąć klucza wykorzystanego do zabezpieczenia wiadomości.

- Przejęcie sesji – Atak polega na przejęciu tokena sesji utworzonej pomiędzy urządzeniem a bramą aplikacyjną albo pomiędzy dwoma urządzeniami.LAAFFI nie korzysta z sesji, więc nie ma możliwości przejęcia sesji. Może się jednak zdarzyć sytuacja, że atakujący przejmie klucz do komunikacji pomiędzy dwoma urządzeniami utworzony przez węzeł rejestru rozproszonego. W takiej sytuacji atakujący ma możliwość odszyfrowania komunikacji pomiędzy tymi urządzeniami, oraz może podszywać się za dowolne urządzenie w tej komunikacji. Aby temu zapobiec istotna jest ochrona klucza wykorzystywanego do tej komunikacji.

- Atak typu DoS/DDoS (ang. denial-of-service attack/distributed denial-of-service attack) – W przypadku warstwy sieciowej, atak może polegać na zagłuszaniu urządzeń prowadzących komunikację. Atakujący nadaje sygnał wykorzystując te same częstotliwości, co urządzenia komunikujące się ze sobą. Tak samo jak w przypadku ataku DoS/DDoS w warstwie percepcji, sposób ochrony stron komunikacji przed tego typu atakiem jest bardzo ograniczony.

- Atak na wykorzystywane protokoły i technologie – Protokół LAAFFI nie wymaga do poprawnego działania określonych protokołów sieciowych. Dlatego też można wybrać

jakie protokoły sieciowe będą odpowiednie do wykorzystania w zakładanym scenariuszu oraz wykorzystać mechanizmy, które będą niwelowały możliwość przeprowadzenia skutecznych ataków na wykorzystany protokół.

W warstwach aplikacyjnej i przetwarzania danych, które obejmują rejestr rozproszony oraz bramę aplikacyjną atakujący może próbować wykonać następujące ataki:

- Atak typu DoS/DDoS (ang. denial-of-service attack/distributed denial-of-service attack) – Ze względu na to, że LAAFFI wykorzystuje rejestr rozproszony jest podatny na ten atak, jeżeli wykonywane jest zbyt dużo operacji wymagających dodanie danych do rejestru np. rejestracja urządzeń, zmiana uprawnień. W innych przypadkach nie ma ograniczeń skalowania węzłów rejestru rozproszonego. Dodanie kolejnych węzłów pozwala na obsługę większej liczby operacji w ciągu sekundy, a tym samym wymusza na atakującym zwiększenie liczby operacji, aby przeprowadzić ten atak z sukcesem. Ochrona przed atakami tego typu jest niezmiernie trudna, ale brama aplikacyjna może mieć możliwość odrzucania wiadomości, które są duplikatami lub mają niepoprawny format. Taki proces odrzucania może ograniczyć liczbę wiadomości, która musi zostać obsłużona przez węzły rejestru rozproszonego. Dodatkowo można próbować ochronić organizację przed atakiem DoS/DDoS wykorzystując usługi ochrony przed tego typu atakami oferowanymi przez operatorów telekomunikacyjnych, lub Web Application Firewall (WAF).

- Ochrona danych przed niepowołanym dostępem – W przypadku tego ataku ważne jest, aby uniemożliwić dostęp do danych osobom niepowołanym. Z tego też powodu LAAFFI wymusza wykorzystanie prywatnego rejestru rozproszonego z węzłami uprawnionymi. Implementacja LAAFFI wykorzystuje Hyperledger Fabric, który umożliwia stworzenie prywatnych kanałów, a tym samym ogranicza dostępność do danych zapisanych w rejestrze rozproszonym z zachowaniem konieczności przestrzegania warunków zapisanych w kodach łańcuchowych.

- Modyfikacja uprawnień przez osoby nieupoważnione – W protokole LAAFFI modyfikacja uprawnień jest możliwa tylko poprzez kody łańcuchowe, a tym samym każda modyfikacja wymaga spełniania warunków zapisanych w kodzie łańcuchowym. Jeżeli kod łańcuchowy jest poprawnie napisany, to tylko właściciel urządzenia lub podmiot uprawniony może modyfikować uprawnienia urządzenia IoT.

- Atak polegający na nadużywaniu władzy (ang. abuse of authority) – W przypadku

LAAFFI zminimalizowano możliwość powodzenia ataku nadużywania władzy poprzez wykorzystanie kodów łańcuchowych, których zadaniem jest weryfikowanie każdej operacji wykonywanej na danych zapisanych w rejestrze rozproszonym. Z tego też powodu każda organizacja ma m.in. możliwość modyfikacji uprawnień do swoich urządzeń IoT, ale nie może to być wykonane przez podmioty nieuprawnione.

- Dodanie złośliwego węzła rejestru rozproszonego lub bramy aplikacyjnej – W przypadku przygotowanej implementacji protokołu LAAFFI, w której wykorzystano Hyperledger Fabric uwierzytelnianie węzłów i bram aplikacyjnych opiera się na PKI. Tylko podmiot posiadający certyfikat i odpowiednią parę kluczy ma możliwość utworzyć węzeł rejestru rozproszonego i bramy aplikacyjnej.

- Ataki na wykorzystane technologie – Mogą to być ataki wykorzystujące podatności w językach oprogramowania [2], czy też w samym oprogramowaniu [3]. Przed tymi atakami można się ochronić aktualizując wykorzystywane oprogramowanie. Ataki mogą być ukierunkowane również na konkretne rozwiązanie w oprogramowaniu i wykorzystywać wady tego oprogramowania. W przypadku opracowanej implementacji protokołu LAAFFI atakujący może próbować wykorzystywać wady Hyperledger Fabric. Atakiem, przed którym trudno jest się ochronić, jest atak typu DDoS. Utrudnieniem wykonania ataku DDoS jest zwiększenie liczby węzłów, ich parametrów oraz przepustowości sieci. Kolejnym atakiem, który można wykonać jest atak typu wormhole [9] przez członków organizacji budujących kanał. Jeżeli pojedynczy członek kanału działa złośliwie możliwe jest ujawnienie wszystkich informacji przechowywanych w kanale. Mechanizm kontroli dostępu w Hyperledger Fabric ściśle bazuje na zaufaniu do organizacji, które tworzą rejestr w kanale. Przed tym atakiem, możemy się ochronić poprzez przechowywanie zaszyfrowanych danych lub przynajmniej anonimizacją danych, jeżeli jest to wystarczające. Ewentualnie można wykorzystać dane prywatne, gdzie dane są przechowywane poza rejestrem rozproszonym, a w rejestrze przechowywany jest tylko wartość funkcji skrótu z tych danych. Poza tymi atakami istnieją także problemy bezpieczeństwa. Najistotniejszym z nich jest ochrona MSP przed wyciekiem klucza prywatnego urzędu certyfikacyjnego oraz administratorów. Jeżeli atakujący przejmie kontrolę nad MSP danej

[2] https://www.cvedetails.com/vulnerability-list/vendor_id-14185/Golang.html
[3] https://www.cvedetails.com/vulnerability-list/vendor_id-18415/product_id-117539/year-2022/Hyperledger-Fabric.html

organizacji lub uzyska uprawnienia administratora, to będzie mógł w imieniu tej organizacji wykonać dowolną operację, m.in. utworzyć nowe węzły, zgłaszać zmiany kodów łańcuchowych i konfiguracji systemowej Hyperledger Fabric. Kompromitacja jednej organizacji może doprowadzić do kompromitacji całego rejestru rozproszonego. Kolejnym problemem są możliwe błędy w kodach łańcuchowych. Kod łańcuchowy w Hyperledger Fabric uruchamiany jest w piaskownicy, co utrudnia wykorzystanie luk bezpieczeństwa ze względu na to, że proces jest odizolowany oraz nie ma dostępu do kontenera z poziomu sieci. Jednak w momencie przejęcia kontroli nad węzłem na którym jest uruchomiony kod łańcuchowy, atakujący ma otwartą drogę do analizy oraz wykorzystania znalezionych podatności. Podatności mogą mieć różny skutek, mogą one doprowadzić, m.in. do udostępnienia informacji podmiotowi nieuprawnionemu. Aby uniknąć błędów podczas pisania kodów łańcuchowych należy kody łańcuchowe gruntownie przetestować przed wdrożeniem. Można do tego wykorzystać statyczną i dynamiczną analizę [113, 160] czy też fuzzery [48]. W kontekście bezpieczeństwa rejestrów rozproszonych należy także prowadzić analizę zdarzeń oraz utworzyć alerty jeżeli zostaną wykryte próby atakowania węzłów rejestru rozproszonego [148].

- Wstrzyknięcie złośliwego kodu – Aby zapobiec tego typu atakowi, należy wdrożyć walidację danych w kodzie bramy aplikacyjnej oraz w kodzie łańcuchowym. Za utrzymanie oraz aktualizację bramy aplikacyjnej odpowiedzialny jest każdy z członków federacji. W związku z tym każdy z nich musi wdrożyć taki mechanizm. Powinna istnieć także możliwość przeprowadzenia testów bezpieczeństwa bram aplikacyjnych każdej z organizacji przez pozostałe organizacje. W przypadku kodów łańcuchowych wymagana jest zgoda wszystkich członków, aby kod łańcuchowy został wdrożony. Dlatego przed zaakceptowaniem nowej wersji kodu łańcuchowego należy przeprowadzić testy bezpieczeństwa nowej wersji kodu łańcuchowego.

- Nieautoryzowany dostęp – LAAFFI wykorzystuje Hyperledger Fabric, który wymaga, aby każdy użytkownik musiał uwierzytelnić się z wykorzystaniem PKI. Aby zapobiec nieautoryzowanemu dostępowi, należy wydawać certyfikaty tylko dla osób upoważnionych. Ważne jest także odpowiednie zarządzanie dostępami osób, tak aby miały one dostęp tylko do wymaganych typów danych oraz funkcji.

- Utrata danych – LAAFFI korzysta z rejestru rozproszonego do przechowywania danych,

w związku z tym każdy węzeł rejestru posiada kopię danych. Utrata jednego węzła nie powoduje niedostępności danych. Danych zapisanych do rejestru nie można także usunąć bądź nadpisać.

4.3 Formalna weryfikacja modeli bezpieczeństwa protokołu

Podczas tworzenia protokołów komunikacyjnych mogą wystąpić błędy, które mogą zostać nieznalezione na etapie projektowania, a które po wdrożeniu protokołu mogą doprowadzić do ujawnienia informacji przesyłanej pomiędzy stronami. Z tego też powodu ważne jest posiadanie dowodu potwierdzającego spełnienie cech bezpieczeństwa informacji przez opracowywany protokół. W celu ograniczenia ryzyka wystąpienia błędu można zastosować weryfikację formalną. Weryfikacja formalna potwierdza spełnienie założonych właściwości bezpieczeństwa, które są wymagane w konkretnym środowisku operacyjnym. Do cech bezpieczeństwa opracowanego protokołu, które należy potwierdzić, aby protokół mógł być uznany za użyteczny w rzeczywistym środowisku należą m.in.: poufność, uwierzytelnienie stron oraz odporność na atak powtórzeniowy.

Obecnie najpopularniejsze są dwa typy formalnych modeli analizy bezpieczeństwa: model symboliczny i obliczeniowy. Model symboliczny [26] lub model "Dolev-Yao"charakteryzuje się tym, że prymitywy kryptograficzne są uznawane za czarne skrzynki tzn. uznawane są za elementy, które nie są poddawane analizie, nie można ich rozbić na etapy oraz uznaje się, że nie zawierają błędów. Kolejną cechą charakterystyczną dla tego modelu jest budowa wiadomości przesyłanych między stronami w oparciu o prymitywy (szyfrowanie/deszyfrowanie, obliczanie funkcji skrótu, generowanie wartości losowej itd.). Atakujący ma ograniczenia w postaci wykonywania działań tylko przy użyciu tych prymitywów, zna wszystkie publiczne informacje o protokole, może rozpocząć dowolną liczbę sesji protokołu, może tworzyć, czytać i modyfikować każdą przesyłaną wiadomość między stronami, może także jeżeli zna klucz szyfrować i deszyfrować wiadomości. W tym typie modelu nie można nadawać zmiennym wartości algebraicznych lub numerycznych. Model symboliczny jest prosty do zautomatyzowania weryfikacji cech bezpieczeństwa. Obecnie istnieją narzędzia realizujące weryfikację w modelu symbolicznym w sposób automatyczny: Proverif [25], Scyther [38], Tamarin [181], Verifpal [104].

W modelowaniu protokołu komunikacji problemem jest nieskończona liczba stanów protokołu, co doprowadza do nierozstrzygnięcia rezultatu weryfikacji protokołu. Wpływ na to

ma możliwość wystąpienia wielu jednoczesnych sesji protokołu, czy też same wiadomości tworzone przez atakującego mogą mieć nieskończony rozmiar. Aby nie dopuścić do takiej sytuacji, stosuje się ograniczenia liczby stanów, ograniczenia liczby możliwych jednoczesnych sesji, stosowania uproszczeń w modelowaniu lub także wymagana jest interakcja użytkownika.

Drugim typem modelu jest model obliczeniowy [26]. Charakteryzuje się on tym, że wiadomości są ciągami bitów, prymitywy kryptograficzne są funkcjami przyjmującymi i zwracającymi dane jako ciągi bitów. Atakujący w tym modelu jest probabilistyczną maszyną Turinga, z założeniami bezpieczeństwa określającymi czego atakujący nie może robić. Ten typ modelu też nie jest w stanie wykryć wszystkich możliwości przeprowadzenia ataków, np. ataków kanałem bocznym, które mogą dostarczyć nowych informacji. Model obliczeniowy jest znacznie bardziej realistyczny od modelu symbolicznego, ale nie w każdym przypadku umożliwia dowiedzenie cech bezpieczeństwa informacji. Przykładem narzędzi pozwalających zweryfikować model obliczeniowy są: EasyCrypt [17] oraz CryptoVerif [24].

Żaden z modeli nie jest w stanie wykryć błędów implementacji protokołu. Istnieją co prawda narzędzia potrafiące wygenerować model symboliczny z kodu programu, ale mają one ograniczenia m.in. w postaci użytego języka programowania [3, 94]. Tego typu narzędzia nie są dopracowane, ale problem ten jest cały czas poruszany w wielu pracach badawczych [3, 13].

Model atakującego obejmuje dwa typy atakujących: wewnętrzni (członkowie organizacji oraz atakujący, który uzyskał kontrolę nad przynajmniej jednym z komponentów organizacji np. urządzeniem IoT) oraz zewnętrzni. Zewnętrzny atakujący może przechwycić komunikację pomiędzy urządzeniem IoT a bramą aplikacyjną lub innym urządzeniem IoT i uzyskać informacje na temat identyfikatora urządzenia, indeksów używanych parametrów, wartości nonce i szyfrogramu. Zewnętrzny atakujący może przechwycić taką komunikację, oraz wysłać duplikat do jednej ze stron tej komunikacji, lub też innego urządzenia IoT. Duplikat będzie zignorowany przez strony komunikacji, natomiast inne urządzenia IoT nie będą w stanie utworzyć poprawnego klucza do odczytania wiadomości. W związku z tym wiadomość będzie odrzucona. Zewnętrzny atakujący może również próbować odszyfrować przesyłany szyfrogram, ale jeżeli zostaną zachowane odpowiednie metody bezpieczeństwa m.in. zastosowanie algorytmów kryptograficznych uznanych za bezpieczne, utworzenie klucza o wysokiej entropii, to atakujący może próbować wykorzystać tylko atak typu brute force. Przeprowadzenie z sukcesem tego ataku będzie co najmniej bardzo czasochłonne.

W przypadku atakującego operującego z wewnątrz organizacji należy wziąć pod uwagę jego uprawnienia, jeżeli atakujący nie ma uprawnień administracyjnych w danym urządzeniu

IoT, to nie może odczytać tablicy parametrów urządzenia IoT, nie ma dostępu do danych przechowywanych na urządzeniu. W zależności od posiadanych uprawnień może komunikować się z innymi komponentami sieci IoT. Ważne jest w takim przypadku posiadanie odpowiedniego systemu wychwytującego podejrzane próby uwierzytelnienia. System SIEM może pomóc w wykrywaniu przejętych urządzeń i nieuczciwych administratorów, analizując działania podejmowane przez urządzenie i identyfikując odchylenia od zwykłego wzorca zachowania. W przypadku wykrycia podejrzanego ruchu urządzenie powinno być odizolowane od systemu. Taka izolacja powinna być zrealizowana poprzez blokadę komunikacji urządzenia IoT z bramą aplikacyjną każdej organizacji. W przypadku atakującego, który uzyska uprawnienia administracyjne, może on uzyskać klucz do komunikacji z urządzeniem IoT należącym do organizacji, w której ma on uprawnienia administratora. Dlatego ważne jest, aby monitorować wszystkie próby uzyskania kluczy dla urządzeń IoT z wykorzystaniem np. systemu SIEM.

W ramach sprawdzenia spełnienia przez protokół LAAFFI cech bezpieczeństwa weryfikacji poddano wszystkie trzy operacje tj. rejestrację urządzenia IoT, komunikację urządzenia IoT z węzłem rejestru rozproszonego, podczas której wykonywane jest zadanie zlecone przez urządzenie IoT oraz komunikację pomiędzy urządzeniami IoT. Do weryfikacji wykorzystano dwa narzędzia: Verifpal oraz Tamarin. Wybór dwóch narzędzi różniących się metodą weryfikacji pozwolił na uzyskanie większej pewności rezultatów.

4.3.1 Weryfikacja z wykorzystaniem Verifpal

Verifpal jest jednym z narzędzi umożliwiających modelowanie współczesnych protokołów komunikacyjnych i weryfikację spełnienia cech bezpieczeństwa. Cechą charakterystyczną tego narzędzia, która jest podkreślana przez autora jest prosta, intuicyjna składania umożliwiająca łatwiejsze tworzenie modelu dla osób niezaznajomionych z zagadnieniami modelowania protokołów komunikacyjnych. Verifpal jest narzędziem weryfikującym modele symboliczne w sposób automatyczny. Użytkownicy mogą korzystać tylko ze zdefiniowanych prymitywów (np. szyfrowanie symetryczne/asymetryczne, funkcja skrótu, podpisywanie wiadomości i wiele innych). Verifpal oferuje także zbiór cech bezpieczeństwa, które mogą być zweryfikowane: tajność przesyłanych danych, uwierzytelnienie stron komunikacji, możliwość przeprowadzenia ataku powtórzenia, oraz brak możliwości powiązania (ang. unlinkability). Został wykorzystany do zbadania bezpieczeństwa wielu protokołów m.in.: Signal, TLS, ProtonMail [105]. Wyniki analizy tych protokołów przeprowadzonych przy pomocy Verifpal pokrywały się z analizą

przeprowadzoną przez inne narzędzia.

Weryfikacja modelu polega na wykonaniu pięciu faz [104]. W pierwszej fazie atakujący biernie obserwuje protokół i poznaje wszystkie wartości udostępniane publicznie między stronami komunikacji. W drugiej fazie budowany jest stan atakującego zawierający wszystkie poznane wartości. W fazie trzeciej wykonywane są transformacje na wartościach zapisanych w stanie atakującego. W fazie czwartej Verifpal przygotowuje tablicę mutacji, która zawiera wszystkie możliwe podstawienia wartości ze stanu atakującego. W fazie piątej przeprowadzane są próby wykonania protokołu dla wszystkich kombinacji w tablicy mutacji. Następnie następuje powrót do stanu pierwszego. Verifpal wykonuje ten proces tak długo, jak długo atakujący uczy się nowych wartości. Po każdej fazie Verifpal sprawdza, czy znalazł sprzeczność z którąkolwiek z cech bezpieczeństwa określonych w modelu i informuje użytkownika, jeśli taka sprzeczność zostanie znaleziona.

Aby osiągnąć ograniczoną przestrzeń stanów, Verifpal stosuje techniki heurystyczne polegające na podziale analizy na etapy, w których Verifpal stopniowo pozwala na modyfikację większej liczby zmiennych, ograniczeniu liczby wejść i wyjść do każdego prymitywu. Verifpal w celu zwiększenia wydajności weryfikacji działa wielowątkowo. Niestety, w przypadku Verifpal nie jest formalnie udowodnione, że nie pominie on ataku [104]. Autorzy uzasadniają to potrzebą zastosowania pewnego kompromisu, którego celem jest ograniczenie czasu analizy kosztem ewentualnej utraty wiarygodności rezultatu w odniesieniu do mało prawdopodobnych ataków dla protokołów stosowanych w rzeczywistych konstrukcjach.

4.3.2 Rejestracja urządzenia IoT w rejestrze rozproszonym

W każdym z modeli budowanych dla Verifpal należy zdefiniować atakującego działającego w sposób aktywny, czyli takiego, który ma możliwość modyfikacji danych przesyłanych pomiędzy uczestnikami komunikacji.

Zgodnie z przedstawionym w rozdz. 3 opisem protokołu podczas rejestracji urządzenia A, generuje ono tablicę parametrów *paramtable* oraz wartość nonce $nonceAH$, posiada zapisany jednorazowy klucz OTP oraz identyfikator ITD. W rzeczywistym systemie poza tablicą parametrów przesyłany jest także znacznik czasu. Jednakże ze względu, że Verifpal nie ma możliwość operowania na znacznikach czasu, uznano, że tworzenie i przesyłanie znacznika czasu jest bezcelowe. Także tablica parametrów jest oznaczona jako wartość losowa, ponieważ w modelu symbolicznym nie operuje się na tablicach i listach. Tablica parametrów *paramtable*

jest szyfrowana z wykorzystaniem algorytmu AEAD przy użyciu tymczasowego klucza *OTP* oraz wartości jednorazowej *nonceAH*. Do węzła rejestru rozproszonego przesyłane są: tymczasowy identyfikator *ITD*, *nonceAH* oraz szyfrogram *E_DataAH*. Fragment modelu realizujący te działania przedstawiono na listingu 4.1.

```
attacker[active]
principal DeviceA[
  generates paramtable  //Generuje tablice parametrow
  generates nonceAH //Generuje nonce do wyslania danych
  knows private OTP //Posiada tymczasowy klucz
  knows private TID //Posiada tymczasowe ID
  E_DataAH = AEAD_ENC(OTP, paramtable, nonceAH) //Szyfruje tablice parametrow
]
DeviceA -> HLF: TID, nonceAH, E_DataAH // Przesylanie do HLF
```

Listing 4.1: Fragment modelu odpowiadającego za opis działań wykonywanych przez urządzenie A w procesie rejestracji urządzenia

Węzeł rejestru rozproszonego również posiada tymczasowy identyfikator urządzenia *TID* oraz jednorazowy klucz *OTP*. Przy użyciu *nonceAH* i *OTP* odszyfrowuje otrzymany szyfrogram *E_DataAH*. Otrzymana odszyfrowana wartość to tablica parametrów *D_DataAH*. Węzeł rejestru rozproszonego generuje stały identyfikator urządzenia *ID*. Tablica parametrów oraz *ID* w rzeczywistym systemie podczas wykonywania rejestracji urządzenia są zapisywane do rejestru rozproszonego. Generowany jest klucz *key*, który w rzeczywistym systemie pochodzi z podzbioru parametrów określonych przez *param_numbers*. Ponieważ w modelu Verifpal nie jest możliwe przedstawienie tego działania, dlatego dalej przyjęto założenie upraszczające, że klucz *key* jest znany obu stronom komunikacji. Identyfikator *ID* jest szyfrowany z wykorzystaniem klucza *key* oraz wygenerowanej wartości nonce *nonceHA*. Wynikiem szyfrowania jest szyfrogram *E_DataHA*. Do urządzenia *A* przesyłany jest wartość nonce *nonceHA* oraz szyfrogram *E_DataHA*. Zamodelowane działania przedstawiono w formie listingu 4.2.

W dalszych opisach modeli na oprogramowania Verifpal nie zawarto już informacji o konieczności weryfikacji znaczników czasu w przesłanych w wiadomości oraz uznano, że klucz jest znany obydwu stronom komunikacji. Uproszczenia te wprowadzono ze względu, że Verifpal nie ma możliwości porównywania znaczników czasu oraz operowania na tabelach.

```
principal HLF[
  knows private OTP    //Posiada tymczasowy klucz urzadzenia
  knows private TID    //Posiada tymczasowy ID urzadzenia
  D_DataAH = AEAD_DEC(OTP,E_DataAH,nonceAH) //Deszyfruje dane, D_DataAH to
      paramtables
  generates ID  //Generuje ID dla urzadzenia
  knows private key //Tworzy klucz do komunikacji na podstawie paramtables i
      paramnumbers (nie ma mozliwosci potraktowania Paramtables jako tablicy/
      listy i wyboru konkretnych elementow z niej)
  generates nonceHA    //Generowanie nonce
  E_DataHA = AEAD_ENC(key,ID,nonceHA)    //Szyfrowanie ID
]
HLF -> DeviceA: nonceHA, E_DataHA // Przesylanie do urzadzenia IoT
```

Listing 4.2: Fragment modelu odpowiadającego za opis działań wykonywanych przez węzeł rejestru rozproszonego podczas rejestracji urządzenia A

Urządzenie *A* po otrzymaniu odpowiedzi od węzła odszyfrowuje otrzymany szyfrogram *E_DataHA* przy użyciu klucza *key*, który jest znany urządzeniu oraz nonce *nonceHA* przesłanego w wiadomości. Odszyfrowany identyfikator *DataID* jest uznany za stały identyfikator urządzania, a samo urządzenie za zarejestrowane. Fragment modelu realizujący te działania przedstawiono w listingu 4.3.

```
    principal DeviceA[
  knows private key //Zna klucz, ktory jest uzyty do zaszyfrowania wiadomosci
      (nie ma mozliwosci potraktowania paramtables jako tablicy/listy i wyboru
      konkrentych elementow z niej)
  D_DataHA = AEAD_DEC(key,E_DataHA,nonceHA)    //Odszyfrowanie danych -
      odszyfrowany identyfikator ID
]
```

Listing 4.3: Fragment modelu odpowiadającego za opis działań wykonywanych przez urządzenie A po odebraniu odpowiedzi w procesie rejestracji

Na końcu modelu rejestracji urządzenia znajduje się lista zapytań, które są weryfikowane w ramach tego modelu. Weryfikacji podlega poufność przesyłanej tablicy parametrów (*paramtable*) oraz identyfikatora nadanego urządzeniu (*ID*). Zweryfikowane zostało także uwierzytelnienie komunikacji, a także możliwość rozróżnienia komunikatów pomiędzy przebiegami protokołu. Wszystkie weryfikowane cechy bezpieczeństwa zostały spełnione. Lista weryfikowanych zapytań została przedstawiona w listingu 4.4.

```
queries[
  confidentiality? paramtable
  confidentiality? ID
  authentication? DeviceA -> HLF: E_DataAH
  authentication? HLF -> DeviceA: E_DataHA
  freshness? E_DataAH
  freshness? E_DataHA
]
```

Listing 4.4: Lista zapytań weryfikowana przez Verifpal w procesie rejestracji urządzenia

4.3.3 Komunikacja urządzenia IoT z rejestrem rozproszonym

W tym modelu również został zdefiniowany aktywny atakujący. Urządzenie *A* „wie" jakie dane *dataA* ma wysłać do rejestru rozproszonego. Następuje generowanie wartości nonce *nonceAH*. Dane *dataA* są szyfrowane z wykorzystaniem klucza *keyAH* oraz *nonceAH* wykorzystując algorytm AEAD. Do węzła rejestru rozproszonego jest wysyłana wartość nonce *nonceAH* a także szyfrogram *E_ DataAH*. Na listingu 4.5 przedstawiono fragment modelu przedstawiający działania wykonywane przez urządzanie IoT.

```
attacker[active]
principal DeviceA[
  knows private dataA    //Posiada dane, ktore chce wyslac
  generates nonceAH    //Generowanie nonce
  knows private keyAH    //Tworzy klucz (Tak samo jak w procesie rejestracji)
  E_DataAH = AEAD_ENC(keyAH, dataA, nonceAH)   //Szyfrowanie danych
]
DeviceA -> HLF:   nonceAH, E_DataAH //Przesylanie do rejestru rozproszonego
```

Listing 4.5: Fragment modelu odpowiadającego za opis działań podejmowanych przez urządzenie A wysyłającego dane do rejestru rozproszonego

Po otrzymaniu danych przez węzeł dokonuje on odszyfrowania szyfrogramu *E_ DataAH* wykorzystując klucz *keyAH* (odtworzony na podstawie wartości określonych parametrów zapisanych w tablicy podczas rejestracji urządzenia) oraz przesłaną wartość *nonceAH*. W celu wysłania odpowiedzi *reply*, węzeł generuje nowy klucz *keyHA* oraz wartość nonce *nonceHA*. Odpowiedź *reply* jest szyfrowana przy pomocy algorytmu AEAD z wykorzystaniem utworzonego klucza *keyHA* oraz nonce *nonceHA*. Do urządzenia *A* przesyłany jest szyfrogram *E_ DataHA*,

i wartość nonce *nonceHA*. Model realizujący ten fragment protokołu został przedstawiony na listingu 4.6.

```
principal HLF[
  knows private keyAH      // Tworzy klucz do odszyfrowania
  D_DataAH = AEAD_DEC(keyAH,E_DataAH,nonceAH) //Odszyfrowuje dane
  generates reply    //Wykonuje operacje na odebranych danych i generuje
      odpowiedz
  generates nonceHA    //Generowanie nonce
  knows private keyHA    //Tworzy klucz do komunikacji z urzadzeniem A
  E_DataHA = AEAD_ENC(keyHA,reply,nonceHA)   //Szyfruje dane
]
HLF -> DeviceA: nonceHA,E_DataHA //przesyla dane do urzadzenia A
```

Listing 4.6: Fragment modelu odpowiadającego za opis działań wykonywanych przez rejestr rozproszony po odebraniu danych od urządzenia A

Urządzenie *A* zna klucz *keyHA* oraz przy pomocy tego klucza i otrzymanej wartości *nonceHA* odszyfrowuje szyfrogram *E_DataHA*. Uzyskuje w ten sposób odpowiedź od węzła rejestru rozproszonego. Na listingu 4.7 przedstawiono fragment modelu odpowiadający za czynności wykonywane przez urządzenie *A*.

```
principal DeviceA[
  knows private keyHA //Tworzy klucz do odszyfrowania danych - na podstawie
      numerow parametrow i tablicy parametrow
  D_DataHA = AEAD_DEC(keyHA,E_DataHA,nonceHA)    //Odszyfrowuje odpowiedz
]
```

Listing 4.7: Fragment modelu odpowiadającego za opis działań wykonywanych przez urządzenie A po odebraniu danych od rejestru rozproszonego

W tym modelu również zweryfikowano poufność przesłanych danych, uwierzytelnienie stron komunikacji oraz możliwość rozróżnienia komunikatów przesłanych w różnych przebiegach protokołu. Zapytania weryfikowane przez Verifpal dla tego modelu zostały przedstawione na listingu 4.8. Wszystkie weryfikowane cechy bezpieczeństwa zostały zweryfikowane z wynikiem pozytywnym.

```
queries[
  confidentiality? dataA
  confidentiality? reply
  authentication? DeviceA -> HLF: E_DataAH
  authentication? HLF -> DeviceA: E_DataHA
  freshness? E_DataAH
  freshness? E_DataHA
]
```

Listing 4.8: Lista zapytań weryfikowana przez Verifpal w procesie komunikacji urządzenia z rejestrem rozproszonym

Komunikacja pomiędzy urządzeniami IoT

Komunikacja pomiędzy urządzeniami wymaga wcześniejszego uzyskania klucza od węzła rejestru rozproszonego przez urządzenie IoT, które chce nawiązać komunikację z innym urządzeniem IoT. Atakujący tak samo jak w poprzednich dwóch modelach może modyfikować przesyłane wiadomości. W celu uzyskania klucza do komunikacji urządzenie A szyfruje identyfikator urządzenia B przy pomocy algorytmu AEAD z wykorzystaniem klucza *keyAH* oraz nonce *nonceAH*. Szyfrogram *E_DataAH* wraz z *nonceAH* jest przesyłany do węzła rejestru rozproszonego. Model realizujący ten fragment został przedstawiony na listingu 4.9.

```
attacker[active]
principal DeviceA[
  knows private dataAB  //Zna dane ktore chce wyslac do urzadzenia B
  knows public IDB //Zna ID urzadzenia B
  generates nonceAH    //Generowanie nonce
  knows private keyAH    //Tworzy klucz (Tak samo jak w procesie rejestracji)
  E_DataAH = AEAD_ENC(keyAH,IDB,nonceAH)   //Szyfrowanie danych
]
DeviceA -> HLF: nonceAH,E_DataAH //Przesylanie do rejestru rozproszonego
```

Listing 4.9: Fragment modelu odpowiadającego za opis działań wykonywanych przez urządzenie A wysyłającego żądanie do rejestru rozproszonego wygenerowania klucza do komunikacji z innym urządzeniem

Węzeł rejestru rozproszonego po otrzymaniu wiadomości odszyfrowuje szyfrogram *E_DataAH* przy pomocy klucza *keyAH* oraz przesłanego nonce *nonceAH*. Przy pomocy algorytmu AEAD z wykorzystaniem klucza *keyHA* oraz nonce *nonceHA* szyfrowany jest utworzony

klucz do komunikacji między urządzeniami *keyAB*. Szyfrogram *E_DataHA* wraz z *nonceHA* są przesyłane do urządzenia *A*. Na listingu 4.10 przedstawiono fragment modelu opisujący czynności wykonywane przez węzeł rejestru rozproszonego.

```
principal HLF[
  knows private keyAH // Tworzy klucz do odszyfrowania
  D_DataAH = AEAD_DEC(keyAH,E_DataAH,nonceAH) //Odszyfrowuje dane -
      zaszyfrowany jest identyfikator urzadzenia B
  knows private keyAB //Tworzony klucz do komunikacji miedzy urzadzeniami A i B
  generates nonceHA    //Generowanie nonce
  knows private keyHA    //Tworzy klucz (Tak samo jak w procesie rejestracji)
  E_DataHA = AEAD_ENC(keyHA,keyAB,nonceHA)   //Szyfruje dane
]
HLF -> DeviceA: nonceHA,E_DataHA //Przesylanie do urzadzenia A
```

Listing 4.10: Fragment modelu odpowiadającego za opis działań wykonywanych przez rejestr rozproszony po odebraniu żądania wygenerowania klucza do komunikacji między urządzeniami A i B

Urządzenie *A* po otrzymaniu wiadomości, odszyfrowuje ją i dzięki temu uzyskuje klucz *keyAB* do komunikacji z urządzeniem *B*. Odpowiedni fragment modelu realizujący te kroki został przedstawiony na listingu 4.11.

```
principal DeviceA[
  knows private keyHA //Tworzy klucz do odszyfrowania danych
  D_DataHA = AEAD_DEC(keyHA,E_DataHA,nonceHA) //Odszyfrowuje klucz uzywany do
      komunikacji z urzadzeniem B
]
```

Listing 4.11: Fragment modelu odpowiadającego za opis działań wykonywanych przez urządzenie A przed wysyłaniem danych do urządzenia B

W modelu tym tak samo jak w poprzednich zweryfikowano poufność przesyłanych danych, uwierzytelnienie stron komunikacji oraz możliwość rozróżnienia komunikatów przesłanych w różnych przebiegach protokołu. Weryfikowane cechy zostały przedstawione na listingu 4.12.

```
queries[
  confidentiality? IDB
  confidentiality? keyAB
  authentication? DeviceA -> HLF: E_DataAH
  authentication? HLF -> DeviceA: E_DataHA
  freshness? E_DataAH
  freshness? E_DataHA
]
```

Listing 4.12: Lista zapytań weryfikowana przez Verifpal w procesie nawiązywania komunikacji urządzenia A z urządzeniem B

Verifpal podczas analizy właściwości tego modelu znalazł jeden możliwy atak. Został on przedstawiony na listingu 4.13.

```
Result : confidentiality? idb -
          idb (idb) is obtained by Attacker.
```

Listing 4.13: Wynik wykonania modelu nawiązania komunikacji przedstawiający pierwszy z możliwych ataków

Odnosi się on do niezapewnienia tajności przesyłanego identyfikatora urządzenia B do węzła rejestru rozproszonego. Jest to spowodowane tym, że identyfikator jest znany publicznie. Uznano, że nie jest to problem, który wpływa na bezpieczeństwo systemu. W rzeczywistym systemie identyfikator jest przesyłany wraz ze znacznikiem czasu, który jest weryfikowany dlatego nie ma możliwości wykonania ataku powtórzenia.

Drugi model obejmuje komunikację pomiędzy urządzeniami A i B po tym, jak urządzenie A otrzymało klucz $keyAB$. Dane $dataAB$ są szyfrowane przez urządzenie A przy pomocy algorytmu AEAD z wykorzystaniem klucza $keyAB$ oraz generowanej wartości nonce $nonceAB$ dla każdej przesłanej wiadomości. Otrzymany szyfrogram E_DataAB jest wraz z $nonceAB$ jest przesyłany do urządzenia B. Na listingu 4.14 przedstawiono fragment modelu realizujący nawiązanie komunikacji urządzenia A z urządzeniem B.

```
attacker[active]
principal DeviceA[
  knows private keyAB //Zna klucz do komunikacji z urzadzeniem B
  knows private dataAB
  generates nonceAB //Generowanie nonce
  E_DataAB = AEAD_ENC(keyAB,dataAB,nonceAB) //Szyfruje dane
]
DeviceA -> DeviceB: nonceAB,E_DataAB
```

Listing 4.14: Fragment modelu przestawiający operacje wykonywane przez urzadzenia A.

W rzeczywistym rozwiązaniu urządzenie B na podstawie numerów parametrów, swojej tablicy parametrów oraz przesłanego znacznika czasu potrafi utworzyć klucz *keyAB*. Jeżeli klucz ten wraz z przesłaną wartością nonce umożliwia odszyfrowanie przesłanego szyfrogramu, oznacza to, że urządzenie A ma prawo do komunikacji z urządzeniem B, ponieważ w innym przypadku węzeł rejestru nie wydałby poprawnego klucza. W zamodelowanym modelu założono, że klucz *keyAB* jest znany urządzeniu B. Urządzenie B wykorzystując ten klucz i wartość nonce *nonceAB* może odszyfrować *E_DataAB*. W wyniku tej operacji urządzenie B uzyskuje dane *DataAB*, na których wykonuje jakieś operacje. Odpowiedź *reply* jest szyfrowana przy pomocy algorytmu AEAD wykorzystując klucz *keyAB* oraz nową wartość nonce *nonceBA*. Powstaje szyfrogram *E_DataBA*, który jest przesyłany wraz z *nonceBA* do urządzenia A. Urządzenie A przy pomocy posiadanego klucza *keyAB* oraz wartości otrzymanej wartości nonce *nonceBA* potrafi odszyfrować *E_DataBA*, w wyniku czego otrzymuje odpowiedź *reply*. Listingi 4.15, 4.16 przedstawiają zamodelowaną komunikację pomiędzy urządzeniami B i A.

```
principal DeviceB[
  knows private keyAB //Tworzy klucz (Tak samo jak w procesie rejestracji)
  D_DataAB = AEAD_DEC(keyAB,E_DataAB,nonceAB) //Odszyfrowuje dane przeslane od
      urzadzenia A
  generates reply //Generowanie odpowiedzi
  generates nonceBA //Generowanie nonce
  E_DataBA = AEAD_ENC(keyAB_2,reply,nonceBA)  //Szyfruje dane
]
```

Listing 4.15: Fragment modelu przestawiający operacje wykonywane przez urzadzenia B po odebraniu wiadomości od urządzenia A

```
DeviceB -> DeviceA: nonceBA,E_DataBA
principal DeviceA[
  D_DataBA = AEAD_DEC(keyAB,E_DataBA,nonceBA) //Odszyfrowuje dane uzywane do
      komunikacji
]
```

Listing 4.16: Fragment modelu przestawiający operacje wykonywane przez urządzenie A po otrzymaniu odpowiedzi od urządzenia B.

Model ten weryfikował te same właściwości co poprzednie modele tj. poufność komunikacji, uwierzytelnienie przesyłanych wiadomości oraz możliwość rozróżnienia komunikatów przesłanych w różnych przebiegach protokołu. Lista weryfikowanych zapytań została przedstawiona na listingu 4.17.

```
queries[
  confidentiality? dataAB
  confidentiality? reply
  authentication? DeviceA -> DeviceB: E_DataAB
  authentication? DeviceB -> DeviceA: E_DataBA
  freshness? E_DataAB
  freshness? E_DataBA
]
```

Listing 4.17: Lista zapytań weryfikowana przez Verifpal w procesie komunikacji urządzenia A z urządzeniem B

W wyniku weryfikacji tego modelu Verifpal wskazał możliwość niespełnienia uwierzytelnienia przesyłanych wiadomości pomiędzy urządzeniami A i B. Verifpal pokazał, że możliwy jest atak, polegający na zamianie przesyłanych wiadomości, tzn. wiadomość przesyłana z urządzenia A do urządzenia B jest przechwycona przez atakującego i wysłana ponownie do urządzenia A. W drugim przypadku sytuacja jest analogiczna, tzn. atakujący przechwytuje wiadomość przesyłaną z urządzenia B do urządzenia A i w kolejnej iteracji przesyła ją do urządzenia B. W obydwu przypadkach wynik analizy modeli przeprowadzony przez Verifpal potwierdził, że nie zostanie zapewnione uwierzytelnienie stron, jednakże w rzeczywistym systemie obydwie wiadomości nie zostaną zrealizowane przez urządzenia. Jest to spowodowane tym, że urządzenia otrzymają dane, które nie będą miały oczekiwanego formatu tzn. urządzenie A otrzyma wiadomość zbudowaną jako żądanie, a oczekuje odpowiedzi, natomiast urządzenie B oczekuje żądania, a otrzyma odpowiedź. Obydwa ataki zostały przedstawione na listingach 4.18 i 4.19.

```
Result : authentication? Devicea -> Deviceb: e_dataab - When:
    nonceab -> nonceba <- mutated by Attacker (originally nonceab)
    e_dataab -> AEAD_ENC(keyab, reply, nonceba) <- mutated by Attacker (
        originally AEAD_ENC(keyab, dataab, nonceab))
    d_dataab -> reply <- obtained by Attacker
    d_databa -> reply <- obtained by Attacker
    e_dataab (AEAD_ENC(keyab, reply, nonceba)), sent by Attacker and not by
        Devicea, is successfully used in AEAD_DEC(keyab, AEAD_ENC(keyab, reply,
         nonceba), nonceba) within Deviceb's state.
```

Listing 4.18: Wynik wykonania modelu komunikacji urządzeń przedstawiający pierwszy z możliwych ataków

```
Result : authentication? Deviceb -> Devicea: e_databa - When:
    d_dataab -> dataab <- obtained by Attacker
    nonceba -> nonceab <- mutated by Attacker (originally nonceba)
    e_databa -> AEAD_ENC(keyab, dataab, nonceab) <- mutated by Attacker (
        originally AEAD_ENC(keyab, reply, nonceba))
    d_databa -> dataab <- obtained by Attacker
    e_databa (AEAD_ENC(keyab, dataab, nonceab)), sent by Attacker and not by
        Deviceb, is successfully used in AEAD_DEC(keyab, AEAD_ENC(keyab, dataab
        , nonceab), nonceab) within Devicea's state.
```

Listing 4.19: Wynik wykonania modelu komunikacji urządzeń przedstawiający drugi z możliwych ataków

Inną kwestią w przypadku tego modelu jest możliwość wysłania wielokrotnie tej samej wiadomości tzn. przeprowadzenie ataku powtórzeniowego. Sposób minimalizacji możliwości przeprowadzenia tego ataku został opisany w sekcji 4.2.

4.3.4 Weryfikacja z wykorzystaniem Tamarin

Tamarin podobnie jak Verifpal jest narzędziem umożliwiającym weryfikację modelu symbolicznego. Tamarin wykorzystuje etykietowane reguły przepisywania wielozbiorów (ang. multiset rewriting rules) do przedstawienia działania protokołu, a dokładniej przejść stanów protokołu [129]. Stan protokołu jest zbiorem faktów oraz akcji pomiędzy tymi zbiorami. Każde przejście jest opatrzone adnotacją czasową, co umożliwia zdefiniowanie określenie kolejności wystąpienia stanów. Do przedstawienia funkcji kryptograficznych wykorzystywane są prymitywy kryptograficzne, które nie muszą pochodzić tylko ze zdefiniowanego zbioru, ale mogą być

utworzone przez użytkownika. Własności bezpieczeństwa, które mają być sprawdzone są definiowane przez użytkownika w postaci formuł logiki pierwszego rzędu [129]. Weryfikacja modelu może nastąpić w trybie interaktywnym lub automatycznym [130]. W trybie interaktywnym to użytkownik wybiera jaki następny krok będzie wykonany. Aby wesprzeć użytkownika w tym wyborze, Tamarin wizualizuje przebieg dostępnych opcji. Zazwyczaj tryb interaktywny używany jest do zrozumienia automatycznie wygenerowanych dowodów i zbadania kwestii, gdy wynik analizy modelu jest nierozstrzygalny. W trybie automatycznym natomiast narzędzie łączy dedukcję i rozumowanie równościowe z heurystyką w celu wyszukiwania dowodów twierdzeń zdefiniowanych w modelu. Jeśli zautomatyzowane wyszukiwanie dowodu zakończy się, zwracany jest dowód poprawności albo kontrprzykład, reprezentujący atak, który narusza zdefiniowane twierdzenie. Tamarin został wykorzystany do znalezienia podatności w powszechnie wykorzystywanych rozwiązaniach np. 5G [19], TLS 1.3 [39, 40].

4.3.5 Rejestracja urządzenia IoT w rejestrze rozproszonym

Kluczową operacją zdefiniowaną w opisie protokołu 3.3 jest rejestracja urządzenia IoT, której wynikiem jest przesłanie tablicy parametrów do rejestru rozproszonego oraz przesłanie do urządzenia IoT identyfikatora. Na listingu 4.20 przedstawiono fragment początkowy modelu operacji rejestracji, który zawiera zdefiniowane funkcje, z których skorzystano, a także zdefiniowane dwa równania, które wykorzystano w modelu. Równania te opisują weryfikację poprawności przesłania danych, a także deszyfrowanie szyfrogramu.

```
theory DeviceRegistration
begin
functions:
aead/3, verify/3, true/0,decrypt/3
equations: verify(aead(k, p, a), a, k)=true
equations: decrypt(aead(k, p, a), a, k)=p
```

Listing 4.20: Fragment modelu określający wykorzystane funkcje oraz reguły

Kolejne elementy modelu opisują reguły przejść między stanami, gdzie stan składa się z symbolicznej reprezentacji wiedzy przeciwnika o wiadomościach przesłanych, informacji o świeżo wygenerowanych wartościach, oraz stanie protokołu. Wejściem i wyjściem każdej reguły jest stan. Przykładowo na listingu 4.21 przedstawiono regułę nazwaną *Setup*, która określa przejście ze stanu wygenerowania klucza *Fr(~key)* do stanu, w którym klucz jest przypisany do dwóch obiektów: urządzenia *$IoT* oraz rejestru rozproszonego *$HLF*. W środku

w strzałce zdefiniowane są akcje (w dokumentacji Tamarin ten element nazwany jest "action facts"), które nie wpływają na stan, ale są wykorzystywane w procesie dowodzenia twierdzeń.

Pierwsza z reguł (listing 4.21) nazwana *Setup* opisuje proces utworzenia tymczasowego klucza *key*, a następnie przypisanie go do dwóch tożsamości: urządzenia IoT *IoT* oraz węzła rejestru rozproszonego *HLF*. Kolejna reguła nazwana *IoT_1* określa przejście pomiędzy stanem utworzenia tożsamości a wysłaniem wiadomości. Stan wejściowy tworzy dwie zmienne reprezentujące losowe wartości: *nonce* oraz *new_key*. Ze względu na brak możliwości zamodelowania przesyłania tablicy parametrów, zdecydowano się wykorzystać zmienną *new_key*, która będzie symbolizowała współdzielony klucz pomiędzy urządzeniem *IoT* a rejestrem rozproszonym *HLF*. Zmienna *new_key* będzie symbolizowała także tajny klucz wykorzystywany do zabezpieczenia komunikacji. W rzeczywistym systemie, klucz ten bazuje na podzbiorze parametrów z przesłanej tablicy parametrów. Stan wyjściowy tej reguły zawiera wiadomość z urządzenia *IoT* do *HLF*, zawierającą *nonce* a także szyfrogram zawierający nowy klucz *new_key* utworzony z wykorzystaniem algorytmu AEAD na podstawie tymczasowego klucza *key* oraz wartości *nonce*. Zapis *Out()* oznacza, że dane zapisane wewnątrz nawiasu są przesyłane kanałem publicznym tzn. ujawniane są potencjalnemu atakującemu. Stan wyjściowy zawiera także przypisanie nowego klucza do urządzenia *IoT*. Wymienione reguły zostały przedstawione na listingu 4.21.

```
rule Setup:
    [ Fr(~key) ]
--[ID_IoT($IoT,~key),ID_Server($HLF,~key)]->
    [!Identity($IoT, ~key), !Identity($HLF, ~key)]
rule IoT_1:
    [ Fr(~nonce), Fr(~new_key), !Identity($IoT, ~key),!Identity(HLF, ~key)]
--[IoT2HLF($IoT,HLF,<~key,~nonce>),ID_IoT($IoT,~new_key)]->
    [ Out(<$IoT,HLF,~nonce,aead(~key,~new_key,~nonce)>),!New_Identity($IoT, ~
        new_key)]
```

Listing 4.21: Fragment modelu rejestracji urządzenia opisujący proces przypisywania klucza tymczasowego oraz przesłanie danych z urządzenia do węzła rejestru rozproszonego

Następna reguła *HLF_1*, przedstawiona na Listingu 4.22 reprezentuje działania wykonane przez węzeł rejestru rozproszonego. Reguła okresla przejście ze stanu z wygenrowaną losową wartością *nonce2* i odebraną wiadomością od urządzenia *IoT* przez węzeł rejestru rozproszonego. Dane zapisane wewnątrz nawiasów *In()* oznaczają, że są one odebrane z kanału publicznego. Natomiast wyjściem tej reguły przejścia jest stan, w którym wiadomość z węzła

rejestru rozproszonego *HLF* do urządzenia *IoT* zawierająca *nonce2*, a także szyfrogram zawierający odpowiedź *Response* utworzony z wykorzystaniem algorytmu AEAD na podstawie klucza *new_key* oraz wartości *nonce2*. Stan wyjściowy zawiera także przypisanie do rejestru rozproszonego *HLF* nowego klucza współdzielonego (reprezentującego tablicę parametrów) *new_key*.

Ostatnia reguła *IoT_2* określa przejście ze stanu po odebraniu wiadomości przez urządzenie *IoT* od węzła rejestru rozproszonego *HLF*. Stanem wyjściowym jest stan pusty. Obydwie reguły zostały przedstawione w fragmencie modelu zawartym w listingu 4.22.

```
rule HLF_1:
let EncMessage = aead(~key,new_key,nonce)
new_key = decrypt(EncMessage,nonce,~key)
in
    [ Fr(~nonce2),!Identity($HLF, ~key), In(<IoT,$HLF,nonce,EncMessage>) ]
--[HLF2IoT($HLF,IoT,<new_key,~nonce2>),Eq(verify(EncMessage,nonce,~key),true),
    ID_Server($HLF,new_key)]->
    [ Out(<$HLF,IoT,~nonce2,aead(new_key,'Response',~nonce2)>),!New_Identity(
        $HLF, new_key)]

rule IoT_2:
let EncMessageFromHLF = aead(new_key,'Response',~nonce2)
in
    [!New_Identity($IoT, new_key),In(<HLF,$IoT,~nonce2,EncMessageFromHLF>) ]
--[Receive(HLF,$IoT,<new_key,~nonce2>),Eq(verify(EncMessageFromHLF,~nonce2,
    new_key),true)]->
    [ ]
```

Listing 4.22: Fragment modelu opisujący proces utworzenia nowej tożsamości oraz przypisywania jej do urządzenia IoT

Poza regułami dodana jest także jedna restrykcja, nazwana *Dual_IoTs*, która zabrania utworzenia dwóch urządzeń *IoT* z tym samym kluczem *key*. W przeciwnym razie nie doszłoby do uwierzytelnienia urządzenia IoT, ponieważ węzeł rejestru rozproszonego nie ma pewności, które urządzenie rzeczywiście się z nim kontaktuje. Ograniczenie zostało przedstawione na listingu 4.23.

```
restriction dual_IoTs:
"
    All IoT key #i. (ID_IoT(IoT,key) @ #i)
    ==> not (Ex #j IoT2 . ID_IoT(IoT2,key) @j )//& #i=#j)
"
```

Listing 4.23: Fragment modelu opisującego restrykcję zabraniającą tworzenia dwóch tożsamości z tym samym kluczem

Weryfikacja tajności klucza tymczasowego *key* została zdefiniowana w twierdzeniu o nazwie *key_Secrecy_Temp*. Twierdzenie to definiuje, że dla każdej akcji *IoT2HLF* nie wystąpi taka chwila czasu *k1*, że atakujący pozna klucz *key*. Poza kluczem tymczasowym wykorzystywany jest także klucz *new_key* (tablica parametrów). Twierdzenie definiujące tajność tego klucza zostało opisane w twierdzeniu o nazwie *key_Secrecy*. Dla akcji *HLF2IoT* oraz *Receive* nie istnieje taka chwila *k2*, że atakujący pozna klucz (tablicę parametrów) *new_key*. W obydwu twierdzeniach określano także, że urządzenie *IoT* nie może być węzłem rejestru rozproszonego *HLF*. Listing 4.24 przedstawia fragment modelu zawierający definicje twierdzeń o tajności kluczy.

```
lemma key_Secrecy_Temp:
"
All IoT HLF key nonce #i. (
    IoT2HLF(IoT,HLF,<key,nonce>) @ #i &
    not (IoT = HLF)
)==> not(Ex #k1 . K(key) @ #k1 )
"

lemma key_Secrecy:
"
All IoT HLF new_key nonce2 #j #k. (
    HLF2IoT(HLF,IoT,<new_key,nonce2>) @ #j &
    Receive(HLF,IoT,<new_key,nonce2>) @ #k &
    #j < #k &
    not (IoT = HLF)
)==> not(Ex #k2 . K(new_key) @ #k2 )
"
```

Listing 4.24: Fragment modelu opisujący definicję twierdzeń o tajności kluczy

Weryfikacja uwierzytelnienia i możliwości przeprowadzenia ataku powtórzeniowego została

wykonana na podstawie definicji zawartych w [122]. Uwierzytelnienie jest tożsame z definicją nieróżnowartościowego porozumienia (ang. non-injective agreement). Definicja ta określa, że agent a w roli A osiąga porozumienie z agentem b w roli B w sprawie komunikatu M, jeśli za każdym razem, gdy a kończy wykonywanie protokołu, pozornie z b w roli B, to b wcześniej wykonywał protokół pozornie z a oraz b działał w roli B w swoim przebiegu, i obydwie strony zgodziły się co do komunikatu M [18]. Odnosząc to do badanego protokołu oznacza to, że uwierzytelnienie jednej strony następuje w sytuacji, gdy druga odpowie na wysłany komunikat M. Aby zweryfikować, czy nastąpiło uwierzytelnienie opracowano dwa twierdzenia odnoszące się do potwierdzenia komunikacji pomiędzy urządzeniem IoT a węzłem rejestru rozproszonego. Twierdzenie *noninjectiveagreement_IoT* opisuje uwierzytelnienie komunikatu wysłanego przez urządzenie IoT do węzła rejestru rozproszonego. Twierdzenie to definiuje, że dla każdej akcji *IoT2HLF* istnieje akcja *HLF2IoT* pomiędzy tym samym urządzeniem IoT oraz węzłem rejestru rozproszonego. Podobną definicję w kontekście uwierzytelnienia węzła rejestru rozproszonego przedstawiono w twierdzeniu *noninjectiveagreement_HLF*, gdzie dla każdej akcji *HLF2IoT* istnieje akcja *Receive* dla tego urządzenia IoT oraz węzła rejestru rozproszonego. Listing 4.25 przedstawia definicje twierdzeń nieróżnowartościowego porozumienia.

```
lemma noninjectiveagreement_IoT:
"
All IoT HLF key nonce #i . (
    IoT2HLF(IoT,HLF,<key,nonce>) @i
) ==> (Ex #j new_key nonce2 . HLF2IoT(HLF,IoT,<new_key,nonce2>) @j & #i < #j &
    not(Ex #k1 . K(key) @ #k1 ))
"

lemma noninjectiveagreement_HLF :
"
All IoT HLF new_key nonce #i . (
    HLF2IoT(HLF,IoT,<new_key,nonce>) @i
) ==> (Ex #j . Receive(HLF,IoT,<new_key,nonce>) @j & #i < #j & not(Ex #k1 . K(
    new_key) @ #k1))
"
```

Listing 4.25: Fragment modelu opisujący twierdzenia nieróżnowartościowego porozumienia

Rozszerzeniem definicji nieróżnowartościowego porozumienia jest różnowartościowe porozumienie (ang. injective agreement). Definicja tego pojęcia jest tożsama z nieróżnowartościowym porozumieniem, ale dodatkowo zawiera pojęcie unikalności strony uwierzytelnianej [181].

W odniesieniu do badanego protokołu oznacza to, że w momencie wykonywania protokołu nie istnieje trzecia strona poza urządzeniem *IoT* oraz węzłem rejestru rozproszonego mogąca wykonać ten sam przebieg protokołu. Twierdzenie nazwane *injectiveagreement_IoT* określa, że dla każdej akcji *IoT2HLF* istnieje *HLF2IoT*, która dzieje się później oraz nie występuje inna akcja *IoT2HLF* pomiędzy innym urządzeniem IoT a tym samym węzłem rejestru rozproszonego korzystający z tego samego klucza i wartości nonce. Podobnie skonstruowane jest twierdzenie *injectiveagreement_HLF*, które określa, że dla każdej akcji *HLF2IoT* istnieje akcja *Receive* oraz nie istnieje akcja *HLF2IoT* pomiędzy innym węzłem rejestru rozproszonego a tym samym urządzeniem IoT wykorzystując ten sam klucz, oraz wartość nonce. Na listingu 4.26 przedstawiono fragment modelu opisujący twierdzenia różnowartościowego porozumienia.

```
lemma injectiveagreement_IoT :
"
All IoT HLF key nonce #i . (
    IoT2HLF(IoT,HLF,<key,nonce>) @i
) ==> (Ex #j new_key nonce2. HLF2IoT(HLF,IoT,<new_key,nonce2>) @j & #i < #j &
    not (Ex IoT2 #i2. IoT2HLF(IoT2,HLF,<key,nonce>)@i2 & not (#i2 = #i)) &
    not(Ex #k1 . K(key) @ #k1))
"

lemma injectiveagreement_HLF :
"
All IoT HLF new_key nonce #i . (
    HLF2IoT(HLF,IoT,<new_key,nonce>) @i
) ==> (Ex #j. Receive(HLF,IoT,<new_key,nonce>) @j & #i < #j &
    not (Ex HLF2 #i2. HLF2IoT(HLF2,IoT,<new_key,nonce>)@i2 & not (#i2 = #i)) &
    not(Ex #k1 . K(new_key) @ #k1))
"
end
```

Listing 4.26: Fragment modelu opisujący twierdzenia różnowartościowego porozumienia

Każda z właściwości opisana za pomocą twierdzeń została potwierdzona poprzez weryfikację opracowanego modelu. Na rysunkach 4.2,4.3,4.4 przedstawiono potwierdzenie sukcesu weryfikacji zdefiniowanych twierdzeń. Oczywiście w sytuacji, gdy pozbędziemy się restrykcji niepozwalającej na przypisanie klucza tymczasowego do większej liczby urządzeń IoT to twierdzenia różnowartościowego porozumienia nie zostaną potwierdzenie pozytywnie.

```
lemma key_Secrecy_Temp:
  all-traces
  "∀ IoT HLF key nonce #i.
         ((IoT2HLF( IoT, HLF, <key, nonce> ) @ #i) ∧ (¬(IoT = HLF))) ⇒
         (¬(∃ #k1. K( key ) @ #k1))"
simplify
by contradiction /* from formulas */

lemma key_Secrecy:
  all-traces
  "∀ IoT HLF new_key nonce2 #j #k.
         ((((HLF2IoT( HLF, IoT, <new_key, nonce2> ) @ #j) ∧
            (Receive( HLF, IoT, <new_key, nonce2> ) @ #k)) ∧
           (#j < #k)) ∧
          (¬(IoT = HLF))) ⇒
         (¬(∃ #k2. K( new_key ) @ #k2))"
simplify
by solve( !Identity( $HLF, ~key ) ▶₁ #j )
```

Rysunek 4.2: Potwierdzenie spełnienia przez protokół tajności wykorzystywanych kluczy dla operacji rejestracji urządzenia IoT

```
lemma noninjectiveagreement_IoT:
  all-traces
  "∀ IoT HLF key nonce #i.
         (IoT2HLF( IoT, HLF, <key, nonce> ) @ #i) ⇒
         (∃ #j new_key nonce2.
           ((HLF2IoT( HLF, IoT, <new_key, nonce2> ) @ #j) ∧
            (#i < #j)) ∧
           (¬(∃ #k1. K( key ) @ #k1)))"
simplify
by contradiction /* from formulas */

lemma noninjectiveagreement_HLF:
  all-traces
  "∀ IoT HLF new_key nonce #i.
         (HLF2IoT( HLF, IoT, <new_key, nonce> ) @ #i) ⇒
         (∃ #j.
           ((Receive( HLF, IoT, <new_key, nonce> ) @ #j) ∧
            (#i < #j)) ∧
           (¬(∃ #k1. K( new_key ) @ #k1)))"
simplify
by solve( !Identity( $HLF, ~key ) ▶₁ #i )
```

Rysunek 4.3: Potwierdzenie spełnienia przez protokół nieróżnowartościowego porozumienia dla operacji rejestracji urządzenia IoT

```
lemma injectiveagreement_IoT:
  all-traces
  "∀ IoT HLF key nonce #i.
         (IoT2HLF( IoT, HLF, <key, nonce> ) @ #i) ⇒
         (∃ #j new_key nonce2.
           (((HLF2IoT( HLF, IoT, <new_key, nonce2> ) @ #j) ∧
             (#i < #j)) ∧
            (¬(∃ IoT2 #i2.
                (IoT2HLF( IoT2, HLF, <key, nonce> ) @ #i2) ∧
                (¬(#i2 = #i))))) ∧
           (¬(∃ #k1. K( key ) @ #k1)))"
simplify
by contradiction /* from formulas */

lemma injectiveagreement_HLF:
  all-traces
  "∀ IoT HLF new_key nonce #i.
         (HLF2IoT( HLF, IoT, <new_key, nonce> ) @ #i) ⇒
         (∃ #j.
           (((Receive( HLF, IoT, <new_key, nonce> ) @ #j) ∧
             (#i < #j)) ∧
            (¬(∃ HLF2 #i2.
                (HLF2IoT( HLF2, IoT, <new_key, nonce> ) @ #i2) ∧
                (¬(#i2 = #i))))) ∧
           (¬(∃ #k1. K( new_key ) @ #k1)))"
simplify
by solve( !Identity( $HLF, ~key ) ▶₁ #i )
```

Rysunek 4.4: Potwierdzenie spełnienia przez protokół różnowartościowego porozumienia dla operacji rejestracji urządzenia IoT

4.3.6 Komunikacja urządzenia IoT z rejestrem rozproszonym

Kolejną operacją zdefiniowaną w opisie protokołu 3.3 jest komunikacja urządzenia IoT z rejestrem rozproszonym. Weryfikacja tej operacji jest prostszą wersją operacji rejestracji urządzenia. Jedyną różnicą jest to, że w tej operacji urządzenia od początku komunikacji wykorzystują ten sam klucz do zabezpieczenia komunikacji. Fragment modelu opisujący proces komunikacji urządzenia IoT z rejestrem rozproszonym został przedstawiony na listingach 4.27, 4.28.

```
theory Device2Ledger
begin
functions:
aead/3, verify/3, true/0
equations: verify(aead(k, p, a), a, k)=true
```

Listing 4.27: Fragment modelu określający wykorzystane funkcje oraz reguły

```
rule Setup:
    [ Fr(~key) ]
--[ID_IoT($IoT,~key),ID_Server($HLF,~key)]->
    [Identity($IoT, ~key), Identity($HLF, ~key)]

rule IoT_1:
    [ Fr(~nonce), Identity($IoT, ~key), Identity(HLF, ~key) ]
--[IoT2HLF($IoT,HLF,<~key,~nonce>)]->
    [ Out(<$IoT,HLF,~nonce,aead(~key,'Dane',~nonce)>) ]

rule HLF_1:
let EncMessage = aead(~key,'Dane',~nonce)
in
    [ Fr(~nonce2),Identity($HLF, ~key), In(<IoT,$HLF,~nonce,EncMessage>)  ]
--[HLF2IoT($HLF,IoT,<~key,~nonce2>),Eq(verify(EncMessage,~nonce,~key),true)]->
    [ Out(<$HLF,IoT,~nonce2,aead(~key,'Response',~nonce2)>)]

rule IoT_2:
let EncMessageFromHLF = aead(~key,'Response',~nonce2)
in
    [ Identity($IoT, ~key), In(<HLF,$IoT,~nonce2,EncMessageFromHLF>)  ]
--[Receive(HLF,$IoT,<~key,~nonce2>),Eq(verify(EncMessageFromHLF,~nonce2,~key),
    true)]->
    [ ]
```

Listing 4.28: Fragment modelu opisujący komunikację urządzenia IoT z rejestrem rozproszonym

W tym modelu również dodano ograniczenie, że dwa urządzenia IoT nie mogą posiadać tego samego klucza. Fragment modelu przedstawiający to ograniczenie został przedstawiony na listingu 4.29.

```
restriction dual_IoTs:
"
    All IoT key #i. (
        ID_IoT(IoT,key) @ #i
    ) ==> not (Ex #j IoT2 . ID_IoT(IoT2,key) @j )//& #i=#j)
"
```

Listing 4.29: Fragment modelu opisującego restrykcję zabraniającą tworzenia dwóch tożsamości z tym samym kluczem

Twierdzenie *Key_ Secrecy* określa, że podczas wykonywania protokołu, nie dojdzie do ujawnienia klucza wykorzystywanego do zabezpieczenia komunikacji. Twierdzenie dotyczące nieróżnowartościowego i różnowartościowego porozumienia są identyczne jak w przypadku operacji rejestracji urządzenia. Definicje twierdzeń, które są sprawdzane w tym modelu zostały przedstawione na listingach 4.30, 4.31.

```
lemma Key_Secrecy:
"
All IoT HLF key nonce nonce2 #i #j. (
    IoT2HLF(IoT,HLF,<key,nonce>) @ #i &
    HLF2IoT(HLF,IoT,<key,nonce2>) @ #j &
    #i < #j &
    not (IoT = HLF)
)==> not(Ex #k1 . K(key) @ #k1)
"
lemma noninjectiveagreement_IoT:
"
All IoT HLF key nonce #i . (
    IoT2HLF(IoT,HLF,<key,nonce>) @i
) ==> Ex #j nonce2 . HLF2IoT(HLF,IoT,<key,nonce2>) @j & #i < #j & not(Ex #k1 .
    K(key) @ #k1)
"
lemma noninjectiveagreement_HLF :
"
All IoT HLF key nonce #i . (
    HLF2IoT(HLF,IoT,<key,nonce>) @i
) ==> (Ex #j . Receive(HLF,IoT,<key,nonce>) @j & #i < #j & not(Ex #k1 . K(key)
    @ #k1))
"
```

Listing 4.30: Fragment modelu opisujący definicje twierdzeń weryfikowanych przez Tamarin w modelu komunikacji urządzenia z rejestrem rozproszonym

```
lemma injectiveagreement_IoT :
"
All IoT HLF key nonce #i . (
    IoT2HLF(IoT,HLF,<key,nonce>) @i
) ==> (Ex #j nonce2. HLF2IoT(HLF,IoT,<key,nonce2>) @j & #i < #j &
    not (Ex IoT2 #i2. IoT2HLF(IoT2,HLF,<key,nonce>)@i2 & not (#i2 = #i)) &
    not(Ex #k1 . K(key) @ #k1))
"
lemma injectiveagreement_HLF :
"
All IoT HLF key nonce #i . (
    HLF2IoT(HLF,IoT,<key,nonce>) @i
) ==> (Ex #j . Receive(HLF,IoT,<key,nonce>) @j & #i < #j &
    not (Ex HLF2 #i2. HLF2IoT(HLF2,IoT,<key,nonce>)@i2 & not (#i2 = #i)) &
    not(Ex #k1 . K(key) @ #k1))
"
end
```

Listing 4.31: Fragment modelu opisujący definicje twierdzeń weryfikowanych przez Tamarin w modelu komunikacji urządzenia z rejestrem rozproszonym

Wynikiem weryfikacji opracowanego modelu komunikacji urządzenia IoT z rejestrem rozproszonym jest potwierdzenie wszystkich twierdzeń, co zostało przedstawione na rysunkach 4.5, 4.6, 4.7.

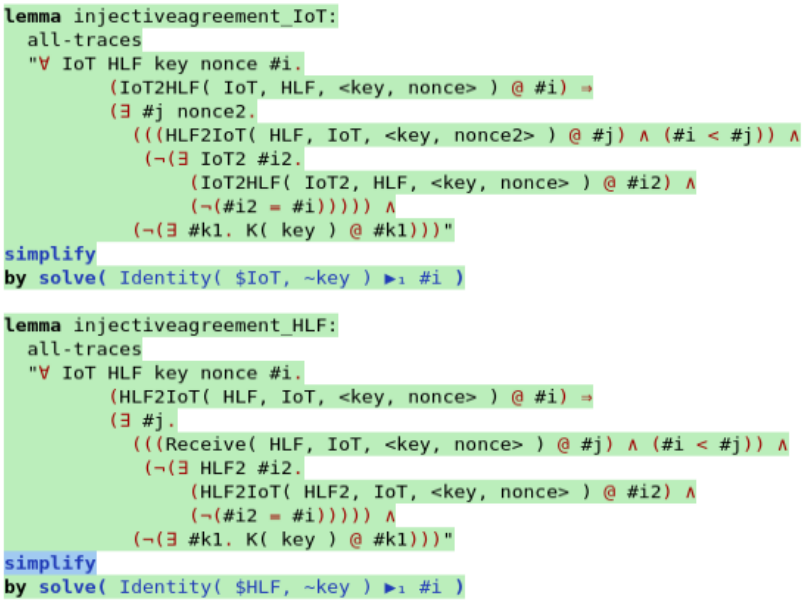
```
lemma Key_Secrecy:
  all-traces
  "∀ IoT HLF key nonce nonce2 #i #j.
         (((((IoT2HLF( IoT, HLF, <key, nonce> ) @ #i) ∧
            (HLF2IoT( HLF, IoT, <key, nonce2> ) @ #j)) ∧
           (#i < #j)) ∧
          (¬(IoT = HLF))) ⇒
         (¬(∃ #k1. K( key ) @ #k1))"
simplify
by solve( Identity( $IoT, ~key ) ▶₁ #i )
```

Rysunek 4.5: Potwierdzenie spełnienia przez protokół tajności wykorzystywanego klucza dla operacji komunikacji urządzenia IoT z rejestrem rozproszonym

Komunikacja pomiędzy urządzeniami IoT

Model komunikacji pomiędzy urządzeniami IoT zawiera zdefiniowane funkcje, z których skorzystano, a także zdefiniowane dwa równania, które wykorzystano w modelu. Równania te

Rysunek 4.6: Potwierdzenie spełnienia przez protokół nieróżnowartościowego porozumienia dla operacji komunikacji urządzenia IoT z rejestrem rozproszonym



Rysunek 4.7: Potwierdzenie spełnienia przez protokół różnowartościowego porozumienia dla operacji komunikacji urządzenia IoT z rejestrem rozproszonym

opisują weryfikację poprawności przesłania danych, a także deszyfrowanie szyfrogramu. Zdefiniowane funkcje oraz równania zostały przedstawione na listingu 4.32.

```
theory Device2Device
begin
functions:
aead/3, decrypt/3, verify/3, true/0, h/1
equations: decrypt(aead(k, p, a), a, k)=p
equations: verify(aead(k, p, a), a, k)=true
```

Listing 4.32: Fragment modelu określający wykorzystane funkcje oraz reguły w modelu komunikacji urządzeń IoT między sobą

Pierwsza reguła przejścia pomiędzy stanami przedstawia proces utworzenia dwóch zmiennych reprezentujących klucze potrzebne do komunikacji z oboma urządzeniami IoT oraz przypisania ich do odpowiednich tożsamości. Reguła *IoT_A_1* opisuje przejście pomiędzy stanem utworzenia tożsamości a stanem wysłaniem wiadomości do węzła rejestru rozproszonego. Wiadomość ta zawiera zaszyfrowany identyfikator urządzenia *IoT_B*. W regule *HLF_1* opisano przejście pomiędzy otrzymaniem wiadomości od urządzenia *IoT_A* a wysłaniem wiadomości zawierającej klucz do komunikacji z urządzeniem *B*. Klucz ten na potrzeby opracowanego modelu tworzony jest jako funkcja skrótu z nazw urządzeń IoT oraz klucza urządzenia *B*. Na listingach 4.33, 4.34 przedstawiono fragmenty modelu opisujące proces uzyskania przez urządzenie *A* klucza do komunikacji z urządzeniem *B*.

```
rule Setup:
    [ Fr(~keyA),Fr(~keyB) ]
--[ID_IoT_A($IoT_A,~keyA),ID_HLF($HLF,~keyA),ID_HLF($HLF,~keyB),ID_IoT_B($IoT_B
    ,~keyB)]->
    [!Identity($IoT_A, ~keyA), !Identity($HLF, ~keyA),!Identity($HLF, ~keyB),!
        Identity($IoT_B, ~keyB)]
```

Listing 4.33: Fragment modelu opisujący przypisanie tożsamości oraz proces uzyskania przez urządzenie A klucza do komunikacji z urządzeniem B - cześć 1

```
rule IoT_A_1:
    [ Fr(~nonce), !Identity($IoT_A, ~keyA), !Identity(HLF, ~keyA) ]
--[IoT_A_HLF($IoT_A,HLF,<~keyA,~nonce>)]->
    [ Out(<$IoT_A,HLF,~nonce,aead(~keyA,$IoT_B,~nonce)>) ]

rule HLF_1:
let EncMessage = aead(~keyA,IoT_B,~nonce)
keyAB = h(<IoT_A,IoT_B,~keyB>)
IoT_B = decrypt(EncMessage,~nonce,~keyA)
in
    [ Fr(~nonce2),!Identity($HLF, ~keyA),!Identity($HLF, ~keyB), In(<IoT_A,$HLF
        ,~nonce,EncMessage>) ]
--[HLFIoT_A($HLF,IoT_A,<~keyA,~nonce2>),Ex(verify(EncMessage,~nonce,~keyA),true
    )]->
    [ Out(<$HLF,IoT_A,~nonce2,aead(~keyA,h(<IoT_A,IoT_B,~keyB>),~nonce2)>),!
        SessionKey(IoT_A,IoT_B,keyAB)]
```

Listing 4.34: Fragment modelu opisujący przypisanie tożsamości oraz proces uzyskania przez urządzenie A klucza do komunikacji z urządzeniem B - cześć 2

Reguła *IoT_A_2* określa przejście ze stanu odebrania wiadomości od rejestru rozproszonego przez urządzenie A, a wysłaniem wiadomości do urządzenia B, zabezpieczonej z wykorzystaniem klucza otrzymanego z rejestru rozproszonego. Proces przejścia pomiędzy odebraniem wiadomości od urządzenia A przez urządzenie B, a wysłaniem odpowiedzi został przedstawiony w regule *IoT_B_1*. Ostatnia reguła *IoT_A_3* przedstawia odebranie odpowiedzi przez urządzenie A. Reguły opisujące komunikację pomiędzy urządzeniami przedstawiono na listingach 4.35,4.36.

```
rule IoT_A_2:
let EncMessageFromHLF= aead(~keyA,keyAB,~nonce2)
keyAB = decrypt(EncMessageFromHLF,~nonce2,~keyA)
in
    [ Fr(~nonce3), !Identity($IoT_A, ~keyA), In(<HLF,$IoT_A,~nonce2,
        EncMessageFromHLF>),!SessionKey($IoT_A,$IoT_B,keyAB) ]
--[Receive_HLF(HLF,$IoT_A,<~keyA,~nonce2>),IoT_A_IoT_B($IoT_A,$IoT_B,<keyAB,~
    nonce3>),Ex(verify(EncMessageFromHLF,~nonce2,~keyA),true),ID_IoT_AB($IoT_A,
    keyAB)]->
    [ Out(<$IoT_A, $IoT_B, ~nonce3, aead(keyAB,'Data',~nonce3)>), ]
```

Listing 4.35: Fragment modelu opisujący komunikację pomiędzy urządzeniami - cześć 1

```
rule IoT_B_1:
let
EncMessageFromIoT_A = aead(keyAB, 'Data',~nonce3)
in
    [ Fr(~nonce4), !Identity($IoT_B, ~keyB), In(<$IoT_A,$IoT_B,~nonce3,
        EncMessageFromIoT_A>),!SessionKey($IoT_A,$IoT_B,keyAB)  ]
--[IoT_B_IoT_A($IoT_B,$IoT_A,<keyAB,~nonce4>),Ex(verify(EncMessageFromIoT_A,~
    nonce3,keyAB),true),ID_IoT_AB($IoT_B,keyAB)]->
    [ Out(<$IoT_B, $IoT_A, ~nonce4, aead(keyAB, 'Response',~nonce4)>)  ]

rule IoT_A_3:
let
EncMessageFromIoT_B = aead(keyAB, 'Response',~nonce4)
in
    [ !Identity($IoT_A, ~keyA), In(<$IoT_B,$IoT_A,~nonce4,EncMessageFromIoT_B>)
        ,!SessionKey($IoT_A,$IoT_B,keyAB)  ]
--[Receive($IoT_B,$IoT_A,<keyAB,~nonce4>),Ex(verify(EncMessageFromIoT_B,~nonce4
    ,keyAB),true)]->
    [  ]
```

Listing 4.36: Fragment modelu opisujący komunikację pomiędzy urządzeniami - cześć 2

W opracowanym modelu utworzono także ograniczenie zabraniające na utworzenie większej liczby urządzeń z identycznym kluczem. Ograniczenia przedstawiono na listingu 4.37.

```
restriction dual_IoT_A:
"
    All IoT key #i. (
        ID_IoT_A(IoT,key) @ #i
    ) ==> not (Ex #j IoT2 . ID_IoT_A(IoT2,key) @j )//& #i=#j)
"
restriction dual_IoT_B:
"
    All IoT key #i. (
        ID_IoT_B(IoT,key) @ #i
    ) ==> not (Ex #j IoT2 . ID_IoT_B(IoT2,key) @j )//& #i=#j)
"
```

Listing 4.37: Fragment modelu opisującego restrykcję zabraniającą tworzenia dwóch tożsamości z tym samym kluczem dla obydwu urządzeń IoT

Weryfikacja tajności klucza wymaga utworzenia dwóch twierdzeń dla dwóch kluczy, jednego użytego do zabezpieczenia komunikacji pomiędzy urządzeniem A a węzłem rejestru rozproszonego co zostało przedstawione w twierdzeniu nazwanym *Key_Secrecy_HLF_A*. Drugim kluczem jest klucz użyty do komunikacji pomiędzy urządzeniami IoT A i B. Tajność tego klucza jest weryfikowana przez twierdzenie *Key_Secrecy_AB*. Definicje twierdzeń o tajności przedstawiono na listingu 4.38.

```
lemma Key_Secrecy_HLF_A:
"
All IoT HLF Key nonce nonce2 #i #j .
(
    IoT_A_HLF(IoT,HLF,<Key,nonce>) @ #i &
    HLFIoT_A(HLF,IoT,<Key,nonce2>) @ #j &
    #i < #j &
    not (IoT = HLF)
)==> not(Ex #k1 . K(Key) @ #k1)
"

lemma Key_Secrecy_AB:
"
All IoT_A IoT_B Key nonce nonce2 #i #j .
(
    IoT_A_IoT_B(IoT_A,IoT_B,<Key,nonce>) @ #i &
    IoT_B_IoT_A(IoT_B,IoT_A,<Key,nonce2>) @ #j &
    #j < #i &
    not (IoT_A = IoT_B)
)==> not(Ex #k . K(Key) @ #k)
"
```

Listing 4.38: Fragment modelu opisujący twierdzenia tajności kluczy

Weryfikacja spełnienia nieróżnowartościowego porozumienia wymagała przygotowania czterech twierdzeń, po jednym dla każdej wysłanej wiadomości. Na listingu 4.39 przedstawiono definicje twierdzeń nieróżnowartościowego porozumienia.

```
lemma noninjectiveagreement_IoT:
"
All IoT HLF key nonce #i . (
    IoT_A_HLF(IoT,HLF,<key,nonce>) @i
) ==> Ex #j nonce2 . HLFIoT_A(HLF,IoT,<key,nonce2>) @j & #i < #j & not(Ex #k1 .
    K(key) @ #k1)
"

lemma noninjectiveagreement_HLF:
"
All IoT HLF key nonce #i . (
    HLFIoT_A(HLF,IoT,<key,nonce>) @i
) ==> Ex #j . Receive_HLF(HLF,IoT,<key,nonce>) @j & #i < #j & not(Ex #k1 . K(
    key) @ #k1)
"

lemma noninjectiveagreement_A_B:
"
All IoT_A IoT_B key nonce #i . (
    IoT_A_IoT_B(IoT_A,IoT_B,<key,nonce>) @i
) ==> Ex #j nonce2 . IoT_B_IoT_A(IoT_B,IoT_A,<key,nonce2>) @j & #i < #j & not(
    Ex #k1 . K(key) @ #k1)
"

lemma noninjectiveagreement_B_A:
"
All IoT_A IoT_B key nonce #i . (
    IoT_B_IoT_A(IoT_B,IoT_A,<key,nonce>) @i
) ==> Ex #j. Receive(IoT_B,IoT_A,<key,nonce>) @j & #i < #j & not(Ex #k1 . K(key
    ) @ #k1)
"
```

Listing 4.39: Fragment modelu opisujący twierdzenia nieróżnowartościowego porozumienia

Potwierdzenie spełnienia różnowartościowego porozumienia w przygotowanym modelu również czterech twierdzeń, po jednym dla każdej wysłanej wiadomości. Definicję twierdzeń różnowartościowego porozumienia przedstawiono na listingu 4.40.

```
lemma injectiveagreement_IoT:
"
All IoT HLF key nonce #i . (
    IoT_A_HLF(IoT,HLF,<key,nonce>) @i
) ==> Ex #j nonce2 . HLFIoT_A(HLF,IoT,<key,nonce2>) @j & #i < #j & not (Ex IoT2
    #i2. IoT_A_HLF(IoT2,HLF,<key,nonce>)@i2 & not (#i2 = #i)) & not(Ex #k1 . K
    (key) @ #k1)
"

lemma injectiveagreement_HLF:
"
All IoT HLF key nonce #i . (
    HLFIoT_A(HLF,IoT,<key,nonce>) @i
) ==> Ex #j . Receive_HLF(HLF,IoT,<key,nonce>) @j & #i < #j & not (Ex HLF2 #i2.
    HLFIoT_A(HLF2,IoT,<key,nonce>)@i2 & not (#i2 = #i)) & not(Ex #k1 . K(key)
    @ #k1)
"

lemma injectiveagreement_A_B:
"
All IoT_A IoT_B key nonce #i . (
    IoT_A_IoT_B(IoT_A,IoT_B,<key,nonce>) @i
) ==> Ex #j nonce2 . IoT_B_IoT_A(IoT_B,IoT_A,<key,nonce2>) @j & #i < #j & not (
    Ex IoT_A2 #i2. IoT_A_IoT_B(IoT_A2,IoT_B,<key,nonce>)@i2 & not (#i2 = #i))&
    not(Ex #k1 . K(key) @ #k1)
"

lemma injectiveagreement_B_A:
"
All IoT_A IoT_B key nonce #i . (
    IoT_B_IoT_A(IoT_B,IoT_A,<key,nonce>) @i
) ==> Ex #j. Receive(IoT_B,IoT_A,<key,nonce>) @j & #i < #j & not (Ex IoT_B2 #i2
    . IoT_B_IoT_A(IoT_B2,IoT_A,<key,nonce>)@i2 & not (#i2 = #i)) & not(Ex #k1 .
    K(key) @ #k1)
"
```

Listing 4.40: Fragment modelu opisujący twierdzenia różnowartościowego porozumienia

Wszystkie twierdzenia zostały udowodnione pozytywnie, co zostało przedstawione na rysunkach 4.8,4.9,4.10.

```
lemma Key_Secreccy_HLF_A:
  all-traces
  "∀ IoT HLF Key nonce nonce2 #i #j.
         ((((IoT_A_HLF( IoT, HLF, <Key, nonce> ) @ #i) ∧
            (HLFIoT_A( HLF, IoT, <Key, nonce2> ) @ #j)) ∧
           (#i < #j)) ∧
          (¬(IoT = HLF))) ⇒
         (¬(∃ #k1. K( Key ) @ #k1))"
simplify
by solve( !Identity( $IoT_A, ~keyA ) ▶₁ #i )

lemma Key_Secreccy_AB:
  all-traces
  "∀ IoT_A IoT_B Key nonce nonce2 #i #j.
         ((((IoT_A_IoT_B( IoT_A, IoT_B, <Key, nonce> ) @ #i) ∧
            (IoT_B_IoT_A( IoT_B, IoT_A, <Key, nonce2> ) @ #j)) ∧
           (#i < #j)) ∧
          (¬(IoT_A = IoT_B))) ⇒
         (¬(∃ #k. K( Key ) @ #k))"
simplify
by solve( !Identity( $IoT_A, ~keyA ) ▶₁ #i )
```

Rysunek 4.8: Potwierdzenie spełnienia przez protokół tajności wykorzystywanych kluczy dla operacji komunikacji pomiędzy urządzeniem IoT a węzłem rejestru rozproszonego oraz pomiędzy urządzeniami IoT

4.4 Podsumowanie

W rozdziale przedstawiono analizę bezpieczeństwa obejmującą:

- analizę entropii parametrów wykorzystanych do tworzenia klucza,
- analizę jakościową możliwości przeprowadzenia prawdopodobnych ataków,
- analizę formalną.

Analiza entropii parametrów potwierdziła, że wykorzystanie pojedynczych odpowiedzi pochodzących z układu PUF oraz parametrów sprzętowych i programowych urządzenia może być niewystarczające do generowania bezpiecznych kluczy o długości min. 128 bitów. W związku, z czym celowe jest wykorzystanie kilku odpowiedzi PUF lub też parametrów urządzenia IoT, aby osiągnąć wystarczającą entropię dla tworzonego klucza. Oszacowanie średniej odległości Hamminga pozwoliło na uzyskanie informacji o parametrach sprzętowo-programowych urządzeń IoT, które najlepiej nadają się do identyfikacji urządzeń i generowania kluczy kryptograficznych. Wśród parametrów, których powinno się unikać znalazły się oba parametry związane z kartą pamięci. Spowodowane jest to tym, że wykorzystano parametry z kart pamięci kupionych jednocześnie w większej liczbie, co przełożyło się na posiadanie kart o zbliżonych

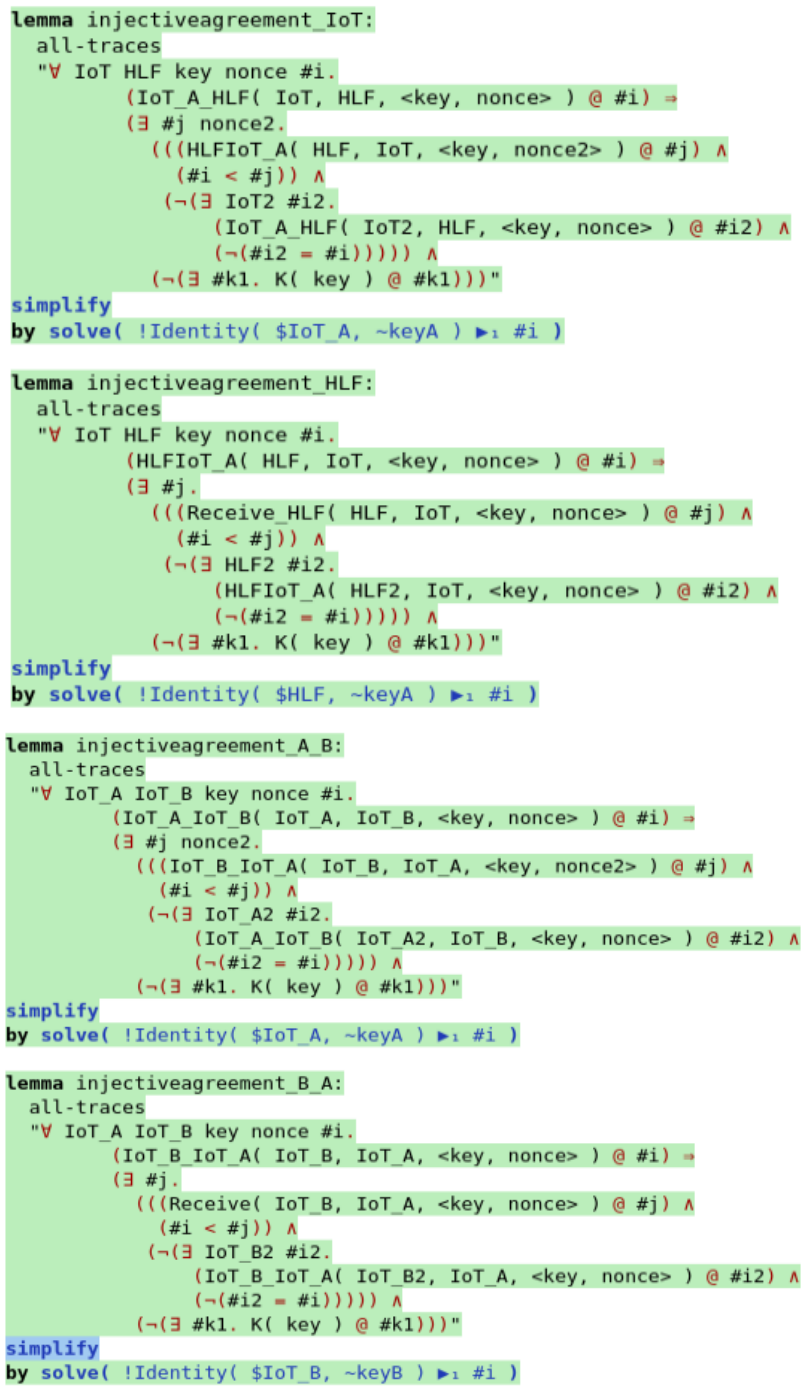
```
lemma noninjectiveagreement_IoT:
  all-traces
  "∀ IoT HLF key nonce #i.
         (IoT_A_HLF( IoT, HLF, <key, nonce> ) @ #i) ⇒
         (∃ #j nonce2.
           ((HLFIoT_A( HLF, IoT, <key, nonce2> ) @ #j) ∧ (#i < #j)) ∧
           (¬(∃ #k1. K( key ) @ #k1)))"
simplify
by solve( !Identity( $IoT_A, ~keyA ) ▶₁ #i )

lemma noninjectiveagreement_HLF:
  all-traces
  "∀ IoT HLF key nonce #i.
         (HLFIoT_A( HLF, IoT, <key, nonce> ) @ #i) ⇒
         (∃ #j.
           ((Receive_HLF( HLF, IoT, <key, nonce> ) @ #j) ∧
            (#i < #j)) ∧
           (¬(∃ #k1. K( key ) @ #k1)))"
simplify
by solve( !Identity( $HLF, ~keyA ) ▶₁ #i )

lemma noninjectiveagreement_A_B:
  all-traces
  "∀ IoT_A IoT_B key nonce #i.
         (IoT_A_IoT_B( IoT_A, IoT_B, <key, nonce> ) @ #i) ⇒
         (∃ #j nonce2.
           ((IoT_B_IoT_A( IoT_B, IoT_A, <key, nonce2> ) @ #j) ∧
            (#i < #j)) ∧
           (¬(∃ #k1. K( key ) @ #k1)))"
simplify
by solve( !Identity( $IoT_A, ~keyA ) ▶₁ #i )

lemma noninjectiveagreement_B_A:
  all-traces
  "∀ IoT_A IoT_B key nonce #i.
         (IoT_B_IoT_A( IoT_B, IoT_A, <key, nonce> ) @ #i) ⇒
         (∃ #j.
           ((Receive( IoT_B, IoT_A, <key, nonce> ) @ #j) ∧
            (#i < #j)) ∧
           (¬(∃ #k1. K( key ) @ #k1)))"
simplify
by solve( !Identity( $IoT_B, ~keyB ) ▶₁ #i )
```

Rysunek 4.9: Potwierdzenie spełnienia przez protokół nieróżnowartościowego porozumienia dla operacji komunikacji pomiędzy urządzeniem IoT a węzłem rejestru rozproszonego oraz pomiędzy urządzeniami IoT

numerach seryjnych. Sytuacja ta nie miała miejsca w przypadku urządzeń Raspberry Pi, pomimo że tak samo jak karty zostały zakupione w większej liczbie jednocześnie.

Analiza jakościowa przedstawia jasno, że ważnym elementem bezpieczeństwa protokołu jest ochrona wartości parametrów przechowywanych w rejestrze rozproszonym. Newralgicznym elementem opracowanego protokołu jest odpowiednia synchronizacja czasu oraz wymiana danych pomiędzy węzłami o otrzymanych wiadomościach w celu ochrony przed atakiem metodą powtórzenia.

Modelowanie protokołu z wykorzystaniem Verifpal i Tamarin wymagało wprowadzenia kilku uproszczeń. Pierwszym z nich jest oznaczenie, że klucz został wygenerowany losowo,



Rysunek 4.10: Potwierdzenie spełnienia przez protokół różnowartościowego porozumienia dla operacji komunikacji pomiędzy urządzeniem IoT a węzłem rejestru rozproszonego oraz pomiędzy urządzeniami IoT

a nie na podstawie podzbioru parametrów. Drugim uproszczeniem, jest przesyłanie tego klucza, zamiast tablicy parametrów. Obydwa uproszczenia nie powinny mieć wpływu na wynik

przeprowadzonej weryfikacji modeli operacji przewidzianych w LAAFFI. Ostatnim uproszczeniem, które trzeba było zastosować jest brak analizy wpływu czasu na możliwość przeprowadzenia ataku metodą powtórzenia. Spowodowane jest to tym, że żadne znalezione narzędzie formalnej weryfikacji protokołu nie ma takiej możliwości. Weryfikacja formalna modeli zakończyła się pozytywnym wynikiem, ponieważ znalezione ataki są niemożliwe do zrealizowania w rzeczywistym środowisku. Wykorzystanie narzędzi formalnej weryfikacji było niezbędne do potwierdzenia, że opracowany protokół zapewnia wymaganie tajności, uwierzytelnienia stron oraz ochronę przed atakiem metodą powtórzenia.

W następnym rozdziale zostanie przedstawiona analiza wydajności protokołu LAAFFI.

Rozdział 5

Badania wydajnościowe opracowanego protokołu

W tym rozdziale przedstawiono wyniki badań wydajnościowych opracowanego protokołu dla urządzeń IoT przy wykorzystaniu różnych protokołów kryptograficznych. Badania te musiały być poprzedzone sprawdzeniem, jaka konfiguracja protokołu CoAP jest najwydajniejsza. Oceniono także wydajność opracowanego protokołu dla rejestru rozproszonego, a także skalowalność rejestru rozproszonego. Istotnym wymaganiem dla opracowanego protokołu było zapewnienie minimalnego nadmiaru przesyłania danych pomiędzy urządzeniem IoT a bramą aplikacyjną. Z tego też powodu wykonano badania porównawcze protokołu LAAFFI z innymi podobnymi rozwiązaniami. Opisano także model sieci Petriego, który może pomóc wyznaczyć wydajność dla różnych konfiguracji rejestru rozproszonego przy różnej liczbie napływających transakcji.

5.1 Metodyka przeprowadzonych badań wydajnościowych

Podczas przeprowadzania badań wydajnościowych (benchmarków) systemów rozproszonego przetwarzania danych (a do takich należy zaliczyć zaproponowany ramowy system IoT z wykorzystaniem Hyperledger Fabric), konieczne jest uwzględnienie trzech głównych wskaźników: opóźnienia, przepustowości i stopnia wykorzystania zasobów sieciowych (przepustowości sieci). Opóźnienie jest miarą czasu oczekiwania [70] na wykonanie operacji przez urządzenie IoT oraz rejestr rozproszony. Przepustowość jest miarą szybkości wykonywania pracy [70]. Termin ten będzie używany do wykonanej liczby operacji lub transakcji w czasie sekundy. Wykorzystanie zasobów sieciowych też jest rozumiane jako przepustowość, ale wyrażona w liczbie bajtów przesłanych w ciągu sekundy.

Problem oceny wydajności systemu ramowego z wykorzystaniem opracowanego proto-

kołu jest dość złożony, bo na ogólną ocenę wydajności funkcjonowania całego systemu mogą mieć wpływ zarówno parametry wejściowe (np. konfiguracja systemu, użyte protokoły warstw niższych, dodatkowe komponenty systemu działające w tle) jak i scenariusze przetwarzania (obciążenie węzłów). Parametrami, od których głównie zależy wydajność badanego systemu ramowego są m.in.:

1. obciążenie węzłów (np. liczba zleconych transakcji/jednostkę czasu),
2. konfiguracja procesu osiągnięcia konsensusu,
3. wykorzystane protokoły,
4. wykorzystany typ sieci (np. Wi-Fi, LoRA, Zigbee),
5. konfiguracja sprzętowa bramy aplikacyjnej, węzłów rejestru rozproszonego,
6. wykonywanie innych zadań (np. aktualizacja systemu, wykonywanie kopii zapasowej),
7. liczba bram aplikacyjnych, liczba węzłów rejestru rozproszonego,
8. liczba organizacji tworzących federację.

Głównym celem badań jest potwierdzenie użyteczności opracowanego rozwiązania (protokołu i architektury ramowej systemu) w określonych zastosowaniach. Aby to zrobić, trzeba rozważyć podstawowe scenariusze użytkowania systemu, dla których należy przeprowadzić testy wydajnościowe. Główne scenariusze użycia to:

1. przeprowadzenie rejestracji urządzenia – najważniejszymi wskaźnikami są: liczba operacji rejestracji urządzeń w ciągu sekundy, opóźnienie procesu rejestracji, liczba przesłanych nadmiarowych bajtów pomiędzy urządzeniem IoT a bramą aplikacyjną,
2. ustanowienie bezpiecznego kanału komunikacji z rejestrem rozproszonym oraz innymi usługami – do najbardziej istotnych wskaźników należy zaliczyć liczbę operacji szyfrowania oraz deszyfrowania danych w ciągu sekundy w zależności od ilości danych, liczbę operacji weryfikacji uprawnień w ciągu sekundy, a także opóźnienie wykonania operacji przez węzły rejestru rozproszonego,
3. ustanowienie bezpiecznego kanału komunikacji pomiędzy urządzeniami należącymi do różnych domen – najbardziej istotnymi wskaźnikiem jest liczba operacji wygenerowania klucza do komunikacji pomiędzy urządzeniami IoT, liczba operacji weryfikacji uprawnień

w ciągu sekundy, a także opóźnienie wykonania operacji przez węzły rejestru rozproszonego.

Aby stwierdzić, czy proponowane rozwiązanie jest skalowalne, należałoby zbadać jak wskaźniki związane z wydajnością węzłów rejestru rozproszonego zmieniają się w zależności od konfiguracji sprzętowej węzłów, liczby węzłów w organizacji, oraz liczby organizacji tworzących federację.

Dodatkowo należy sprawdzić jak na wydajność protokołu LAAFFI wpływa wykorzystanie różnych algorytmów AEAD oraz HMAC. Badania wydajnościowe dla urządzeń IoT przeprowadzono na urządzeniach Raspberry Pi (RPi) ze względu na ich popularność oraz możliwość wyboru urządzeń o różnej konfiguracji sprzętowej pochodzącej od tego samego producenta, oraz wykorzystująca ten sam system operacyjny przystosowany do działania na tych urządzeniach.

5.2 Badanie wydajnościowe urządzeń Raspberry Pi

5.2.1 Badanie protokołu CoAP

Celem tego badania jest ocena wydajności protokołu CoAP dla różnych typów protokołu warstwy czwartej modelu ISO\OSI. Dodatkowo dla protokołu CoAP wykorzystującego TCP [30] zbadano wpływ tworzenia sesji na wydajność. Sesja pomiędzy klientem a serwerem CoAP może być utworzona na czas:

- wysłania jednej wiadomości,
- wysłania większej liczby wiadomości (wykonania całego testu).

Badanie polegało na wykonaniu 10 testów. Każdy test polegał na wysłaniu 10000 wiadomości składających się ze zmiennej liczby bajtów (od 128B do 102 400B). Wiadomość składała się z losowych znaków i była identyczna dla każdego testu oraz scenariusza. Wiadomość wysyłana była z urządzenia IoT do serwera przy użyciu protokołu CoAP. Serwer odsyłał wiadomość niezwłocznie po jej otrzymaniu bez jej zmiany. Urządzenie weryfikowało, czy wiadomość wysłana i odebrana są identyczne. Badania przeprowadzono dla czterech scenariuszy:

Scenariusz 1. - używany protokół warstwy aplikacji CoAP wykorzystujący TCP. Każdy test obejmował utworzenie jednej sesji, w ramach której wysłano 10000 wiadomości.
Scenariusz 2. - używany protokół warstwy aplikacji CoAP wykorzystujący TCP. Każdy test

obejmował tworzenie osobnej sesji, w ramach której wysłano jedną wiadomość.
Scenariusz 3. - używany protokół warstwy aplikacji CoAP wykorzystujący UDP. Każda wiadomość wysyłana jest z jednego portu źródłowego.
Scenariusz 4. - używany protokół warstwy aplikacji CoAP wykorzystujący UDP. Każda wiadomość wysyłana jest z losowego portu źródłowego.

Środowisko do badania składało się z trzech komponentów:

- urządzenia IoT – Raspberry Pi 4B 8GB RAM,
- przełącznika – Modem Technicolor CGA4236TCH1,
- maszyny wirtualnej z systemem operacyjnym Ubuntu 22.04 LTS.

Dokładne parametry komponentów są przedstawione w tabelach A1, A4, A5 zawartych w załączniku do rozprawy. Schemat połączenia komponentów został przedstawiony na rysunku 5.1. Urządzenie IoT z przełącznikiem, a także przełącznik z komputerem, na którym była uruchomiona maszyna wirtualna połączono z użyciem kabla Ethernet kat. 5e. Na urządzeniu IoT uruchomiono program klienta, który wysyłał dane do serwera CoAP uruchomionego na maszynie wirtualnej. Obydwa programy klienta i serwera napisano w języku Golang w wersji 1.18.3 wykorzystując bibliotekę go-coap w wersji 2.6.0 [1]. Zapytania zawierały ścieżkę jednoznakową: $/a$.

Rysunek 5.1: Schemat połączenia sieciowego pomiędzy urządzeniem IoT a serwerem CoAP

Liczbę obsłużonych operacji na sekundę dla czterech scenariuszy przedstawiono w tabeli 5.1.

Na podstawie wyników przedstawionych w tabeli 5.1 można wywnioskować, że najlepszym rozwiązaniem jest wykorzystanie CoAP bazującego na TCP i nawiązywaniu jednej sesji dla całego procesu wymiany informacji. W przypadku badania protokołu CoAP wykorzystującego UDP dochodziło do przesyłania duplikatów pakietów. Jeżeli do takiej sytuacji doszło, to połączenie kończyło się błędem i test nie mógł być dokończony. Sytuacja taka wydarzyła się podczas wykonywania badań dla dwóch scenariuszy: trzeciego scenariusza i wiadomości

[1] `https://github.com/plgd-dev/go-coap`

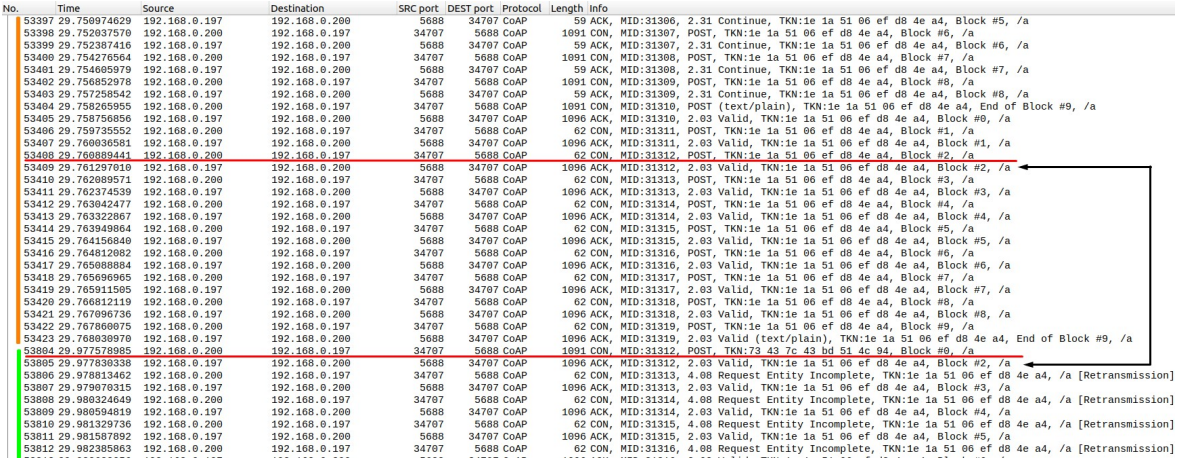
Tabela 5.1: Wyniki badania wydajności protokołu CoAP

Liczba bajtów w wiadomości [B]	**Liczba operacji na sekundę**			
	scenariusz 1.	**scenariusz 2.**	**scenariusz 3.**	**scenariusz 4.**
128	1059,70	302,87	1013,38	804,68
256	1001,22	263,42	1003,91	774,54
512	951,84	258,64	963,66	774,54
1024	886,01	261,13	939,41	761,16
2048	885,14	257,32	325,33	276,73*
5120	738,71	251,67	-	-
10240	566,11	254,09	-	-
51200	231,70	121,65	-	-
102400	147,01	88,00	-	-

większych niż 2kB oraz czwartego scenariusza i wiadomości większych niż 1kB, z tym że dla wiadomości 2kB udało się wykonać dwa testy bez wywołania duplikatu*.

Do wysłania duplikatu w przypadku protokołu CoAP bazującego na UDP dochodzi w sytuacji, gdy w ciągu 247 sekund (domyślna wartość dla EXCHANGE_LIFETIME) zostaną wysłane dwa pakiety pochodzące z tego samego adresu IP, portu źródłowego oraz zawierające ten sam identyfikator wiadomości (Message ID). W protokole CoAP każda wiadomość zawiera poza identyfikatorem wiadomości także unikatowy token. W sytuacji, gdy wysyłane jest więcej niż 1024 bajty danych to wiadomość jest dzielona na bloki. Każdy blok w wiadomości ma identyczny token, ale identyfikator wiadomości jest inny dla każdego bloku. Serwer otrzymując blok od klienta zapisuje po swojej stronie: adres IP i port źródłowy, z którego pochodzi blok, identyfikator wiadomości (Message ID), otrzymany token oraz czas otrzymania wiadomości. W odpowiedzi na żądanie klienta serwer odsyła odpowiedź z tokenem zapisanym w swojej pamięci dla adresu IP i portu źródłowego oraz identyfikatora wiadomości. Jeżeli klient wyśle drugi blok w czasie krótszym niż 247 sekund (domyślna wartość dla EXCHANGE_LIFETIME) z tego samego adresu IP, portu źródłowego z tym samym identyfikatorem wiadomości, to serwer odeśle odpowiedź z tokenem, który był użyty w poprzednim bloku. Klient taką odpowiedź odrzuci, ponieważ zawiera inny token niż ten wysłany w żądaniu do serwera. Aby temu zapobiec należy wysłać mniej niż 65536 bloków w czasie 247 sekund (domyślna wartość dla EXCHANGE_LIFETIME).

Na rysunku 5.2 przedstawiono zdarzenie wysłania duplikatu podczas wykonywania testu przesłania wiadomości o rozmiarze 10kB wykorzystując CoAP z UDP. Na pomarańczowo zaznaczono pierwszą wiadomość, natomiast na zielono wiadomość, w której pojawił się duplikat. Na czerwono podkreślono pakiet oryginalny i jego duplikat – każdy z nich ma ten sam adres źródłowy, port źródłowy oraz identyfikator wiadomości (MID). Serwer na obydwa żądania odpowiedział identycznym tokenem (TKN) – zaznaczono czarną strzałką. Po otrzymaniu złego tokenu klient wysyłał retransmisję, ponieważ odpowiedź była inna niż oczekiwana.

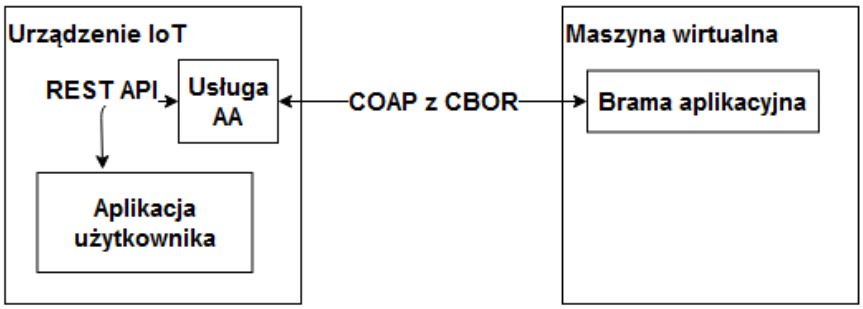


Rysunek 5.2: Duplikat wiadomości przesłanej od klienta do serwera

5.2.2 Badanie wydajności protokołu w urządzeniu IoT z Raspberry Pi

Celem badania była ocena wydajności protokołu LAAFFI dla różnych wielkości przesyłanych wiadomości oraz określenie wpływu na wydajność stosowanych protokołów kryptograficznych.

Badanie polegało na wykonaniu czterech testów. Każdy test polegał na wysłaniu 50000 wiadomości składających się ze zmiennej liczby bajtów (od 128B do 102 400B). Wiadomość składała się z losowych znaków alfanumerycznych i była identyczna dla każdego testu. Na urządzeniu IoT została uruchomiona aplikacja użytkownika, która wysyłała dane do usługi AA wykorzystując REST API. Usługa AA również została uruchomiona na urządzeniu IoT. Zadaniem usługi AA było wysłanie danych wykorzystując LAAFFI do bramy aplikacyjnej. Brama aplikacyjna została uruchomiona na maszynie wirtualnej. Brama aplikacyjna nie wykonywała żadnych operacji na danych, tylko odsyłała otrzymane wiadomości do usługi AA uruchomionej na urządzeniu IoT. Usługa AA odszyfrowywała wiadomość i przekazywała wynik do aplikacji użytkownika. Aplikacja użytkownika po otrzymaniu odpowiedzi wysyłała kolejne żądanie. Schemat logiczny połączenia usług przedstawiono na rysunku 5.3.



Rysunek 5.3: Schemat połączenia logicznego pomiędzy usługami

Tabela 5.2: Algorytmy wybrane do badania wydajności LAAFFI na urządzeniach IoT. WK – wielkość klucza, WN – wielkość nonce, WS – wielkość skrótu

Numer badania	**AEAD**			**HMAC**	
	Algorytm	**WK (bit)**	**WN (bit)**	**Algorytm**	**WS (bit)**
1	AES(GCM)	256	96	SHA-256	256
2	Xoodyak	128	128	Xoodyak	128
3	Ascon	128	128	Xoodyak	128
4	XChaCha20-Poly1305	256	192	SHA-256	256

Ponieważ w LAAFFI można wykorzystać różne algorytmy kryptograficzne, to podczas przeprowadzenia tego badania sprawdzono dodatkowo jak na wydajność opracowanego protokołu wpływa wykorzystanie różnych algorytmów funkcji skrótu, AEAD i HMAC. Algorytm funkcji skrótu był identyczny, jak ten wykorzystany w funkcji HMAC. Do badania wybrano algorytmy przedstawione w tabeli 5.2.

Algorytmy Advanced Encryption Standard (AES) oraz XChaCha20-Poly1305 zostały wybrane ze względu na powszechność wykorzystania, co może umożliwić innym badaczom porównanie LAAFFI z ich rozwiązaniami. Natomiast algorytmy Ascon i Xoodyak zostały wybrane ze względu na to, że są finalistami konkursu [117] oraz tylko dla tych finalistów udało się znaleźć implementację w języku Goolang, co umożliwiło zbadanie wydajności. W dniu 7 lutego 2023 NIST ogłosił, że zwycięzcą konkursu został Ascon [118].

Do badania wydajności LAAFFI wykorzystano CoAP z TCP ponieważ dla większych wiadomości wykorzystanie protokołu CoAP z UDP wymagałoby ograniczenia wydajności co wpłynęłoby negatywnie na wyniki. W dalszych badaniach założono, że sesja TCP będzie tworzona dla każdej wiadomości, ponieważ weryfikacja wydajności dla gorszego przypadku

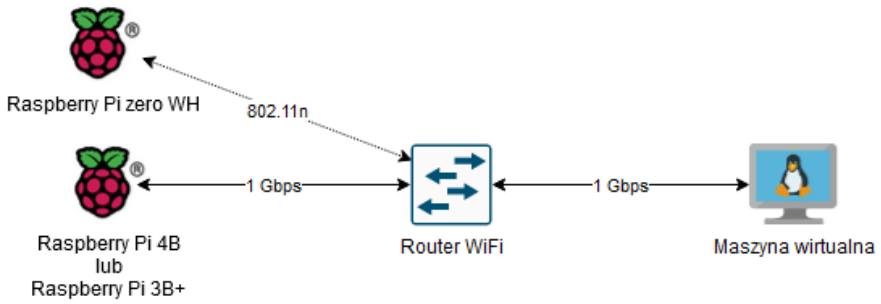
udowodni, że w przypadku wysyłania wiadomości w ramach jednej sesji uzyskano by lepsze wyniki. Badanie wydajności LAAFFI wymagało skonfigurowania nowego środowiska. Środowisko do badania wydajności opracowanego protokołu składało się z 3 komponentów:

- urządzenia IoT,
- punkt dostępowy Wireless Fidelity (Wi-Fi), przełącznik – Modem Technicolor CGA4236TCH1
- Maszyny wirtualnej z systemem operacyjnym Ubuntu 22.04 LTS

Badanie zostało wykonane dla urządzeń IoT o różnej wydajności:

- Raspberry Pi 4B 8GB pamięci RAM
- Raspberry Pi 3B+ 1GB pamięci RAM
- Raspberry Pi zero WH 512MB pamięci RAM

Szczegółowe zestawienie parametrów wykorzystanych komponentów przedstawiono w tabelach A1, A2, A3, A4, A5 zamieszczonych w załączniku. Schemat połączenia sieciowego komponentów przedstawiono na rysunku 5.4. Pomiędzy urządzeniem Raspberry Pi zero WH a punktem dostępowym Wi-Fi wykorzystano połączenie bezprzewodowe wykorzystujące standard The Institute of Electrical and Electronics Engineers (IEEE) 802.11n ze względu na to, że to urządzenie IoT nie jest wyposażone z przewodowy interfejs sieciowy. W roli punktu dostępowego wykorzystano urządzenie Technicolor CGA4236TCH1, który posiada także interfejsy Ethernet, które wykorzystano do podłączenia pozostałych urządzeń IoT.



Rysunek 5.4: Schemat połączenia sieciowego pomiędzy urządzeniami IoT a serwerem CoAP

Badanie wydajności opracowanego protokołu – badanie numer 1

W pierwszym badaniu wykorzystano AES [174] w trybie Galois/Counter Mode (GCM) [51]. AES jest symetrycznym szyfrem blokowym, który został wybrany przez NSA jako zalecany do użycia w zapewnieniu poufności informacji o każdej klasyfikacji [10]. NSA zaleca wykorzystanie kluczy o długości 256 bit, dlatego w przeprowadzonych badaniach wykorzystano klucz o takiej długości. Uznaje się, że AES o długości 256 bitów jest bezpieczny w przypadku ataków z wykorzystaniem obecnie dostępnych komputerów kwantowych [28, 150]. Wykorzystanie trybu pracy GCM gwarantuje integralność zaszyfrowanych danych. Klucz jest tworzony zgodnie ze sposobem opisanym w sekcji 3.1. W tym badaniu wykorzystano algorytm SHA-256 jako algorytm funkcji skrótu oraz algorytm wykorzystany w HMAC. Wykorzystanie tego algorytmu umożliwi uzyskanie 256-bitowego klucza wymaganego przez AES. W tabeli 5.3 przedstawiono wyniki badania.

Tabela 5.3: Wyniki badania wydajności opracowanego protokołu – badanie numer 1

Liczba bajtów	**Liczba operacji na sekundę**		
w wiadomości [B]	**Raspberry Pi 4B**	**Raspberry Pi 3B+**	**Raspberry Pi zero WH**
128	243,25	247,02	85,95
256	244,99	243,06	84,69
512	243,82	233,98	76,13
1024	240,87	217,17	68,15
2048	234,38	178,59	55,84
5120	206,1	127,72	37,65
10240	141,39	89,02	26,96
51200	49,44	26,73	7,73
102400	33,79	14,8	-

Badanie wydajności opracowanego protokołu – badanie numer 2

Xoodyak [43] jest jednym algorytmów, które dostały się do finału konkursu organizowanego przez NIST na lekki protokół kryptograficzny odpowiedni dla urządzeń o gorszych parametrach sprzętowych [117]. Xoodyak jest nazwą nie tylko dla algorytmu uwierzytelnionego szyfrowania z powiązanymi danymi (AEAD), czyli poza zapewnieniem poufności danych zapewnia

także ich integralność, ale także dla algorytmu funkcji skrótu oraz kod uwierzytelniania wiadomości (MAC). Każdy z tych algorytmów wykorzystuje nowy schemat permutacji nazwany XooDoo oraz architekturę gąbki (ang. sponge construction). Autorami tego rozwiązania jest *KeccakTeam* – ten sam zespół, który opracował algorytm SHA-3. W przypadku szyfrowania Xoodyak wymaga 128-bitowego klucza. Klucz ten jest tworzony w ten sam sposób jak przestawiono w 3.1. W roli algorytmu HMAC wykorzystano odpowiedni tryb tego algorytmu – Xoodyak.

W tabeli 5.4 przedstawiono wyniki badania. Wykorzystana implementacja Xoodyak nie pochodziła od autorów tego algorytmu, dlatego może zawierać błędy co przyznał nawet sam autor implementacji [88]. Zastosowana implementacja Xoodyak nie była także zoptymalizowana co mogło przyczynić się do gorszych wyników.

Tabela 5.4: Wyniki badania wydajności opracowanego protokołu – badanie numer 2

Liczba bajtów	**Liczba operacji na sekundę**		
w wiadomości [B]	**RPi 4B**	**RPi 3B+**	**RPi zero WH**
128	251,66	257,36	62,84
256	251,33	251,50	61,55
512	250,49	241,60	57,11
1024	247,29	224,53	52,80
2048	241,22	186,95	43,47
5120	230,28	131,42	31,57
10240	159,38	92,74	23,88
51200	53,73	28,65	7,70
102400	32,91	15,95	-

Badanie wydajności opracowanego protokołu – badanie numer 3

Ascon [50] jest zwycięzcą konkursu NIST na lekki protokół kryptograficzny odpowiedni dla urządzeń o gorszych parametrach sprzętowych [117]. Ascon zapewnia uwierzytelnione szyfrowanie z powiązanymi danymi (AEAD) danych. Autorzy Ascon poza algorytmem uwierzytelnionego szyfrowania z powiązanymi danymi opracowali także algorytm funkcji skrótu o tej samej nazwie wykorzystujący ten sam sposób permutacji. Szyfrowanie algorytmem Ascon wymaga klucza o długości 128 bit. W badaniu jako algorytm HMAC wykorzystano ten sam

algorytm co w poprzednim badaniu.

W tabeli 5.5 przedstawiono wyniki badania. Wykorzystana implementacja nie pochodzi od autorów tego algorytmu, dlatego też może zawierać błędy oraz może nie być zoptymalizowana pod kątem wydajności. Nie udało się wykonać badań dla Raspberry Pi zero WH, ponieważ nie znaleziono implementacji algorytmu Ascon dla architektury tego urządzenia. Na podstawie wyników można stwierdzić, że dla urządzeń Raspberry Pi 4B i 3B+ wykorzystanie tego zestawu algorytmów dla opracowanego protokołu jest wydajniejsze niż wykorzystanie Xoodyak w roli AEAD i HMAC.

Tabela 5.5: Wyniki badania wydajności opracowanego protokołu – badanie numer 3

Liczba bajtów	**Liczba operacji na sekundę**		
w wiadomości [B]	**RPi 4B**	**RPi 3B+**	**RPi zero WH**
128	253,40	261,94	-
256	252,10	255,26	-
512	252,16	247,34	-
1024	250,85	233,18	-
2048	243,46	207,67	-
5120	234,11	158,48	-
10240	186,43	117,06	-
51200	63,08	39,04	-
102400	39,30	22,92	-

Badanie wydajności opracowanego protokołu – badanie numer 4

W ostatnim badaniu wykorzystano algorytm uwierzytelnionego szyfrowania z powiązanymi danymi (AEAD) – XChaCha20-Poly1305 [11]. Według badań XChaCha20-Poly1305 jest wydajniejszy niż AES w przypadku wykonywania kodu bez akceleracji sprzętowej (np. AES-NI) [139]. XChaCha20-Poly1305 wymaga klucza o wielkości 256 bit. Algorytm ten różni się od powszechnie wykorzystywanego ChaCha20-Poly1305 tym, że wartość nonce ma wielkość 192 bit zamiast 96 bit. Sam algorytm Chacha20-Poly1305 jest powszechnie wykorzystywany w takich protokołach jak TLS, DTLS, SSH i innych [111, 132]. Jako funkcji skrótu wybrano SHA-256, i na ten sam algorytm wykorzystuje HMAC.

W tabeli 5.6 przedstawiono wyniki badania. Dla urządzeń Raspberry Pi 4B i 3B+ wyko-

rzystanie tego zestawu algorytmów bezpieczeństwa jest wydajniejsze niż wykorzystanie AES-GCM-256 i SHA-256. Natomiast dla Raspberry Pi zero WH można zauważyć niewiele gorszą wydajność niż dla badania wykorzystującego AES-GCM-256 i SHA-256.

Tabela 5.6: Wyniki badania wydajności opracowanego protokołu – badanie numer 4

Liczba bajtów	**Liczba operacji na sekundę**		
w wiadomości [B]	**RPi 4B**	**RPi 3B+**	**RPi zero WH**
128	251,37	257,36	81,60
256	251,58	254,61	79,82
512	249,58	247,27	76,35
1024	249,80	227,89	69,27
2048	243,52	203,98	50,44
5120	238,17	158,54	39,35
10240	190,41	115,76	28,64
51200	66,71	39,60	9,50
102400	40,58	23,21	-

5.3 Badania wydajnościowe operacji wykonywanych z udziałem rejestru rozproszonego

Głównym celem tego badania jest sprawdzenie wydajności oraz skalowalności rejestru rozproszonego. Określenie wydajności rejestru rozproszonego w kontekście możliwości wykonania operacji przewidzianych w LAAFFI jest kluczowe. Bez osiągnięcia odpowiedniej wydajności rejestru rozproszonego, wykorzystanie opracowanego protokołu nie ma sensu. Podczas badania sprawdzono jak liczba organizacji, liczba węzłów w organizacji oraz parametry węzłów wpływają na przepustowość mierzoną liczbą wykonywanych operacji w jednostce czasu (sekundy) (ang. Operations Per Second (OPS)). Zbadano operacje, które są wymagane przez LAAFFI: zaszyfrowanie i odszyfrowanie danych, utworzenie klucza do komunikacji pomiędzy urządzeniami IoT, rejestracja urządzenia. Badanie podzielono na testy. Pojedynczy test to wykonanie 1 operacji 10 000 razy dla konkretnej konfiguracji węzłów oraz sieci rejestru rozproszonego.

Każdy z testów został wykonany dwukrotnie. Wszystkie testy zostały wykonane z użyciem platformy Amazon Web Services (AWS) Elastic Compute Cloud (EC2), co umożliwia ich po-

Tabela 5.7: Parametry węzła

Parametr	**Wartość**
Instancja AWS	c5.xlarge lub c5.2xlarge
CPU	Intel Xeon Platinum 8000 series 4vCPU lub 8vCPU
Pamięć RAM (GB)	8 or 16
Dysk (Gib)	50 EBS
Sieć (Gbps)	Do 10
OS	Ubuntu 20.04.1
Wersja HLF	1.4.8
Wersja Golang	1.15.2
Wersja Docker	1.5-2

wtórzenie z użyciem innej konfiguracji i porównanie wyników z przedstawionymi w tym rozdziale. Tak jak wspomniano w rozdziale 2.4, do implementacji rejestru rozproszonego wybrano Hyperledger Fabric. W badaniach tych wykorzystano Hyperledger Caliper, który umożliwia testowanie wydajności Hyperledger Fabric. Narzędzie to skonfigurowano tak, aby wywoływało konkretną operację na sieci Hyperledger Fabric z odpowiednimi parametrami z określoną częstotliwością. Aby zbadać wydajność i sprawdzić jak się ona zmienia dla różnych konfiguracji, utworzone zostało środowisko badawcze składające się z:

- jednego węzła z Hyperledger Caliper,
- węzłów Hyperledger Fabric. Każdy węzeł miał uruchomione usługi Peer i Ordering. Liczba węzłów oraz ich konfiguracja sprzętowa zależała od testu. Parametry konfiguracji węzłów Hyperledger Fabric została przedstawiona w tabeli 5.7. Natomiast parametry konfiguracyjne sieci Hyperledger Fabric przedstawiono w tabeli 5.8.

W tabeli 5.9 przedstawiono wyniki dla operacji, które nie wymagały osiągnięcia konsensusu, ponieważ nie zmieniały stanu rejestru rozproszonego. Wydajność Hyperledger Fabric w tych przypadkach oznacza liczbę operacji na sekundę, która może zostać obsłużona przez jedną organizację. To znaczy, że w przypadku dwóch organizacji, każdą z tych wartości należy pomnożyć przez 2, aby uzyskać wydajność całej sieci rejestru rozproszonego. W tabeli 5.10 przedstawiono wyniki dla operacji, które wymagały dodania nowej transakcji do rejestru rozproszonego, a tym samym osiągnięcia konsensusu. W przypadku tego typu operacji wyniki

Tabela 5.8: Konfiguracja Hyperledger Fabric

Parametr	**Wartość**
Liczba organizacji	Od 2 do 4
Liczba węzłów w organizacji	Od 2 do 4
Polityka zatwierdzenia	Jeden węzeł z każdej organizacji
Wartość BatchTimeout	2s
Wartość MaxMessageCount	1000
Wartość AbsoluteMaxBytes	99MB
Wartość PreferredMaxBytes	5MB
Konsensus dla usługi Ordering	EtcdRaft
Liczba węzłów z usługą Ordering	Każdy węzeł z usługą Peer miał również uruchomioną usługę Ordering

oznaczają liczbę operacji na sekundę, która może zostać obsłużona przez całą sieć Hyperledger Fabric.

Wszystkie operacje z wyjątkiem zmiany uprawnień urządzenia IoT oraz rejestracji urządzenia IoT nie wymagają osiągnięcia konsensusu, ponieważ nie są dodawane żadne nowe transakcje do rejestru rozproszonego. W przypadku tych operacji liczba operacji na sekundę rośnie liniowo wraz ze wzrostem liczby węzłów w organizacji. Liczba organizacji zwiększa również liczbę operacji na sekundę, która może być wykonana przez całą sieć. Podwojenie ilości pamięci RAM i liczby vCPU również zwiększyło wydajność, ale w przypadku przeprowadzonych testów nie była ona podwojona.

Dla operacji, które wymagają konsensusu, zwiększenie liczby węzłów w organizacji, również przyczynia się do zwiększenia wydajności. Jednak dodanie nowej organizacji zmniejsza wydajność całej sieci. Jest to spowodowane zastosowaną polityką zatwierdzania, która wymaga, aby przynajmniej jeden węzeł z każdej organizacji zatwierdził transakcję. Podwojenie ilości pamięci RAM i liczby vCPU również zwiększyło wydajność, ale tak jak w przypadku testów, które nie wymagało konsensusu, nie była ona podwojona.

Tabela 5.9: Wydajność Hyperledger Fabric – operacje niewymagające konsensusu

Operacja	**Rozmiar wiadomości**	**Operacje/s**							
		4 vCPU, 8GB RAM					**8 vCPU, 16GB RAM**		
		2 org			**3 org**	**4 org**	**2 org**		
		2 węzły	**3 węzły**	**4 węzły**	**2 węzły**	**2 węzły**	**2 węzły**	**3 węzły**	**4 węzły**
Zaszyfrowanie danych	20B	1643,40	2599,30	3197,55	1449,60	1324,65	2641,50	3346,65	5172,60
	100B	1649,55	2548,85	3122,25	1430,40	1299,80	2574,35	3297,95	5136,15
	1kB	1574,80	2304,75	3098,75	1549,70	1299,30	2549,40	3248,20	5190,10
	15kB	1199,80	1655,15	2310,95	1160,25	1024,60	2396,45	2898,30	4618,50
	30kB	999,80	1574,45	2148,35	924,60	824,90	2198,95	2706,65	4237,70
	50kB	837,15	1274,40	1699,10	774,85	699,90	1863,70	2497,55	3471,05
	100kB	628,40	901,05	1154,90	577,70	524,90	1341,35	1881,25	1990,20
Odszyfrowanie danych	20B	1669,60	2475,30	3147,70	1449,75	1324,75	2133,25	3180,40	5291,85
	100B	1669,70	2498,65	3148,70	1449,60	1299,85	2102,95	3112,50	4304,10
	1kB	1574,80	2348,75	3097,80	1403,90	1299,85	2549,80	2946,30	4343,55
	15kB	1199,85	1847,65	2490,85	1149,75	999,90	2379,60	2798,20	4143,95
	30kB	1074,70	1556,45	2006,80	974,85	874,90	2263,70	2598,80	4092,70
	50kB	899,85	1362,00	1768,85	774,90	767,05	1805,50	2398,20	3641,80
	100kB	649,80	934,90	1171,65	594,90	524,90	1432,15	2171,60	2778,00
Wygenerowanie klucza	-	1654,60	2390,95	3297,35	1399,75	1299,70	1974,25	3223,30	4343,20
Weryfikacja uprawnień	-	1639,65	2498,30	3278,85	1474,55	1434,60	2447,00	3224,00	5184,75

Tabela 5.10: Wydajność Hyperledger Fabric – operacje wymagające osiągnięcia konsensusu

Operacja	**Operacje/s**							
	4 vCPU, 8GB RAM					**8 vCPU, 16GB RAM**		
	2 org			**3 org**	**4 org**	**2 org**		
	2 węzły	**3 węzły**	**4 węzły**	**2 węzły**	**2 węzły**	**2 węzły**	**3 węzły**	**4 węzły**
Zmiana uprawnień	539,40	622,55	646,10	498,65	443,10	784,30	914,55	918,05
Rejestracja urządzenia	500,60	564,00	589,20	464,80	412,70	729,45	793,15	863,80

5.4 Badanie nadmiaru przesyłanych danych pomiędzy urządzeniem IoT a bramą aplikacyjną

Celem tego badania jest sprawdzenie nadmiaru przesyłanych danych pomiędzy urządzeniem IoT a bramą aplikacyjną dla opracowanego protokołu. Istotną cechą opracowanego protokołu, jest nieduża nadmiarowość przesyłanych danych pomiędzy urządzeniem IoT a bramą aplikacyjną. Zbadano także, jak LAAFFI wypada pod tym względem z innymi powszechnie wykorzystywanymi protokołami do zabezpieczenia transmisji klient-serwer w sieciach IoT:Datagram Transport Layer Security (DTLS) oraz TLS. W każdym teście sprawdzono nadmiar przesyłanych danych dla różnych scenariuszy:

- Wykorzystanie protokołu CoAP wraz z DTLS, uwierzytelnianie z wykorzystaniem PKI.
- Wykorzystanie protokołu CoAP wraz z DTLS, uwierzytelnianie z wykorzystaniem Pre-shared key (PSK).
- Wykorzystanie protokołu CoAP wraz z TLS, uwierzytelnianie z wykorzystaniem PKI.
- Protokół LAAFFI wykorzystujący CoAP z TCP oraz AES-GCM-256 i HMAC-SHA-256.
- Protokół LAAFFI wykorzystujący CoAP z UDP oraz AES-GCM-256 i HMAC-SHA-256.
- Protokół CoAP wykorzystujący TCP oraz CBOR. Przesłanie danych zaszyfrowanych z wykorzystaniem AES, uwierzytelnianie stron na podstawie PSK.
- Protokół CoAP wykorzystujący UDP oraz CBOR. Przesłanie danych zaszyfrowanych z wykorzystaniem AES, uwierzytelnianie stron na podstawie PSK.

W tym celu zbudowano środowisko testowe składające się z:

- Urządzenia IoT – Raspberry Pi 4B 8GB RAM
- Modem Technicolor CGA4236TCH1 w roli przełącznika
- Maszyny wirtualnej z systemem operacyjnym Ubuntu 22.04 LTS

Parametry komponentów zostały przedstawione w tabelach: A1, A4, A5 dołączonych w załączniku. Schemat połączenia komponentów został przedstawiony na rysunku 5.5. Do połączenia urządzenia IoT z przełącznikiem, a także przełącznika z komputerem, na którym

była uruchomiona maszyna wirtualna użyto kabla Ethernet kat. 5e. Na urządzeniu IoT uruchomiono program – klient, który wysyłał żądanie do serwera uruchomionego na maszynie wirtualnej. Klient wysyłał pakiety zawierające różne długości tekstu, który został wcześniej wygenerowany tak, aby zawierał losowe ciągi znaków. Serwer po otrzymaniu wiadomości odsyłał odpowiedź zawierającą tylko 2 znaki: 'OK'. Na maszynie wirtualnej uruchomiony był także program Wireshark w wersji 4.1, który zbierał cały ruch sieciowy. Na podstawie wyników programu Wireshark ustalono liczbę pakietów oraz liczbę przesłanych bajtów.

Rysunek 5.5: Schemat połączenia sieciowego pomiędzy urządzeniem IoT a serwerem badanego protokołu

W tabelach 5.12, 5.11 przestawiono wyniki tego badania.

Analizując dane z tabeli 5.11 można zauważyć, że liczba przesłanych bajtów i pakietów nie różni się bardzo pomiędzy opracowanym protokołem oraz rozwiązaniem, w którym jest zastosowany jeden klucz – PSK. LAAFFI w porównaniu do tradycyjnego podejścia wykorzystującego TLS oraz PKI ma mniejszy nadmiar przesyłanych danych. Widocznie jest to szczególnie dla wiadomości zawierających mniej niż 50kB danych. Wynika to z konieczności przesłania certyfikatów w celu uwierzytelniania stron oraz ustanowienia klucza do dalszej komunikacji.

Podobna sytuacja ma miejsce w przypadku wyników w tabeli 5.12. Liczba przesłanych pakietów jest identyczna dla LAAFFI oraz rozwiązania wykorzystującego jeden stały klucz – PSK. Porównując nadmiar przesyłanych danych LAAFFI z DTLS można zauważyć, że LAAFFI posiada znacznie mniejszy nadmiar przesyłanych danych.

Porównując dane z tabeli 5.11 z danymi z tabeli 5.12 widać wyraźnie, że w przypadku mniejszej liczby bajtów koniecznej do przesłania, UDP wymaga przesłania mniejszej liczby pakietów i bajtów niż TCP. Wynika to z konieczności ustanowienia połączenia pomiędzy klientem i serwerem w protokole TCP. Odwrotna sytuacja ma miejsce jednak w przypadku przesyłania większej liczby bajtów. Jest to spowodowane tym, że w przypadku CoAP wykorzystującego UDP w jednym pakiecie można przesłać maksymalnie 1024B danych [163], natomiast w przypadku CoAP wykorzystującego TCP takiego ograniczenia nie ma. Podczas

Tabela 5.11: Wyniki badania nadmiaru przesyłanych danych dla protokołów wykorzystujących TCP

Liczba bajtów	**LAAFFI**		**CoAP + CBOR + AES + PSK**		**TLS + PKI**	
w wiadomości [B]	**L. pakietów**	**L. bajtów**	**L. pakietów**	**L. bajtów**	**L. pakietów**	**L. bajtów**
128	13	1239	14	1263	21	5135
256	13	1369	13	1327	22	5331
512	14	1691	13	1583	22	5588
1024	14	2203	13	2095	21	6032
2048	14	3227	14	3185	21	7056
5120	14	6299	16	6389	21	10156
10240	15	11485	18	11641	26	15632
51200	32	53567	38	53921	36	57382
102400	43	105497	34	104861	40	108944

Tabela 5.12: Wyniki badania nadmiaru przesyłanych danych dla protokołów wykorzystujących UDP

Liczba bajtów	**LAAFFI**		**CoAP + CBOR + AES + PSK**		**DTLS + PSK**		**DTLS + PKI**	
w wiadomości [B]	**L. pakietów**	**L. bajtów**	**L. pakietów**	**L. bajtów**	**L. pakietów**	**L. bajtów**	**L. pakietów**	**L. bajtów**
128	2	431	2	389	9	1276	10	4139
256	2	560	2	518	8	1323	10	4267
512	2	816	2	774	9	1660	10	4523
1024	4	1457	4	1415	8	2172	10	5035
2048	6	2603	6	2561	11	3399	12	6262
5120	12	6041	12	5999	17	7059	18	9922
10240	22	11771	22	11729	27	13268	28	16022
51200	102	57524	102	57638	107	62135	108	64889
102400	202	115183	202	115141	207	123335	208	126089

przeprowadzenia tego badania zauważono, że udało się przesłać 15kB danych w jednym pakiecie TCP. Dodatkowo w przypadku TCP każdy pakiet jest potwierdzany przez drugą stronę, co również zwiększa liczbę wysłanych pakietów, a tym samym przesłanych bajtów.

5.5 Modelowanie wydajności podstawowych operacji opracowanego protokołu z wykorzystaniem czasowych kolorowanych sieci Petriego

5.5.1 Kolorowe czasowe sieci Petriego

W ramach oceny opisywanego rozwiązania zbudowano także model sieci Petriego, który można wykorzystać do weryfikacji wydajności różnych konfiguracji rejestru rozproszonego.

Sieci Petriego i wspierające je narzędzia komputerowe służą do modelowania, symulacji i analizy modeli. Istnieje wiele przykładów ich wykorzystania w różnych obszarach [179, 180], takich jak modelowanie protokołów komunikacyjnych, modelowanie telekomunikacyjnych systemów mobilnych czy projektów oprogramowania systemów rozproszonych. Wykorzystanie sieci Petriego ma również miejsce w przypadku systemów Internetu rzeczy [199] czy analizy rejestrów rozproszonych [162]. Praktyczne zastosowanie sieci Petriego obejmowało również wiele etapów budowy systemu od wymagań po projektowanie, walidację i implementację.

Sieć Petriego przedstawiana jest w formie grafu dwudzielnego, składającego się z miejsc i przejść. Łuki w grafie mogą łączyć tylko węzły, które są różnych typów. Model sieci Petriego poza graficzną reprezentacją danego systemu umożliwia wykonanie symulacji pracy sieci, a dzięki temu, taki model jest wirtualnym prototypem modelowanego systemu. Element, który przechodzi pomiędzy miejscami nazywany jest znacznikiem. Przejście jest aktywne (czyli może zostać odpalone) w momencie, gdy każde z jego miejsc wejściowych zawiera minimum jeden znacznik.

Sieci Petriego mogą być zweryfikowane pod względem spełnienia własności. Pierwszą własnością jest osiągalność, czyli możliwość osiągnięcia stanu M_0 z innego stanu M_1. Drugą własnością sieci Petriego jest żywotność, określająca, że każde przejście lub miejsce może być wykonane. Trzecią własnością sieci Petriego jest ograniczoność sieci określającą nieprzekroczenie zasobów w systemie. Zachowawczość sieci jest osiągnięta jeżeli sieć ma stałą liczbę znaczników. Ostatnią własnością sieci jest odwracalność, czyli możliwość powrotu do stanu poprzedniego z dowolnego innego stanu.

Opracowany model zbudowany został w wykorzystaniem czasowej kolorowej hierarchicznej sieci Petriego. Uwzględnienie czasu tj. rozszerzenia sieci Petriego o aspekt czasowy, umożliwia modelowanie upływu czasu pomiędzy wykonaniem poszczególnych przejść sieci Petriego [179]. Przejścia w czasowych sieciach Petriego mogą mieć określone czasy aktywacji, co pozwala na modelowanie czasu potrzebnego do wykonania danej operacji (czy przebywania systemu w pewnych stanach). Oprócz czasów trwania przejść, czasowe sieci Petriego mogą uwzględniać również wprowadzenie opóźnienia. Jest to istotne w modelowaniu upływu czasu w systemach równoległych lub rozproszonych. Analiza czasowa sieci Petriego, pozwala na symulowanie zachowania systemu w czasie, co pozwala na przewidywanie jego działania i ocenę wskaźników wydajnościowych w szczególności czasu trwania operacji, czasu oczekiwania na aktywację przejścia czy liczby znaczników oczekujących na aktywację przejścia (długość kolejki).

Kolorowa sieć Petriego umożliwia zdefiniowanie typów znaczników, które odpowiadają typom danych występujących w popularnych językach programowania [180]. Umożliwia to wykonywanie różnych operacji na znacznikach. Każde miejsce ma przypisany określony kolor (typ danych) i tylko znaczniki z tym typem może przechowywać. Manipulowanie kolorem odbywa się poprzez wyrażenia zastosowane w przejściach. W sieci kolorowanej, aby przejście było aktywne, jego miejsca wejściowe muszą zawierać nie tylko odpowiednią liczbę znaczników, ale również odpowiednie wartości. Kolorową sieć Petriego wykorzystano w celu zdefiniowania typu znaczników, co umożliwiło uzyskanie dokładnych statystyk podczas symulacji modelu, pozwoliło także na uporządkowanie modelu. Wykorzystanie kolorowych sieci Petriego pozwala na uwzględnienie różnych właściwości obiektów, które mogą być zmieniane podczas przechodzenia znacznika przez przejścia oraz mogą wpływać na zachowanie całej sieci Petriego.

Hierarchizacja sieci pozwala na łączenie wielu niehierarchicznych sieci (zwanych modułami lub stronami) za pomocą podstawionych przejść lub fuzji [180], czyli miejsc łączenia stron. Strony mogą być uporządkowane hierarchicznie, umożliwiając podejście „góra-dół" podczas konstruowania modeli sieci Petriego. Podstawione przejście pozwala na połączenie podstron z nadstronami poprzez gniazda (miejsca w nadstronach) i miejsca portowe (miejsca w podstronach). Fuzja natomiast jest zbiorem miejsc, które są nierozróżnialne tzn. reprezentują jedno miejsce, ale narysowane indywidualnie na różnych stronach. Znacznik, który jest dodawany (usuwany) do miejsca należącego do fuzji jest automatycznie dodawany do (usuwany z) innych miejsc należących do tej samej fuzji. Miejsca należące do fuzji mogą znajdować się na tej samej stronie lub na różnych stronach.

W rozprawie do zamodelowania i symulacji zapisu transakcji w węzłach rejestru rozproszo-

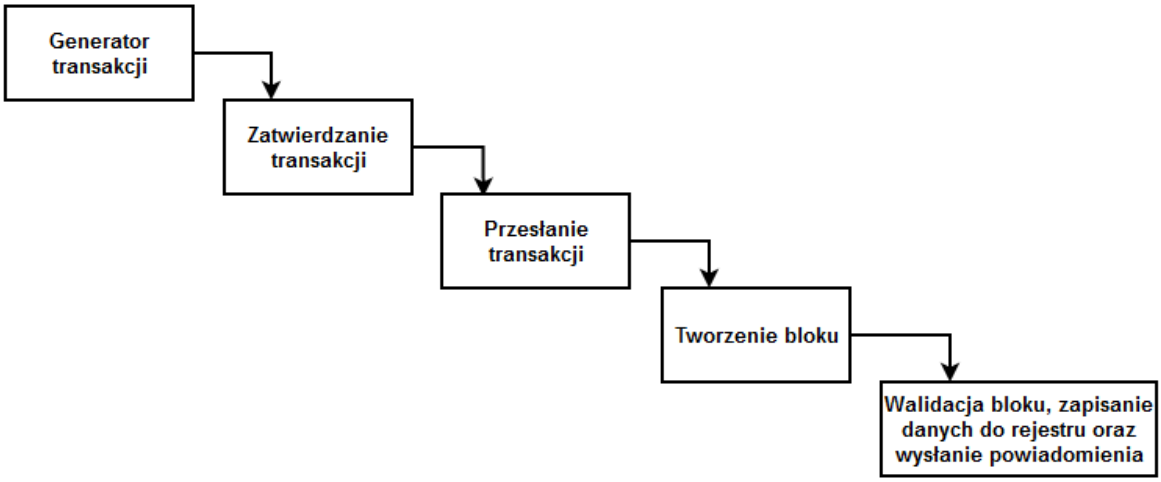
nego wykorzystano narzędzie CPN Tools. Narzędzie to umożliwia utworzenie i analizę modelu czasowej kolorowej hierarchicznej sieci Petriego, a także na przeprowadzenie jej symulacji. Do modelowania wykorzystano także język CPN ML. Nie jest on elementem formalnym definicji sieci Petriego, ale został opracowany dla wykorzystanego narzędzia – CPN Tools. Wywodzi on się z języka Standard ML. Wykorzystanie języka wysokiego poziomu pozwoliło na większe manipulowanie znacznikami, zbieranie statystyk dla różnych modelowanych elementów, a także sparametryzowanie modelu.

5.5.2 Budowa i opis modelu

Podstawowym celem tej części badań było skonstruowanie modelu zapisu transakcji w rejestrze, który pozwalałby na przeprowadzenie badań wydajnościowych funkcjonowania Hyperledger Fabric w zależności m.in. od liczby węzłów rejestru rozproszonego, liczby oraz rozmiaru transakcji przesyłanych do dodania, oraz od parametru bloku, który ma zostać zapisany w rejestrze. Operacja zapisu transakcji w rejestrze wymaga osiągnięcia konsensusu pomiędzy organizacjami, a także spełnienia warunków utworzenia bloku zgodnie z ustalonymi w polityce kanału zasadami. Zasady te określają, jakie i/lub ile organizacji musi podpisać transakcję, aby została ona dodana do rejestru rozproszonego. W zasadach zapisane są również minimalne właściwości bloku z transakcjami, aby blok został utworzony i dodany do rejestru rozproszonego. Opracowanie modelu zapisu transakcji w rejestrze rozproszonym pozwoli na badanie parametrów rejestru rozproszonego dla różnej konfiguracji polityki kanału w zależności od liczby zgłaszanych transakcji i ich rozmiaru. Model sieci Petriego musi dostarczyć informacji o czasie dodania X transakcji w rejestrze rozproszonym, liczbie bloków potrzebnych do zapisania X transakcji, średniej liczby transakcji w bloku, średniego czasu obsługi transakcji (opóźnienia dodania transakcji do rejestru rozproszonego), średniej długości kolejki na węzłach rejestru rozproszonego. Jeżeli średnia długość kolejki będzie większa niż jeden, oznacza to możliwość osiągnięcia maksymalnej przepustowości rejestru rozproszonego.

Aby zamodelować proces dodawania transakcji do rejestru rozproszonego, podzielono go na pięć etapów, gdzie każdy etap przedstawiono jako oddzielny moduł — strona sieci Petriego. Pierwszym etapem jest generowanie transakcji i wysłanie transakcji do organizacji. Drugi etap zawiera zatwierdzanie transakcji przez węzeł organizacji. Liczba stron z drugim etapem jest zależna od liczby organizacji. W trzecim etapie transakcja przesłana jest do usługi budującej blok. Czwarty etap to proces budowania bloku. Natomiast ostatni etap to proces weryfikacji,

zatwierdzania i dodawania bloku do rejestru rozproszonego. Hierarchię stron przedstawiono na rysunku 5.6. Każda z typów stron przedstawionych na rysunku 5.6 może wystąpić wiele razy, ale zawsze w tym samym miejscu w hierarchii. Przygotowany proces dodawania transakcji jest zgodny z procesem wykonywanym przez Hyperledger Fabric i nie jest odpowiedni dla innych implementacji rejestru rozproszonego.

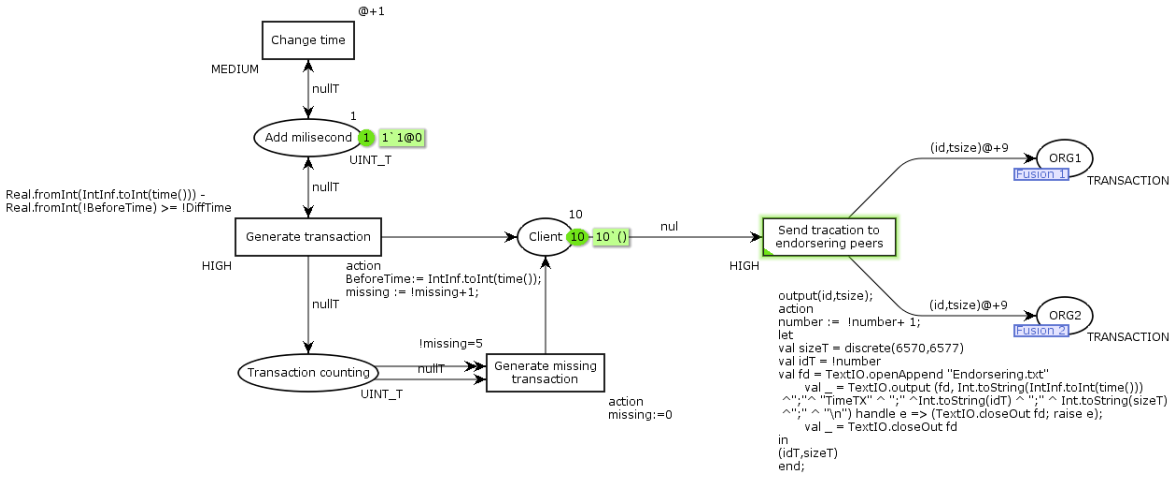


Rysunek 5.6: Wykres hierarchii stron w opracowanym modelu

Podczas budowy modelu przyjęto, że inne parametry środowiska np. przepustowość sieci są niezmienne w czasie. Założono także, że wszystkie parametry sprzętowe są identyczne dla wszystkich węzłów. W przypadku polityki zatwierdzania transakcji założono, że wystarczy, aby jeden z węzłów organizacji potwierdził transakcję, oraz że każda transakcja będzie potwierdzona przez każdą organizację.

Pierwszą stroną jest *Generator transakcji*. Na tej stronie został zamodelowany etap tworzenia transakcji przez klienta. Do ważnych właściwości, jakie ten etap musi spełnić, należy generowanie znaczników z odpowiednią częstotliwością oraz nadawanie każdemu znacznikowi odpowiedniego koloru określającego numer transakcji i wielkość tej transakcji. Ponieważ założono, że jedna jednostka czasu symulacji odpowiada 1 milisekundzie spowodowało to, że dla wartości częstotliwości generowania znaczników niebędącymi dzielnikami wartości 1000 konieczne jest generowanie dodatkowego znacznika w pewnych jednostkach czasu. Na tej stronie znacznik jest reprezentację pojedynczej transakcji. Na rysunku 5.7 przedstawiono stronę modelu odpowiedzialną za proces generowania znaczników.

Przejście *Generate missing transactions* i miejsce *Transaction counting* wraz z przyległymi łukami jest dodane do strony tylko w sytuacji konieczności generowania dodatkowych

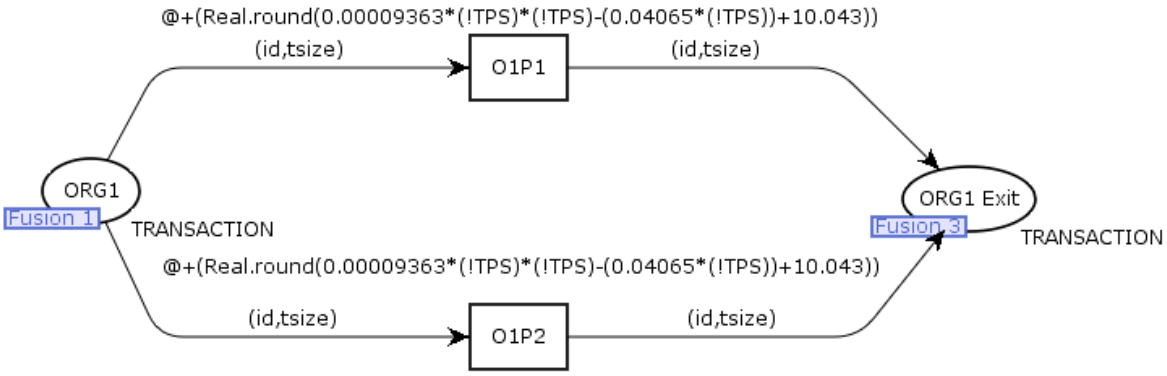


Rysunek 5.7: Model sieci Petriego odpowiedzialny za generowanie transakcji (znaczników)

znaczników dla wartości częstotliwości generowania znaczników niebędących dzielnikami 1000. W innym przypadku nie zachodzi konieczność generowania dodatkowych znaczników. Przejście *Change time* jest wykonywane w momencie, gdy nie można wykonać innej operacji. Przejście to w każdej jednostce symulacji umożliwia utworzenie nowego znacznika w miejscu *Add milisecond*. Pojawienie się znacznika w tym miejscu wymusza sprawdzenie warunku dozoru w przejściu *Generate transaction*. Dzięki temu przejściu możliwe jest generowanie nowych znaczników, ponieważ CPN Tools sprawdza aktywne przejścia tylko w momencie utworzenia znacznika w miejscu poprzedzającym. Jeżeli warunek dozoru w przejściu *Generate transaction* nie będzie spełniony to w momencie pojawienia się znacznika w miejscu *Add milisecond* spowodowałoby, że przejście *Generate transaction* nigdy nie byłoby aktywne, a tym samym nowe znaczniki nie byłyby generowane. Znakowanie początkowe występuje w miejscach: *Add milisecond* – 1 znacznik, który umożliwia generowanie nowych znaczników, oraz *Client* – 10 znaczników. Przejście *Send transaction to endorsering peers* zmienia kolor znacznika. Od tego momentu znacznik ma swój unikatowy identyfikator oraz wartość określającą rozmiar. Wartość rozmiaru jest losowana spomiędzy wartości 6570 a 6577 bajtów. Wartość ta wynika z rozmiaru transakcji uzyskanych podczas prowadzenia badań wydajnościowych rejestru rozproszonego. Także podczas wykonywania tego przejścia ma miejsce wygenerowanie wpisu do dziennika zdarzeń na potrzeby późniejszej analizy wyników symulacji. Miejsca *ORG1* oraz *ORG2* są miejscami, które są oznaczone fuzją z miejscami na innych stronach opracowanego modelu. Wartość znacznika czasu na łuku pomiędzy *Send transaction to endorsering peers* oraz miejsc *ORG1*, *ORG2* wynika ze średniej wartości czasu przesłania transakcji pomiędzy

klientem a węzłem z usługą Peer i rolą *Zatwierdzający (Endorsering)* podczas wykonania badań wydajnościowych.

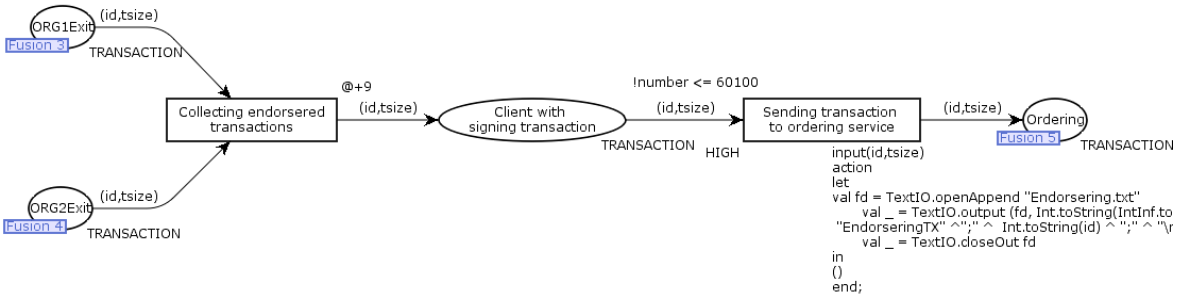
Na rysunku 5.8 przedstawiono kolejny typ strony – *Zatwierdzanie transakcji*. Model na tej stronie jest odpowiedzialny za zasymulowanie procesu zatwierdzania transakcji. Miejsce *ORG1* jest połączone poprzez fuzję z miejscem o tej samej nazwie na stronie *Generator transakcji*. Funkcja przy przejściach *O1P1*, *O1P2* służąca do wyliczenia czasu trwania tego etapu została wyznaczona na podstawie wyników badań wydajnościowych Hyperledger Fabric. Miejsce *ORG1Exit* jest połączone fuzją z miejscem na kolejnej stronie. Na rysunku 5.8 przedstawiono tylko jedno wystąpienie tej strony, w przygotowanym modelu wykorzystano dwie takie strony, po jednej dla każdej organizacji. W zaprezentowanej stronie zamodelowano tylko dwa węzły (odpowiednio *O1P1*, *O1P2*), ale model strony można rozszerzyć o kolejne.



Rysunek 5.8: Model sieci Petriego zatwierdzania transakcji w Hyperledger Fabric

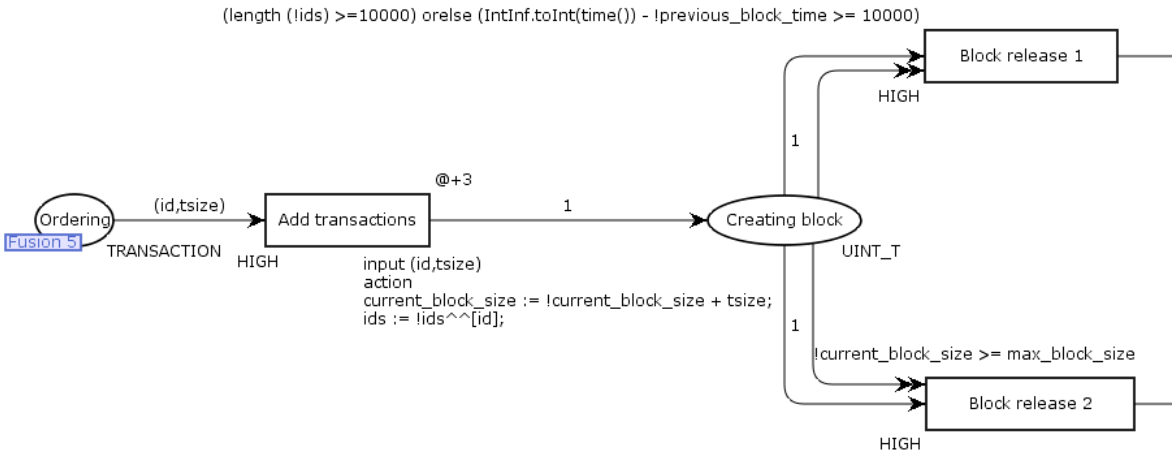
Na kolejnej stronie zamodelowano proces przesyłania transakcji zatwierdzonej do usługi Ordering. Cały model tej strony został przedstawiony na rysunku 5.9. Miejsca *ORG1Exit* i *ORG2Exit* są połączone fuzją z odpowiednimi miejscami na stronie *Zatwierdzanie transakcji*. Wartość czasu przy przejściu *Collecting endorsered transactions* została wyliczona w badaniach wydajnościowych rejestru rozproszonego. Podczas wykonywania przejścia *Collecting endorsered transactions* generowany jest wpis do dziennika zdarzeń na potrzeby późniejszej analizy wyników. Miejsce *Ordering* jest połączone fuzją z miejscem na kolejnej stronie.

Rysunek 5.10 przedstawia przedostatnią stronę (*Tworzenie bloku*), na której znajduje się model usługi Ordering. Miejsce Ordering jest połączone fuzją z miejscem o tej samej nazwie znajdującym się na stronie *Przesyłanie transakcji*. Przejście *Add transactions* zawiera kod odpowiedzialny za tworzenie parametrów bloków na podstawie właściwości nadchodzących znaczników (transakcji). Utworzenie bloku i wysłanie go dalej następuje, gdy właściwości



Rysunek 5.9: Model sieci Petriego przesłania transakcji do usługi Ordering w Hyperledger Fabric

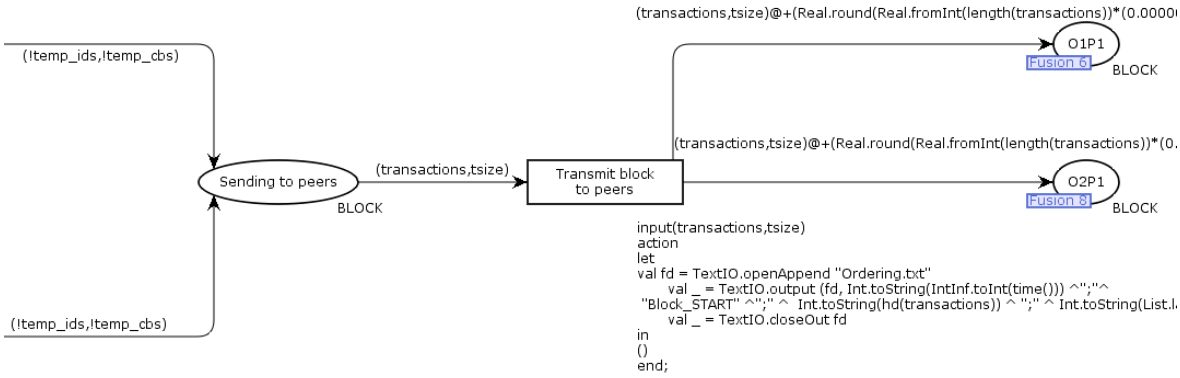
będą spełniały jeden z 2 warunków: liczba transakcji osiągnie wartość określoną przed przeprowadzeniem symulacji lub w sytuacji, gdy zostanie osiągnięty określony rozmiar. Znacznik z miejsca *Creating block* przejdzie tylko przez jedno z przejść *Block release 1* lub *Block release 2* w zależności, który z warunków zostanie spełniony. Na rysunku 5.10 nie przedstawiono kodów przy przejściach *Block release 1* lub *Block release 2* ze względu na to, że zajmują one bardzo dużo miejsca. Kody te są odpowiedzialne za zmianę koloru znacznika (który teraz jest reprezentacją bloku) oraz wygenerowania wpisu zawierającego długość kolejek do dziennika zdarzeń na potrzeby późniejszej analizy wyników symulacji.



Rysunek 5.10: Model sieci Petriego budowania bloku w Hyperledger Fabric – część I

Na rysunku 5.11 przedstawiono dalszy przebieg modelu na tej stronie. Znacznik, który teraz reprezentuje blok jest przesyłany do węzłów organizacji z usługą Peer posiadającej rolę lidera. Wartość znacznika czasu na łuku pomiędzy *Transmit block to peers* oraz miejsc *O1P1*, *O2P1* wynika ze średniej wartości czasu przesłania transakcji pomiędzy usługą Ordering a wę-

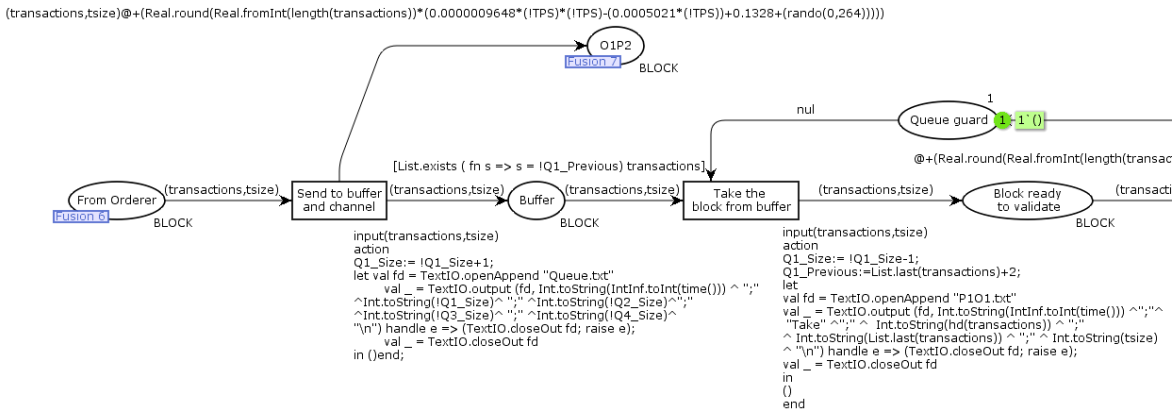
złami z usługą Peer posiadającymi rolę lidera podczas wykonania badań wydajnościowych. Cała funkcja nie zmieściła się na tym rysunku. Kod przy przejściu *Transmit block to peers* odpowiada za dodanie wpisu określającego początek czasu przetwarzania całego bloku. do dziennika zdarzeń Miejsca *O1P1* i *O2P1* są połączone fuzją z miejscami na kolejnej stronie.



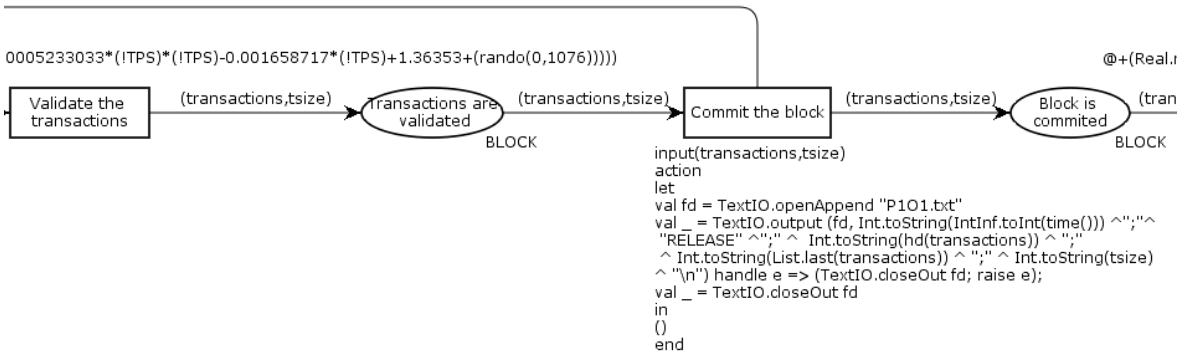
Rysunek 5.11: Model sieci Petriego budowania bloku w Hyperledger Fabric – część II

Ostatnia strona zawiera model, który prezentuje proces weryfikacji bloku oraz proces rozsyłania powiadomienia do klientów. Na rysunkach 5.12 do 5.14, przestawiono model tylko dla jednej organizacji, dla pozostałych strona ta będzie wyglądała bardzo podobnie. Miejsce *From orderer* jest połączone fuzją z miejscem *O1P1* znajdującym się na poprzedniej stronie. Łuk od przejścia *Send to buffer and channel* do miejsca *O1P2* oraz miejsce *O1P2* występuje tylko na stronie, która modeluje działanie węzła posiadającego rolę lider w ramach 1 organizacji. W przypadku pozostałych węzłów (nieposiadających roli lidera) miejsce początkowe jest połączone fuzją z miejscem *O1P2* – pozostałe węzły otrzymują bloki od lidera usługi Peer, a nie od usługi Ordering. Znaczniki mogą przejść przez przejście *Take the block from buffer* tylko w sytuacji, gdy znacznik znajdujący się w miejscu *Buffer* reprezentujący blok zawiera transakcję o numerze większym niż ostatni numer transakcji z poprzedniego znacznika oraz w miejscu *Queue guard* znajduje się znacznik. Przejście *Take the block from buffer* także zawiera kod odpowiedzialny za wygenerowanie zdarzenia do dziennika zdarzeń na potrzeby dalszej analizy wyników symulacji. Miejsce *Queue guard* jest wykorzystywane, aby ograniczyć liczbę jednocześnie weryfikowanych znaczników do 1. Wartość znacznika czasu przy przejściu *Validate the transactions* wynika ze średniej wartości czasu walidacji transakcji podczas wykonania badań wydajnościowych. Gdy znacznik przejdzie przez przejście *Commit the block* generowany jest dodatkowy znacznik do miejsca *Queue guard* i tym samym możliwe jest rozpoczęcie walidacji kolejnego znacznika. Kod przy przejściu *Commit the block* jest odpowiedzialny za wygenero-

wanie zdarzenia do dziennika zdarzeń określającego czas końca przetwarzania bloku. Wartość znacznika czasu przy przejściu *Send confirmation to client* wynika ze średniej wartości czasu przesyłania powiadomienia o dodaniu transakcji do rejestru rozproszonego podczas wykonania badań wydajnościowych. Kod przy przejściu *Log notification* jest odpowiedzialny za wygenerowanie zdarzenia do dziennika zdarzeń określającego czas otrzymania powiadomienia przez klienta.

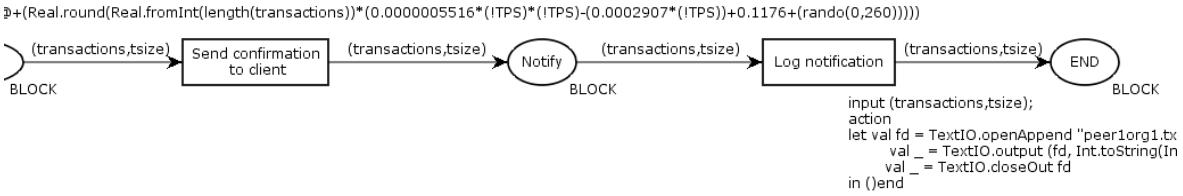


Rysunek 5.12: Model sieci Petriego zatwierdzania bloku w Hyperledger Fabric – część I



Rysunek 5.13: Model sieci Petriego zatwierdzania bloku w Hyperledger Fabric – część II

Do wykonania modelu wymagane było określenie czasu trwania poszczególnych etapów zatwierdzania transakcji. Aby określić czasy trwania każdego z etapów wykonano badania wydajnościowe rejestru rozproszonego, podczas których zebrano wszystkie potrzebne charakterystyki tj. czas trwania każdego etapu, czas przesyłania danych, rozmiar transakcji. Wykonano cztery badania, dla różnej liczby węzłów w organizacji, czasu budowania bloku (BatchTimeout), liczby transakcji w bloku (MaxMessageCount). Wartości te przedstawiono w tabeli: 5.13.



Rysunek 5.14: Model sieci Petriego zatwierdzania bloku w Hyperledger Fabric – część III

Tabela 5.13: Konfiguracja rejestru rozproszonego podczas przeprowadzania badań, których wyniki wykorzystano w budowie modelu

Liczba węzłów w organizacji	**Czas budowania bloku (BatchTimeout) [sek]**	**Liczba transakcji w bloku (MaxMessageCount)**
2	2	1000
2	10	1000
3	2	10000
3	10	10000

Każde badanie składało się z kilku testów, podczas których wysyłane były transakcje z różną częstotliwością (100,200,300,400,500,550 transakcji na sekundę). W tabeli 5.14 przedstawiono wszystkie skonfigurowane właściwości Hyperledger Fabric podczas prowadzenia tych badań.

Wyniki badań są bardzo obszerne, dlatego dołączono je w postaci plików Excel: `https://gitlab.com/MichalJarosz/doktorat/-/tree/main/Rozdzia%C5%82%205/Sie%C4%87%20Petri`.

Wyniki badań zostały następnie odpowiednio przygotowane według poniższych punktów:

1. Dla każdego bloku z testów obliczono czas wykonywania poszczególnych etapów.
2. Wyniki z punktu pierwszego zgrupowano dla wszystkich badań według etapów oraz częstotliwości napływu transakcji.
3. Dla każdej grupy obliczono średnią oraz odchylenie standardowe obsługi pojedynczej transakcji w bloku. Uzyskano dzięki temu średni czas oraz odchylenie standardowe obsługi transakcji dla różnych częstotliwości napływu transakcji.
4. Wyznaczono funkcje, które posłużyły do wyliczenia czasu trwania poszczególnych eta-

Tabela 5.14: Konfiguracja Hyperledger Fabric w badaniach potrzebnych do zbudowania modelu sieci Petriego

Parametr	**Wartość**
Liczba organizacji	2
Liczba węzłów w organizacji	Od 2 do 3
Polityka zatwierdzenia	Jeden węzeł z każdej organizacji
Wartość BatchTimeout	2s lub 10s
Wartość MaxMessageCount	1000 lub 10000
Wartość AbsoluteMaxBytes	99MB
Wartość PreferredMaxBytes	5MB
Konsensus dla usługi Ordering	EtcdRaft
Liczba węzłów z usługą Ordering	Każdy węzeł z usługą Peer miał również uruchomioną usługę Ordering

pów w modelu sieci Petriego. Do utworzenia przybliżonej funkcji ciągłej skorzystano z usługi: `http://curve.fit/`, która wykorzystuje bibliotekę ODRPACK [27].

Jeżeli wartość średniej czasu trwania etapu nie zależała od częstotliwości napływu transakcji do rejestru rozproszonego to w modelu wykorzystano zbadaną, stałą wartość czasu.

5.5.3 Analiza rezultatów

Na podstawie zdarzeń wygenerowanych podczas symulacji opracowanej sieci Petriego wyznaczono charakterystyki:

1. czasu trwania zapisania 50000 transakcji w rejestrze rozproszonym,
2. wymaganej liczby bloków do zapisania 50000 transakcji,
3. średniej liczby transakcji w bloku,
4. średniego czasu obsługi transakcji,
5. średniej długości kolejki podczas trwania symulacji.

W tabelach 5.15 do 5.18 przedstawiono wyniki symulacji dla badań o różnej konfiguracji Hyperledger Fabric.

Tabela 5.15: Różnica pomiędzy badaniem a wynikami modelu sieci Petriego dla 2 org, 2 węzły w każdej organizacji, maksymalny czas tworzenia bloku 2s, 1000 transakcji w bloku. Legenda: B - Badanie, S - Symulacja, R - Różnica

Liczba generowanych transakcji na sekundę	**100**			**200**			**400**			**500**		
Operacja	**B**	**S**	**R [%]**	**B**	**S**	**R [%]**	**B**	**S**	**R [%]**	**B**	**S**	**R [%]**
Czas trwania [ms]	500466	500881	100,08	250935	251931	100,40	126935	128192	100,99	108816	101730	93,49
Liczba bloków	243	249	102,47	124	125	100,81	123	125	101,63	99	99	100
Liczba transakcji na sekundę	99,91	99,94	100,03	199,25	199,48	100,11	393,90	395,08	100,30	459,49	491,87	107,05
Średni czas obsługi transakcji	1,38	1,34	96,74	1,70	1,66	97,53	2,69	2,52	93,76	8,44	2,95	34,99
Średnia długość kolejki	1	1	–	1	1	–	1,09	1,27	–	3,31	1,64	–

Tabela 5.16: Różnica pomiędzy badaniem a wynikami modelu sieci Petriego dla 2 org, 2 węzły w każdej organizacji, maksymalny czas tworzenia bloku 10s, 10000 transakcji w bloku. Legenda: B - Badanie, S - Symulacja, R - Różnica

Liczba generowanych transakcji na sekundę	**100**			**200**			**400**			**500**		
Operacja	**B**	**S**	**R [%]**	**B**	**S**	**R [%]**	**B**	**S**	**R [%]**	**B**	**S**	**R [%]**
Czas trwania [ms]	504085	500873	99,36	251188	251184	100	130752	128214	98,06	106890	102017	95,44
Liczba bloków	99	100	101,01	75	75	100	75	75	100	70	70	100
Liczba transakcji na sekundę	99,19	99,94	100,76	199,05	199,28	100,11	382,40	393,44	102,89	467,77	490,55	104,87
Średni czas obsługi transakcji	4,44	4,48	100,86	2,97	2,93	98,72	2,68	2,54	94,85	6,46	3,10	48,03
Średnia długość kolejki	1	1	–	1	1	–	1,05	1,11	–	2,17	1,23	–

Tabela 5.17: Różnica pomiędzy badaniem a wynikami modelu sieci Petriego dla 2 org, 3 węzły w każdej organizacji, maksymalny czas tworzenia bloku 2s, 1000 transakcji w bloku. Legenda: B - Badanie, S - Symulacja, R - Różnica

Liczba generowanych transakcji na sekundę	**100**			**200**			**400**			**500**		
Operacja	**B**	**S**	**R [%]**	**B**	**S**	**R [%]**	**B**	**S**	**R [%]**	**B**	**S**	**R [%]**
Czas trwania [ms]	501204	500776	99,96	251270	251957	100,27	126864	128283	101,12	101808	102353	100,53
Liczba bloków	243	249	102,47	124	125	100,81	125	125	100	99	99	100
Liczba transakcji na sekundę	99,76	99,96	100,20	198,99	199,46	100,24	394,12	394,80	100,17	491,12	488,88	99,54
Średni czas obsługi transakcji	1,37	1,34	98,10	1,70	1,65	97,32	2,53	2,50	98,70	2,76	2,87	103,88
Średnia długość kolejki	1	1	–	1	1	–	1,24	1,20	–	1,11	1,57	–

Tabela 5.18: Różnica pomiędzy badaniem a wynikami modelu sieci Petriego dla 2 org, 3 węzły w każdej organizacji, maksymalny czas tworzenia bloku 10s, 10000 transakcji w bloku. Legenda: B - Badanie, S - Symulacja, R - Różnica

Liczba generowanych transakcji na sekundę	**100**			**200**			**400**			**500**		
Operacja	**B**	**S**	**R [%]**	**B**	**S**	**R [%]**	**B**	**S**	**R [%]**	**B**	**S**	**R [%]**
Czas trwania [ms]	504533	500759	99,25	250979	250726	99,90	130669	127371	97,48	102464	102668	100,20
Liczba bloków	99	100	101,01	75	75	100	75	75	100	70	70	100
Liczba transakcji na sekundę	99,10	99,96	100,87	199,22	199,64	100,21	382,65	396,05	103,50	487,98	487,44	99,89
Średni czas obsługi transakcji	4,44	4,47	100,68	2,94	2,96	100,61	2,59	2,55	98,76	3,02	3,31	109,54
Średnia długość kolejki	1	1	–	1	1	–	1,04	1,08	–	1,03	1,29	–

Tabela 5.19: Współczynniki zmienności dla operacji walidacji transakcji i przesłania bloku od usługi Ordering do usługi Peer w zależności od częstotliwości napływu transakcji

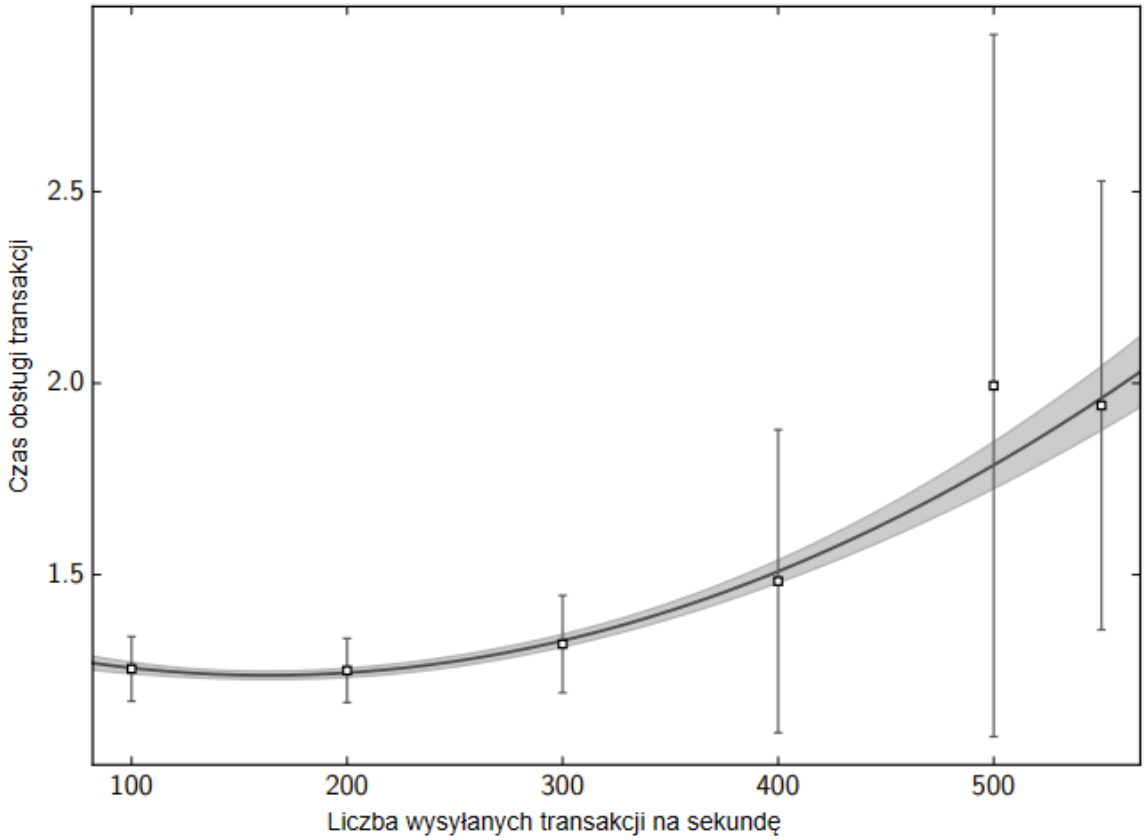
Liczba transakcji/sekundę	**Przesłanie bloku do lidera organizacji**	**Walidacja transakcji**
100	21,19%	6,71%
200	23,11%	6,69%
300	50,59%	9,63%
400	71,00%	26,67%
500	128,54%	45,97%
550	93,26%	30,16%

Na podstawie wyników symulacji można wywnioskować, że opracowany model jest zgodny z eksperymentalnym środowiskiem w sposób zadowalający. Duża różnica dla częstotliwości napływu transakcji wynoszącym 500 transakcji na sekundę jest skutkiem niedokładnego dopasowania wyznaczonej funkcji dla tej wartości. Dla takiej wartości wysyłanych transakcji na sekundę występuje największa różnica między wartościami otrzymanymi podczas badań a wartościami funkcji użytymi podczas symulacji. Na rysunkach 5.15 i 5.16 przedstawiono przebieg dwóch funkcji, na podstawie tych 2 rysunków widać wyraźnie, że:

- dla czasu walidacji transakcji wartość funkcji dla *x=500* wynosi 1,786 odbiega od średniej wartości wyliczonej w ramach badań oznaczonej punktem (500; 1,994);

- dla czasu przesłania bloku od usługi Ordering do usługi Peer wartość funkcji dla *x=500* wynosi 0,2049 i również odbiega od średniej wartości wyliczonej w ramach badań oznaczonej punktem (500; 0,1634).

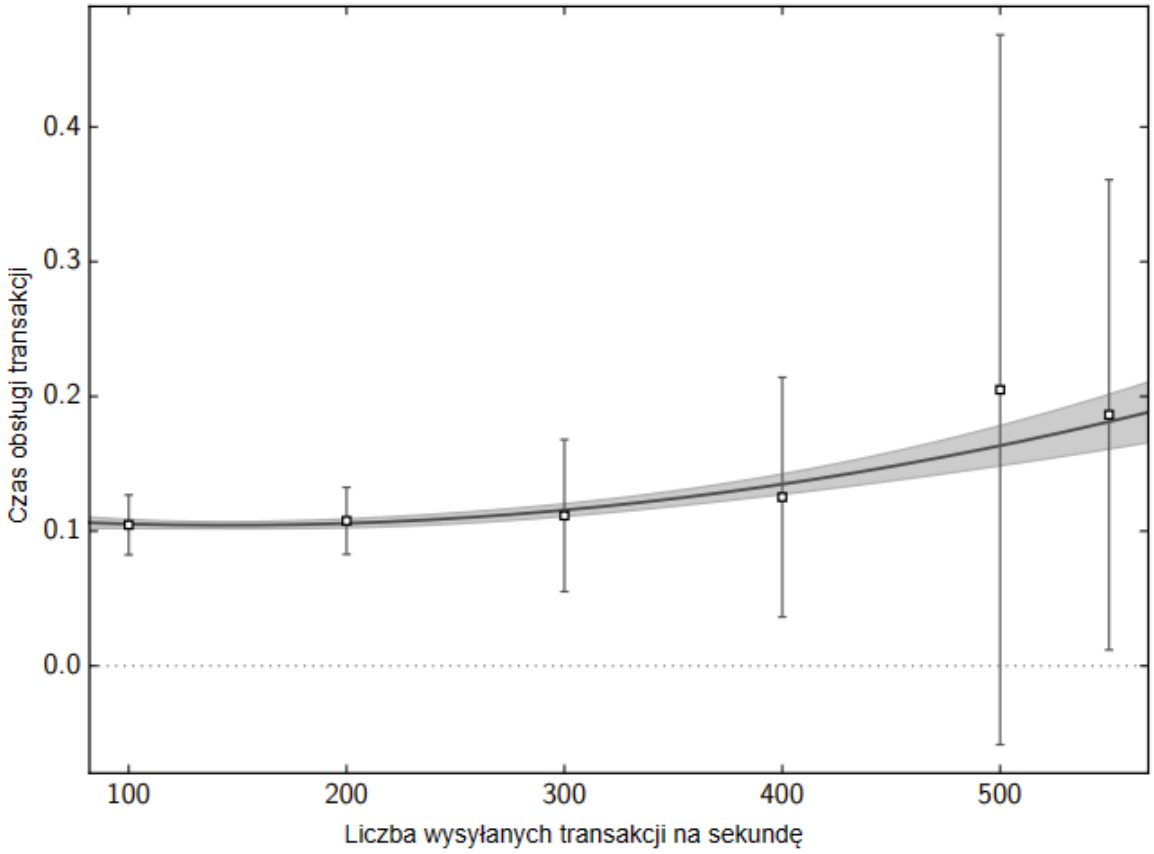
W tabeli 5.19 przedstawiono wartości współczynnika zmienności dla przeprowadzonych badań dla dwóch etapów w zależności od częstotliwości napływu nowych transakcji. Jak widać wyraźnie dla częstotliwości 500 transakcji/sekundę wartość współczynnika zmienności jest największa. Wyniki te są oznaką, że sieć jest na granicy stabilnej pracy. W tej sytuacji część bloków jest kolejkowana co oznacza, że węzły nie są w stanie na bieżąco przetwarzać nadchodzących bloków, a to wynika z tego, że procesor nie nadąża za wykonywaniem wielu różnych operacji – długość kolejki zadań na procesorze wynosi ponad 1.0.



Rysunek 5.15: Wykres funkcji czasu walidacji transakcji wykorzystanej w modelu sieci Petriego wraz z naniesionymi punktami reprezentującymi średnią tego obliczonymi na podstawie badań

5.6 Podsumowanie

W niniejszym rozdziale przedstawiono badania wydajnościowe operacji protokołu wykonywanych z wykorzystaniem rejestru HyperLedger Fabric (HLF). Na podstawie wyników badania protokołu CoAP uznano, że najwydajniejszym rozwiązaniem będzie utworzenie jednej sesji wykorzystującej TCP i w ramach tej sesji wysyłania wielu wiadomości. Jest to spowodowane tym, że tworzenie nowej sesji dla każdej wiadomości wymusza wysłania wielu dodatkowych pakietów (uzgodnienie trójetapowe – 3-way handshake), co przekłada się na zużycie zasobów urządzenia oraz wymaga dodatkowego czasu. Wykorzystanie protokołu CoAP z UDP powoduje wystąpienie duplikatów przy wysłaniu w jak najkrótszym czasie wielu tysięcy wiadomości każdej o rozmiarze ponad 2kB danych. Niezależnie od wykazania, że utworzenie jednej sesji CoAP z TCP jest wydajniejsze, do dalszych badań wybrano scenariusz, w którym sesja jest tworzona do wysłania pojedynczej wiadomości ze względu na dwa powody:



Rysunek 5.16: Wykres funkcji czasu przesyłania bloków od usługi Ordering do usługi Peer wykorzystanej w modelu sieci Petriego wraz z naniesionymi punktami reprezentującymi średnią tego czasu obliczonymi na podstawie badań

1. Założony został gorszy scenariusz. Jeżeli wyniki badania wydajności dla najgorszego scenariusza są uznane za wystarczające, to w pozostałych przypadkach również można będzie je uznać za wystarczające.

2. Tworzenie jednej sesji do wysłania pojedynczej komunikacji jest powszechnie stosowaną praktyką. Utrzymywanie sesji nie ma sensu, również w sytuacji, gdy rzadko przesyłane są małe porcje danych.

Na podstawie wyników badań LAAFFI, które wykonano dla różnych algorytmów HMAC, funkcji skrótu oraz AEAD można stwierdzić, że opracowany protokół jest wystarczająco wydajny, aby wykorzystać go w urządzeniach IoT posiadających własny system operacyjny. Pomiędzy wynikami badań dla różnych algorytmów HMAC oraz AEAD nie widać większej różnicy. Porównując badania z uwzględnieniem długości stosowanego klucza można stwierdzić, że dla LAAFFI:

1. wykorzystanie XChaCha20-Poly1305 i HMAC-SHA-256 jest wydajniejsze niż wykorzystanie AES-GCM-256 i HMAC-SHA-256 dla Raspberry Pi 4B i 3B+,

2. wykorzystanie Ascon i Xoodyak jest wydajniejsze niż wykorzystanie Xoodyak w obydwu przypadkach dla Raspberry Pi 4B i 3B+.

Przedstawiono także wyniki badań wydajności rejestru rozproszonego dla operacji, które będą wykonywane podczas realizacji opracowanego protokołu. Dla każdej przetestowanej konfiguracji rejestru rozproszonego uzyskano wyniki, które uznane są za zadowalające. Wydajność rejestru można w prosty sposób zwiększać poprzez dodawanie nowych węzłów lub poprzez poprawę konfiguracji sprzętowej węzłów. W przypadku operacji niewymagających osiągnięcia konsensusu przy dodawaniu nowych węzłów nie trzeba brać pod uwagę ograniczenia w komunikacji pomiędzy węzłami, bo taka nie następuje. Natomiast dla operacji wymagających osiągnięcia konsensusu, dodanie nowych węzłów wymusza komunikację w celu osiągnięcia wspólnych wartości. Liczba przesłanych komunikatów zależy od konfiguracji zasad zatwierdzenia transakcji. W tym przypadku można dojść do sytuacji, w którym dodanie kolejnego węzła nie wpłynie na zwiększenie wydajności, a nawet może doprowadzić do spadku wydajności [138]. W pracy dowiedziono także, że zwiększenie zasobów sprzętowych węzłów oraz liczby węzłów rejestru zwiększa wydajność całej sieci. Wydajność sieci rejestru rozproszonego maleje wraz ze zwiększaniem się liczby organizacji wymaganych do zatwierdzenia bloku.

Poza badaniem wydajnościowym protokołu dokonano także analizy ilości nadmiarowych danych oraz porównano LAAFFI z innymi podobnymi rozwiązaniami. Wynik porównania pokazuje, że LAAFFI nie posiada znacząco większego nadmiaru przesyłanych danych niż wykorzystanie tylko jednego klucza podczas całej komunikacji. LAAFFI ma także mniejszy nadmiar przesyłanych bajtów niż protokół TLS czy DTLS.

Biorąc pod uwagę wyniki zbadanych wskaźników można uznać, że protokół LAAFFI jest przystosowany do realizacji głównych scenariuszy opisanych w 5.1.

W rozdziale przedstawiono również model zapisu transakcji w rejestrze rozproszonym z wykorzystaniem czasowych kolorowanych sieci Petriego, który może posłużyć do wykonania symulacji rejestru rozproszonego podczas dodawania danych do rejestru. Wyniki opracowanego modelu są zadowalające, ponieważ dla większości przypadków różnica między wynikami eksperymentalnymi a symulacyjnymi wynosi kilka procent. Wykorzystanie modelu symulacyjnego wymusza posiadanie implementacji badanego protokołu oraz wykonania testów wydajnościowych tej implementacji przed przystąpieniem do wykonania badań w oparciu o model sieci Pe-

triego. Jest to wymagane, aby wyliczyć parametry funkcji czasu wykonywania poszczególnych etapów w zależności od liczby transakcji napływających w ciągu sekundy. W celu uzyskania lepszych wyników należy dokonać pomiarów w środowisku odizolowanym od innych czynników m.in. uruchomionych innych maszyn wirtualnych, uruchomionych usług. W środowisku, z którego zasobów korzystają także inne osoby lub są wykonywane jakieś prace (np. synchronizacja maszyn wirtualnych między węzłami) nie jest możliwe dokonanie badań bez zakłóceń. Oczywiście posiadanie odizolowanego środowiska zwiększa koszt przeprowadzenia takiego badania. Problemem wykorzystania tego modelu jest dość duży nakład pracy potrzebny na jego zbudowanie. Jednakże zakładając, że jest on już opracowany i właściwie sparametryzowany z możliwością łatwego dostosowywania parametrów, zależnych od konfiguracji rejestru rozproszonego to koszt badań z wykorzystaniem modelu jest mniejszy niż badania na rzeczywistym systemie. Dodatkową zaletą jest łatwiejsze zbadanie parametrów wydajnościowych dla różnej konfiguracji rejestru rozproszonego.

Rozdział 6

Podsumowanie pracy oraz kierunki dalszych badań

6.1 Podsumowanie pracy

Uwierzytelnianie i autoryzacja urządzeń Internetu rzeczy ma kluczowe znaczenie w zapewnieniu bezpieczeństwa w sytuacjach wymagających współpracy pomiędzy różnymi organizacjami, które nie zawsze mają do siebie zaufanie. W pierwszym rozdziale pracy przeanalizowano, jakie wymagania musi spełniać system, którego celem jest zapewnienie uwierzytelniania i autoryzacji w środowiskach federacyjnych. W ramach analizy literatury zauważono, że w wielu publikacjach przedstawiono wykorzystanie rejestru rozproszonego jako elementu systemu uwierzytelniania. Technologia ta ma cechy, które są przydatne w środowiskach federacyjnych i uznano, że wykorzystanie rejestru rozproszonego może pomóc zrealizować część ze zdefiniowanych wymagań. Przedstawione przez wielu badaczy rozwiązania nie spełniają wszystkich zdefiniowanych wymagań systemu uwierzytelniania i autoryzacji urządzeń IoT w środowiskach federacyjnych. Dodatkowo w wielu propozycjach przedstawionych w publikacjach nie wykorzystano wszystkich zalet, jakie oferuje technologia rejestrów rozproszonych, a wykorzystano rejestr rozproszony tylko jako bazę danych.

W rozdziale drugim przeanalizowano, jakie cechy powinna mieć implementacja rejestru rozproszonego, która posłuży do przygotowania wersji PoC systemu uwierzytelniania i autoryzacji urządzeń IoT w środowiskach federacyjnych. Na podstawie wskazanych cech dokonano analizy obiecujących implementacji i ostatecznie wybrano Hyperledger Fabric. Wybór ten pozwala spełnić kilka warunków dotyczących systemu uwierzytelniania i autoryzacji urządzeń IoT w środowiskach federacyjnych określonych w rozdziale pierwszym. Hyperledger Fabric, tak samo jak inne rejestry rozproszone, jest zdecentralizowany (warunek 4), tzn. nie istnieje jedna główna organizacja, która może kontrolować cały system. Implementacja ta wspiera kanały

prywatne, które umożliwiają przechowywanie danych prywatnych każdej organizacji tylko na węzłach do niej należących (warunek 5). Poza tym istnieje możliwość wykorzystania kodów łańcuchowych, które mogą określać zasady dostępu do danych przechowywanych w rejestrze rozproszonym (warunek 3) oraz mogą wspomóc ochronę danych poprzez wykonywanie na danych operacji kryptograficznych (warunek 5). Wdrażając Hyperledger Fabric, zalecane jest posiadanie kilku węzłów, aby zagwarantować ciągłość operacji biznesowych (warunek 9). Wybrana implementacja rejestru rozproszonego jest też powszechnie wykorzystywana, przez co istnieją usługi chmurowe umożliwiające szybkie wdrożenie tej usługi (warunek 1). W ramach rozprawy został zaproponowany nowy, oryginalny protokół, który został opisany w rozdziale III. Bazuje on na znajomości klucza, który został wykorzystany do zaszyfrowania/odszyfrowania przesyłanych danych. Jeżeli obie strony potrafią komunikować się ze sobą, wykorzystując wspólny klucz ustalany na podstawie cech dystynkcyjnych urządzenia IoT, to założono, że obie strony komunikacji są uwierzytelnione. Zaproponowane rozwiązanie umożliwia skorzystanie z jednego z trzech różnych źródeł do utworzenia klucza uwierzytelniającego. Pierwszym z nich jest możliwość skorzystania z wartości sprzętowych i programowych każdego urządzenia IoT, co jest podejściem niespotkanym w literaturze, drugim wykorzystanie układu PUF, natomiast trzecim wykorzystanie losowych ciągów znaków. Opracowany protokół cechuje modularność, ponieważ nie został zaprojektowany z myślą o wykorzystaniu konkretnych protokołów, technologii czy też rozwiązań. To od organizacji wdrażającej zależy, jakie protokoły kryptograficzne, komunikacyjne, metody autoryzacji zostaną zastosowane. LAAFFI może być wdrożony lokalnie lub/i z wykorzystaniem usług chmurowych, a sam proces można zautomatyzować i dzięki temu uzyskać niezwłoczną interoperacyjność (warunek 1). W przypadku każdej z operacji urządzenie IoT nawiązuje połączenie z węzłem rejestru rozproszonego ze względu na to, że jest on odpowiedzialny za autoryzacje operacji, co pozwala na zapewnienie rozliczalności (warunek 8).

Analiza entropii wykazała, że wykorzystanie każdego z zaproponowanych źródeł oferuje wystarczającą entropię, aby uzyskany klucz uwierzytelniający był uznany za trudny do odgadnięcia. W przypadku układów PUF oraz wartości fizycznych i oprogramowania konieczne jest wykorzystanie kilku parametrów. Liczba tych wartości jest zależna od liczby bitów entropii, jaką federacja chce uzyskać. Z przeprowadzonej jakościowej analizy ataków, które można przeprowadzić względem LAAFFI wynika, że bardzo ważnym elementem jest zachowanie poufności wartości parametrów przechowywanych w rejestrze rozproszonym. W przypadku ich upublicznienia każdy, kto będzie w ich posiadaniu jest w stanie podszyć się pod urządze-

nie oraz może odczytać przesyłane wiadomości. Istotnym elementem jest także zapewnienie poprawnej synchronizacji czasu wszystkich komponentów systemu LAAFFI, aby umożliwić komunikację z urządzeniem IoT jednocześnie zapobiegać atakom powtórzenia. Przygotowane modele formalne przeanalizowane z wykorzystaniem narzędzi Verifpal i Tamarin wykazały spełnienie przez LAAFFI cech poufności, uwierzytelnienia obu stron oraz odporności na atak powtórzeniowy. Analiza bezpieczeństwa przedstawiona w tym rozdziale pozwala stwierdzić, że protokół LAAFFI można uznać za bezpieczny (warunek 7).

Przeprowadzono także analizę wydajności opracowanego protokołu dla urządzeń Raspberry Pi. Badanie wydajnościowe protokołu LAAFFI dla urządzeń IoT wykazało, że różnica pomiędzy wykorzystaniem zwykłych protokołów kryptograficznych oraz protokołów uznawanych za "lekkie"nie jest duża. Przygotowana implementacja pozwoliła wykazać, że protokół LAAFFI można wykorzystywać dla różnych urządzeń IoT z własnym systemem operacyjnym (warunek 2). Dużą zaletą LAAFFI jest bardzo mały nadmiar przesyłanych danych. W przypadku porównania z protokołem TLS i DTLS, LAAFFI osiągnął lepsze wyniki dla każdego zbadanego rozmiaru wiadomości. Natomiast porównując go z przypadkiem, gdy oba urządzenia posiadają współdzielony klucz i tylko przy jego pomocy zabezpieczają transmisję, LAAFFI okazał się lepszy dla każdego rozmiaru wiadomości mniejszego niż 100kB (warunek 2). Wykonano także badanie wydajnościowe implementacji dla rejestru rozproszonego i wykazano, że LAAFFI może obsłużyć setki operacji na sekundę, jeżeli rejestr jest zbudowany z dwóch węzłów, a w sytuacji gdy ta wydajność jest niewystarczająca, można dodać kolejne węzły i tym samym zwiększyć liczbę obsługiwanych operacji przez rejestr rozproszony w sposób zbliżony do liniowego (warunek 6). Można także zwiększyć liczbę rdzeni procesora dla węzła rejestru rozproszonego, co również przekłada się na wzrost wydajności. Zbudowano także model sieci Petriego, który pozwala na badanie wydajności protokołu LAAFFI dla różnej konfiguracji rejestru rozproszonego.

Biorąc pod uwagę powyższe, można stwierdzić, że wszystkie warunki postawione przed systemem uwierzytelniania i autoryzacji urządzeń IoT w środowiskach federacyjnych zostały spełnione. W ramach pracy zrealizowano wszystkie cele: przedstawiono opracowany protokół (rozdział 3), wykonano implementację tego protokołu (rozdział 3), oceniono skalowalność i wydajność (rozdział 5), przygotowano jakościową i formalną ocenę bezpieczeństwa (rozdział 4), a także przygotowano model wydajnościowy podstawowych operacji na rejestrze z wykorzystaniem kolorowanych czasowych sieci Petriego (rozdział 4). W związku z pozytywną realizacją celów można stwierdzić, że teza postawiona w pracy została potwierdzona.

6.2 Kierunki dalszych prac

Podczas pisania pracy napotkano na problem braku zaleceń dotyczących budowy federacyjnych systemów Internetu rzeczy. Znaleziono zalecenia dotyczące m.in. stosowania zasad i praktyk bezpieczeństwa w cyklu życia systemów IoT [89] czy też bezpieczeństwa urządzeń IoT [56], ale żadne ze znalezionych zaleceń nie odnosiły się do współpracy pomiędzy organizacjami wykorzystującymi systemy IoT. Utworzenie federacji systemów IoT nie jest prostym zadaniem, ponieważ wymaga ono określenia pewnych założeń dotyczących m.in.: przetwarzania danych, metod komunikacji, wykorzystywanych algorytmów, uprawnień do obiektów itd. dopasowanych do środowisk IoT. Dlatego też powinny istnieć zalecenia dotyczące budowy federacyjnych systemów IoT, które określałyby, jakie wymagania musi spełniać system, który ma być częścią środowiska federacyjnego. Takie zalecenia powinny zawierać także informacje o algorytmach, protokołach, procedurach i technologiach, jakie należy wykorzystać w takim środowisku, aby proces budowania federacji był możliwie jak najszybszy, uwzględniając przy tym kwestie bezpieczeństwa takie jak prywatność czy też rozliczalność.

Zagadnieniem, które należy również zbadać jest ocena możliwości wykorzystania rejestrów rozproszonych w środowiskach o niskiej przepustowości sieci. Obecne implementacje przeznaczone są do zastosowań w zwykłym środowisku, a nie takim o ograniczonej przepustowości komunikacji. Podczas przygotowywania badań nie istniały implementacje rejestrów rozproszonych, które były przeznaczone do wykorzystania w sieciach IoT oraz były dojrzałe. Należy zbadać, czy implementacje takie jak IOTA mogą być wykorzystywane w środowisku o niskiej przepustowości sieci.

Następnym problemem wartym sprawdzenia jest możliwość przeprowadzenia ataków kwantowych na implementację rejestru rozproszonego. Ponieważ protokół LAAFFI bazuje na kryptografii symetrycznej, to odporny jest na ataki kwantowe. Jednak wszystkie znane autorowi pracy implementacje rejestrów rozproszonych wykorzystują kryptografię asymetryczną, przez co może istnieć w przyszłości zagrożenie ujawnienia elementu kryptograficznego wykorzystanego do zabezpieczenia komunikacji z węzłami rejestru rozproszonego.

Wykorzystane narzędzia umożliwiające analizę formalną opracowanego protokołu bazują na modelach utworzonych przez autora pracy, a nie na konwersji kodu protokołu napisanego w języku programowania na model formalny. Pomimo że prowadzone są prace umożliwiające taką konwersję [26], a tym samym analizę kodu utworzonego protokołu, to są one jeszcze na wczesnym etapie i mają różne ograniczenia, np. możliwość konwersji tylko z pewnych

języków oprogramowania (C, F#, Java) lub też wykorzystanie tylko bibliotek dostarczonych wraz z językiem programowania. Przykładem takiego narzędzia jest FS2PV [1], a narzędzie Spi2Java [2] umożliwia konwersję modelu na kod programu.

[1] https://www.microsoft.com/en-us/download/details.aspx?id=52339
[2] http://spi2java.polito.it/

Rozdział 7

Parametry urządzeń wykorzystywanych w badaniach wydajności

Tabela A1: Konfiguracja sprzętowa Raspberry Pi 4B

Parametr	**Raspberry Pi 4B**
Procesor	Broadcom BCM2711
Liczba rdzeni	4 x Cortex-A72
Taktowanie	1.5 GHz
Karta pamięci	Samsung Evo Plus 32GB
Pamięć RAM	8GB
Wyjścia USB	2x 2.0, 2x 3.0
Inne wyjścia	2x HDMI, 3,5mm jack, DSI
Interfejs kamery	CSI
Komunikacja bezprzewodowa	IEEE 802.11 b/g/n/ac oraz Bluetooth 5.0
Komunikacja przewodowa	10/100/1000 Ethernet
Urządzenia peryferyjne niskiego poziomu	40× GPIO (w tym I2C, UART, SPI)
System operacyjny	Rasbpian OS 5.15.30 (2022-04-04) arm64

Tabela A2: Konfiguracja sprzętowa Raspberry Pi 3B+

Parametr	**Raspberry Pi 3B+**
Procesor	Broadcom BCM2837B0
Liczba rdzeni	4 x Cortex-A53
Taktowanie	1.4 GHz
Karta pamięci	Samsung Evo Plus 32GB
Pamięć RAM	1GB
Wyjścia USB	4x 2.0
Inne wyjścia	1x HDMI, 3,5mm jack, DSI
Interfejs kamery	CSI
Komunikacja bezprzewodowa	IEEE 802.11 b/g/n/ac oraz Bluetooth 4.2 BLE
Komunikacja przewodowa	10/100/1000 Ethernet (rzeczywiście 300Mb/s)
Urządzenia peryferyjne niskiego poziomu	40× GPIO (w tym I2C, UART, SPI)
System operacyjny	Rasbpian OS 5.15.30 (2022-04-04) arm64

Tabela A3: Konfiguracja sprzętowa Raspberry Pi zero WH

Parametr	**Raspberry Pi zero WH**
Procesor	Broadcom BCM2835
Liczba rdzeni	1 x ARM1176JZF-S
Taktowanie	1.0 GHz
Karta pamięci	Samsung Evo Plus 32GB
Pamięć RAM	512 MB
Wyjścia USB	1x 2.0
Inne wyjścia	1x HDMI
Interfejs kamery	CSI
Komunikacja bezprzewodowa	IEEE 802.11 b/g/n oraz Bluetooth 4.1 BLE
Komunikacja przewodowa	brak
Urządzenia peryferyjne niskiego poziomu	40× GPIO (w tym I2C, UART, SPI)
System operacyjny	Rasbpian OS 5.15.30 (2022-04-04) armhf

Tabela A4: Konfiguracja komputera, na którym uruchomiono maszynę wirtualną Ubuntu 22.04

Parametr	**HP EliteBook 835 G7**
Procesor	Ryzen 7 PRO 4750
Typ pamięci RAM	DDR4
Wielkość pamięci RAM	64 GB
Taktowanie pamięci RAM	3200 MHz
Dysk twardy	Samsung 970 EVO Plus 2TB
Karta Wi-Fi	Intel Wi-Fi 6 AX200
Karta graficzna	Zintegrowana
Karta Ethernet	Dell DA300
System operacyjny	Windows 10 Pro
Wirtualizator	VMware Workstation 16.2.4

Tabela A5: Konfiguracja sprzętowa Technicolor CGA4236TCH1

Parametr	**Technicolor CGA4236TCH1**
Komunikacja bezprzewodowa	IEEE 802.11ax, 2.4 GHz (3×3), 5 GHz (4×4)
Standard transmisji danych HFC	DOCSIS 3.1 oraz (Euro)PacketCable 2.0
Komunikacja przewodowa	4x GbE LAN
Wyjścia USB	1x 3.1
Porty FXS	2
Pamięć Flash	512 MB
Pamięć RAM	512 MB

Bibliografia

[1] E. Abulibdeh i in. „DRAM-Based PUF Utilizing the Variation of Adjacent Cells". W: *IEEE Transactions on Information Forensics and Security* (2024), s. 1–1. DOI: 10.1109/TIFS.2024.3354115.

[2] ACT-IAC. „Blockchain Playbook for the U.S. Federal Government". W: (2019), s. 1–105. URL: https://www.actiac.org/documents/act-iac-white-paper-blockchain-playbook-us-federal-government.

[3] M. Aizatulin. *Verifying Cryptographic Security Implementations in C Using Automated Model Extraction*. 2020. DOI: 10.48550/arXiv.2001.00806.

[4] M. Akil i in. „Privacy-Preserving Identifiers for IoT: A Systematic Literature Review". W: *IEEE Access* 8 (2020), s. 168470–168485. DOI: 10.1109/ACCESS.2020.3023659.

[5] M. S. Ali i in. „Applications of Blockchains in the Internet of Things: A Comprehensive Survey". W: *IEEE Communications Surveys & Tutorials* 21.2 (2019), s. 1676–1717. DOI: 10.1109/COMST.2018.2886932.

[6] S. AlJanah, N. Zhang i S. W. Tay. „A Survey on Smart Home Authentication: Toward Secure, Multi-Level and Interaction-Based Identification". W: *IEEE Access* 9 (2021), s. 130914–130927. DOI: 10.1109/ACCESS.2021.3114152.

[7] N. Alsaeed i F. Nadeem. „Authentication in the Internet of Medical Things: Taxonomy, Review, and Open Issues". W: *Applied Sciences* 12.15 (2022). ISSN: 2076-3417. DOI: 10.3390/app12157487.

[8] M. Alshahrani i I. Traore. „Secure mutual authentication and automated access control for IoT smart home using cumulative Keyed-hash chain". W: *Journal of Information Security and Applications* 45 (2019), s. 156–175. ISSN: 2214-2126. DOI: 10.1016/j.jisa.2019.02.003.

[9] N. Andola i in. „Vulnerabilities on Hyperledger Fabric". W: *Pervasive and Mobile Computing* 59 (2019), s. 101050. ISSN: 1574-1192. DOI: 10.1016/j.pmcj.2019.101050.

[10] *Announcing the Commercial National Security Algorithm Suite 2.0*. URL: https://media.defense.gov/2022/Sep/07/2003071834/-1/-1/0/CSA_CNSA_2.0_ALGORITHMS_.PDF.

[11] S. Arciszewski. *XChaCha: eXtended-nonce ChaCha and AEAD_XChaCha20_Poly1305*. Internet-Draft draft-irtf-cfrg-xchacha-03. Work in Progress. Internet Engineering Task Force, sty. 2020. 18 s. URL: https://datatracker.ietf.org/doc/draft-irtf-cfrg-xchacha/03/.

[12] J.-P. Aumasson. *Nowoczesna kryptografia. Praktyczne wprowadzenie do szyfrowania*. Wydawnictwo Naukowe PWN, 2018. ISBN: 9788301200695.

[13] M. Avalle, A. Pironti i R. Sisto. „Formal verification of security protocol implementations: a survey". W: *Formal Aspects of Computing* 26.1 (sty. 2014), s. 99–123. ISSN: 1433-299X. DOI: 10.1007/s00165-012-0269-9.

[14] E. S. Babu i in. „Blockchain-based Intrusion Detection System of IoT urban data with device authentication against DDoS attacks". W: *Computers and Electrical Engineering* 103 (2022), s. 108287. ISSN: 0045-7906. DOI: 10.1016/j.compeleceng.2022.108287.

[15] A. V. Barenji i B. Montreuil. „Open Logistics: Blockchain-Enabled Trusted Hyperconnected Logistics Platform". W: *Sensors* 22.13 (2022). ISSN: 1424-8220. DOI: 10.3390/s22134699.

[16] E. Barka i in. „Securing the Web of Things with Role-Based Access Control". W: *Codes, Cryptology, and Information Security*. Cham: Springer International Publishing, 2015, s. 14–26. ISBN: 978-3-319-18681-8.

[17] G. Barthe i in. „EasyCrypt: A Tutorial". W: *Foundations of Security Analysis and Design VII: FOSAD 2012/2013 Tutorial Lectures*. Red. A. Aldini, J. Lopez i F. Martinelli. Cham: Springer International Publishing, 2014, s. 146–166. ISBN: 978-3-319-10082-1. DOI: 10.1007/978-3-319-10082-1_6. URL: https://doi.org/10.1007/978-3-319-10082-1_6.

[18] D. Basin. *Symbolic verification of cryptographic protocols using Tamarin Part 3: Security Properties and Algorithmic Verification*. Sierp. 2018. URL: https://resources.mpi-inf.mpg.de/departments/rg1/conferences/vtsa18/slides/basin-lecture3.pdf.

[19] D. Basin i in. „A Formal Analysis of 5G Authentication". W: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. CCS '18. Toronto, Canada: Association for Computing Machinery, 2018, s. 1383–1396. ISBN: 9781450356930. DOI: 10.1145/3243734.3243846.

[20] F. R. Batubara, J. Ubacht i M. Janssen. „Challenges of Blockchain Technology Adoption for E-Government: A Systematic Literature Review". W: *Proceedings of the 19th Annual International Conference on Digital Government Research: Governance in the Data Age*. dg.o '18. Delft, The Netherlands: Association for Computing Machinery, 2018. ISBN: 9781450365260. DOI: 10.1145/3209281.3209317.

[21] M. Bellare. „New Proofs for NMAC and HMAC: Security without Collision Resistance". W: *Journal of Cryptology* 28.4 (paź. 2015), s. 844–878. ISSN: 1432-1378. DOI: 10.1007/s00145-014-9185-x.

[22] R. Berg i V. Den. „Entropy analysis of physical unclonable functions". W: 2012. URL: https://api.semanticscholar.org/CorpusID:16474701.

[23] M. Bhandary, M. Parmar i D. Ambawade. „A Blockchain Solution based on Directed Acyclic Graph for IoT Data Security using IoTA Tangle". W: *2020 5th International Conference on Communication and Electronics Systems (ICCES)*. 2020, s. 827–832. DOI: 10.1109/ICCES48766.2020.9137858.

[24] B. Blanchet. „A Computationally Sound Mechanized Prover for Security Protocols". W: *IEEE Transactions on Dependable and Secure Computing* 5.4 (2008), s. 193–207. DOI: 10.1109/TDSC.2007.1005.

[25] B. Blanchet. „Automatic Verification of Security Protocols in the Symbolic Model: the Verifier ProVerif". W: *Foundations of Security Analysis and Design VII, FOSAD Tutorial Lectures*. Red. A. Aldini, J. Lopez i F. Martinelli. T. 8604. Lecture Notes in Computer Science. Springer, 2014, s. 54–87.

[26] B. Blanchet. „Security Protocol Verification: Symbolic and Computational Models". W: *Principles of Security and Trust*. Red. P. Degano i J. D. Guttman. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, s. 3–29. ISBN: 978-3-642-28641-4. DOI: 10.1007/978-3-642-28641-4_2.

[27] P. Boggs, R. Byrd i J. Rogers. Dostęp : 2023-01-20. 1992. URL: docs.scipy.org/doc/external/odrpack_guide.pdf.

[28] X. Bonnetain, M. Naya-Plasencia i A. Schrottenloher. „Quantum Security Analysis of AES". W: *IACR Transactions on Symmetric Cryptology* 2019.2 (czer. 2019), s. 55–93. DOI: 10.13154/tosc.v2019.i2.55-93.

[29] C. Bormann i P. Hoffman. *Concise Binary Object Representation (CBOR)*. RFC 7049. IETF, paź. 2013. DOI: 10.17487/RFC7049.

[30] C. Bormann i in. *CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets*. Spraw. tech. 8323. Lut. 2018. 54 s. DOI: 10.17487/RFC8323.

[31] S. Brakeville i B. Perepa. *Blockchain basics: Introduction to distributed ledgers*. Maj 2019. URL: https://developer.ibm.com/tutorials/cl-blockchain-basics-intro-bluemix-trs/.

[32] R. G. Brown. *Dieharder: A Random Number Test Suite*. 2023. URL: https://webhome.phy.duke.edu/~rgb/General/dieharder.php.

[33] A. K. C. Bormann M. Ersue. *Terminology for Constrained-Node Networks*. Spraw. tech. 7228. IETF, paź. 2015. 17 s. DOI: 10.17487/RFC7228.

[34] C. Chaplin i in. *802.11i Overview*. Spraw. tech. 802.11i. IEEE, 2005. URL: https://ieee802.org/16/liaison/docs/80211-05_0123r1.pdf.

[35] G. Cheng i in. „A Blockchain-Based Mutual Authentication Scheme for Collaborative Edge Computing". W: *IEEE Transactions on Computational Social Systems* 9.1 (2022), s. 146–158. DOI: 10.1109/TCSS.2021.3056540.

[36] *Cisco Annual Internet Report (2018–2023)*. Spraw. tech. Cisco, 2020.

[37] R. Colopy i J. Chopra. *SRAM Characteristics as Physical Unclonable Functions*. Worcester Polytechnic Institute, Project Number: MQP-BS2-0803. 2009. URL: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&%20doi=-%208c601d1daa6f321aefa57bdb506d7a9cec062d35.

[38] C. Cremers. *The Scyther Tool*. https://people.cispa.io/cas.cremers/scyther/index.html. 2014. URL: https://people.cispa.io/cas.cremers/scyther/index.html.

[39] C. Cremers i in. „A Comprehensive Symbolic Analysis of TLS 1.3". W: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. CCS '17. Dallas, Texas, USA: Association for Computing Machinery, 2017, s. 1773–1788. ISBN: 9781450349468. DOI: 10.1145/3133956.3134063.

[40] C. Cremers i in. „Automated Analysis and Verification of TLS 1.3: 0-RTT, Resumption and Delayed Authentication". W: *2016 IEEE Symposium on Security and Privacy (SP)*. 2016, s. 470–485. DOI: 10.1109/SP.2016.35.

[41] B. Cremonezi i in. „Survey on Identity and Access Management for Internet of Things". W: (sierp. 2020). DOI: 10.21203/rs.3.rs-66793/v1.

[42] *Cryptographic Mechanisms: Recommendations and Key Lengths*. Sty. 2024. URL: https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TG02102/BSI-TR-02102-1.html.

[43] J. Daemen i in. „Xoodyak, a lightweight cryptographic scheme". W: *IACR Transactions on Symmetric Cryptology* 2020.S1 (czer. 2020), s. 60–87. DOI: 10.13154/tosc.v2020.iS1.60-87.

[44] H.-N. Dai, Z. Zheng i Y. Zhang. „Blockchain for Internet of Things: A Survey". W: *IEEE Internet of Things Journal* 6.5 (2019), s. 8076–8094. DOI: 10.1109/JIOT.2019.2920987.

[45] Q. Dang. *Recommendation for Existing Application-Specific Key Derivation Functions*. Spraw. tech. Grud. 2011. DOI: 10.6028/NIST.SP.800-135r1.

[46] Q. Dang. *The Keyed-Hash Message Authentication Code (HMAC)*. en. Lip. 2008. DOI: 10.6028/NIST.FIPS.198-1.

[47] *Definicja blockchain*. 2019. URL: https://sjp.pl/blockchain.

[48] M. Ding i in. *HFContractFuzzer: Fuzzing Hyperledger Fabric Smart Contracts for Vulnerability Detection*. 2021. DOI: 10.48550/arXiv.2106.11210.

[49] S. Ding i in. „A Novel Attribute-Based Access Control Scheme Using Blockchain for IoT". W: *IEEE Access* 7 (2019), s. 38431–38441. DOI: 10.1109/ACCESS.2019.2905846.

[50] C. Dobraunig i in. *Ascon v1.2*. Submission to Round 1 of the NIST Lightweight Cryptography project. 2019. URL: https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/ascon-spec.pdf.

[51] M. J. Dworkin. *SP 800-38D. Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*. Spraw. tech. Gaithersburg, MD, USA, 2007. DOI: 10.6028/NIST.SP.800-38D.

[52] Ethereum. *Introduction to Smart Contracts*. Dostępne w: https://docs.soliditylang.org/en/latest/introduction-to-smart-contracts.html.

[53] Ethereum. *Strona projektu*. URL: https://ethereum.org/pl/.

[54] EU. *General Data Protection Regulation 2016/679*. 2016.

[55] H. Fabric. *Writing Your First Chaincode*. Dostępne w: https://hyperledger-fabric.readthedocs.io/en/release-2.5/chaincode4ade.html.

[56] M. Fagan i in. *IoT Device Cybersecurity Guidance for the Federal Government: Establishing IoT Device Cybersecurity Requirements*. Spraw. tech. List. 2021. DOI: 10.6028/NIST.SP.800-213.

[57] K. Fan, P. Song i Y. Yang. „ULMAP: Ultralightweight NFC Mutual Authentication Protocol with Pseudonyms in the Tag for IoT in 5G". W: *Mobile Information Systems* 2017 (kw. 2017), s. 2349149. ISSN: 1574-017X. DOI: 10.1155/2017/2349149.

[58] L. Feiten, M. Sauer i B. Becker. *On Metrics to Quantify the Inter-Device Uniqueness of PUFs*. Cryptology ePrint Archive, Paper 2016/320. https://eprint.iacr.org/2016/320. 2016. URL: https://eprint.iacr.org/2016/320.

[59] T. M. Fernández-Caramés i P. Fraga-Lamas. „A Review on the Use of Blockchain for the Internet of Things". W: *IEEE Access* 6 (2018), s. 32979–33001. DOI: 10.1109/ACCESS.2018.2842685.

[60] J. T. Force. *Security and Privacy Controls for Information Systems and Organizations*. SP 800-53. NIST, 2020. DOI: 10.6028/NIST.SP.800-53r5.

[61] A. S. Foundation. *Apache CouchDB® 3.2.0 Documentation*. Dostępne w: https://docs.couchdb.org/en/stable/.

[62] T. L. Foundation. *Hyperledger Fabric*. Dostępne w: https://www.hyperledger.org/projects/fabric.

[63] P. Fremantle i in. „Federated Identity and Access Management for the Internet of Things". W: *2014 International Workshop on Secure Internet of Things*. 2014, s. 10–17. DOI: 10.1109/SIoT.2014.8.

[64] S. Ghaleb i in. „Mobility management for IoT: a survey". W: *EURASIP Journal on Wireless Communications and Networking* 2016 (lip. 2016). DOI: 10.1186/s13638-016-0659-4.

[65] F. Goichon i in. „Entropy transfers in the Linux Random Number Generator". W: (wrz. 2012).

[66] J. Golosova i A. Romanovs. „The Advantages and Disadvantages of the Blockchain Technology". W: *2018 IEEE 6th Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE)*. 2018, s. 1–6. DOI: 10.1109/AIEEE.2018.8592253.

[67] Google. *LevelDB github repository*. Dostępne w: https://github.com/google/leveldb.

[68] P. Grassi, M. Garcia i J. Fenton. *Digital Identity Guidelines*. SP 800-63-3. NIST, 2017. DOI: 10.6028/NIST.SP.800-63-3.

[69] P. Grassi i in. *Digital Identity Guidelines: Federation and Assertions*. SP 800-63C. NIST, 2017. DOI: 10.6028/NIST.SP.800-63c.

[70] B. Gregg. *Systems Performance: Enterprise and the Cloud*. Pearson, 2020. ISBN: 9780136821694. URL: https://www.oreilly.com/library/view/systems-performance-2nd/9780136821694/.

[71] C. Gu i in. „A Modeling Attack Resistant Deception Technique for Securing Lightweight-PUF-Based Authentication". W: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40.6 (2021), s. 1183–1196. DOI: 10.1109/TCAD.2020.3036807.

[72] J. Guajardo. „Physical Unclonable Functions (PUFs)". W: *Encyclopedia of Cryptography and Security*. Red. H. C. A. van Tilborg i S. Jajodia. Boston, MA: Springer US, 2011, s. 929–934. ISBN: 978-1-4419-5906-5. DOI: 10.1007/978-1-4419-5906-5_912.

[73] J. Guajardo i in. „FPGA Intrinsic PUFs and Their Use for IP Protection". W: *Cryptographic Hardware and Embedded Systems - CHES 2007*. Red. P. Paillier i I. Verbauwhede. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, s. 63–80. ISBN: 978-3-540-74735-2. DOI: 10.1007/978-3-540-74735-2_5.

[74] Z. Guoping i T. Jiazheng. „An extended role based access control model for the Internet of Things". W: *2010 International Conference on Information, Networking and Automation (ICINA)*. T. 1. 2010, s. V1-319-V1–323. DOI: 10.1109/ICINA.2010.5636381.

[75] S. Gusmeroli, S. Piccione i D. Rotondi. „A capability-based security approach to manage access control in the Internet of Things". W: *Mathematical and Computer Modelling* 58.5 (2013), s. 1189–1205. ISSN: 0895-7177. DOI: 10.1016/j.mcm.2013.02.006.

[76] M. El-hajj i in. „A Survey of Internet of Things (IoT) Authentication Schemes". W: *Sensors* 19.5 (2019). ISSN: 1424-8220. DOI: 10.3390/s19051141.

[77] A. Haleem i in. „Blockchain technology applications in healthcare: An overview". W: *International Journal of Intelligent Networks* 2 (2021), s. 130–139. ISSN: 2666-6030. DOI: 10.1016/j.ijin.2021.09.005.

[78] Hashscan. *Przeglądarka bloków*. URL: https://hashscan.io/mainnet/blocks?p=2010014001&k=11.

[79] V. Hassija i in. „A Survey on IoT Security: Application Areas, Security Threats, and Solution Architectures". W: *IEEE Access* PP (czer. 2019), s. 1–1. DOI: 10.1109/ACCESS.2019.2924045.

[80] Hedera. *Hedera - State and History*. Dostępne w: https://docs.hedera.com/hedera/core-concepts/state-and-history.

[81] C. Herder i in. „Physical Unclonable Functions and Applications: A Tutorial". W: *Proceedings of the IEEE* 102.8 (2014), s. 1126–1141. DOI: 10.1109/JPROC.2014.2320516.

[82] Holochain. *Strona projektu*. URL: https://www.holochain.org/what-holochain/.

[83] H. Hou. „The Application of Blockchain Technology in E-Government in China". W: *2017 26th International Conference on Computer Communication and Networks (ICCCN)*. 2017, s. 1–4. DOI: 10.1109/ICCCN.2017.8038519.

[84] V. Hu i in. *Guide to attribute based access control (ABAC) definition and considerations*. Sierp. 2019. DOI: 10.6028/NIST.SP.800-162.

[85] Hyperledger Fabric. *A Blockchain Platform for the Enterprise*. Dostępne w: https://hyperledger-fabric.readthedocs.io/en/release-2.2/.

[86] Hyperledger Fabric. *SDK*. Dostępne w: https://github.com/hyperledger/fabric-sdk-go.

[87] IETF. *A summary of security-enabling technologies for IoT devices*. URL: https://datatracker.ietf.org/doc/html/draft-ietf-iotops-security-summary-01.

[88] Inmcm. *Inmcm/xoodoo: Golang implementation of xoodoo permutation and XOOFF-F/Xoodyak crypto suites*. URL: https://github.com/inmcm/xoodoo.

[89] *Internet of Things Security Acquisition Guidance*. Spraw. tech. URL: https://www.cisa.gov/sites/default/files/publications/20_0204_cisa_sed_internet_of_things_acquisition_guidance_final_508.pdf.

[90] IOTA. *The Coordinator - PoA Consensus*. https://wiki.iota.org/learn/protocols/coordinator/. 2016. URL: https://wiki.iota.org/learn/protocols/coordinator/.

[91] ITU-T. *ITU-T: NGN Identity Management Framework - Recommendation Y.2720. Technical report*. Spraw. tech. ITU-T, 2009. URL: http://www.itu.int/rec/T-REC-Y.2720-200901-I/en.

[92] M. Jarosz, K. S. Wrona i Z. Zielinski. „Formal verification of security properties of the Lightweight Authentication and Key Exchange Protocol for Federated IoT devices". W: *Proceedings of the 17th Conference on Computer Science and Intelligence Systems, FedCSIS 2022, Sofia, Bulgaria, September 4-7, 2022*. Red. M. Ganzha i in. T. 30. Annals of Computer Science and Information Systems. 2022, s. 617–625. DOI: 10.15439/2022F169.

[93] J. Jindou, X. Qiu i C. Cheng. „Access Control Method for Web of Things Based on Role and SNS". W: paź. 2012, s. 316–321. ISBN: 978-1-4673-4873-7. DOI: 10.1109/CIT.2012.81.

[94] C. F. K. Bhargavan i A. Gordon. *F7: Refinement Types for F#*. 2014. URL: https://www.microsoft.com/en-us/research/project/f7-refinement-types-for-f/overview/.

[95] S. Kala i S. Nalesh. „Chapter 24 - Security and challenges in IoT-enabled systems". W: *System Assurances*. Red. P. Johri i in. Emerging Methodologies and Applications in Modelling. Academic Press, 2022, s. 437–445. ISBN: 978-0-323-90240-3. DOI: 10.1016/B978-0-323-90240-3.00024-2.

[96] B. Kaliski. *PKCS #5: Password-Based Cryptography Specification Version 2.0*. Spraw. tech. 2898. Sierp. 2000. 34 s. DOI: 10.17487/RFC2898.

[97] N. Kaliya i M. Hussain. „Framework for privacy preservation in iot through classification and access control mechanisms". W: *2017 2nd International Conference for Convergence in Technology (I2CT)*. 2017, s. 430–434. DOI: 10.1109/I2CT.2017.8226166.

[98] J. Kang i in. „Blockchain for Secure and Efficient Data Sharing in Vehicular Edge Computing and Networks". W: *IEEE Internet of Things Journal* 6.3 (2019), s. 4660–4670. DOI: 10.1109/JIOT.2018.2875542.

[99] A. V. D. M. Kayem, S. G. Akl i P. Martin. *Adaptive Cryptographic Access Control*. Springer, 2010. ISBN: 978-1-4614-2642-4. DOI: 10.1007/978-1-4419-6655-1.

[100] C. Keller i in. „Dynamic memory-based physically unclonable function for the generation of unique identifiers and true random numbers". W: *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2014, s. 2740–2743. DOI: 10.1109/ISCAS.2014.6865740.

[101] S. Khan i in. „Utilizing manufacturing variations to design a tri-state flip-flop PUF for IoT security applications". W: *Analog Integrated Circuits and Signal Processing* 103.3 (sty. 2020), s. 477–492. ISSN: 1573-1979. DOI: 10.1007/s10470-020-01642-9.

[102] J.-Y. Ko, S.-G. Lee i C.-H. Lee. „Real-time Mandatory Access Control on SELinux for Internet of Things". W: *2019 IEEE International Conference on Consumer Electronics (ICCE)*. 2019, s. 1–6. DOI: 10.1109/ICCE.2019.8662112.

[103] C. Koball i in. „IoT Device Identification Using Unsupervised Machine Learning". W: *Information* 14.6 (2023). ISSN: 2078-2489. DOI: 10.3390/info14060320.

[104] N. Kobeissi. *Verifpal User Manual*. 2023. URL: https://verifpal.com/res/pdf/manual.pdf.

[105] N. Kobeissi, G. Nicolas i M. Tiwari. *Verifpal: Cryptographic Protocol Analysis for the Real World*. Cryptology ePrint Archive, Paper 2019/971. 2019. URL: https://eprint.iacr.org/2019/971.

[106] J. Koo i Y.-G. Kim. „Interoperability of device identification in heterogeneous IoT platforms". W: *2017 13th International Computer Engineering Conference (ICENCO)*. 2017, s. 26–29. DOI: 10.1109/ICENCO.2017.8289757.

[107] J. Koo, S.-R. Oh i Y.-G. Kim. „Device Identification Interoperability in Heterogeneous IoT Platforms". W: *Sensors* 19.6 (2019). DOI: 10.3390/s19061433.

[108] Krzysztof Piech. *Leksykon pojęć na temat technologii blockchain oraz kryptowalut*. 2016. URL: https://www.gov.pl/documents/31305/0/leksykon_pojec_na_temat_technologii_blockchain_i_kryptowalut.pdf/77392774-1180-79ab-4dd5-089ffab37602.

[109] N. Kshetri. „Can Blockchain Strengthen the Internet of Things?" W: *IT Professional* 19.4 (2017), s. 68–72. DOI: 10.1109/MITP.2017.3051335.

[110] N. M. Kumar i P. K. Mallick. „The Internet of Things: Insights into the building blocks, component interactions, and architecture layers". W: *Procedia Computer Science* 132 (2018). International Conference on Computational Intelligence and Data Science, s. 109–117. ISSN: 1877-0509. DOI: 10.1016/j.procs.2018.05.170.

[111] A. Langley i in. *ChaCha20-Poly1305 Cipher Suites for Transport Layer Security (TLS)*. RFC 7905. Czer. 2016. DOI: 10.17487/RFC7905.

[112] P. Leach, M. Mealling i R. Salz. *A Universally Unique IDentifier (UUID) URN Namespace*. Spraw. tech. 4122. Lip. 2005. 31 s. DOI: 10.17487/RFC4122.

[113] P. Li i in. „A Vulnerability Detection Framework for Hyperledger Fabric Smart Contracts Based on Dynamic and Static Analysis". W: *Proceedings of the 26th International Conference on Evaluation and Assessment in Software Engineering*. EASE '22. Gothenburg, Sweden: Association for Computing Machinery, 2022, s. 366–374. ISBN: 9781450396134. DOI: 10.1145/3530019.3531342.

[114] R. Li i in. „Blockchain for Large-Scale Internet of Things Data Storage and Protection". W: *IEEE Transactions on Services Computing* 12.5 (2019), s. 762–771. DOI: 10.1109/TSC.2018.2853167.

[115] W. Li i in. „Cross-Domain Authentication Scheme for IoT Devices Based on Block-Chain". W: *2022 IEEE 13th International Conference on Software Engineering and Service Science (ICSESS)*. 2022, s. 67–73. DOI: 10.1109/ICSESS54813.2022.9930157.

[116] G. Liang i in. „Distributed Blockchain-Based Data Protection Framework for Modern Power Systems Against Cyber Attacks". W: *IEEE Transactions on Smart Grid* 10.3 (2019), s. 3162–3173. DOI: 10.1109/TSG.2018.2819663.

[117] *Lightweight cryptography*. URL: https://csrc.nist.gov/Projects/lightweight-cryptography.

[118] *Lightweight Cryptography Standardization Process: NIST Selects Ascon*. https://csrc.nist.gov/News/2023/lightweight-cryptography-nist-selects-ascon. Dostęp : 09-02-2023.

[119] Q. Liu i in. „An Access Control Model for Resource Sharing Based on the Role-Based Access Control Intended for Multi-Domain Manufacturing Internet of Things". W: *IEEE Access* 5 (2017), s. 7001–7011. DOI: 10.1109/ACCESS.2017.2693380.

[120] X. Liu i in. „A Blockchain-Based Medical Data Sharing and Protection Scheme". W: *IEEE Access* 7 (2019), s. 118943–118953. DOI: 10.1109/ACCESS.2019.2937685.

[121] Y. Liu i in. „Machine Learning for the Detection and Identification of Internet of Things Devices: A Survey". W: *IEEE Internet of Things Journal* 9.1 (2022), s. 298–320. DOI: 10.1109/JIOT.2021.3099028.

[122] G. Lowe. „A hierarchy of authentication specifications". W: *Proceedings 10th Computer Security Foundations Workshop*. 1997, s. 31–43. DOI: 10.1109/CSFW.1997.596782.

[123] LTO Network. *LTO_ Network On-chain Identities and Credentials Technical Implementation*. URL: https://www.ltonetwork.com/documents/LTO%20Network%20-%20Identities%20Tech.pdf.

[124] Y. Ma i in. „A survey of blockchain technology on security, privacy, and trust in crowdsourcing services". W: *World Wide Web* 23.1 (sty. 2020), s. 393–419. ISSN: 1573-1413. DOI: 10.1007/s11280-019-00735-4.

[125] R. Maes. „Physically Unclonable Functions: Constructions, Properties and Applications". Prac. dokt. Katholieke Universiteit Leuven – Faculty of Engineering, 2012.

[126] R. Maes i I. Verbauwhede. „Physically Unclonable Functions: A Study on the State of the Art and Future Research Directions". W: *Towards Hardware-Intrinsic Security: Foundations and Practice*. Red. A.-R. Sadeghi i D. Naccache. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, s. 3–37. ISBN: 978-3-642-14452-3. DOI: 10.1007/978-3-642-14452-3_1.

[127] M. Manso i in. „Connecting the Battlespace: C2 and IoT Technical Interoperability in Tactical Federated Environments". W: *MILCOM 2022 - 2022 IEEE Military Communications Conference (MILCOM)*. 2022, s. 1045–1052. DOI: 10.1109/MILCOM55135.2022.10017950.

[128] T. McGrath i in. „A PUF taxonomy". W: *Applied Physics Reviews* 6 (mar. 2019), s. 011303. DOI: 10.1063/1.5079407.

[129] S. Meier. „Advancing automated security protocol verification". en. Doctoral Thesis. Zürich: ETH Zurich, 2013. DOI: 10.3929/ethz-a-009790675.

[130] S. Meier i in. „The TAMARIN Prover for the Symbolic Analysis of Security Protocols". W: *Computer Aided Verification*. Red. N. Sharygina i H. Veith. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, s. 696–701. ISBN: 978-3-642-39799-8. DOI: 10.1007/978-3-642-39799-8_48.

[131] E. Mik. „Smart contracts: terminology, technical limitations and real world complexity". W: *Law, Innovation and Technology* 9.2 (2017), s. 269–300. DOI: 10.1080/17579961.2017.1378468.

[132] D. Miller i S. Josefsson. *The chacha20-poly1305@openssh.com authenticated encryption cipher*. Internet-Draft draft-josefsson-ssh-chacha20-poly1305-openssh-00. Work in Progress. Internet Engineering Task Force, list. 2015. 5 s. URL: https://datatracker.ietf.org/doc/draft-josefsson-ssh-chacha20-poly1305-openssh/00/.

[133] S. M. Mostafa, I. M. Darwish i M. R. Saadi. „Improved lightweight security approach routing protocol in internet of things". W: *Internet of Things* 11 (2020), s. 100208. ISSN: 2542-6605. DOI: 10.1016/j.iot.2020.100208.

[134] O. E. Mouaatamid, M. Lahmer i M. Belkasmi. „Internet of Things Security: Layered classification of attacks and possible Countermeasures". W: *Electronic Journal of Information Technology* (2016). URL: https://api.semanticscholar.org/CorpusID:54878156.

[135] D. Mukhopadhyay. „PUFs as Promising Tools for Security in Internet of Things". W: *IEEE Design & Test* 33.3 (2016), s. 103–115. DOI: 10.1109/MDAT.2016.2544845.

[136] Multichain. *Dokumentacja projektu*. URL: https://docs.multichain.org/getting-started/introduction.

[137] S. Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. Dostęp : 2015-07-01. Grud. 2008. URL: https://bitcoin.org/bitcoin.pdf.

[138] Q. Nasir i in. „Performance Analysis of Hyperledger Fabric Platforms". W: *Security and Communication Networks* 2018 (wrz. 2018), s. 3976093. ISSN: 1939-0114. DOI: 10.1155/2018/3976093.

[139] Y. Nir i A. Langley. *ChaCha20 and Poly1305 for IETF Protocols*. RFC 8439. Czer. 2018. DOI: 10.17487/RFC8439.

[140] K. C. Okafor, B. Adebisi i K. Anoh. „Lightweight multi-hop routing protocol for resource optimisation in edge computing networks". W: *Internet of Things* 22 (2023), s. 100758. ISSN: 2542-6605. DOI: 10.1016/j.iot.2023.100758.

[141] Open System Consultants. „RadSec a secure, reliable RADIUS Protocol". W: *Open System Consultants Pty. Ltd.* 0 (2012), s. 1–6. URL: http://www.open.com.au/radiator/radsec-whitepaper.pdf.

[142] *OWASP Top Ten Proactive Controls 2018*. Dostęp : 09-02-2023. URL: https://owasp.org/www-project-proactive-controls/v3/en/c7-enforce-access-controls.

[143] Z. Paral i S. Devadas. „Reliable and efficient PUF-based key generation using pattern matching". W: *2011 IEEE International Symposium on Hardware-Oriented Security and Trust*. 2011, s. 128–133. DOI: 10.1109/HST.2011.5955010.

[144] M. Pehl i in. „Spatial Context Tree Weighting for Physical Unclonable Functions". W: *2020 European Conference on Circuit Theory and Design (ECCTD)*. 2020, s. 1–4. DOI: 10.1109/ECCTD49232.2020.9218325.

[145] B. Pousse. *Short communication: An interpretation of the Linux entropy estimator*. Cryptology ePrint Archive, Paper 2012/487. https://eprint.iacr.org/2012/487. 2012. URL: https://eprint.iacr.org/2012/487.

[146] M. Pradhan i in. „Toward an Architecture and Data Model to Enable Interoperability between Federated Mission Networks and IoT-Enabled Smart City Environments". W: *IEEE Communications Magazine* 56.10 (2018), s. 163–169. DOI: 10.1109/MCOM.2018.1800305.

[147] M. Pradhan i in. „Toward an Architecture and Data Model to Enable Interoperability between Federated Mission Networks and IoT-Enabled Smart City Environments". W: *IEEE Communications Magazine* 56.10 (2018), s. 163–169. DOI: 10.1109/MCOM.2018.1800305.

[148] B. Putz, F. Böhm i G. Pernul. „HyperSec: Visual Analytics for blockchain security monitoring". W: *CoRR* abs/2103.14414 (2021). DOI: 10.48550/arXiv.2103.14414.

[149] A. Rejeb, J. G. Keogh i H. Treiblmaier. „Leveraging the Internet of Things and Blockchain Technology in Supply Chain Management". W: *Future Internet* 11.7 (2019). ISSN: 1999-5903. DOI: 10.3390/fi11070161.

[150] A. S. Richard Evers. „Quantum computers will not cause all forms of conventional encryption to become insecure". W: *kryptera.ca* (2019).

[151] R. Ross i in. *Developing Cyber-Resilient Systems: A Systems Security Engineering Approach*. SP 800-160 Vol. 2 Rev. 1. NIST, 2021. DOI: 10.6028/NIST.SP.800-160v2r1.

[152] A. Rukhin i in. *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*. SP 800-22 Rev.1a. NIST, 2010. DOI: 10.6028/NIST.SP.800-22r1a.

[153] J. K. Sadeghi R., V. R. Prybutok i B. Sauser. „Theoretical and practical applications of blockchain in healthcare information management". W: *Information & Management* 59.6 (2022), s. 103649. ISSN: 0378-7206. DOI: 10.1016/j.im.2022.103649.

[154] K. Sadique, R. Rahmani i P. Johannesson. „Identity Management in Internet of Things: A Software-Defined Networking Approach". W: sty. 2020, s. 495–504. ISBN: 978-981-15-0828-8. DOI: 10.1007/978-981-15-0829-5_48.

[155] S. Sallam i B. D. Beheshti. „A Survey on Lightweight Cryptographic Algorithms". W: *TENCON 2018 - 2018 IEEE Region 10 Conference*. 2018, s. 1784–1789. DOI: 10.1109/TENCON.2018.8650352.

[156] M. L. Santos i in. „FLAT: Federated lightweight authentication for the Internet of Things". W: *Ad Hoc Networks* 107 (2020), s. 102253. ISSN: 1570-8705. DOI: 10.1016/j.adhoc.2020.102253.

[157] M. Saqib i A. H. Moon. „A Systematic Security Assessment and Review of Internet of Things in the Context of Authentication". W: *Comput. Secur.* 125.C (lut. 2023). ISSN: 0167-4048. DOI: 10.1016/j.cose.2022.103053. URL: https://doi.org/10.1016/j.cose.2022.103053.

[158] S. Sciancalepore i in. „Attribute-Based Access Control Scheme in Federated IoT Platforms". W: kw. 2017, s. 123–138. ISBN: 978-3-319-56876-8. DOI: 10.1007/978-3-319-56877-5_8.

[159] J. Sengupta, S. Ruj i S. Das Bit. „A Comprehensive Survey on Attacks, Security Issues and Blockchain Solutions for IoT and IIoT". W: *Journal of Network and Computer Applications* 149 (2020), s. 102481. ISSN: 1084-8045. DOI: 10.1016/j.jnca.2019.102481.

[160] N. K. Shah i in. „Smart Contract Vulnerability Detection Techniques for Hyperledger Fabric". W: *2023 IEEE 8th International Conference for Convergence in Technology (I2CT)*. 2023, s. 1–7. DOI: 10.1109/I2CT57861.2023.10126362.

[161] Z. Shah i in. „Routing Protocols for Mobile Internet of Things (IoT): A Survey on Challenges and Solutions". W: *Electronics* 10.19 (2021). ISSN: 2079-9292. DOI: 10.3390/electronics10192320.

[162] M. A. Shahriar i in. „Modelling Attacks in Blockchain Systems using Petri Nets". W: *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. 2020, s. 1069–1078. DOI: 10.1109/TrustCom50675.2020.00142.

[163] Z. Shelby, K. Hartke i C. Bormann. *The Constrained Application Protocol (CoAP)*. Spraw. tech. 7252. Czer. 2014. 112 s. DOI: 10.17487/RFC7252.

[164] M. Shen i in. „Blockchain-Assisted Secure Device Authentication for Cross-Domain Industrial IoT". W: *IEEE Journal on Selected Areas in Communications* 38.5 (2020), s. 942–954. DOI: 10.1109/JSAC.2020.2980916.

[165] N. Shi i in. „BacS: A blockchain-based access control scheme in distributed internet of things". W: *Peer-to-Peer Networking and Applications* 14.5 (wrz. 2021), s. 2585–2599. ISSN: 1936-6450. DOI: 10.1007/s12083-020-00930-5.

[166] M. Shiozaki, Y. Hori i T. Fujino. *Entropy Estimation of Physically Unclonable Functions with Offset Error*. Cryptology ePrint Archive, Paper 2020/1284. 2020. URL: https://eprint.iacr.org/2020/1284.

[167] S. Sicari i in. „Security, privacy and trust in Internet of Things: The road ahead". W: *Computer Networks* 76 (sty. 2015). DOI: 10.1016/j.comnet.2014.11.008.

[168] J. Singh i in. „Accountability in the IoT: Systems, Law, and Ways Forward". W: *Computer* 51.7 (2018), s. 54–65. DOI: 10.1109/MC.2018.3011052.

[169] A. Singhal, T. Winograd i K. Scarfone. *Guide to Secure Web Services*. SP 800-95. NIST, 2007. DOI: 10.6028/NIST.SP.800-95.

[170] A. Skarmeta, D. Garcia Carrillo i A. Olivereau. „End-Node Security". W: *Internet of Things Security and Data Protection*. Red. S. Ziegler. Cham: Springer International Publishing, 2019, s. 45–69. ISBN: 978-3-030-04984-3. DOI: 10.1007/978-3-030-04984-3_3.

[171] R. Soltani i in. „Distributed Ledger Technologies and Their Applications: A Review". W: *Applied Sciences* 12.15 (2022). ISSN: 2076-3417. DOI: 10.3390/app12157898.

[172] W. Stallings. *Cryptography and Network Security: Principles and Practice*. 7th. USA: Prentice Hall Press, 2017. ISBN: 1292158581.

[173] W. Stallings. *Operating Systems Internals and Design Principles*. 9th. USA: Prentice Hall Press, 2018. ISBN: 1-292-21429-5.

[174] N. I. of Standards i Technology. „Advanced Encryption Standard". W: *NIST FIPS PUB 197* (2001). DOI: 10.6028/NIST.FIPS.197-upd1.

[175] Steve Syfuhs. *Why is Kerberos Terrible?* 2018. URL: https://syfuhs.net/2018/12/31/why-is-kerberos-terrible/.

[176] M. M. Steven Michael Bellovin. „Limitations of the Kerberos Authentication System". W: *Proceedings of the Winter 1991 USENIX Conference* (1991), s. 253–267. ISSN: 1570-8705. DOI: 10.7916/D84B372N.

[177] M. Stoyanova i in. „A Survey on the Internet of Things (IoT) Forensics: Challenges, Approaches, and Open Issues". W: *IEEE Communications Surveys & Tutorials* 22.2 (2020), s. 1191–1221. DOI: 10.1109/COMST.2019.2962586.

[178] S. Symington, W. Polk i M. Souppaya. *Trusted Internet of Things (IoT) Device Network-Layer Onboarding and Lifecycle Management*. Spraw. tech. NIST, 2020. DOI: 10.6028/NIST.CSWP.16.ipd.

[179] W. Szmuc. „Modelowanie wybranych diagramów języka UML 2.0 z zastosowaniem kolorowych sieci Petriego". Prac. dokt. Akademia Górniczo-Hotnicza im. Stanisława Staszica w Krakowie, 2014.

[180] M. Szpyrka. *Sieci Petriego w modelowaniu i analizie systemów współbieżnych*. Inżynieria Oprogramowania - Wydawnictwa Naukowo-Techniczne. Wydawnictwa Naukowo-Techniczne, 2008. ISBN: 9788320433784. URL: https://books.google.pl/books?id=YALSKwAACAAJ.

[181] T. T. Team. *Tamarin-Prover Manual*. Grud. 2023. URL: https://tamarin-prover.com/manual.

[182] F. Tehranipoor i in. „DRAM-Based Intrinsic Physically Unclonable Functions for System-Level Security and Authentication". W: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 25.3 (2017), s. 1085–1097. DOI: 10.1109/TVLSI.2016.2606658.

[183] Thomas' Digital Garden. *Myths about /dev/urandom*. 2021. URL: Dost%C4%99pne%20w:%20%5Curl%7Bhttps://www.2uo.de/myths-about-urandom/#from-linux-48-onward%7D.

[184] F. Tong i in. „CCAP: A Complete Cross-Domain Authentication Based on Blockchain for Internet of Things". W: *IEEE Transactions on Information Forensics and Security* 17 (2022), s. 3789–3800. DOI: 10.1109/TIFS.2022.3214733.

[185] D. K. Tosh i in. „Blockchain-Empowered Secure Internet -of- Battlefield Things (IoBT) Architecture". W: *MILCOM 2018 - 2018 IEEE Military Communications Conference (MILCOM)*. 2018, s. 593–598. DOI: 10.1109/MILCOM.2018.8599758.

[186] M. S. Turan i in. *Recommendation for the Entropy Sources Used for Random Bit Generation*. SP 800-90B. NIST, 2018. DOI: 10.6028/NIST.SP.800-90B.

[187] M. S. Turan i in. *Status Report on the Final Round of the NIST Lightweight Cryptography Standardization Process*. IR 8454. NIST, 2023. DOI: 10.6028/NIST.IR.8454.

[188] use.ink. *Why Rust for Smart Contracts?* Dostępne w: https://use.ink/why-rust-for-smart-contracts.

[189] use.ink. *Why WebAssembly for Smart Contracts?* Dostępne w: https://use.ink/why-webassembly-for-smart-contracts.

[190] vyperlang.org. *Vyper*. Dostępne w: https://docs.vyperlang.org/en/stable.

[191] J. Wang i in. „Trust and Attribute-Based Dynamic Access Control Model for Internet of Things". W: *2017 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*. 2017, s. 342–345. DOI: 10.1109/CyberC.2017.47.

[192] X. Wang i in. „A Certificateless-Based Authentication and Key Agreement Scheme for IIoT Cross-Domain". W: *Security and Communication Networks* 2022 (paź. 2022), s. 3693748. ISSN: 1939-0114. DOI: 10.1155/2022/3693748.

[193] N. Will. *Development of a Measurement Setup for Large Scale SRAM PUF Evaluation*. Ruhr-Universität Bochum. 2017. URL: https://informatik.rub.de/wp-content/uploads/2021/11/BA_Will.pdf.

[194] K. Wrona, F. Micevski-Scharf i M. Jarosz. „Security Accreditation and Software Approval with Smart Contracts". W: *IEEE Communications Magazine* 59.2 (2021), s. 56–62. DOI: 10.1109/MCOM.001.2000802.

[195] K. Wüst i A. Gervais. „Do you Need a Blockchain?" W: *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*. 2018, s. 45–54. DOI: 10.1109/CVCBT.2018.00011.

[196] X. Xiao, F. Guo i A. Hecker. „A Lightweight Cross-Domain Proximity-Based Authentication Method for IoT Based on IOTA". W: *2020 IEEE Globecom Workshops (GC Wkshps*. 2020, s. 1–6. DOI: 10.1109/GCWkshps50303.2020.9367500.

[197] R. Xu i in. „A Federated Capability-based Access Control Mechanism for Internet of Things (IoTs)". W: *CoRR* abs/1805.00825 (2018). DOI: 10.48550/arXiv.1805.00825.

[198] D. Yaga i in. *Blockchain Technology Overview*. Spraw. tech. 8202. NIST, 2018.

[199] R. Yang, B. Li i C. Cheng. „A Petri Net-Based Approach to Service Composition and Monitoring in the IOT". W: *2014 Asia-Pacific Services Computing Conference*. 2014, s. 16–22. DOI: 10.1109/APSCC.2014.11.

[200] Y. Yang i in. „A Blockchain-Based Multidomain Authentication Scheme for Conditional Privacy Preserving in Vehicular Ad-Hoc Network". W: *IEEE Internet of Things Journal* 9.11 (2022), s. 8078–8090. DOI: 10.1109/JIOT.2021.3107443.

[201] S. Yoon, J. Kim i Y.-S. Jeon. „Security Considerations based on Classification of IoT Device Capabilities". W: 2017.

[202] Y. Zhang i in. „A Lightweight Authentication Scheme Based on Consortium Blockchain for Cross-Domain IoT". W: *Security and Communication Networks* 2022 (sty. 2022), s. 9686049. ISSN: 1939-0114. DOI: 10.1155/2022/9686049.

[203] X. Zhu i Y. Badr. „A Survey on Blockchain-Based Identity Management Systems for the Internet of Things". W: *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. 2018, s. 1568–1573. DOI: 10.1109/Cybermatics_2018.2018.00263.

[204] G. Zyskind, O. Nathan i A. Pentland. „Enigma: Decentralized Computation Platform with Guaranteed Privacy". W: *CoRR* abs/1506.03471 (2015). DOI: 10.48550/arXiv.1506.03471.