Kauno technologijos universitetas

Informatikos fakultetas

Blokų grandinėmis grįstas galinių įrenginių autentifikavimo ūko kompiuterijoje metodas

Baigiamasis magistro krypties studijų projektas

Žilvinas Radzevičius

Projekto autorius

prof. Algimantas Venčkauskas

Vadovas

Kaunas, 2019

Kauno technologijos universitetas

Informatikos fakultetas

Blokų grandinėmis grįstas galinių įrenginių autentifikavimo ūko kompiuterijoje metodas

Baigiamasis magistro krypties studijų projektas

Informacijos ir informacinių technologijų sauga (6211BX008)

Žilvinas Radzevičius

Projekto autorius

prof. Algimantas Venčkauskas

Vadovas

doc. Stasys Maciulevičius

Recenzentas

Kaunas, 2019

Kauno technologijos universitetas

Informatikos fakultetas

Žilvinas Radzevičius

Blokų grandinėmis grįstas galinių įrenginių autentifikavimo ūko kompiuterijoje metodas

Akademinio sąžiningumo deklaracija

Patvirtinu, kad mano, Žilvino Radzevičiaus, baigiamasis projektas tema „Blokų grandinėmis grįstas galinių įrenginių autentifikavimo ūko kompiuterijoje metodas“ yra parašytas visiškai savarankiškai ir visi pateikti duomenys ar tyrimų rezultatai yra teisingi ir gauti sąžiningai. Šiame darbe nei viena dalis nėra plagijuota nuo jokių spausdintinių ar internetinių šaltinių, visos kitų šaltinių tiesioginės ir netiesioginės citatos nurodytos literatūros nuorodose. Įstatymų nenumatytų piniginių sumų už šį darbą niekam nesu mokėjęs.

Aš suprantu, kad išaiškėjus nesąžiningumo faktui, man bus taikomos nuobaudos, remiantis Kauno technologijos universitete galiojančia tvarka.

(vardą ir pavardę įrašyti ranka)	(parašas)

Radzevičiaus, Žilvino. Blokų grandinėmis grįstas galinių įrenginių autentifikavimo ūko kompiuterijoje metodas. Magistro studijų baigiamasis projektas vadovas prof. Algimantas Venčkauskas; Kauno technologijos universitetas, informatikos fakultetas.

Studijų kryptis ir sritis: Informatikos inžinerija, informacijos ir informacinių technologijų sauga.

Reikšminiai žodžiai: blokų grandinės, ūko kompiuterija, paskirstytosios sistemos, daiktų internetas, autentifikacija.

Kaunas, 2019. 60 p.

Santrauka

Magistrantūros studijų baigiamajame darbe atliktas blokų grandinėmis grįsto galinių įrenginių autentifikavimo ūko kompiuterijoje metodo tyrimas ir taikymas. Spartus daiktų interneto įrenginių kiekio ir jų panaudojimo informacinėse sistemose augimas sukelia naujas saugos ir architektūrines problemas, kurių sprendimui nepakanka naudoti tradicinio debesų kompiuterijos veikimo modelio. Dėl šios priežasties, pradėtas naudoti naujas kompiuterijos architektūrinis modelis, vadinamas ūko kompiuterija. Galinių įrenginių autentifikavimas ūko kompiuterijoje yra viena iš didžiausių problemų daiktų interneto srityje, autentifikavimo spragos kelia grėsmę tiek vartotojams, tiek sistemoms, kuriose šie įrenginiai veikia. Naudojant blokų grandinių technologijas siekiama išspręsti galinių įrenginių autentifikavimo uždavinį.

Baigiamojo darbo įvade apžvelgiama darbo problematika ir aktualumas, iškeliami tikslai ir uždaviniai. Aptariama, kokie bus šios sistemos privalumai, ir kokią svarbą bei naudą teikia kuriamas metodas.

Analizės dalyje atliekama ūko kompiuterijos architektūros, blokų grandinių sistemų, galinių įrenginių transporto protokolų ir žinučių brokerių sistemų palyginimas ir analizė. Formuluojami darbo tikslai, uždaviniai ir siekiami sistemos privalumai. Pateikiamas siekiamo sprendimo apibrėžimas, projektavimo gairės ir analizės išvados.

Projektinėje dalyje pateikiami sistemos projektavimo etapai ir esminės kuriamo autentifikavimo metodo funkcijos sistemoje. Projektuojama sistemos architektūra ūko kompiuterijos sluoksnyje ir sistemą sudarančių komponentų tarpusavio sąveika. Taip pat, projektuojami galinių įrenginių autentifikavimo procesai blokų grandinėse. Pateikiama projektuojamos blokų grandinių sistemos architektūra ir duomenų struktūros. Skyriaus pabaigoje pateikiamos projektinės dalies įžvalgos ir išvados.

Sprendimo realizavimo dalyje pristatomas sistemos realizavimo modelis, prototipo kūrimui naudojami įrankiai ir sistemos. Detalizuojamas sistemos konfigūravimas, funkcionalumas ir panaudos atvejai bei plėtojimo galimybės.

Tyrimo dalyje tiriami sistemos kiekybiniai ir kokybiniai rodikliai. Pateikiami tyrimo rezultatai, aptariami sukurto sistemos prototipo trūkumai, apribojimai ir tolimesnio plėtojimo galimybės.

Darbo pabaigoje įvardijami pasiekti rezultatai ir išvados, apibendrinančios visą projektą ir aprašančios darbo autoriaus pastebėjimus. Taip pat, pateikiamas literatūros sąrašas ir priedai.

Radzevičius, Žilvinas. Method for Fog Computing End Device's Authentication Based on BlockChain. Master's Final Degree Project supervisor prof. Algimantas Venčkauskas; Faculty of Informatics, Kaunas University of Technology.

Study field and area: Informatics Engineering, Information and Information Technology Security.

Keywords: blockchain, fog computing, distributed systems, internet of things, authentication.

Kaunas, 2019. 60 p.

Summary

The master thesis presents implementation and research of a method for fog computing end devices' authentication based on Blockchain. Rapid growth of the Internet of Things devices and their usage in information systems introduce new security and architectural issues which cannot be solved by using traditional Cloud Computing architecture. Therefore, a new architecture known as fog computing is being used to overcome security threats and fill in the architectural gaps of traditional Cloud Computing solutions. Authentication of end devices is the most common security issue that introduces security threats to end users and systems that incorporate end devices. Usage of Blockchain technologies aims to solve end device authentication problems.

Introduction covers the overview of main problems and their relevance, the aims and tasks of the work are also raised. Brief overview of the to-be developed method, its virtues, importance and benefits are provided in the introduction section as well.

Analysis section covers the analysis and comparison of fog computing architecture, Blockchain systems, end devices' transport protocols, message broker systems. The aims of the work are being stated along with the tasks and functionality that system must provide. Description of the required solution, development guidelines and analysis conclusions are drawn.

System design section provides system planning phases and describes fundamental functions required for the authentication system. The architecture of authentication system and its interaction between system components is being designed in the fog computing layer. End device authentication functions in Blockchain are being modelled. Data structures and architectural diagrams of system's design are being presented as well. At the end of this section insights and conclusions are drawn.

During solution development section system development model, tools and systems used for development are being presented. Detailed system configuration instructions, functionality, use cases and future development possibilities are provided in this section as well.

Research chapter provides the experiments and insights of system's quantitative and qualitative indicators. Results of tests, experiments, flaws, constraints and future development opportunities are discussed at the end of analysis section.

Turinys

Lentelių sąrašas

Paveikslų sąrašas

Santrumpų ir terminų sąrašas

Terminai:

IoT (angl. Internet of Things) – įvairaus pobūdžio fizinių įrenginių tinklas, kuris turi galimybę prisijungti prie interneto ir keistis duomenimis.

Debesija (angl. Cloud) – interneto paslaugų visuma, jungianti įvairiuose serveriuose esančius informacijos išteklius ir programinę įrangą, sudaranti sąlygas jais naudotis.

MQ (angl. Message Queuing) – duomenų žinučių pavidale siuntimas ir jų eiliškumo užtikrinimas.

Blokų grandinė (angl. Blockchain) – technologija, kuri remiasi nuolatos augančiais įrašų sąrašais, įrašai yra susiejami ir apsaugojami kriptografiškai.

HTTP (angl. Hyper Text Transfer Protocol) – Hipertekstų persiuntimo protokolas saityno duomenims (ištekliams) persiųsti

Web aplikacija – žiniatinklio programa pasiekiama naudojant internetines naršykles.

API (angl. Application Programming Interface) – susitarimų ir procedūrų rinkinys ryšiams tarp atskirų programų ir operacinės sistemos realizuoti.

Įvadas

Magistro baigiamasis projektas, atliktas pagal informacijos ir informacinių sistemų technologijų saugos magistro studijų programą (programos kodas 6211BX008).

Darbo naujumas ir aktualumas

Populiarėjant IoT (angl. Internet of Things) prietaisų naudojimo pritaikymui informacinių technologijų srityje, didėja ir galinių prietaisų saugumo, autentifikavimo ir identiteto valdymo problemos aktualumas informacinių technologijų ūkyje. Sistemose, kuriose naudojama daug IoT prietaisų, reikia didelio interneto pralaidumo bei duomenų apdorojimo spartos, šių resursų ar pakankamos jų spartos galiniai įrenginiai turi labai mažai. Todėl reikalingas papildomas sluoksnis tarp debesijos ir galinių įrenginių. Dėl šios priežasties pradėta naudoti nauja architektūra, kurioje veikia tarpinis sluoksnis tarp galinių IoT įrenginių ir debesijos paslaugų, vadinamas ūko kompiuterijos sluoksniu. Blokų grandinių technologijų populiarumui augant, ši technologija pradėta taikyti ne tik kriptovaliutų sistemose ir finansiniame sektoriuje. Blokų grandinių technologijos siekia užtikrinti sistemos ir jos duomenų decentralizuotumą, duomenų apsaugą, naudojant kriptografinius sprendimus, duomenų nekintamumą ir informacijos auditą.

Ūko kompiuterijos architektūrinis modelis teikia daug privalumų ir papildo standartinį debesų kompiuterijos architektūrinį modelį. Kadangi dauguma IoT prietaisų neturi didelių skaičiavimo resursų bei spartaus duomenų perdavimo galimybių, naudojami papildomi ūko prietaisai, kurie suteikia galimybę naudoti skaičiavimo resursus ūko sluoksnyje ir optimizuoti tinklo srautą prieš perduodant duomenis į debesijos paslaugas. Ūko prietaisai, gali būti klasikiniai serveriai ar specializuoti įrenginiai, kurie keičiasi informacija su galiniais prietaisais. Didžiausias skirtumas nuo debesijos modelio yra tai, kad ūko prietaisai yra geografiškai arti galinių prietaisų, atvirkščiai nei nuo debesijos paslaugų.

Kadangi IoT prietaisai yra įvairaus pobūdžio ir paskirčių, reikalingas šių prietaisų saugus autentifikavimas sistemose. Tradicinių autentifikavimo sistemų taikymai ūko kompiuterijos architektūroje nėra našūs ir sunkiai pritaikomi judriame ir nuolat kintančiame ūko kompiuterijos sluoksnyje. Paskirstyta autentifikavimo sistema ūko sluoksnyje leistų įgyvendinti saugų ir patikimą būdą autentifikuoti galinius prietaisus. Blokų grandinių technologijų taikymas galinių įrenginių autentifikavimui, gali užtikrinti autentifikavimo informacijos kaupimo saugumą, sistemos decentralizuotumą ir informacijos audito galimybes.

Darbo tikslas ir uždaviniai

Darbo tikslas – sukurti galinių įrenginių autentifikavimo metodą ūko kompiuterijoje, taikant blokų grandinių technologiją. Darbo metu bus tiriamos blokų grandinių technologijos ir jų pritaikymo galimybės autentifikuoti galinius įtaisus ūko kompiuterijos architektūroje.

Uždaviniai:

1. atlikti analizę ir ištirti esamus blokų grandinių technologijų taikymus autentifikacijai atlikti ūko kompiuterijoje;
2. suprojektuoti sistemą, paremtą blokų grandininių technologijomis ūko kompiuterijoje autentifikacijos uždaviniui spręsti;

3. realizuoti prototipą galinių įrenginių autentifikacijai, pritaikant blokų grandinių technologijas ūko kompiuterijoje;

4. eksperimentiškai ištirti ir įvertinti sukurtą metodą.

Darbo rezultatai ir jų svarba

Ištirtas ūko kompiuterijos architektūrinis modelis ir apibrėžta projektuojamos sistemos veikimo vieta ūko kompiuterijos architektūriniame modelyje. Atlikta esamų autentifikavimo metodų taikymo ūko kompiuterijoje analizė. Atlikta blokų grandinių sistemų analizė ir šių sistemų taikymas ūko kompiuterijoje. Ištirti lengvasvoriai protokolai, naudojami duomenų perdavimui tarp galinių įrenginių ir ūko sluoksnio. Remiantis atlikta analize, suprojektuota galinių įrenginių autentifikavimo sistema. Pagal projektavimo metu apibrėžtą sistemos architektūrinį modelį, sukurta blokų grandinėmis grįsta sistema, skirta galinių įrenginių autentifikavimui ūko kompiuterijoje. Atliktas sukurto sistemos prototipo kiekybinių ir kokybinių rodiklių tyrimas ir palyginimas su esamomis autentifikavimo sistemomis.

Darbo struktūra

Darbą sudaro 4 skyriai, išvados, naudotos literatūros sąrašas ir priedai.

Analizės skyriuje analizuojami kuriamo metodo komponentai, problematika ir įgyvendinimo galimybės bei panašios sistemos.

Projektinės dalies skyriuje pateikiama sistemos reikalavimų specifikacija, veikimo modelis, veikimo aprašymas su atitinkamomis diagramomis.

Realizacijos skyriuje aprašomas sistemos prototipo realizavimas ir veikimas. Aptariami sistemos trūkumai ir tobulinimo galimybės.

Tyrimo skyriuje aptariamas sistemos našumas ir tiriami sistemos kokybiniai ir kiekybiniai rodikliai, pateikiamos tyrimo ir eksperimentų išvados.

1. Galinių įrenginių autentifikavimo metodų ūko kompiuterijoje analizė

1.1. Analizės tikslas

Analizės metu yra analizuojama literatūra, susijusi su ūko kompiuterija, jos architektūra ir taikymu. Išsiaiškinamos probleminės sritys, saugumo trūkumai, esami autentifikavimo sprendimai ūko kompiuterijoje, jų pritaikymo galimybės ir alternatyvos. Taip pat analizuojamas blokų grandinių technologijos pritaikymo galimybės galinių įrenginių autentifikavimui ūko kompiuterijoje. Apžvelgiamos publikacijos ir esami produktai, teikiantys įrenginių autentifikavimo funkcionalumą.

1.2. Tyrimo objektas, sritis ir problema

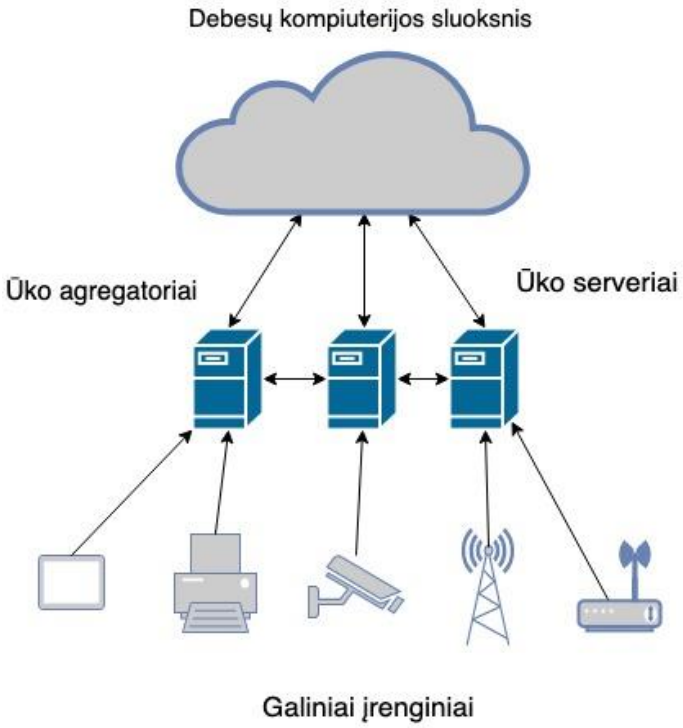
Pagrindiniai šio darbo tyrimo objektai yra ūko kompiuterijos architektūra, blokų grandinių technologijos ir galinių įrenginių autentifikavimas. Ūko kompiuterijos sąvoka yra plati, todėl analizės metu bus nustatyti esminiai tyrimo objektai ir jų probleminės sritys. Kadangi darbo tikslas yra sukurti blokų grandinėmis grįstą galinių įrenginių autentifikavimo ūko kompiuterijoje metodą, visų pirmiausia reikia išanalizuoti ūko kompiuterijos architektūrą ir veikimo pricipus; blokų grandinių technologijų ir platformų taikomumą autentifikacijos uždaviniui spręsti; nustatyti vietas, kuriose galima įgyvendinti galinių įrenginių autentifikavimo mechanizmą ir priimti sprendimą, kuriame ūko kompiuterijos sluoksnyje veiks autentifikacijos sistema.

Kadangi galiniai įrenginiai yra labai skirtingo pobūdžio ir turi skirtingus techninius parametrus, yra sudėtinga naudoti esamas autentifikacijos priemones. Todėl reikalingas naujas autentifikacijos sprendimas, kuris įgyvendintų galinių įrenginių autentifikacijos uždavinį. Ši sistema neturėtų ženkliai padidinti resursų naudojimo galiniuose įrenginiuose, kadangi IoT prietaisai turi labai ribotus skaičiavimo ir duomenų perdavimo išteklius, dėl to gali sumažėti įrenginių našumas.

Klasikinių autentifikavimo ir saugumo sprendimų taikymas ūko kompiuterijos architektūroje nėra tinkamas dėl aparatinės įrangos techninių reikalavimų šioms sistemoms. LDAP (angl. Lightweight Directory Access Protocol) ar *Active Domain* sistemoms reikia daug sisteminių bei aparatūrinių resursų [1], kurių galiniai įrenginiai dažniausiai negali skirti autentifikacijai atlikti, tačiau tai nėra būdinga visiems įrenginiams, kadangi galiniai įrenginiai yra labai skirtingi, tiek turimais resursais, tiek programine įranga. Dėl skirtingų įrenginių gausos yra sunku pritaikyti esamas technologijas autentifikacijos uždaviniui ūko kompiuterijoje spręsti. Standartiniai blokų grandinių technologijų taikymai, kurių panaudos atvejai reikalauja viešųjų ir privačiųjų raktų porų generavimo galiniuose įrenginiuose, taip pat gali sudaryti papildomų resursų sąnaudų galiniuose įrenginiuose. Dėl to, blokų grandinių technologijų ir sistemų taikymas galinių įrenginių autentifikavimo metode, turi būti adekvačiai parinktas sistemos projektavimo metu. Projektuojama sistema, veikianti ūko kompiuterijos architektūroje, neturi ženkliai padidinti aparatūrinių resursų sąnaudų ir duomenų perdavimo trukmės galiniuose įrenginiuose autentifikacijai atlikti.

1.3. Blokų grandinėmis grįsto autentifikavimo metodo ūko kompiuterijoje analizė

Autentifikacija ūko kompiuterijoje modelyje yra atliekama tarp galinių įtaisų ir ūko kompiuterijos architektūros įrenginių, bendru atveju autentifikacija atliekama ūko-galinių įrenginių sluoksnyje. Bendra ūko kompiuterijos pavyzdinė architektūra pavaizduota 1.1 pav.



1.1 pav. Ūko kompiuterijos architektūros pavyzdinė diagrama

Debesija – paslauga ar fizinių serverių visuma, kurioje atliekami pagrindiniai skaičiavimai ar kaupiami duomenys. Įprastu atveju, galiniai įrenginiai siunčia duomenis tiesiogiai į debesis, tokiu atveju yra užimamas labai didelis interneto tinklo srautas ir mažinamas bendras tinklo pralaidumas. Nėra naudinga siųsti visus surinktus duomenis į debesijos paslaugas iš karto, priklausomai nuo pobūdžio, vieni duomenys yra reikalingi realiuoju laiku, o kiti duomenys yra agreguojami ir kaupiami ilgalaikei statistikai.

Ūko sluoksnį sudaro ūko serveriai ir ūko agregatoriai [2]. Šis sluoksnis veikia kaip tarpinė stotis tarp debesų ir galinių įrenginių. Ūko serveriai ir agregatoriai gali būti klasikiniai serveriai ar kompiuteriai, tačiau jie yra kiek įmanoma geografiškai arčiau galinių įrenginių. Tokiu būdu sumažinami vėlinimo laikai tarp galinių prietaisų ir debesų, tai yra labai svarbu galinių įrenginių informacijai perduoti. Tarp ūko ir debesų sluoksnio gali veikti įprastiniai autentifikacijos metodai, kadangi tai yra plačiai naudojama debesų architektūroje, dėl to gali nereikėti naujų administravimo priemonių ir saugumo sprendimų debesų-ūko sluoksnyje. Ūko sluoksnyje nėra būtinas agregatorių panaudojimas, tačiau jų panaudojimas gali optimizuoti ar išplėsti sistemos veikimą.

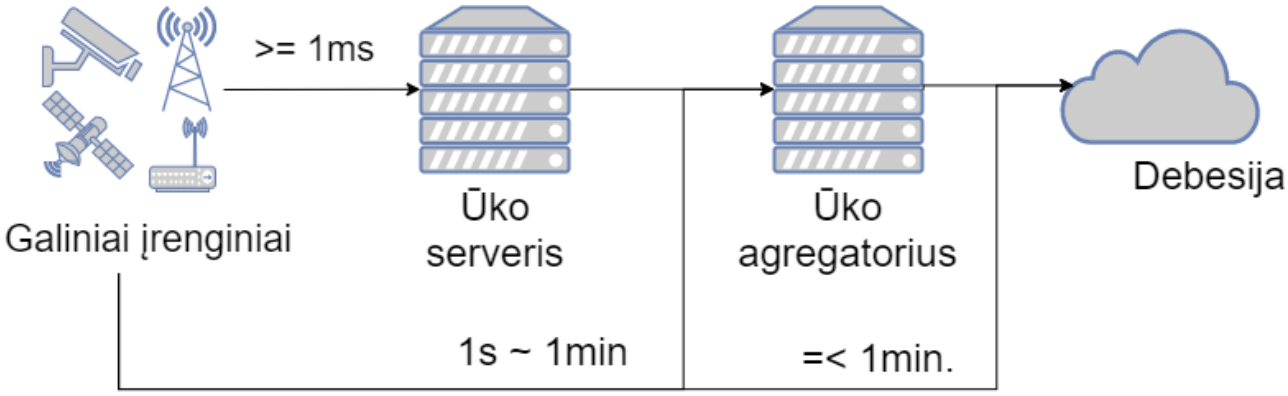
Galinių įrenginių sluoksnis apima visus IoT ir mobiliuosius įrenginius. Galiniai įrenginiai yra skirstomi pagal duomenų ir skaičiavimo operacijų apdorojimo svarbą. Šie įrenginiai gali būti mobilieji telefonai, termometrai, apsaugos ar IP kameros, judesio davikliai ir kito pobūdžio prietaisai. Galiniai įrenginiai internetu ar kitomis duomenų perdavimo priemonėmis perduoda duomenis ir informaciją kitoms sistemoms. Šie duomenys gali būti kritiškai svarbūs, dėl to duomenų vėlinimas turi būti kuo mažesnis. Tokie prietaisai, kaip termometrai ar judesio davikliai gali veikti žmogaus gyvybės saugumą ar kritinių sistemų veikimą. Kadangi galiniai įrenginiai turi nevienodas ryšio, interneto spartos ir skaičiavimo galimybes, duomenys dažniausiai yra apdorojami debesyse. Taip siekiama sumažinti galinių įrenginių ribotų skaičiavimo resursų naudojimą, tačiau užimamas interneto srautas ir didinamas tinklo vėlinimo laikas. Todėl skaičiavimų atlikimas ūko sluoksnyje,

mažina duomenų perdavimo vėlinimą ir optimizuoja tinklo duomenų srautą, kuriuo duomenys siunčiami į debesijos paslaugas.

Galinių įrenginių autentifikacija yra labai svarbus aspektas ūko kompiuterijos architektūroje, kadangi neautentifikuoti įrenginiai gali pažeisti sistemą ar klastoti duomenis, kuriuos sistema kaupia ar apdoroja. Netinkamas ar nepakankamas galinių įrenginių administravimas ir saugumo reikalavimų nesilaikymas, kelia grėsmę ne tik taikomosioms sistemoms, nesaugūs galiniai įrenginiai gali tapti stambaus mąsto kibernetinių atakų įrankiu [3]. Prietaisų ir vartotojų įvairovė galinių įrenginių sluoksnyje apsunkina autentifikacijos uždavinį. Galinis įrenginys gali būti išmaniųjų namų sistema, kurioje įvairūs galiniai prietaisai siunčia informaciją bendram naudojimui debesyse ar bendrai rajono statistikai kaupti. Kadangi įrenginiai namuose gali būti kompiuteriai ar išmanieji telefonai, jų saugumo spragos gali veikti saugų ūko kompiuterijos veikimą.

Ūko kompiuterijos sluoksnio prietaisų svarba gali būti skirstomi pagal duomenų apdorojimo greičio reikalavimus [4] [5]. Skirtingo lygio sistemos ir jų keliami reikalavimai gali veikti duomenų apdorojimo svarbumą ir greitį, šie reikalavimai yra individualūs kiekvienai sistemai. Bendros rekomendacijos duomenų apdorojimo greičiui pagal svarbą pavaizduotos 1.2 pav.

- Prietaisai ir paslaugos, kurios reikalauja atsakymo greičio iki milisekundės ar mažesnio, duomenys turi būti apdorojami ūko serveriuose, kurie yra arčiausiai minimo prietaiso, tai gali būti daugiausiai rajono atstumu. Šie duomenys turi būti apdoroti beveik realiuoju laiku, tai gali būti medicinos prietaisai ar termometrai kritinėse sistemose.
- Duomenys, kurie turi būti apdoroti nuo sekundės iki minutės gali būti apdorojami ar kaupiami tiek ūko serveriuose tiek ūko agregatoriuose. Šio tipo duomenys gali būti kaupiami nuo valandos iki savaitės laiko.
- Trečios kategorijos duomenys gali būti apdoroti per minutes, dienas ar savaites, šio tipo duomenys dažniausiai kaupiami debesyse ilgą laiką. Tai gali būti duomenys skirti analitikai.



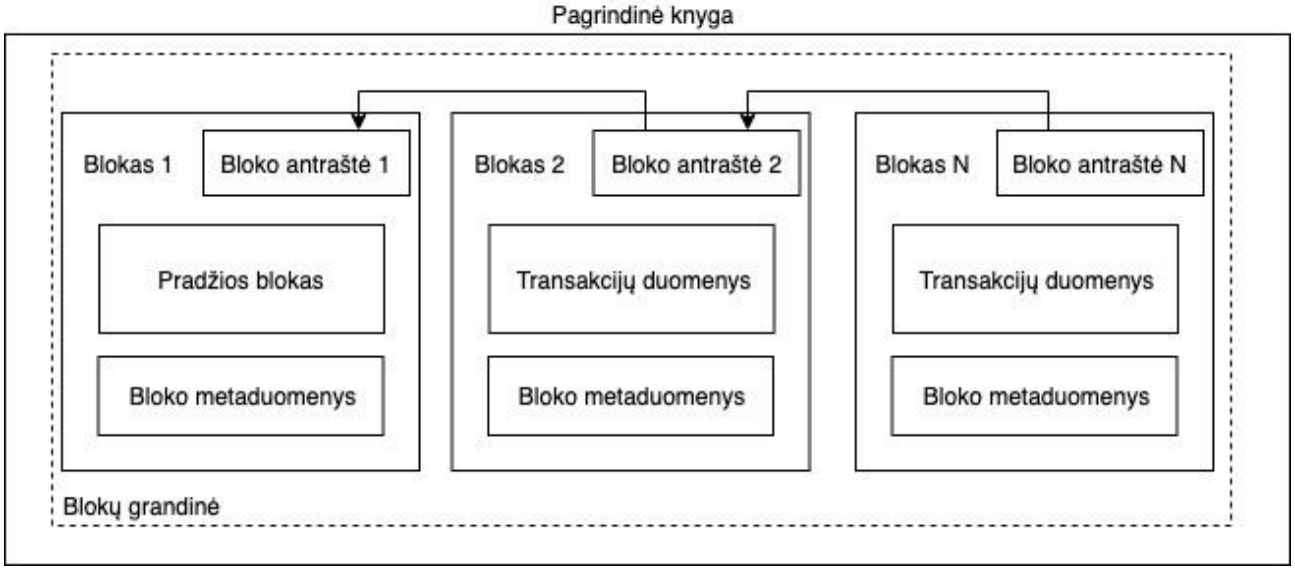
1.2 pav. Duomenų apdorojimo spartos reikalavimų diagrama

Kadangi galiniuose prietaisuose esanti aparatūrinė įranga turi ribotus išteklius, autentifikacijai netinkamos naudoti esamos, klasikinės autentikavimo sistemos. Todėl privalu naudoti paskirstyto tipo autentifikacijos sprendimus, kurie užtikrintų sistemos plečiamumą, augant galinių įrenginių skaičiui sistemoje. Šiam uždaviniui atlikti galima naudoti blokų grandinių technologijomis paremtą

sistemą, kuri kauptų įrašus apie sistemoje naudojamus galinius įrenginius ir leistų atlikti saugią ir mažai resursų reikalaujančią autentifikavimą. Galiniai IoT įrenginiai naudoja gerai žinomus „Bluetooth“, „WiFi“, „NFC“ ar mobiliojo ryšio protokolų standartus duomenų perdavimui. Taip pat yra sukurti lengvasvoriai ir specializuoti protokolai, pritaikyti specialiai IoT įrenginiams.

Šiuo metu blokų grandinių technologijos nėra plačiai taikomos autentifikacijai užtikrinti, tačiau yra sukurtos sistemos ir karkasai šiam uždaviniui atlikti. Ūko kompiuterijos modelyje blokų grandinių technologijomis paremta autorizacijos sistema galėtų veikti keliuose ūko kompiuterijos sluoksniuose. Dėl ribotų galinių įrenginių išteklių negalime naudoti blokų grandinėmis paremtos sistemos, kurios veikimui užtikrinti blokų grandinės pagrindinė knyga (angl. ledger) įrašams kaupti būtų saugojama galiniuose įrenginiuose. Tačiau autentifikavimą galima atlikti ūko sluoksnyje, kuriame veiktų pasiskirstyta blokų grandinių sistema, kurioje yra saugojami įrašai apie galinius įrenginius ir realizuojamas autentifikacijos mechanizmas. Taip pat autentifikacijos sistemą galima realizuoti tiek ūko, tiek debesų kompiuterijos sluoksniuose, naudojant hierarchinę sistemą, kurioje bus kaupiami įrašai ir apie ūko serverius, ir agregatorius ir galinius įrenginius, registruotus ūko sluoksnyje.

Blokų grandinių technologija pirmą kartą panaudota 2009 metais, kuriant bitkoino kriptovaliutą [6], nuo to laiko išaugo domėjimasis šia technologija. Ši technologija yra naudojama kaip virtuali buhalterijos arba pagrindinė knyga, kurioje saugomi įrašai apie vartotojų balansus, tai pirmoji sistema išsprendusi dvigubo išleidimo problemą, nenaudojant trečiosios šalies įsikišimo ar centralizuotos duomenų bazės. Bendruoju atveju pagrindinėje knygoje saugojami transakcijų įrašai kaupiami duomenų vienetuose vadinamais blokais, kiekvienas blokas išskyrus pradinį bloką siejamas su ankstesniu bloku, naudojant ankstesnio bloko kriptografinės santraukos reikšmę. Todėl naudojamas terminas blokų grandinės, dėl duomenų blokų jungimo ir duomenų struktūros, blokų grandinių duomenų struktūra pateikiama 1.3 pav. Decentralizuotumas ir duomenų nekintamumas yra vienos iš pagrindinių blokų grandinių technologijos ypatybių, šios technologijos pritaikymas autentifikavimui teikia privalumų, pavyzdžiui decentralizuotą įrašų kaupimą ir informacijos nekintamumo ir audito užtikrinimą. Decentralizuotas įrašų kaupimas gali užtikrinti duomenų pasiekiamumą ir integralumą. Taip pat pagrindinės knygos kopijų ir įrašų kaupimas keliuose įrenginiuose gali padėti apsisaugoti nuo suklastotų (angl. rogue) galinių įrenginių ir ūko sluoksnio serverių ūko kompiuterijos architektūroje.



1.3 pav. Pagrindinės knygos duomenų struktūra

Blokų grandinių technologijos dažniausiai yra naudojamos kurti kriptovaliutoms, tačiau prieš keletą metų ši technologija pradėta naudoti debesijos platformoms ir aplikacijoms realizuoti. Tai išpopuliarino *Ethereum* kompanija, be decentralizuotų aplikacijų platformos taip pat siūlanti savo kriptovaliutą. Ši kompanija taip pat sukūrė *Solidity* programavimo kalbą, kuri yra lengvai pritaikoma kurti projektus, kurie naudoja blokų grandinių technologijas. *Hyperledger* projektų grupė, kurios kūrimą pradėjo „Linux Foundation“, taip pat yra sparčiai populiarėjančių atvirojo kodo blokų grandinių karkasų ir įrankių rinkinių kūrėjai [7]. Blokų grandinės ir jų platformos yra skirstomos į dvi pagrindines kategorijas – privačias ir viešas blokų grandines.

Viešos blokų grandinės yra skirtos viešam naudojimui, tokio tipo blokų grandinės dažniausiai yra naudojamos kriptovaliutų platformose. Kadangi blokų grandinė yra vieša, nėra asmenų ar organizacijų, kurios valdytų ar administruotų blokų grandinę. Tai reiškia, kad bet kas gali skaityti, rašyti ir audituoti blokų grandinę, ši blokų grandinė yra visiškai atvira, tai leidžia peržiūrėti visus duomenis, bet kuriuo blokų grandinės gyvavimo metu. Kadangi viešos blokų grandinės nėra administruojamos vienos organizacijos ar asmens, sprendimai yra priimami naudojant darbo įrodymo (angl. PoW – Proof of Work) ar intereso įrodymo (angl. PoS – Proof of Stake) konsensuso mechanizmus [8].

Privačiosios blokų grandinės yra skirtos naudoti apibrėžtos, privačios organizacijos viduje arba gali būti dalinai prieinamos iš organizacijos išorės. Šio tipo blokų grandinės gali būti pritaikomos platesniam produktų spektrui, kadangi duomenys gali būti skaitomi ir rašomi tik gavus leidimą šioms operacijoms atlikti. Privačiosios blokų grandinės išplečia viešųjų blokų grandinių funkcionalumą, kadangi organizacijos gali administruoti blokų grandinėje vykstančias transakcijas ir prieigos teises skaityti ar vykdyti operas blokų grandinių įrašuose. Šis platformos tipas naudingas kuriant blokų grandinių aplikacijas skirtas verslo produktams ar aplikacijoms kurti, kadangi suteikiamos blokų grandinės administravimo bei konfigūravimo galimybės. Atvirkščiai nuo viešųjų blokų grandinių sistemų, priklausomai nuo platformos įgyvendinimo, privačiose blokų grandinėse duomenys blokų grandinėje gali būti keičiami.

Blokų grandinių sistemos turi įgyvendintus sprendimus apsisaugoti nuo tokių klasikinių saugumo problemų, kaip identiteto valdymas, informacijos neišsiginamumo ir duomenų bei sistemos audito atžvilgiu. Tačiau blokų grandinėmis grįstos sistemos taip pat turi pažeidžiamumų, pavyzdžiui 51% ataka, kuri išnaudoja darbo įrodymo konsensuso mechanizmus [9] [10]. Šio tipo ataka veiksminga tuo atveju, jeigu ūko sluoksnyje esantys serveriai nustotų veikti, ir tinkle atsirastų nauji, žalingi serveriai, turintys suklastotas pagrindinės knygos kopijas, tokiu atveju, didesnė dalis serverių turinčių savo blokų grandines galės perimti autentifikacijos proceso vykdymą. Tokiu būdu IoT įrenginių siunčiama informacija galėtų būti perimta. Blokų grandinės funkcionalumas yra pagrįstas viešojo rakto infrastruktūra, prietaisas turės pasirašyti žinutę savo privačiuoju raktu autentifikavimo metu. Tai gali būti pažeidžiamumo vieta, tokiu atveju, jeigu įrenginys yra pavogtas ar piktavalis gauna prieigą prie privačiojo rakto. Turint prieigą prie privačiojo rakto, galima apsimesti IoT įrenginiu ar serveriu ir skleisti suklastotą informaciją blokų grandinėje. Šiai problemai spręsti galime pasitelkti fiziškai neklonuojamų funkcijų metodus, šios funkcijos gali panaudoti fizinius įrenginio parametrus, kaip dalį informacijos autentifikavimo metu arba pakeisti privačiojo ir viešojo rakto poras. Kadangi privačiosios blokų grandinių sistemos yra diegiamos privačios infrastruktūros tinkle, tinklo saugos ir prieigos kontrolės priemonės turi būti užtikrinamos tinklo ir sistemų administratorių.

Bitcoin Core – pirmoji sistema, pritaikyta kriptovaliutai išgauti ir taip pat pirmasis šiuolaikinis blokų grandinių technologijos pritaikymas kompiuterijoje. Kadangi *Bitcoin Core* platinama atvirojo kodo licencija, dauguma blokų grandinių sistemų remiasi *Bitcoin Core* programiniu kodu, funkcijomis ir bibliotekomis [11]. Ši sistema dažniausiai yra naudojama privačių blokų grandinių kūrimui ar naujam funkcionalumui testuoti.

Solidity – programavimo kalba, sukurta ir naudojama išmaniųjų kontraktų kūrimui ir programavimui *Ethereum* blokų grandinių platformoje. Išmanusis kontraktas – programinio kodo funkcijų bei duomenų ir jų būsenos rinkinys, kuris yra patalpintas specifiniame adrese *Ethereum* blokų grandinėje. Kiekvieno išmaniame kontrakte gali būti apibrėžti metodai, įvykiai ar būsenos kintamieji, tai leidžia naudoti išmaniuosius kontraktus transakcijų valdymui tarp blokų, blokų grandinės tinkle. *Ethereum* blokų grandinė yra viešojo tipo, todėl transakcijos ir duomenys yra kaupiami viešo naudojimo blokų grandinėje, didelės organizacijos gali išaugti viešojo tipo blokų grandinės teikiamų galimybių lūkesčius. *Ethereum* išmanieji kontraktai ir *Solidity* programavimo kalba yra dažniausiai naudojami kurti bendro naudojimo decentralizuotas aplikacijas [12] [13].

Hyperledger Fabric – pamatinis karkasas, skirtas kurti blokų grandinėmis paremtus produktus, aplikacijas ir sprendimus. Ši platforma yra skirta kurti privačiojo tipo leidimais grįstus blokų grandinių tinklus, kurie suteikia konfidencialumo, lankstumo, atsparumo ir plečiamumo galimybes blokų grandinės tinklo kūrėjui. Taip pat ši sistema pasižymi moduline architektūra, tačiau išlaiko bendrąsias blokų grandinių savybes, kaip išmaniųjų kontraktų ir apskaitos knygų funkcionalumą [14]. Lyginant su kitomis sistemomis, *Fabric* teikia kūrėjui daugiau laisvės konfigūruoti ir plėtoti sistemą, pavyzdžiui duomenų šifravimas, prieigos kontrolė ir matomumas yra apsprendžiamas sistemos architekto, numatytieji nustatymai yra pilnai konfigūruojami sistemos kūrimo gyvavimo ciklo metu.

Hyperledger Composer – įrankis, leidžiantis greitai kurti blokų grandinių sistemų ir aplikacijų prototipus. *Composer* veikia *Fabric* pagrindu, tačiau suteikia aplikacijų kūrėjams įrankius ir galimybę kurti greitas koncepcines aplikacijas, paremtas blokų grandinėmis [15]. Sukurti biznio tinklų rinkiniai gali būti perkeliami į *Fabric* sistemas tolimesniam plėtojimui bei integracijai vystyti.

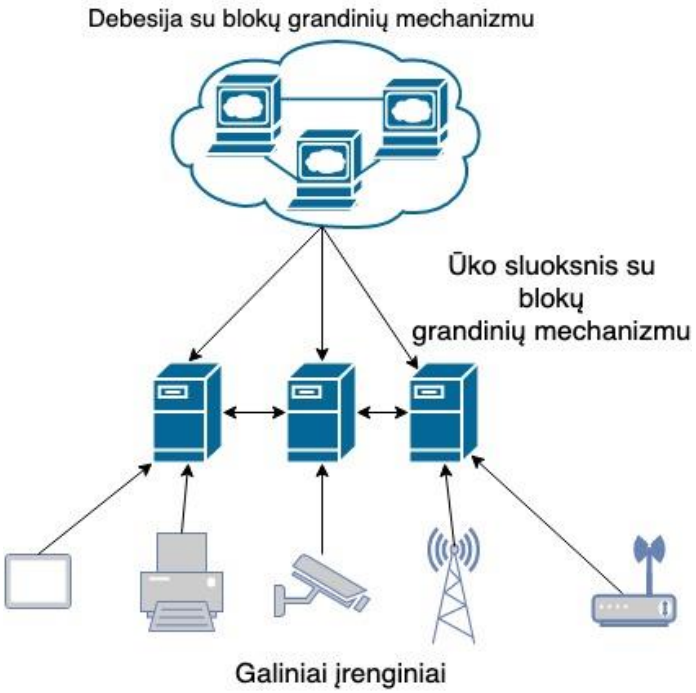
Multichain – platforma, naudojama kurti privačius blokų grandinių tinklus. *Multichain* išplečia ir papildo *Bitcoin Core* funkcionalumą išplėstiniam naudojimui kurti blokų grandinėmis paremtas aplikacijas. Pagrindinis *Multichain* privalumas – aukštų programavimo kalbų palaikymas, kiekviena kalba naudoja atskirą API aplikacijų kūrimui. Tačiau *Multichain* nenaudoja išmaniųjų kontraktų ir kriptovaliutų konceptų [16] [17].

1.1 lentelė. Blokų grandinių sistemų palyginimas

	Bitcoin Core	***Solidity***	***Hyperledger Fabric***	***Hyperledger Composer***	***Multichain***
Paremta kriptovaliutos modeliu	Taip	Taip	Ne	Ne	Ne
Išmanieji kontraktai	Ne	Taip	Taip	Taip	Ne
Konsensuso mechanizmai	darbo įrodymo	darbo įrodymo	Keletas	Keletas	Keletas
Taikymo sritis	Kriptovaliutos	Kripto valiutos, decentralizuotos aplikacijos	Stambios veiklos blokų	Blokų grandinių aplikacijų	Privačiųjų/viešųjų blokų grandinių ir aplikacijų kūrimas

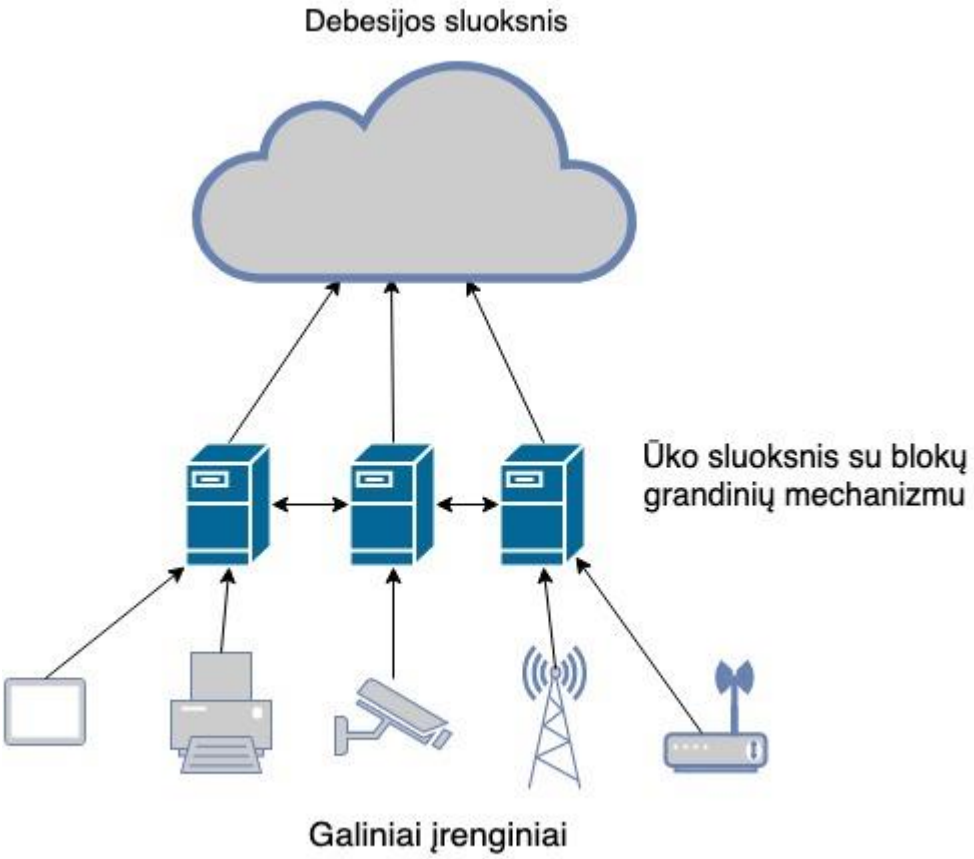
			grandinių kūrimas	prototipų kūrimas	
Moduliarumas	Ne	Ne	Taip	Taip	Taip
Plečiamumas	Ne	Ne	Taip	Taip	Taip
API	Ne	Taip	Taip	Taip	Taip

Remiantis 1.1 lentelėje pateiktais palyginimais galima teigti, kad viešojo tipo blokų grandinių naudojimas kurti galinių įrenginių autentifikavimo metodą teikia mažiau funkcionalumo, kadangi autentifikavimo mechanizmas turi integruotis su kitomis sistemomis ūko kompiuterijos sluoksniuose, taip pat galinių įrenginių duomenys gali kisti, o duomenų keitimas viešose blokų grandinėse nėra galimas. Taip pat privačiose blokų grandinėse gali veikti tik sistemoje registruoti ir sertifikatus turintys įrenginiai, tai padeda apsisaugoti nuo 51% atakų. Įprasti kriptovaliutomis grįsti procesai ir duomenų modeliai gali būti pritaikomi autentifikavimo informacijai saugoti, tačiau didėja privačiųjų raktų nutekėjimo rizika. Taip pat viešojo tipo blokų grandinių sistemos nesuteikia plačių API galimybių, kurių prireiks vykdyti procesams autentifikavimo metu.



1.4 pav. Dvigubo blokų grandinių panaudojimo architektūros modelis

Architektūriniame modelyje, pavaizduotame 1.4 pav., matoma autentifikacijos architektūra, kurioje blokų grandinių mechanizmas yra pritaikomas tiek debesijos, tiek ūko sluoksnyje. Debesų ir ūko sluoksnis sąveikauja tarpusavyje ir sukuria hierarchinę autentifikavimo sistemą. Tai gali būti patogu administruojant didelį kiekį ūko kompiuterijos organizacinių vienetų zonų, tai leistų pritaikyti tas pačias taisykles naujiems ūko regionams ar administruoti keletą zonų vienu metu. Taip pat galima išvengti duomenų praradimo, kadangi blokų grandinių technologija remiasi pagrindinės knygos įrašais, kurie yra paskirstyti tarp daugelio prietaisų. Įvykus trikdžiams ūko sluoksnyje, galiniai įrenginiai galėtų autentifikuotis tiesiai į debesų sluoksnį, taip galima padidinti tolerancijį trikdžiams.



1.5 pav. Blokų grandinių panaudojimo ūko sluoksnyje modelis

Architektūrinis modelis, naudojantis blokų grandinių technologiją tik ūko sluoksnyje, pavaizduotas 1.5 pav. Laikantis šio principo, paskirstytoji blokų grandinės pagrindinė knyga ir blokų grandinių transakcijos veiktų, tik tarp ūko sluoksnyje esančių prietaisų, todėl mažėja duomenų patikimumas. Įvykus trikdžiui ūko sluoksnyje, nauji ir esami prietaisai negalėtų autentifikuotis sistemoje, taip sistema nustotų veikti ir duomenys gali užpildyti duomenų kaupykles galiniuose ir ūko įrenginiuose. Tačiau šio tipo architektūra gali būti lengvai pritaikoma naudojimui su debesijos paslauga, kuri turi realizuotą blokų grandinių panaudos mechanizmą. Tačiau tiek hierachinės blokų grandinių tiek vieno sluoksnio blokų grandinės sistemos architektūros yra pažeidžiamos sisteminių trikdžių.

Kadangi ūko architektūroje veikia įvairiausio tipo daiktų interneto įrenginiai, juos valdyti ir administruoti reikalingi specializuoti, lengvasvoriai protokolai, kadangi įprasti interneto protokolai ir sprendimai dažniausiai pasižymi didelėmis resursų sąnaudomis. Dėl IoT įrenginių gausos pradėti kurti nauji protokolai, kurie sprendžia didelio resursų poreikio problemą, šie protokolai sprendžia tiek infrastruktūrinius, aptikimo, duomenų perdavimo ir apdorojimo bei semantinius klausimus.

Dėl didelės prietaisų įvairovės galinių įrenginių sluoksnyje, ūko sluoksnis turi gebėti komunikuoti su įvairaus tipo IoT bei mobiliaisiais įrenginiais. Todėl svarbu nustatyti kokiais protokolais remsis numatoma sistema ir metodas. Priklausomai nuo sistemos reikalavimų, galima naudoti įvairius protokolus, kurie užtikrintų reikiamą funkcionalumą. Esant nejudriam IoT įrenginių tinklui galima naudoti standartinius protokolus, priartinus ūko sluoksnį kuo arčiau galinių įrenginių, tokiu atveju duomenys būtų siunčiami ūko įrenginiui naudojant *Bluetooth*, Wi-Fi ar kitus tradicinius duomenų perdavimo kanalus. Tačiau uždavinys yra autentifikuoti galinius IoT prietaisus ūko sluoksnyje, todėl reikia surasti kuo labiau universalesnį metodą ir protokolą šiam uždaviniui atlikti.

Galiniai įrenginiai ir ūko serveriai turės keistis duomenimis ir informacija, kadangi galiniai įrenginiai gali turėti įvairios spartos ir galimybių techninę įrangą, sprendimui reikia naudoti specializuotus duomenų perdavimo protokolus. Įprasti duomenų perdavimo protokolai, kaip TCP, UDP ir HTTP nėra palankūs naudoti dėl ribotų išteklių IoT įrenginiuose. Daiktų interneto programose ir aplinkose naudojami lengvasvoriai transporto ir duomenų keitimosi protokolai, gali būti pritaikomi ir projektuojamoje sistemoje. Šie protokolai buvo projektuojami ir kuriami specialiai IoT ar mažus resursus turintiems įrenginiams valdyti ar keistis informacija tarpusavyje. Šie protokolai, kitaip vadinami žinučių siuntimo šeimos (angl. MQ – Message Queuing) protokolais, leidžia IoT prietaisams ar kompiuteriams sparčiai keistis informacija tarpusavyje. Tačiau didėjant galinių įrenginių įvairovei, tarp jų atsiranda ir prietaisų su labai mažomis baterijos talpomis, bei duomenų apdorojimo apribojimais, todėl reikia išsirinkti geriausiai tinkamą, protokolą.

MQTT (Message Queue Telemetry Transport) – vienas seniausių ir labiausiai paplitusių MQ protokolų [18], sukurtas užtikrinti paprastumui ir lengvumui duomenų perdavime. Tačiau šis protokolas neteikia saugumo išplėtimų [19] ir turi ribotą API rinkinį, protokolo naudojimo paprastumas yra vienas iš didžiausių protokolo pranašumų. Skirtingai nuo kitų protokolų šis protokolas neturi žinučių eilių mechanizmo ir remiasi prenumeravimo-skelbimo modeliu, nors protokolo pavadinimas leidžia manyti kitaip. MQTT protokolas projektuotas ir kurtas naudoti įterptinėse sistemose ar įrenginiuose su dideliais resursų apribojimais.

STOMP (Simple/Streaming Text Oriented Messaging Protocol) – protokolas paremtas tekstinių duomenų perdavimu, savo veikimu panašus į HTTP protokolą. Šis protokolas orientuotas į lengvumą ir suderinamumą tarp skirtingų prietaisų sistemoje, siekiant sukurti paprastą ir mažai resursų naudojantį protokolą. Protokolas neturi išplėstinių nustatymų žinučių siuntimui, žinučių eilių valdymą ir temas valdo žinučių brokeriai, siunčiant duomenis STOMP protokolu reikia nurodyti tik brokerio adresą ir norimus perduoti duomenis.

CoAP (Constrained Application Protocol) – specializuotas IoT protokolas, pritaikytas naudojimui įrenginiams turintiems ribotus išteklius, bei veikiantiems mažo pralaidumo tinkluose. Protokolas puikiai tinka perduoti duomenims tarp prietaisų, protokolas gali veikti bet kokiame prietaise, kuris palaiko UDP protokolą. Šis protokolas neturi charakteristikų būdingų MQ protokolams, protokolas projektuotas naudoti paprastam duomenų perdavimui tarp dviejų prietaisų ir didelis dėmesys skirtas protokolo lengvumui. CoAP duomenys gali būti perduodami naudojant net SMS [20] žinutes ar kitus duomenų perdavimo apvalkalus. Kadangi šis protokolas neturi tiesioginės integracijos su žinučių brokeriais [18], duomenų apdorojimas turi būti atliekamas kitų specializuotų aplikacijų, pavyzdžiui įgaliotojų (angl. proxy) serverių ar REST aplikacijų.

AMQP (Advanced Message Queuing Protocol) – protokolas projektuotas, norint pakeisti jau esamus MQ protokolus, dvi stipriausios charakteristikos, patikimumas ir suderinamumas. Taip pat šis protokolas siūlo platų funkcionalumą, susijusį su žinučių siuntimo valdymu, kaip pristatymo garantijos, žinučių grupavimas, lankstumas, transakcijos ir saugumas. Tačiau protokolas nepasižymi lengvumu – reikalauja daugiau resursų [18] nei kiti MQ protokolai. AMQP protokolas projektuotas užtikrinti funkcinį suderinamumą naudojant skirtingas sistemas.

1.2 lentelė. Duomenų perdavimo protokolų palyginimas

	Transporto protokolas	**Siuntimo principas**	**Bendravimas**	**Saugumas**	**Klaidų taškai**

MQTT	TCP/IP	Prenumeravimo-skelbimo	daiktas-serveris, daiktas-daiktas	TLS	Žinučių brokeris
STOMP	TCP/IP	Prenumeravimo-skelbimo	daiktas-serveris	TLS	Pagal sistemos projektavimą
CoAP	UDP/IP	Užklausa ir atsakymas	daiktas-serveris	DTLS	Decentralizuotumas
AMQP	TCP/IP	Prenumeravimo-skelbimo	daiktas-daiktas daiktas-serveris serveris-serveris	TLS	Pagal sistemos projektavimą

Atlikus duomenų perdavimo protokolų analizę galima teigti, kad skirtingi protokolai tinkami naudoti skirtingose taikymo srityse ir įrenginiuose, tačiau projektuojant autentifikavimo sistemą privalu rinktis protokolą, kuris teikia daugiau funkcijų ir galimybių. Įrenginių resursų naudojimo apribojimai dažniausiai taikomi įrenginiams, skirtiems perduoti duomenis, tačiau naudojant CoAP ar STOMP protokolus, duomenys gali būti perduodami į ūko sluoksnį. Tačiau autentifikavimo sistema gali reikalauti sudėtingesnių žinučių siuntimo ir skirstymo funkcijų, kurių STOMP ir CoAP pasiūlyti negali. Remiantis atlikta analize, AMQP ir MQTT duomenų perdavimo protokolai yra plačiausiai naudojami IoT įrenginių tarpe [18]. AMQP suteikia galimybę išplėsti protokolo perdavimo funkcijas ir duomenų perdavimo saugumą, tačiau MQTT pasižymi didesniu lengvasvoriškumu.

Kadangi ūko sluoksnyje esantys serveriai turi bendrauti su įvairaus pobūdžio galiniais įrenginiais, kurių pagrindinis duomenų perdavimo būdas yra žinučių siuntimas MQ protokolais, ūko sluoksnyje turi veikti žinučių brokeriai, kurie gali priimti duomenis siunčiamus MQ protokolais iš galinių įrenginių. Žinučių brokeriai ne tik priima duomenis iš įrenginių, bet ir atlieka gautų duomenų persiuntimą gavėjams. Gavėjai gali būti galiniai įrenginiai, prietaisai ar aplikacijos, kurios naudoja sąsają su žinučių brokeriu. Taigi žinučių brokeriai dažniausiai atlieka tarpininkavimo funkciją tarp informacijos siuntėjų ir informacijos gavėjų. Priklausomai nuo naudojamo MQ protokolo ir žinučių brokerio sistemos, žinutės gali būti siunčiamos į eiles ir gaunamos vieno gavėjo arba gali būti naudojami sudėtingesni žinučių pristatymo protokolai. Šie skirstymo protokolai galimi naudoti ne visose žinučių brokerių sistemose ir ne visi protokolai palaiko šį funkcionalumą. Populiariausias ir beveik visuose brokeriuose veikiantis žinučių siuntimo ir gavimo modelis yra prenumeravimo-skelbimo (angl. publish-subscribe) metodas, tačiau sudėtingesni mechanizmai, kaip vėduoklės, temomis ir antraštėmis grįsti skirstymo metodai veikia naudojant specifinius protokolus ir brokerių sistemas.

Žinučių brokeriai yra atsakingi už žinučių pristatymą gavėjui, turi būti užtikrinamas žinučių kaupimas eilėje net jeigu nėra aktyvių gavėjų galinčių žinutes priimti, taip užtikrinama duomenų pristatymo garantija, kad išsiųstos žinutės bus pristatytos gavėjui bet kokiu atveju. Skirtingos sistemos siūlo skirtingus duomenų pristatymo garantijos modelius, dauguma žinučių brokerių siūlo bent kartą, daugiausiai kartą ir būtent kartą pristatymo garantijos modelius. Didžiausių garantijų pristatymo modelių naudojimas dažniausiai neigiamai veikia žinučių brokerio greitaveiką ir spartą, todėl reikia atsakingai pasirinkti norimą duomenų pristatymo garantijos modelį. Žinučių brokeriai yra neatsiejama paskirstytųjų sistemų architektūros dalis.

Apache Kafka – paskirstytoji srautinių duomenų dorojimo sistema, kurią sudaro serverių visuma vadinama klasteriu (angl. cluster). Dėl naudojamo prenumeravimo-skelbimo modelio sistema yra tapatinama su žinučių brokeriais, tačiau *Kafka* naudojama kaip alternatyvi sistema pakeisti žinučių brokerius. *Apache Kafka* sistema dažniausiai naudojama kuomet reikia dirbti ir apdoroti ypatingai

didelius duomenų srautus. Taip pat sistema pasižymi horizontaliojo plečiamumo galimybėmis, todėl bet kada prireikus sistema gali būti išplėsta suprastėjus sistemos funkcionalumui ar našumui. *Kafka* išsiskiria nuo daugumos žinučių brokerių dėl galimybės kaupti ir apdoroti gautas žinutes, kuomet dauguma žinučių brokerių tik perduoda ar laikinai kaupia žinutes iki kol jos pasiekia gavėjus.

RabbitMQ – žinučių brokeris ir tarpininkavimo platforma tarp prietaisų ar programų. Vienas iš populiariausių ir plačiausiai naudojamų žinučių brokerių, sukurtas naudoti kartu su AMQP protokolu, tačiau šiuo metų *RabbitMQ* palaiko daugumą populiariausių MQ protokolų, tai lėmė šio žinučių brokerio platų naudojimą ir populiarumą. Svarbiausias šio žinučių brokerio aspektas – platus žinučių valdymo mechanizmo pasirinkimas. Dauguma žinučių brokerių siūlo prenumeravimo-skelbimo duomenų apsikeitimo modelį, skirtingai nuo daugelio sistemų *RabbitMQ* siūlo keturis žinučių perdavimo mechanizmus, tiesioginį, vėduoklės, temomis ir antraštėmis grįstus mechanizmus. Be šių savybių sistema taip pat palaiko plečiamumo, klaidų toleravimo, apkrovos skirstymo ir replikavimo savybes.

Apache ActiveMQ – žinučių brokeris paremtas *Java* technologijomis. *ActiveMQ* palaiko įvairių programavimo kalbų klientus, tai pritraukia daugiau skirtingų programavimo kalbų vartotojų prisidėti prie *ActiveMQ* sistemos vystymo. Šis žinučių brokeris neturi apibrėžto žinučių dydžio apribojimo, todėl šis žinučių brokeris gali būti taikomas programoms ar sistemos, kuriose reikalingas didelių duomenų perdavimas žinučių forma. JMS standartų laikymasis leidžia šį žinučių brokerį naudoti su dauguma programavimo kalbų, tačiau panaudojimas galiniuose įrenginiuose nėra pranašesnis nei ankščiau minėtuose žinučių brokeriuose.

1.3 lentelė. Žinučių brokerių palyginimas

	Apache Kafka	***RabbitMQ***	***Apache ActiveMQ***
Žinučių siuntimo/gavimo modelis	Prenumeravimo-skelbimo	Tiesioginis, vėduoklės, prenumeravimo-skelbimo, skirstymas antraštėmis	Prenumeravimo-skelbimo
Pristatymo garantijos modeliai	Bent kartą, daugiausiai kartą, būtent kartą	Bent kartą, daugiausiai kartą	Daugiausiai kartą, būtent kartą
Administravimo sąsaja	Primityvi naršyklės sąsaja arba trečiųjų šalių sistemos	Naršyklės sąsaja, pagrindinės funkcijos	Naršyklėje veikiantis komandų interpretatorius
Duomenų apdorojimo sparta	20 tūkst. – 2 mln. žinučių priėmimų per sekundę [21] [22]	5-16 tūkst. žinučių priėmimų per sekundę [22] [23]	5-22 tūkst. žinučių priėmimų per sekundę [22] [24]
Duomenų replikavimas	+	+	+
Klaidu toleravimas	Palaikomas naudojant *Zookeeper* programine įranga	+	+
Palaikomi protokolai	TCP	AMQP, MQTT, STOMP	AQMP, AUTO, MQTT, OpenWire, REST, RSS, Atom, STOMP, WSIF, XMPP

Žinučių brokerio pasirinkimas, projektuojant sistemas, priklauso nuo sprendžiamo uždavinio, perduodamų duomenų kiekio ir sistemoje naudojamų protokolų. Projektuojamoje autentifikavimo sistemoje, kuri veiks ūko kompiuterijoje, reikia užtikrinti žinučių pristatymą bent vieną kartą arba daugiau nei vieną kartą įrenginių autentifikavimo proceso metu. Kadangi galiniai įrenginiai siunčia

duomenis ne tik autentifikavimui atlikti, bet ir atlikti savo užduotį pagal paskirtį, žinučių kiekis gali išaugti priklausomai nuo numatytos taikymo srities. Dėl didelio galinių įrenginių įvairumo, žinučių brokeriai, palaikantys daugiau nei vieną protokolą turi didesnį pranašumą prieš sistemas palaikančias vieną protokolą. *Kafka* naudoja standartinį TCP protokolą, kuris ne visada gali būti pritaikomas naudojimui galiniuose įrenginiuose, ši sistema dažniausiai pasirenkama, kuomet reikia apdoroti didelius duomenų srautus ir juos kaupti. *RabbitMQ ir ActiveMQ* sistemos projektuotos ir pritaikytos darbui su galiniais įrenginiais, šios sistemos palaiko keletą skirtingų MQ protokolų, kurie naudojami IoT prietaisuose duomenims perduoti.

Kadangi naudoti įprastą galinių įrenginių autentifikaciją naudojant vartotojo vardą ir statinį slaptažodį yra nesaugu, reikalinga alternatyvi reikšmė pakeisti slaptažodį. Slaptažodžio alternatyva gali būti VRI (angl. PKI – Private Key Infrastructure) ar PUF (angl. Physical Unclonable Function) [25] [26], tačiau, VRI naudojimas sukelia papildomas sąnaudas galiniuose įrenginiuose, kadangi kriptografinių raktų poros turi būti sugeneruojamos galinio įrenginio pusėje, taip pat privačiojo rakto fizinis saugumas įrenginyje gali būti sukompromituotas. PUF funkcijas gali palaikyti ne visi galiniai įrenginiai, todėl galima alternatyva PUF funkcijoms yra identifikatoriaus generavimas galiniame įrenginyje pasitelkiant įrenginio parametrus, kuriuos suklastoti yra sunku. Šis įrenginio parametrų rinkinys vadinamas įrenginio identifikatoriumi. Įrenginio identifikatorius turi būti tekstinė simbolių eilutė, tekstinių simbolių naudojimas leis užtikrinti suderinamumą autentifikavimo sistemoje. Identifikatorius autentifikavimo procese veiks, kaip galinio įrenginio slaptažodis, tačiau identifikatorius nėra statiškai nurodomas konfigūracijoje ar fiziškai saugojamas įrenginyje. Duomenų perdavimo operacijose – identifikatorius yra generuojamas kiekviena kartą autentifikuojantis sistemoje. Įrenginio identifikatorių sudaryti turi mažiausiai kintanti galinio įrenginio aparatūrinės įrangos informacija. Kadangi įrenginį identifikuojanti informacija gali būti perduodama nesaugiais, viešais kanalais internete, ši informacija turi būti apsaugota duomenų perdavimo metu. Todėl identifikatoriaus sudedamosios dalys turi būti užšifruojamos saugia bent SHA-256 stiprumo vienkrypte funkcija.

MAC adresas – šis parametras yra unikalus 12 simbolių kodas, priskiriamas kiekvienai interneto plokštei, net virtualios interneto kortos turi savo MAC adresą, todėl kiekvienas galinis įrenginys turės retai kintančią MAC adreso reikšmę. Naudojant tik MAC adresą santraukos generavimui nėra pakankama, tačiau papildomų sudedamųjų naudojimas apsunkina identifikatoriaus klastojimą.

RAM informacija – nors šis parametras nėra unikalus kiekvienam įrenginiui, tačiau kiekvienas prietaisas turi RAM atmintį. Identifikatoriaus sudedamoji paremta RAM informacija, nėra tik RAM atminties kiekis, taip pat naudojama ir RAM atminties gamintojo informacija. Dažnu atveju RAM informacija galiniuose įrenginiuose neturėtų kisti, galinio įrenginio gyvavimo ciklo metu.

CPU informacija – kiekvienas įrenginio architektūroje yra CPU modulis, nors CPU informacija nėra unikali kiekvienam įrenginiui, tačiau CPU informacija turėtų nekisti galinio įrenginio veikimo metu. Kaip ir RAM informacijos sudedamoji identifikatoriuje, CPU informacija susideda iš CPU gamintojo informacijos, ir aparatūrinių CPU parametrų.

1.4. Darbo tikslas, uždaviniai, planas ir siekiami privalumai

Darbo tikslas – sukurti galinių įrenginių autentifikavimo metodą ūko kompiuterijoje taikant blokų grandinių technologiją. Darbo metu bus tiriamos blokų grandinių technologijos ir jų pritaikymo galimybės autentifikuoti galinius įtaisus ūko kompiuterijos architektūroje.

Uždaviniai:

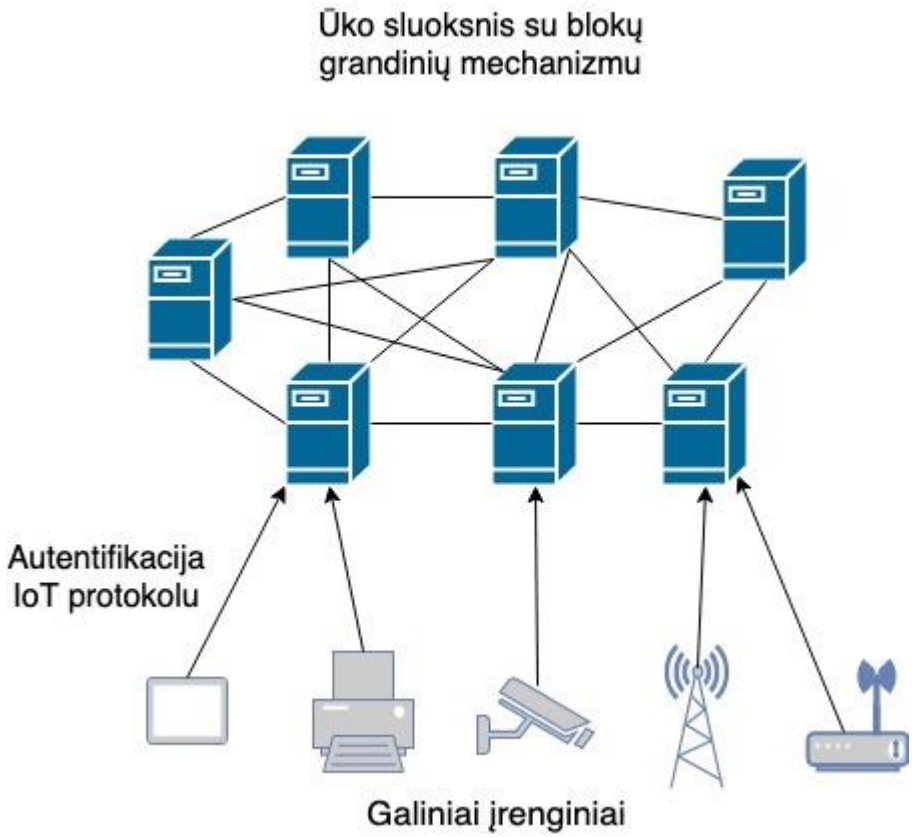
1. atlikti analizę ir ištirti esamus blokų grandinių technologijų taikymus autentifikacijai atlikti ūko kompiuterijoje;
2. suprojektuoti sistemą, paremtą blokų grandininių technologijomis ūko kompiuterijoje autentifikacijos uždaviniui spręsti;
3. realizuoti prototipą galinių įrenginių autentifikacijai, pritaikant blokų grandinių technologijas ūko kompiuterijoje;
4. eksperimentiškai ištirti ir įvertinti sukurtą metodą.

Siekiant sukurti autentifikacijos metodą, skirtą galinių įrenginių autentifikavimui ūko kompiuterijoje, pritaikant blokų grandinių technologijas, pirmiausia ištirta ūko kompiuterijos architektūra, protokolai, naudojami duomenų perdavimui ir blokų grandinių technologijomis paremtos sistemos ir esami sprendimai, atlikti galinių įrenginių autentifikaciją ūko kompiuterijos architektūroje. Remiantis atlikta analize, projektuojama galinių įrenginių autentifikavimo sistema ūko kompiuterijoje naudojanti blokų grandinių technologijas. Suprojektavus sistemą, realizuojamas autentifikavimo sistemos ir įrenginių sąveikos prototipas. Blokų grandinių technologijų panaudojimas ūko kompiuterijos architektūroje galinių įrenginių autentifikavimo uždaviniui spręsti gali pašalinti daug gerai žinomų saugumo ir architektūrinių spragų debesų kompiuterijos architektūroje ir autentifikavimo procese.

1.5. Siekiamo sprendimo apibrėžimas

Projektuojamoje sistemoje turi būti realizuotas galinių įrenginių autentifikacijos mechanizmas, naudojantis blokų grandinių technologijos sprendimus. Kadangi tyrimo subjektai yra ūko sluoksnis ir galiniai įrenginiai, sistema nebus projektuojama pritaikymui debesijos paslaugose, tačiau sistema bus pritaikoma naudojimui kartu su debesijos paslaugomis. Taip bus užtikrinta sistemos plečiamumo galimybė.

Ūko sluoksnyje veiks autentifikavimo mechanizmas, kuris remsis blokų grandinių technologijomis. Tai užtikrins autentifikuojamų galinių įrenginių identitetą ir autentifikavimo mechanizmą. Autentifikavimo mechanizmas turi palaikyti duomenų perdavimo protokolus, kuriuos naudoja galiniai įrenginiai. Autentifikavimo sistema neturi ženkliai padidinti galinių įrenginių išteklių sąnaudų, tokiu būdu nebus sumažinamas IoT įrenginių našumas naudojant sukurtą metodą.



1.6 pav. Numatomos autentifikavimo sistemos architektūrinė diagrama

Kadangi ūko serveriai turi būti geografiškai arti kritinių galinių įrenginių, gali būti naudojamos pirmumo taisyklės ar serverių apkrovos skirstytojai su geografinės vietos nustatymo mechanizmu galininių įrenginių aptarnavimui. Todėl blokų grandinių technologija teikia didelę naudą, kadangi visi ūko serveriai turės įrašus apie sistemoje veikiančius galinius įrenginius, tai sumažina kenkėjiškų galinių įrenginių autentifikacijos galimybę sistemoje.

1.6. Analizės išvados

Atlikus analizę nustatyta, kad esamas ūko kompiuterijos architektūrinis modelis neturi realizuotos specializuotos, našios decentralizuotos autentifikacijos sistemos galiniams IoT įrenginiams. Tačiau numatytoje architektūroje yra galimybė šį funkcionalumą realizuoti panaudojant blokų grandinių technologiją ir esamus ar pritaikant naujai sukurtus specializuotus IoT protokolus galiniams įrenginiams autentifikuoti.

Ūko kompiuterijos architektūra siekia optimizuoti esamą debesų kompiuterijos architektūrą, priartindama papildomus ūko serverius kuo arčiau galinių įrenginių. Ūko serverių geografinis priartinimas gali atlikti pradinius skaičiavimus ir duomenų agregavimą prieš perduodant informaciją į debesų kompiuterijos sistemas. Taip optimizuojamas tinklo pralaidumas ir mažinamas tinklo vėlinimas perduodant informaciją iš galinio įrenginio į ūko sluoksnio serverius.

Autentifikavimo metodui kurti bus naudojamos jau sukurtos sistemos ir įrankiai, tai leis užtikrinti suderinamumą su esamomis daiktų interneto informacinėmis sistemomis. Papildomas ūko sluoksnis gali būti integruojamas esamose daiktas-debesis sistemose.
Remiantis atlikta analize, toliau numatomi spręsti šie uždaviniai:

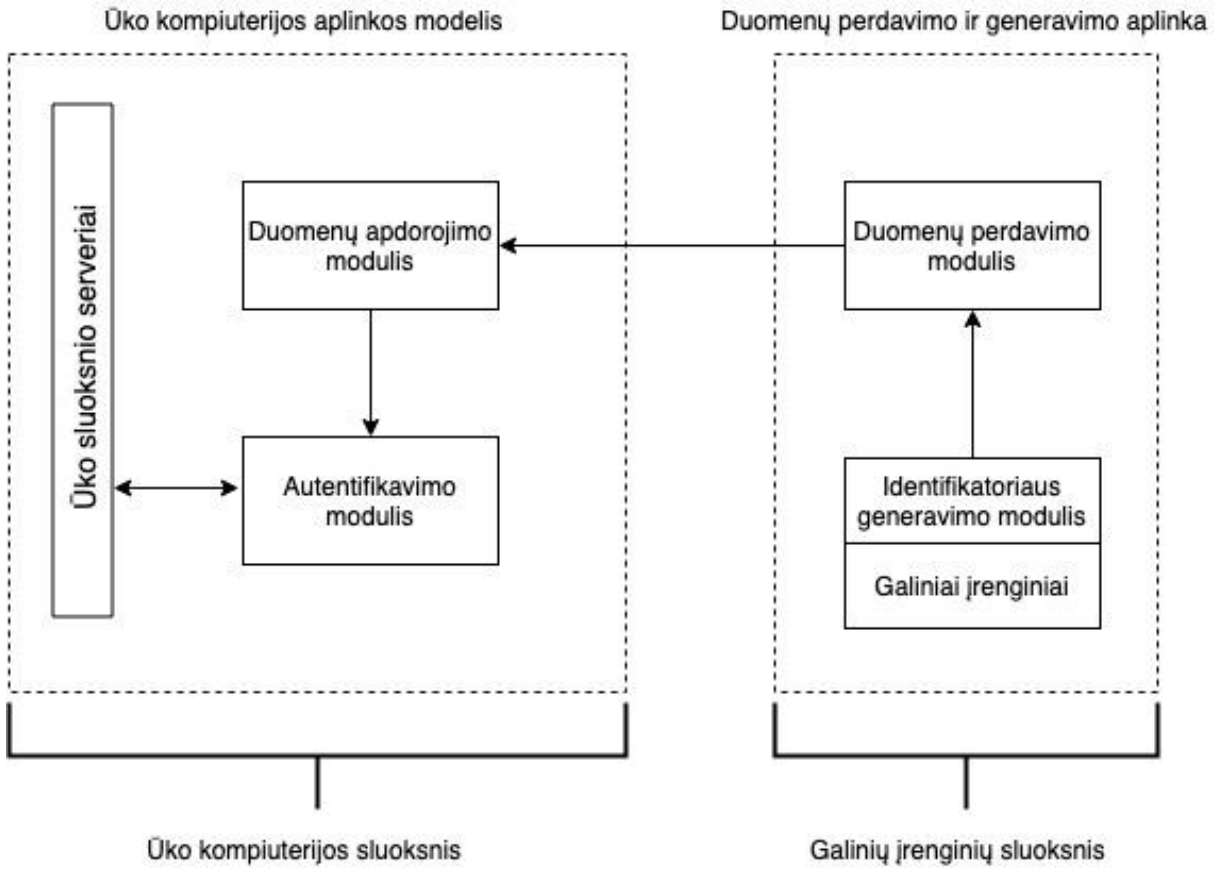
- suprojektuoti sistemą galinių įrenginių autentifikavimui ūko kompiuterijoje, pritaikant blokų grandinių technologijas;
- sukurti sistemos prototipą galinių įrenginių autentifikavimui, naudojant blokų grandinių technologijas ir IoT protokolus ūko kompiuterijos sluoksnyje;
- įvertinti sukurtą metodą ir palyginti gautus tyrimo rezultatus.

2. Galinių įrenginių autentifikavimo sistemos ūko kompiuterijoje metodo projektavimas

Šiame skyriuje aprašomas sistemos projektavimo ir modeliavimo etapas. Projektavimas susideda iš trijų dalių:

1. galinių įrenginių autentifikavimo sistemos ūko kompiuterijos sluoksnyje projektavimo;
2. galinių įrenginių identifikatoriaus generavimo ir duomenų perdavimo mechanizmų projektavimo;
3. blokų grandinėmis grįstos galinių įrenginių autentifikavimo sistemos ūko sluoksnyje metodo projektavimo;

Pirmosios dalies paskirtis yra projektuoti ūko kompiuterijos sluoksnį ir jame vykstančius procesus, taip apibrėžiant gaires, kurios bus naudojamos sistemos prototipui realizuoti. Modeliuojamame ūko kompiuterijos sluoksnyje, taip pat modeliuojamas ir autentifikavimo sistemos sprendimas. Antrojoje dalyje analizuojami duomenų perdavimo protokolai ir autentifikavimo sistemos procesai. Trečioje dalyje pateikiamas projektuojamos sistemos veikimo modelis tarp galinių įrenginių ir ūko kompiuterijos platformos.



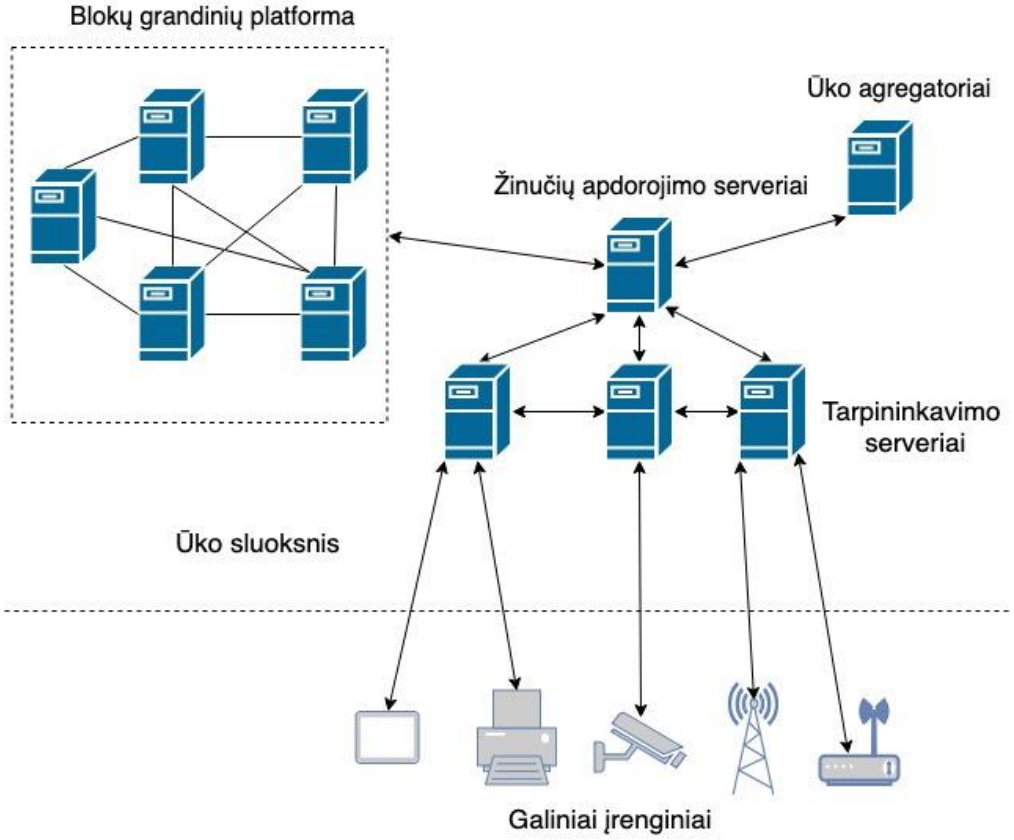
2.1 pav. Autentifikavimo sistemos modelio principinė schema

2.1. Ūko kompiuterijos ir autentifikavimo sistemos projektavimas

Projektuoti galinių įrenginių autentifikavimo sistemą ūko kompiuterijoje yra naudojamas smulkinamasis projektavimas, kadangi sistemos visumą sudaro daug sudedamųjų posistemių. Šiame skyriuje projektuojama autentifikavimo sistema ūko kompiuterijos sluoksnyje, išskiriamos šios posistemės:

- ūko kompiuterijos platforma – tarpusavyje susijusių serverių tinklas, veikiantis ir sudarantis bendrą ūko kompiuterijos sluoksnį;
- autentifikavimo sistema – paskirstytoji galinių įrenginių autentifikavimo sistema, veikianti ūko kompiuterijos sluoksnyje.

Ūko kompiuterijos platformą sudaro įprasti arba specializuoti serveriai ar kompiuteriai, kurie yra kuo labiau geografiškai priartinti prie galinių įrenginių būvimo vietos. Ūko serverių geografinio priartinimo galimybes ir infrastruktūrinius klausimus sprendžia paslaugos ar infrastruktūros administratoriai ar plėtros komandos. Projektuojamoje sistemoje galinių įrenginių autentifikavimas vyks tik tarp ūko ir galinių įrenginių sluoksnių. Numatoma autentifikavimo sistema yra paskirstyta ir veiks skirtinguose ūko kompiuterijos platformos serveriuose. Dėl plataus ūko serverių geografinio pasiskirstymo, vienos grupės serveriai gali būti klasifikuojami į tarpininkavimo ar autentifikavimo serverius, kita – duomenų dorojimo ar kaupimo serverius. Kadangi autentifikavimo sistema remiasi blokų grandinių ir paskirstytųjų sistemų technologiniais sprendimais, autentifikavimo sistemos komponentai veiks skirtinguose serveriuose ar jų grupėse. Tarpininkavimo serveriai tiesiogiai sąveikauja su galiniais įrenginiais, todėl šie serveriai turi sparčiai komunikuoti su galinių įrenginių autentifikavimo sistemą ir galiniais įrenginiais. Autentifikavimo sistema gali neturėti tiesioginio ryšio su galiniais įrenginiais autentifikavimo proceso vykdymo metu. Bendrieji projektuojamos sistemos komponentai pavaizduoti 2.2 pav. Užtikrinti autentifikavimo sistemos našumą, projektavimo metu atsižvelgiama į sistemos komponentų galimybes užtikrinti plečiamumui sistemoms poreikiams augant.



2.2 pav. Projektuojamos sistemos komponentai

Remiantis 2.2 pav. pavaizduotais pagrindiniais ūko kompiuterijos platformos komponentais, visų pirma reikia nuspręsti, kokia blokų grandinėmis grįsta sistema bus naudojama ūko kompiuterijos

platformoje. Sistemai projektuoti ir vystyti bus naudojami esantys sprendimai karkasai ir protokolai, tačiau visų pirma prieš projektuojant sekančius galinių įrenginių autentifikavimo sistemos komponentus reikia pasirinkti tinkamiausią sprendimą galinių įrenginių autentifikavimo uždaviniui spręsti. Remiantis atlikta blokų grandinių platformų, įrankių ir karkasų analize ir palyginimais, žinome, kad blokų grandinės yra dviejų pagrindinių tipų, viešosios ir privačios, abu tipai siūlo skirtingus privalumus, kurie tinka skirtingoms problemoms spręsti.

Projektuojamoje autentifikavimo sistemoje viešojo tipo blokų grandinių naudojimas nėra palankus, kadangi autentifikavimo sistema turi integruotis su kitomis sistemomis ūko kompiuterijos sluoksnyje. Įprasti kriptovaliutomis grįsti procesai ir duomenų modeliai gali būti pritaikomi autentifikavimo informacijai saugoti, tačiau sistemoje prisideda privačiųjų raktų generavimo uždavinys galiniuose įrenginiuose ir didėja privačiųjų raktų nutekėjimo rizika. Palyginus viešas ir privačiojo tipo blokų grandinių sistemas, pastebėta, kad privačiojo tipo blokų grandinės siūlo platesnį API kreipinių rinkinį. Projektuojamoje sistemoje blokų grandinių sistema ir kiti autentifikavimo sistemos komponentai turės sąveikauti tarpusavyje, API naudojimas autentifikavimo metode leis skirtingoms sistemomis komunikuoti tarpusavyje ir vykdyti autentifikavimo procesus. Kadangi skirtingos blokų grandinių sistemos siūlo specifinius API rinkinius ir galimybes, projektavimo metu turi būti pasirinkta blokų grandinių sistema projektuojamoje autentifikavimo sistemoje. Taip pat sistemos projektavimo metu reikia atsižvelgti į komponentų tarpusavio suderinamumą ir plečiamumo galimybes sistemoje. Remiantis atlikta blokų grandinių sistemų analize projektuojamoje sistemoje bus naudojamos privačiojo tipo blokų grandinių sistemos, todėl projektuojamoje sistemoje pasirinkta naudoti *Hyperledger Fabric* blokų grandinių sistema. Naudojant *Hyperledger Fabric* sistemą užtikrinamos autentifikavimo sistemos plečiamumo galimybės, prireikus galima didinti arba mažinti blokų grandinę aptarnaujančių komponentų kiekį sistemoje [27].

Žinučių brokerio pasirinkimas, projektuojant sistemą, priklauso nuo sprendžiamo uždavinio, perduodamų duomenų kiekio ir sistemoje naudojamų protokolų. Projektuojamoje autentifikavimo sistemoje, kuri veiks ūko kompiuterijos sluoksnyje, reikia užtikrinti žinučių pristatymą bent vieną kartą arba daugiau įrenginių autentifikavimo proceso metu. Kadangi galiniai įrenginiai duomenis siųs ne tik autentifikavimui atlikti, bet ir atlikti savo užduotį pagal paskirtį, žinučių kiekis gali išaugti priklausomai nuo numatytos taikymo srities. Dėl didelio galinių įrenginių įvairumo, projektuojant sistemą taip pat reikia pasirinkti žinučių brokerį, kuris palaikytų daugiau nei vieną duomenų perdavimo protokolą. Taip pat sistemoje naudojamo žinučių brokerio parinkimas projektavimo metu, leis tiksliau projektuoti sistemą, kadangi skirtingi žinučių brokeriai siūlo įvairias žinučių skirstymo ir pristatymo galimybes, taip pat nustatomas komponentų suderinamumas ir sistemos plečiamumo galimybės. Atlikus žinučių brokerių sistemų analizę ir palyginimus projektuojamoje sistemoje turime pasirinkti kuo universalesnį ir sistemos plečiamumą užtikrinantį žinučių brokerį. Atsižvelgiant į projektuojamos sistemos reikalavimus ir žinučių brokerių siūlomas galimybes, *RabbitMQ* žinučių brokeris geriausiai atitinka keliamus reikalavimus. Naudojant šį brokerį, galima plėsti sistemos brokerių skaičių pagal poreikį [28], taip pat *RabbitMQ* gali priimti daugelio tipų žinutes iš galinių įrenginių.

2.2. Galinių įrenginių autentifikavimas ir duomenų perdavimas

Šiame poskyryje atliekama galinių įrenginių duomenų perdavimo protokolų analizė ir palyginimas. Taip pat projektuojami galinių įrenginių autentifikacijos procesai ir funkcijos.

Galiniai įrenginiai yra įvairaus tipo įrenginiai, kurie turi įvairias paskirtis, viena iš šių įrenginių paskirčių yra perduoti duomenis kitiems įrenginiams internetu. Kadangi įprasti duomenų perdavimo protokolai, kaip TCP, UDP ir HTTP nėra palankūs naudoti dėl ribotų išteklių IoT įrenginiuose. Įprastinių protokolų naudojimas IoT įrenginiuose naudoja daugiau resursų galiniuose įrenginiuose, siekiant užtikrinti efektyvų galinių įrenginių veikimą buvo sukurti specializuoti žinučių perdavimo protokolai. Šie protokolai leidžia sparčiai ir efektyviai perduoti ir priimti duomenis iš kitų įrenginių.

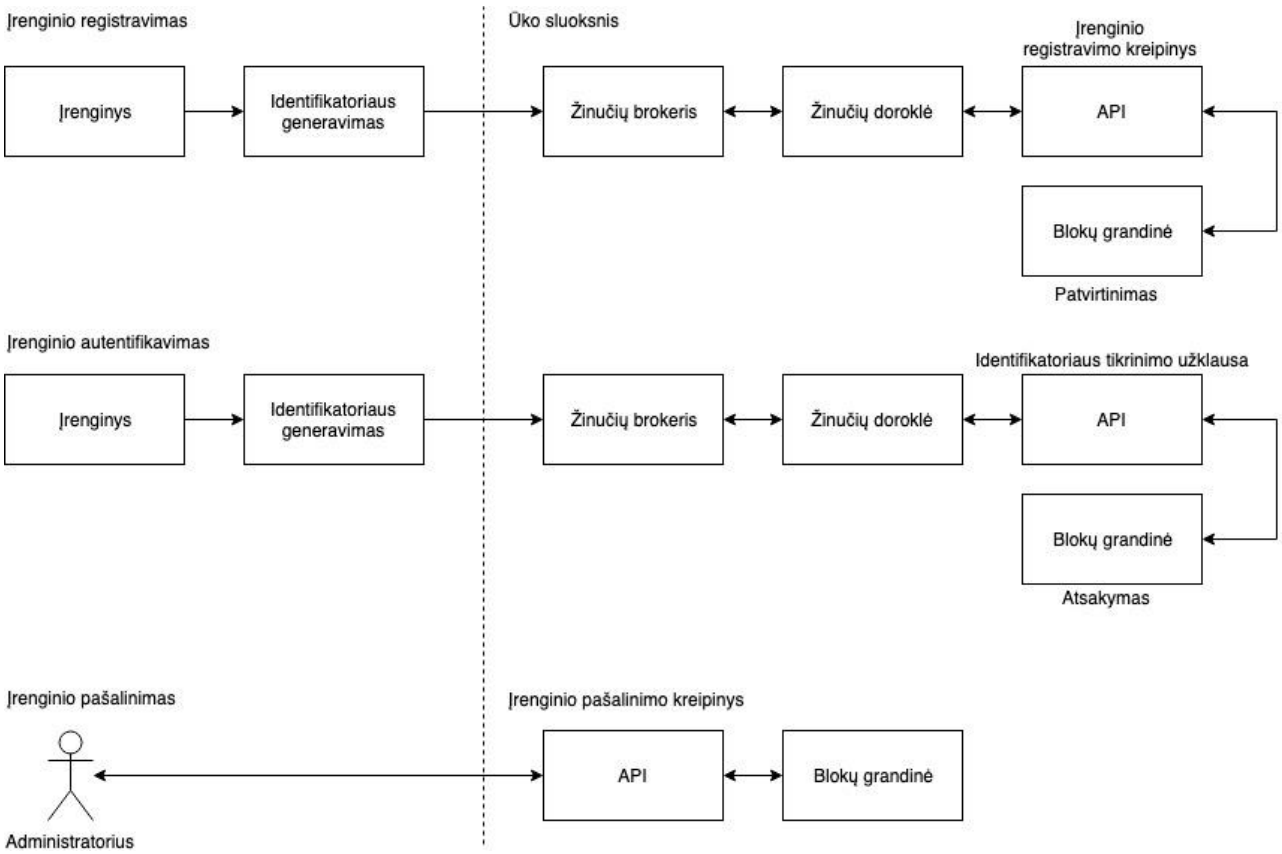
Atlikus pradinį ūko kompiuterijos platformos projektavimo etapą, ir apibrėžtus pagrindinius ūko kompiuterijos platformos būtinus sistemos komponentus, galima projektuoti galinių įrenginių autentifikavimo procesus ir duomenų perdavimo eigą sistemoje. Projektuojant galinių įrenginių autentifikavimo ir su autentifikavimu susijusius procesus apibrėžiamos projektuojamos sistemos gairės ir panaudos atvejai. Kadangi kuriamas autentifikavimo metodas turi būti saugus, todėl duomenų perdavimui turi būti naudojami saugus protokolai. Remiantis atlikta duomenų perdavimo protokolų analize, pastebėta jog visi IoT protokolai gali sparčiai perduoti duomenis tarp įrenginių, tačiau ne visi protokolai pasižymi saugumu ar saugos galimybėmis. Projektuojama sistema, be autentifikavimo funkcionalumo, turi būti suderinama su debesijos paslaugomis ir numatytuoju galinių įrenginių panaudojimu – duomenų perdavimu į debesų kompiuteriją. Todėl projektuojant sistemą pasirinkta naudoti AMQP duomenų perdavimo protokolą, šis nėra lengviausias iš visų aptartų protokolų, tačiau siūlo platų funkcionalumą ir žinučių siuntimo būdus.

2.2.1. Galinių įrenginių autentifikavimo procesų projektavimas

Įrenginio identifikatorių sudaro rečiausiai kintantys įrenginio aparatūrinės informacijos duomenų vienkryptės funkcijos reikšmė arba fiziškai neklonuojamos funkcijos panaudojimas. Įrenginio registravimas yra inicijuojamas galinio įrenginio, registracijai atlikti reikalinga pateikti įrenginio pavadinimą ir identifikatorių, šie duomenys siunčiami žinučių brokeriui, kuris perduoda žinutę apdorojimui. Žinutę gavęs prietaisas kreipiasi į blokų grandinę per API funkciją įrenginio registravimui į blokų grandinę. Galinio įrenginio autentifikavimas vyksta kiekvieną kartą įrenginiui kreipiantis į ūko kompiuterijos sluoksnį, siunčiami duomenys visada turi būti perduodami kartu su įrenginio identifikatoriumi.

Autentifikavimo procesas prasideda nuo galinio įrenginio kuomet duomenys turi būti perduodami į ūko kompiuterijos sluoksnį, galinis įrenginys siunčia žinutę žinučių brokeriui ūko sluoksnyje, kuris perduoda žinutę apdorojimui. Ūko prietaisas apdorojantis žinutes gauna žinutę iš įrenginio ir kreipiasi į blokų grandinę, naudodamas API užklausą įrenginio identitetui patvirtinti. Jeigu įrenginio identifikatorius sutampa su informacija blokų grandinėje duomenys perduodami atitinkamiems ūko sluoksnyje veikiantiems procesams.

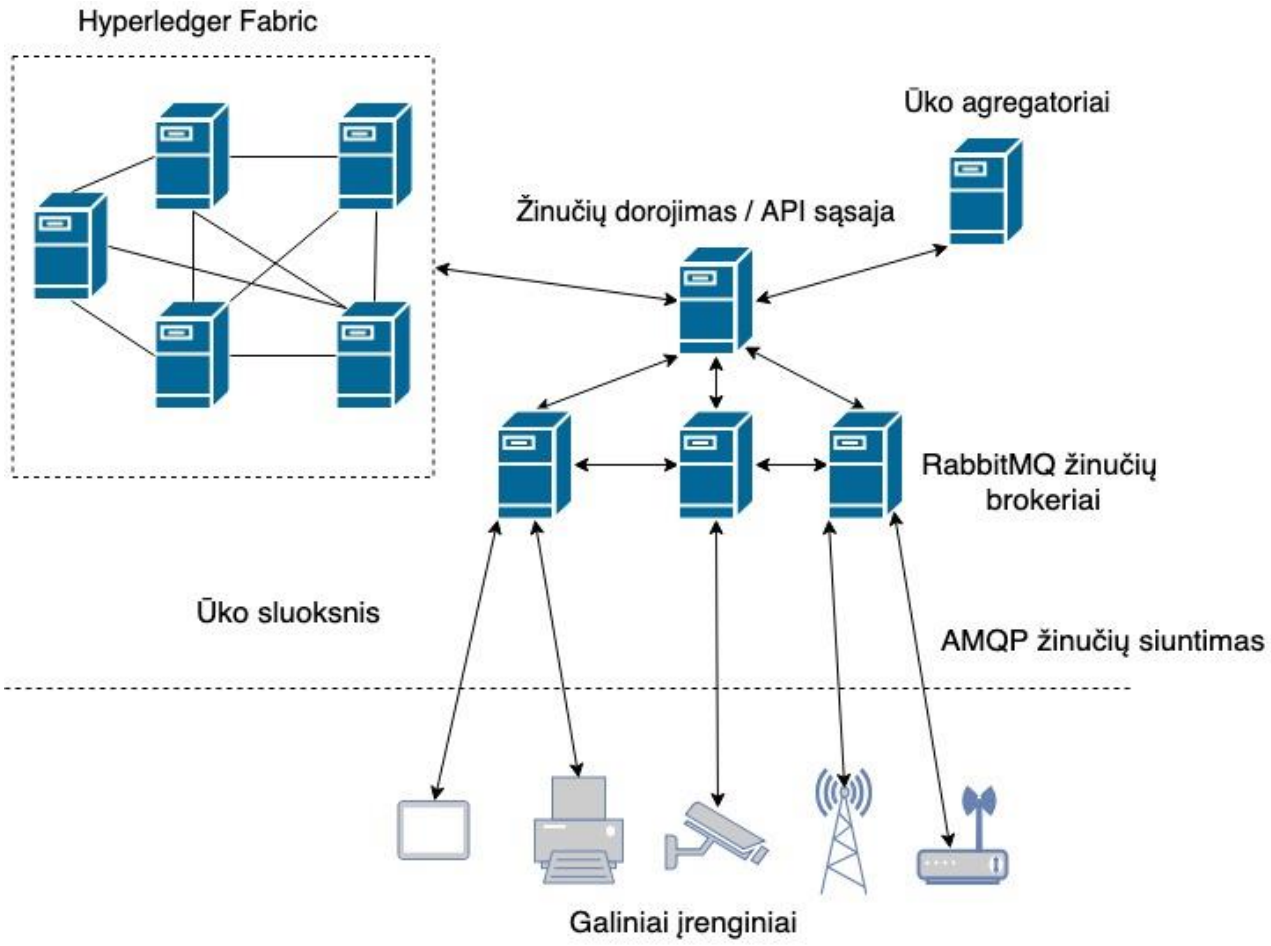
Įrenginio pašalinimo procesas nėra inicijuojamas galinio prietaiso, šis procesas gali būti vykdomas sistemos administratoriaus ar bet kurio ūko sluoksnio serverio, jeigu šį veiksmą įtakoja konsensusas ar kiti veiksmai ar nurodymai. Bet kuris ūko serveris ar ūko administratorius gali kreiptis į blokų grandinę, naudojant API užklausą pašalinti prietaisą iš blokų grandinės.



2.3 pav. Autentifikavimo procesų sekos diagramos

2.3. Galinių įrenginių autentifikavimo sistemos modelis

Remiantis analize ir projekcijomis, atliktomis ankstesniuose skyreliuose, projektuojamas galinių įrenginių autentifikavimo sistemos pradinis architektūrinis modelis, vaizduojamas 2.4 pav. Modelyje pateikiami pagrindiniai sistemos komponentai, pagrindiniai serverių tipai. Blokų grandinių sistemos visuma yra atskirta nuo kitų ūko kompiuterijos sluoksnyje veikiančių serverių. Projektuojant sistemą, blokų grandinių sistema dažnai vartojama kaip loginis vienetas, nedetalizuojant vidinių sistemos komponentų.



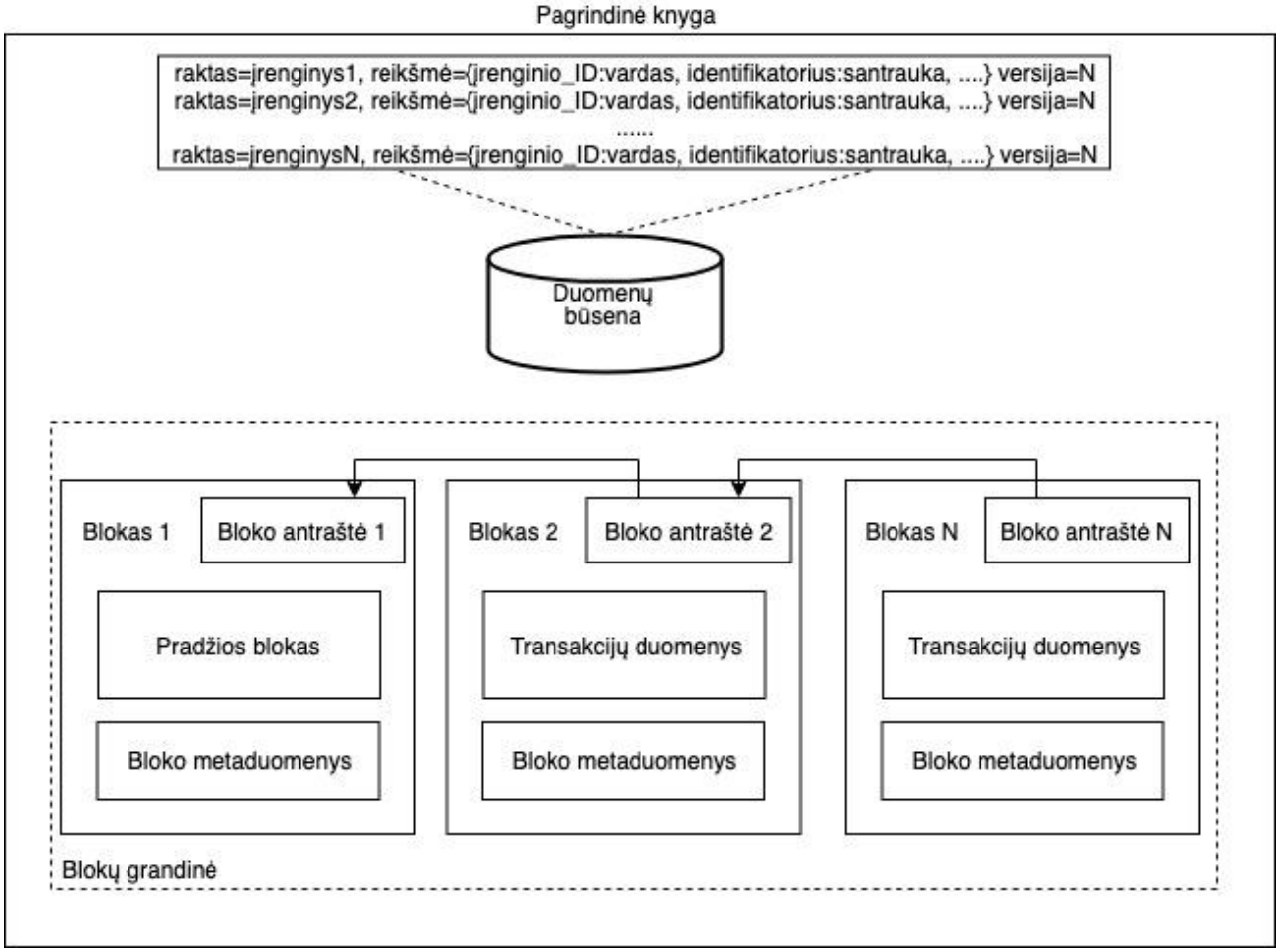
2.4 pav. Projektuojamos sistemos modelis

Ūko kompiuterijos sluoksnį sudaro tarpusavyje tinklą sudarantys serveriai ir įrenginiai. Projektuojamame ūko sluoksnyje išskiriami keturi skirtingi serverių tipai: tarpininkavimo, žinučių apdorojimo, blokų grandinių ir duomenų agregavimo serveriai.

Blokų grandinių platformoje veikia visi paskirstytieji serveriai, sudarantys blokų grandinių platformą. Tarpininkavimo serveriai – geografiškai arčiausiai galinių įrenginių esantys ūko sluoksnio serveriai. Šiuose serveriuose veikia žinučių brokeriai, kurie atlieka duomenų persiuntimą žinučių apdorojimo serveriams. Žinučių apdorojimo serveriai kaip ir tarpininkavimo serveriai turi būti įrengti nedideliu atstumu nuo galinių įrenginių telkties taškų, kadangi šie serveriai pagal žinutės tipą ir funkcijas kreipiasi į blokų grandinių platformą naudodami API kreipinius. Apdorojimo serveriuose taip pat veikia REST API sąsaja su blokų grandinės serveriais. Duomenų agregavimo serveriai atlieka duomenų agregavimo ir apdorojimo funkcijas. Galiniai įrenginiai tiesiogiai kreipiasi tik į tarpininkavimo serverius.

Remiantis šiame skyriuje atlikta analize ir išsiaiškinus projektuojamos sistemos komponentus galima modeliuoti autentifikavimo sistemą. Blokų grandinių serveriuose veiks *Hyperledger Fabric* blokų grandinių sistema, ši serverių grupė yra izoliuota ūko sluoksnyje ir sprendžia tik autentifikavimo uždavinio problemą – kaupiama tik galinių įrenginių autentifikacinė informacija ir vykdomi kiti sisteminiai procesai blokų grandinių funkcionalumą palaikyti. *Hyperledger Fabric* sistemoje išskiriami trys pagrindiniai serverių tipai: vykdymo ir duomenų (angl. Peer), sertifikatų valdymo (angl. CA – Certificate Authority) ir skirstymo serveriai (angl. Orderer). Vykdymo ir duomenų

serveriai vykdo grandinių kodo [14] (angl. chain code) operacijas atlikti skaitymo ir rašymo operacijas blokų grandinės pagrindinėje knygoje, taip pat kiekvienas šio tipo serveris turi pagrindinės blokų grandinės knygos kopijas. Pagrindinę knyga sudaro blokų grandinė ir duomenų būsenos (angl. World State) informacija, pagrindinės knygos sandara pavaizduota 2.5 pav. Duomenų būsena – duomenų bazė kaupianti informaciją rakto-reikšmės formatu, ši duomenų bazė veikia kaip spartinančioji atmintis. Duomenų būsenos gali sparčiai kisti, kadangi duomenys gali būti kuriami, atnaujinami ar trinami, kiekvienas duomenų pakeitimas yra versijuojamas. Blokų grandinė – transakcijų žurnalas, kuriame kaupiami įrašai apie visus atliktus pakeitimus, kurių pasekmė yra duomenų būsena [29]. Transakcijų informacija yra surenkama į blokus, kurie yra prijungiami į blokų grandinės galą. Blokų grandinių duomenų struktūra skiriasi nuo duomenų būsenos, kadangi blokų grandinių įrašai yra nekintami.



2.5 pav. Pagrindinės knygos sandara

Tarpininkavimo serveriuose veiks *RabbitMQ* žinučių brokeris, šių serverių paskirtis atlikti žinučių perdavimą kitiems serveriams, kuriuose veikia žinučių doroklės. Kadangi galiniai įrenginiai gali būti mobilūs prietaisai, duomenys iš galinių įrenginių bus siunčiami į geografiškai arčiausiai esančius tarpininkavimo serverius. Ši funkcionalumą užtikrina apkrovos paskirstytojai arba specializuoti kanoninių vardų įrašai. Žinučių brokeris palaiko keturis skirtingus žinučių skirstymo mechanizmus [30]: tiesioginį, vėduoklės principo, temomis grįstą ir antraštėmis grįstą. Žinučių siuntimo ir skirstymo mechanizmai ir parametrai yra nustatomi kiekvienai aptarnavimo eilei individualiai. Visi skirstymo mechanizmai naudoja skirtingus algoritmus nustatyti kurioms eilėms priklauso siunčiama žinutė.

Tiesioginio tipo skirstymo mechanizmas nukreipia siunčiamas žinutes, naudojant skirstymo raktą, kuris yra lygus skirstymo raktui, nurodytam eilės parametruose. Žinutė keliauja tokia tvarka: siuntėjas išsiunčia žinutę, nurodydamas skirstymo raktą ir skirstymo mechanizmą, *RabbitMQ* serveryje surandama atitinkama eilė ir žinutė išsiunčiama gavėjui. Šio tipo žinučių skirstymas naudingas tuomet, kai reikiama nurodyti žinučių aptarnavimo raktą teksto eilute, siuntėjas ir gavėjas turi žinoti eilės ir skirstymo rakto reikšmę.

Vėduoklės principo skirstymo mechanizmas perduoda žinutes visoms nurodytoms eilėms, šio tipo skirstymo metodai dažniausiai naudojami prenumeravimo-siuntimo žinučių perdavimui realizuoti. Viena išsiųsta žinutė gali būti priimama skirtingų gavėjų, tai yra patogu tuo atveju, jeigu žinutės nėra vieno pobūdžio ir turi būti gaunamos skirtingų gavėjų tenkinant nustatytas sąlygas. Žinutės keliauja tokia tvarka: siuntėjas išsiunčia žinutę į *RabbitMQ* serverį, tuomet vėduoklės skirstymo mechanizmas perduoda žinutes reikiamoms eilėms taip paskirstydamas žinutes, gavėjai, prenumeruojantys žinutes iš tam tikros eilės, gauną eilei skirtą pranešimą.

Temomis grįstas skirstymo mechanizmas taip pat būdingas prenumeravimo-siuntimo perdavimo mechanizmui realizuoti. Šis tipas nukreipia žinutes į eiles, kuriose skirstymo rakte esantys žodžiai sutampa su eilės priskyrimo raktu eilės nustatymuose. Šio tipo skirstymas naudingas kuomet reikia nukreipti žinutes, remiantis keliais kriterijais. Žinutė siunčiama į *RabbitMQ* serverį, nurodant skirstymo raktą ir jam priklausančias temas, atskiriant temas taško (“.”) simboliu, pavyzdžiui: tinklas.prietaisas.vietovė. Skirtingos eilės gali priimti skirtingas žinutes, pavyzdžiui: eilė turinti priskyrimo raktą “*.*.vietovė” gaus visas žinutes kuriose nurodoma tema vietovė, eilė, turinti raktą “#.prietaisas.*” žinutės negaus.

Antraštėmis grįstas skirstymo mechanizmas suteikia galimybę skirstyti žinutes, remiantis žinutės antraštėje esančia informacija. Šis skirstymo tipas panašus į temomis grįstą skirstymo mechanizmą, tačiau suteikia galimybę efektyviau filtruoti ir skirstyti žinutes, nenaudojant iš anksto numatytų paskirstymo raktų. Tačiau eilių nustatymai tampa sudėtingesni, šis būdas tinkamas įvairaus tipo žinutėms perduoti kai yra žinoma imtis kriterijų pagal kuriuos turi būti priimamos žinutės. Šis skirstymo tipas remiasi rakto-reikšmės principu. Prototipo kūrimo metu palankiausia naudoti tiesioginio siuntimo eiles, tačiau pakitus poreikiams nesunku pakeisti siuntimo ir paskirstymo mechanizmus.

Galinių įrenginių autentifikavimui geriausia naudoti tiesioginio tipo siuntimo mechanizmą, kadangi siekiama užtikrinti spartų ir patikimą duomenų perdavimą įrenginių registravimo ir autentikavimo metu. Tolimesniam duomenų apdorojimui ir persiuntimui ūko kompiuterijos sluoksnyje pravartu naudoti vėduoklės ir temomis grįstus skirstymo mechanizmus, taip užtikrinamas geriausias apkrovos paskirstymas ūko sluoksnyje, kadangi visi žinučių apdorojimo serveriai gali autentikuoti galinius įrenginius, tačiau žinutės su tema skirta ilgalaikiam duomenų saugojimui, gali būti perduotos ūko agregatoriams tolimesniems procesams vykdyti.

Žinučių apdorojimo serveriuose veikia žinučių dorojimo paslaugos ir API sąsaja su blokų grandinės platforma, programinė įranga jungiasi prie tarpininkavimo serverio ir laukia duomenų, kuriuos reikia apdoroti. Gavus žinutes tam tikra tema, pavyzdžiui registracijos atveju, serveris kreipiasi į blokų grandinę, naudodamas API šaukinius naujo įrenginio registravimui sistemoje. Kitu atveju, siunčiama užklausa į blokų grandinę įrenginio autentifikacijai atlikti prieš duomenis apdorojant.

Duomenų agregavimo serveriai pasižymi didesnėmis duomenų talpyklomis ir skirtinga programine įranga. Duomenys į šiuos serverius atkeliauja iš žinučių apdorojimo serverių, kurie yra skirti ilgalaikiam saugojimui ar reikalingas duomenų apdorojimas prieš duomenis perduodant į debesijos paslaugas.

Duomenys siunčiami iš galinių įrenginių į ūko tarpininkavimo serverius, perduodami naudojant AMQP protokolą. Didžioji dalis populiariausių programavimo kalbų palaiko AMQP bibliotekas ar klientus, todėl nėra kliūčių pritaikyti AMQP protokolą daugumoje aplikacijų. *RabbitMQ* brokeris gali priimti žinutes iki 512 MB dydžio todėl galime perduoti didelį duomenų kiekį vienoje žinutėje, žinutės kompozicija pavaizduota 2.4 lentelėje.

2.1 lentelė. AMQP žinutės sandara

Antraštė	Pristatymo anotacijos	Žinutės anotacijos	Ypatybės	Aplikacijos ypatybės	Aplikacijos duomenys	Poraštė
			Žinutės pagrindas			

Antraštėje pateikiami penki žinutės atributai:

- Patvarumas(angl. durable) – žinutės toleranciją klaidoms siuntimo metu
- Pirmumas(angl. priority) – žinutės pirmumo reikšmė
- TTL(angl. Time To Live) – pateikiama žinutės TTL reikšmė
- Pirmasis gavėjas (angl. first-acquirer) – jeigu ši reikšmė yra “true”, tuomet žinutė nebuvo priimta kitos jungties
- Pristatymų skaičius (angl. delivery-count) – kiek kartų žinutė buvo nesėkmingai išsiųsta

Ypatybėse pateikiamos šios savybės:

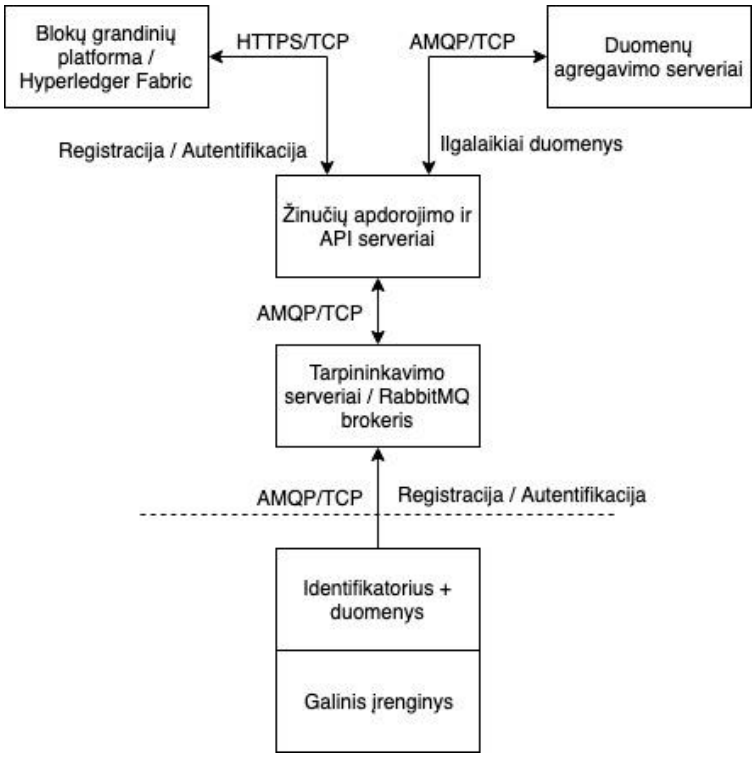
- Žinutės ID – aplikacijos žinutės identifikatorius
- Naudotojo ID – žinutės kūrėjo ID
- Gavėjas – žinutės gavėjo adresas
- Tema – žinutės tema
- Atsakymo adresas – siuntėjo atgalinis adresas
- Koreliacijos ID – aplikacijos koreliacijos identifikatorius
- Turinio tipas – MIME turinio tipas
- Turinio koduotė – MIME turinio tipas, naudojamas turinio tipo pakeitimui nurodyti
- Galiojimo laikas – laikas, kada ši žinutė nebegalioja
- Sukūrimo laikas – laikas, kada buvo sukurta žinutė
- Grupės ID – nurodoma grupė kuriai skirta siunčiama žinutė
- Grupės eiliškumas – eiliškumo numeris skirtas žinutės grupei
- Grupės atsakymo adreso ID – grupės kuriai priklauso atsakymas ID

Aplikacijos duomenys – žinutėje siunčiamų duomenų laukas, kuriame siunčiama pagrindinė žinutės informacija. Tai gali būti metrikos, slaptažodžių santraukos ar kiti norimi perduoti duomenys.

Žinutės poraštė naudojama detalėms apie žinutę ar jos pristatymą, šie duomenys gali būti apskaičiuoti ir įvertinti tik kuomet visą pagrindinės žinutės informacija yra nustatyta.

2.4. Sistemos projektavimo išvados

Šiame skyriuje buvo atliktas blokų grandinėmis grįstos galinių įrenginių autentifikavimo ūko kompiuterijoje sistemos projektavimas ir sistemos veikimo procesų modeliavimas. Remiantis atlikta analize ir projektavimu, galime apibrėžti sistemos veikimo modelį, architektūrą bei autentifikavimo sistemoje naudojamus protokolus. Apibendrintas sistemos veikimo modelis ir naudojami protokolai pavaizduoti 2.6 pav.



2.6 pav. Apibendrintas sistemos veikimo modelis

Siekiant išgryninti galinių įrenginių autentifikavimo metodo ūko kompiuterijoje naudojant blokų grandinėmis grįstas technologijas struktūrą ir architektūrą buvo išskirti trys serverių tipai, veikiantys ūko kompiuterijos sluoksnyje, taip pat atskirai veikianti serverių visuma, sudaranti blokų grandinių platformą. Ūko kompiuterijos serverius sudaro: tarpininkavimo, API ir žinučių apdorojimo serveriai ir duomenų agregavimo serveriai. Blokų grandinių platforma veikia ūko kompiuterijos sluoksnyje, tačiau blokų grandinių platformą sudarantys serveriai su ūko kompiuterijos serveriais duomenimis keičiasi tik API kreipiniais. Sistemos architektūra projektuota atsižvelgiant į plečiamumo galimybes, visų ūko kompiuterijoje veikiančių serverių skaičius gali būti didinamas sistemos našumui ir galimybėms didinti. Tai yra reikalinga, siekiant prisitaikyti prie galinių įrenginių skaičiaus didėjimo.

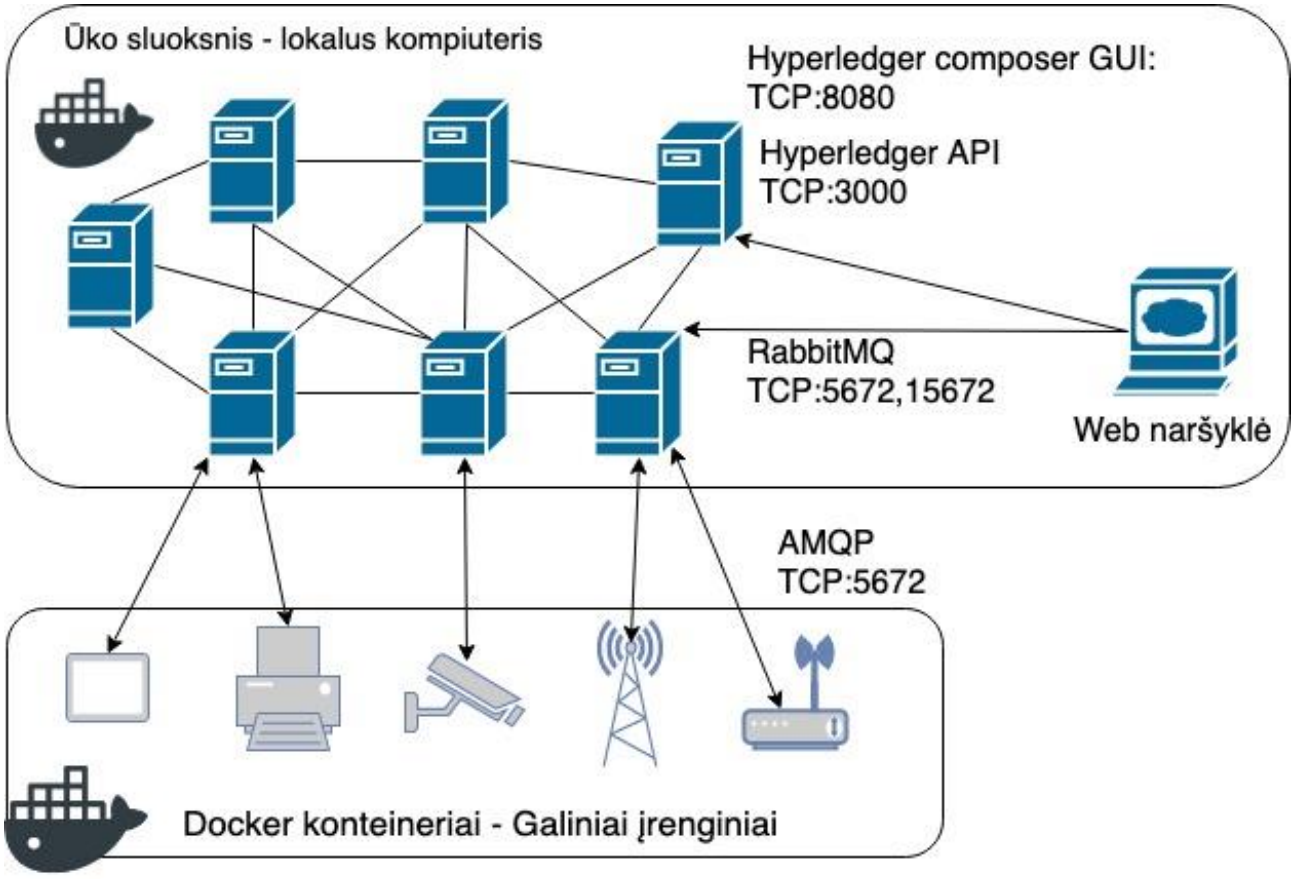
Apibrėžus sistemos architektūrą sekė duomenų perdavimo protokolo pasirinkimo analizė ir autentifikavimo procesų projektavimas. Remiantis ūko kompiuterijos sluoksnio architektūra ir naudojamais komponentais buvo atlikta duomenų perdavimo protokolų analizė. Palyginus esamus ir su apibrėžta architektūra suderinamus duomenų perdavimo protokolus, nuspręsta naudoti AMQP protokolą, kuris pasižymi saugumu ir plačiomis suderinamumo ir plečiamumo galimybėmis. Atlikus visų sistemos komponentų analizę ir priėmus architektūrinius sprendimus buvo projektuojami galinių įrenginių autentifikacijos procesai. Išskirti trys įrenginių autentifikavimo procesai: įrenginio registravimas, įrenginio autentifikacija ir įrenginio pašalinimas iš sistemos. Įrenginio identifikatorius yra generuojama kas kartą kreipiantis į ūko kompiuterijos sluoksnį. Todėl galiniuose įrenginiuose

nėra saugoma slaptažodžio reikšmė, tai leidžia sumažinti slaptažodžių nutekėjimo grėsmę.Taip užtikrinamas galinio įrenginio identitetas. Autentifikavimo procesai yra vykdomi blokų grandinių platformoje, taip pat įrenginio autentifikacinė informacija yra saugoma blokų grandinėje.

3. Blokų grandinėmis grįstos galinių įrenginių autentifikavimo ūko kompiuterijoje metodo realizacija

Suprojektuotos sistemos prototipui realizuoti reikalinga aplinka, kurioje bus modeliuojama virtuali ūko kompiuterijos infrastruktūra ir galinių įrenginių sąveika. Ūko sluoksnio modeliavimui prireiks įdiegti ir sukonfigūruoti *Hyperledger Fabric* blokų grandinių platformą, naudojant *Hyperledger Composer* įrankį, žinučių dorojimo serverį, tarpininkavimo serverį, kuriame veiks *RabbitMQ* žinučių brokeris ir bent vieną galinį įrenginį imituojantį prietaisą. Ūko sluoksnio serveriai gali būti virtualios mašinos arba *Docker* konteineriai, veikiantys tame pačiame virtualiame tinkle. IoT įrenginiai taip pat gali įvairaus tipo virtualios mašinos ar *Docker* konteineriai, kadangi dauguma IoT prietaisų remiasi *Linux* architektūra, galime naudoti *Docker* atvaizdus ar virtualias mašinas su lengvasvorėmis *Linux* operacinėmis sistemomis.

Remiantis informacija ir gairėmis sistemos projektavimo metu, prototipo kūrimui ir blokų grandinių platformai realizuoti bus naudojamas *Hyperledger Composer* įrankis, kuris užtikrins autentifikacijos mechanizmo ir blokų grandinių veikimą ūko kompiuterijos sluoksnyje. *Hyperledger Composer* naudoja *Docker* konteinerius sukurti *Hyperledger Fabric* blokų grandinių sistemą. Autentifikacijos procese ūko sluoksnyje taip pat veiks ir kiti ūko kompiuterijos serveriai, todėl kuriant sistemos prototipą paranku šiuos serverius naudoti viename virtualiame *Docker* tinkle, taip imituojant ūko kompiuterijos sluoksnį. Kadangi modeliuoti ir realizuoti ūko sluoksnį pasirinkta naudoti *Docker* virtualizaciją, galinių įrenginių sluoksniui ir įrenginių realizavimui taip pat galima naudoti *Docker* konteinerius. Pirminė projektuojamo sistemos prototipo architektūra pavaizduota 3.1 pav.

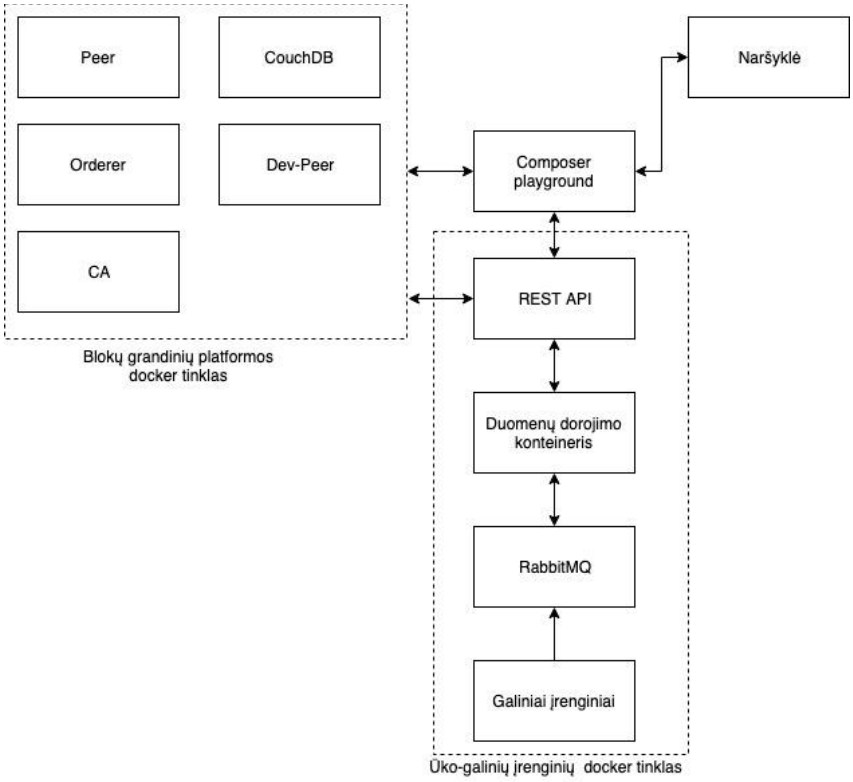


3.1 pav. Pirminė sistemos prototipo architektūra

3.1. Galinių įrenginių autentifikavimo sistemos prototipas

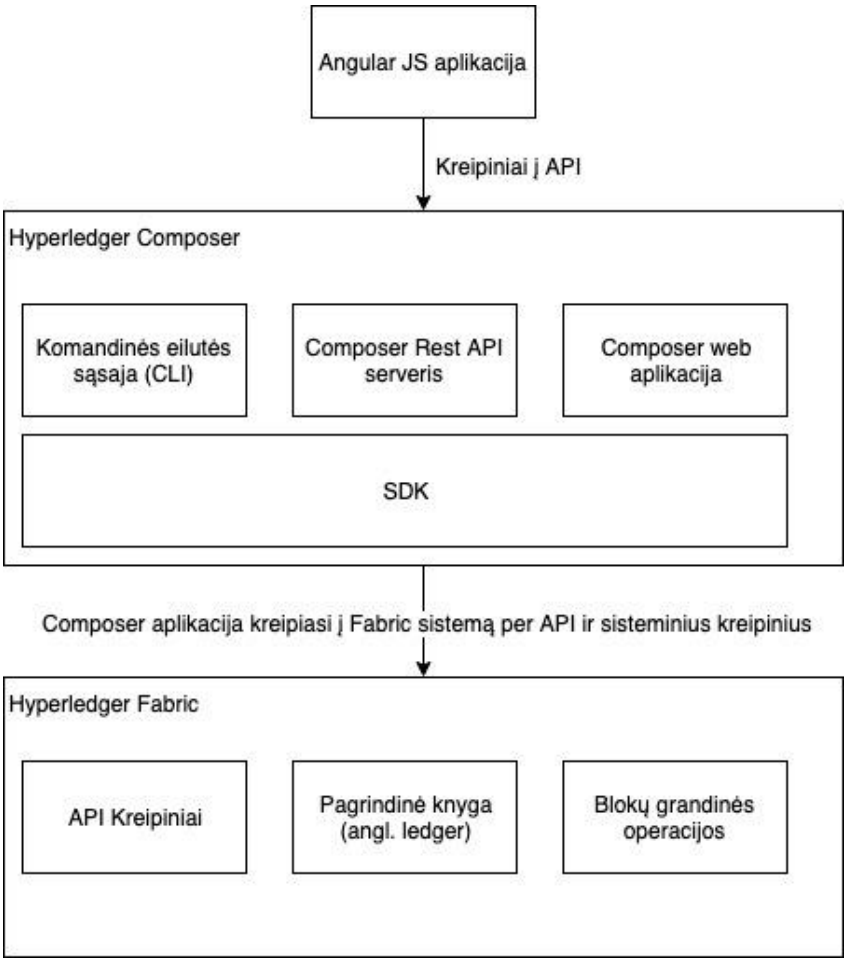
Realizuoti blokų grandinių platformos veikimą, naudojama *Hyperledger Fabric* sistema, tačiau palengvinti prototipo kūrimą taip pat naudojamas *Hyperledger Composer* įrankis, kuris palengvina blokų grandinių aplikacijų prototipų kūrimo procesą, remiantis *Hyperledger Fabric* sistema. *Hyperledger Composer* įrankis naudoja paruoštus scenarijus sukurti virtualią *Hyperledger Fabric* blokų grandinių sistemą, naudojant *Docker* pagalba yra sukuriami penki pradiniai konteineriai: *peer, dev-peer, couchdb, ca* ir *orderer*.

Peer konteineris atlieka blokų grandinių operacijų ir konsensuso mechanizmų vykdymo procesus, *dev-peer* konteineris vykdo grandinių kodo operacijas, kurias patvirtina *peer* konteineris naudojant konsensuso protokolus. Naudojant *Hyperledger Composer* įrankį, pagrindinės knygos kopija yra saugojama ne *peer* bet *couchdb* konteineriuose. CouchDB duomenų bazėje saugojama duomenų būsenos informacija ši duomenų bazė kaupia įrašus apie blokų grandinėje saugomą informaciją, šis sprendimas yra būtinas užtikrinti sistemos našumą, atliekant užklausas ir skaitymo operacijas grandinių kodo vykdymo metu. Taip pat duomenų būsenos duomenų bazės veikia kaip spartinančioji atmintis skaitymo operacijoms blokų grandinėse vykdyti, vykdant užklausas ar skaitymo operacijas *peer* tipo serveriai neturi kiekvieną kartą ieškoti informacijos rekursyviai, keliaujant per blokų grandinės transakcijų istoriją. *CA* konteineris atlieka sertifikatų valdymo funkciją, šis konteineris išduoda VRI paremtus sertifikatus tinklo organizacijoms ir jų vartotojams. *CA* išduoda vieną šakninį ir vieną registravimo sertifikatą kiekvienam autorizuotam sistemos vartotojui. Tačiau galiniai įrenginiai nėra klasifikuojami, kaip sistemos vartotojai, kadangi jų duomenis į blokų grandinę perduoda žinučių dorojimo serveriai. Skirstymo serveriai yra pakeičiami o*rderer* konteineriais, šio konteinerio paskirtis yra skirstyti transakcijas į blokus, skirstymo paslauga veikia nepriklausomai nuo vykdymo serverių. Bendras *Docker* tinklų ir konteinerių architektūrinis modelis pavaizduotas 3.2 pav.



3.2 pav. *Docker* tinklų ir konteinerių architektūra

Kadangi *Hyperledger Composer* pagalba galime naudotis web aplikacijos sąsaja modeliuoti blokų grandinių duomenų modelius ir naudotis API sąsaja, todėl yra kuriama *AngularJS* paremta naršyklinė aplikacija, *Yeoman node.js* modulio pagalba, sugeneruojamas pradinis programinio kodo skeletas, kurio pagalba galime kurti blokų grandinių sistemos duomenų modelių aprašus. Toliau, naudojant *AngularJS* skeletą, kurį modifikuojame apibrėžti blokų grandinių autentifikacijos tinklo dalyvius, transakcijas ir jų logiką, įvykius ir prieigos kontrolės nuostatas . *Hyperledger Composer* karkasą sudaro šie komponentai: *composer-cli* – komandinės eilutės sąsaja, leidžianti vykdyti kreipinius į *Hyperledger Fabric* sistemą, *composer-rest-server* – REST API serveris sudaro galimybę *Angular JS* aplikacijai ar kitiems klientams siųsti API užklausas į *Hyperledger Fabric* sistemą ir atvaizduoti duomenis web aplikacijoje ir atlikti pakeitimus blokų grandinių sistemoje. *Composer playground* – web aplikacija [31], leidžianti dirbti su blokų grandinių platforma. *AngularJS* aplikacijos ir *Hyperledger Fabric* ir *Composer* sistemų sąveikos modelis yra pavaizduotas 3.3 pav pateiktoje schemoje.



3.3 pav. *Hyperledger Composer* karkaso komponentų schema

Blokų grandinių sistemos kūrimui naudojama *Hyperledger Fabric* platforma, naudojant *Composer* įrankį ir jo komponentų pagalbą pirmiausia reikia sukurti blokų grandinės griaučius [32]. Blokų grandinių biznio tinklo šabloninių failų kūrimui naudojamas *Yeoman NPM* modulis, kuris komandinėje eilutėje naudojant komandą:

yo hyperledger-composer:businessnetwork

Biznio tinklų apibrėžimai (angl. business network definition) yra vienas iš pagrindinių *Hyperledger Composer* veikimo principų, šie apibrėžimai aprašo duomenų modelį, transakcijų logiką, autoriaus kontaktus, licenciją ir vardų sritį. Įvykdžius aukščiau nurodytą komandą, nurodomas tinklo pavadinimas, licencija, vardų sritis. Biznio tinklus aprašo šie komponentai:

- Turtas-vienetas (angl. Asset) - aprašo fizines ar kitas objekto savybes, vienetas gali aprašyti blokų grandinėje saugojamų objektų duomenų modelius. Vienetas privalo turėti unikalų identifikatorių(ID), kitos objekto ypatybės gali būti laisvai aprašomos.
- Dalyviai (angl. Participant) - biznio tinklo dalyviai, dalyvis gali turėti turtą ir pateikti užklausas. Dalyvis turi turėti unikalią identifikavimo žymę (angl. ID), kitos ypatybės gali būti naudojamos kaip pagalbiniai duomenys identifikuojantys dalyvį. Autentifikavimo sistemoje dalyviai yra galiniai įrenginiai.
- Transakcijos (angl. Transactions) – funkcijos, vykdančios grandinių kodą, transakcijas naudoja dalyviai atlikdami turto valdymo operacijas. Transakcijos yra vykdomos blokų grandinėje, tačiau inicijuojamos galinių vartotojų, kiekviena transakcija yra patvirtinama naudojant konsensuso protokolais.
- Prieigos teisės (angl. Access control rules) – leidžia pritaikyti prieigos kontrolės taisykles dalyviams biznio tinklo viduje.
- Užklausos ir įvykiai (angl. Queries, Events) – naudojant užklausas galime atlikti duomenų paieškos operacijas turto ir dalyvių paieškai atlikti blokų grandinėje. Užklausos gali būti inicijuojamos REST API pagalba. Įvykiai papildo transakcijų funkcionalumą, įvykiai gali būti iškviečiami įvykus transakcijai, aplikacijos gali prenumeruoti įvykių žinučių gavimą naudojant *Composer* API klientą.

Sukūrus biznio tinklo griaučius pirmiausia aprašomas biznio modelio objektų failas, šiuo atveju programinis kodas pateikiamas org.fog.network.cto faile. Kiekvieno galinio įrenginio informacija pateikiama JSON formatu, įrenginio informacijos duomenų struktūra pateikta 3.1 lentelėje.

3.1 lentelė. Galinio įrenginio informacijos duomenų struktūra

```
{
   "$class": "org.fog.network.Device", //pateikiama biznio tinklo klasė
   "deviceID": "string", //Įrenginio ID(galinio įrenginio pagrindinis kompiuterio vardas)
   "identifier": "string", //Įrenginio identifikatorius
   "metadata": "string", //Papildomi įrenginio siunčiami duomenys
   "revoked": boolean // Reikšmė nurodanti galinio įrenginio narystės galiojimą sistemoje
}
```

Apibrėžus biznio tinklo dalyvių ir objektų struktūrą, kuriama papildoma logika darbui su blokų grandine, funkcijų logika aprašoma logic.js faile projekto *lib* kataloge. Šiame faile aprašomos funkcijos, kurios kreipiasi į blokų grandinę, duomenų paieškai ar duomenų įrašymui ir trynimui blokų grandinėje. Transakcija *RevokeAccess* leidžia keisti galinio prietaiso informaciją blokų grandinėje, susijusią su įrenginio būsena sistemoje, galime pašalinti prieigą prie blokų grandinės sistemos ir ūko sluoksnio. Transakcija *RemoveRevokedDevices* leidžia pašalinti visus įrenginius iš sistemos, kurių narystė yra atšaukiama.

Prieigos kontrolės taisyklės aprašomos *permissions.acl* faile projekto šakniniame kataloge, sukuriamas tinklo prieigos teisių aprašas, kuriame aprašomos taisyklės, kurios nusako kokie dalyviai

gali atlikti tam tikro pobūdžio operacijas sisteminiuose resursuose ar biznio tinkle. Užklausos, kurios yra naudojamos dalyvių ar turto paieškoje blokų grandinėje yra aprašomos *queries.qry* faile, užklausas galime vykdyti transakcijose arba naudojant REST API, sukurtos užklausos gali rasti visus prietaisus pagal prisijungimo ID, rasti prietaisus pagal slaptažodį arba rasti prietaisus, kurių narystė yra atšaukta.

Aprašius visus aukščiau išvardintus failus, galime sukurti biznio tinklo archyvą, kuris apjungia visus programinio kodo failus į archyvą kuris vėliau yra diegiamas į *Hyperledger Fabric* sistemą. Kuriant sistemos prototipą, buvo pasirinktas *fog-auth* projekto pavadinimas.

composer archive create --sourceType dir --sourceName . -a fog-auth@0.0.1.bna

Prieš diegiant tinklo archyvą, reikia sukurti *Hyperledger Fabric* sistemą, sistemos infrastruktūra sukuriama naudojant *Shell* scenarijų: *./startFabric.sh,* kuris sukuria reikiamus *Docker* konteinerius sistemai suskurti. Taip pat reikia sukurti naują administratoriaus prisijungimo plokštę, kuri yra įdiegiama į *Fabric* sistemą ir vėliau naudojama administruoti blokų grandinę, administratoriaus korta sukuriama paleidžiant *Shell* scenarijų: *./createPeerAdminCard.sh,* plokštė išsaugojama networkadmin.card faile. Kada *Fabric* aplinka yra sukurta ir turime reikiamus failus, galime įdiegti ankščiau sugeneruotą tinklo archyvą į *Fabric* sistemą, tai atliekama šia *composer* komandinės eilutės komanda:

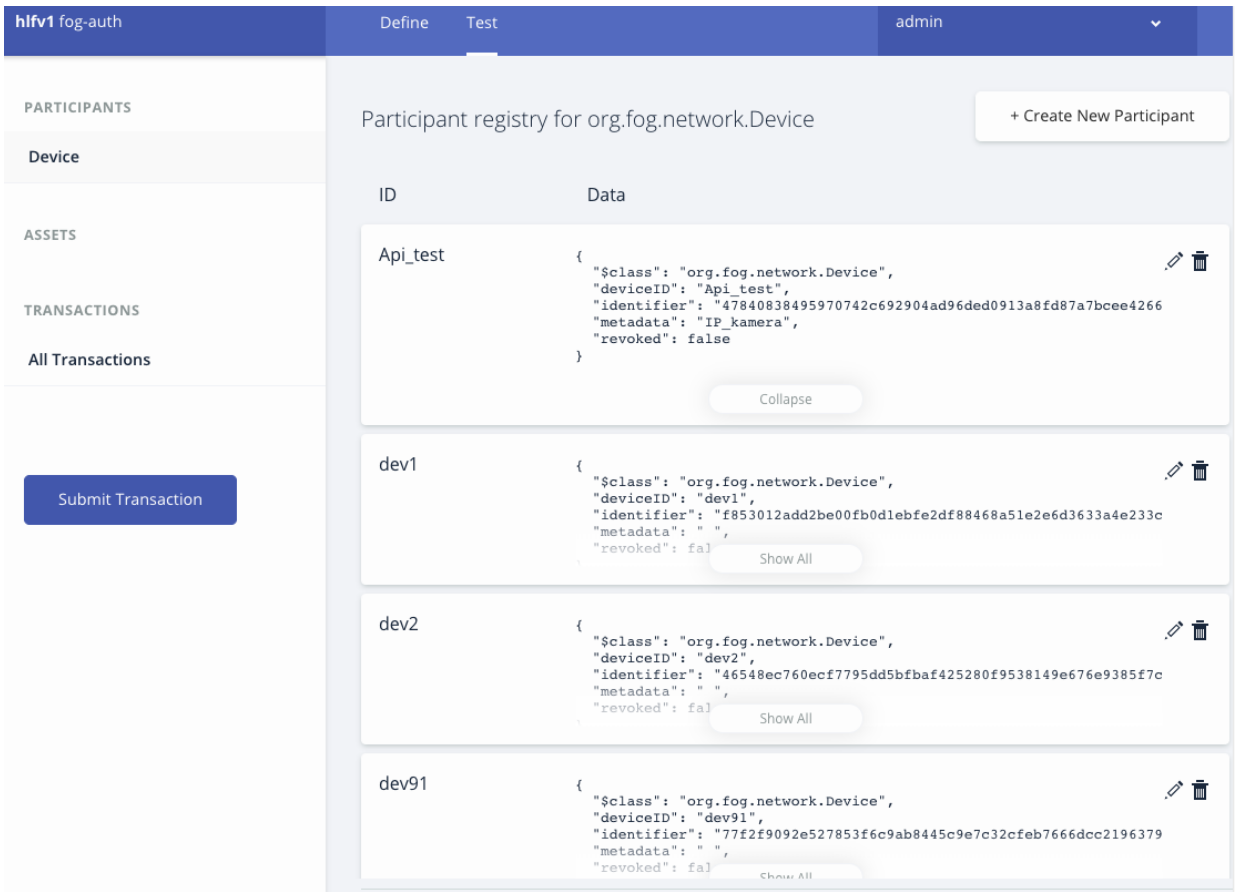
composer network install --card PeerAdmin@hlfv1 --archiveFile fog-auth@0.0.1.bna

Kai tinklas yra įdiegtas *Hyperledger Fabric* aplinkoje, toliau reikia paleisti tinklą, nurodant tinklo vardą, versiją, administratoriaus prisijungimo duomenis ir importuoti administratoriaus plokštės failą:

composer network start --networkName fog-auth --networkVersion 0.0.1 --networkAdmin admin --networkAdminEnrollSecret pass --card PeerAdmin@hlfv1 --file networkadmin.card

composer card import --file networkadmin.card

Įdiegus biznio tinklą ir jį paleidus, galime pradėti naudotis grafine vartotojo sąsaja, tai atliekama naudojant paleidus komandinės eilutės komandą: *composer-playground,* REST API serveris sukuriamas naudojant *Docker-compose* failą. *Hyperledger Composer* grafinę vartotojo sąsają galime pasiekti adresu http://localhost:8080/ , REST API serveris yra pasiekiamas http://localhost:3000/explorer adresu. Naudojant *Composer playground* vartotojo sąsaja galima inicijuoti transakcijas, įtraukti naujus dalyvius, redaguoti biznio tinklo failus ir diegti naujas tinklo versijas, tai suteikia visą pagrindinį funkcionalumą reikalingą atlikti blokų grandinių sistemų modeliavimui ar prototipų kūrimui. Tačiau blokų grandinės sistemos integracijai su kitomis sistemos ir galiniais įrenginiais būtina naudoti REST API serverį, kuris priima API kreipinius ir gali inicijuoti operacijas blokų grandinių sistemoje. Sistemoje registruotus įrenginius taip pat galima peržiūrėti, registruoti ir šalinti naudojant grafinę vartotojo sąsają, 3.4 pav. pavaizduota dalis grafinės vartotojo sąsajos, kurioje matomi sistemoje esantys galiniai įrenginiai, Api_test įrenginys sukurtas naudojant REST API vartotojo sąsają, likę įrenginiai sukurti žinučių aptarnavimo scenarijų, kurie kreipiasi į REST API serverį naudojant POST ir GET API kreipinius.



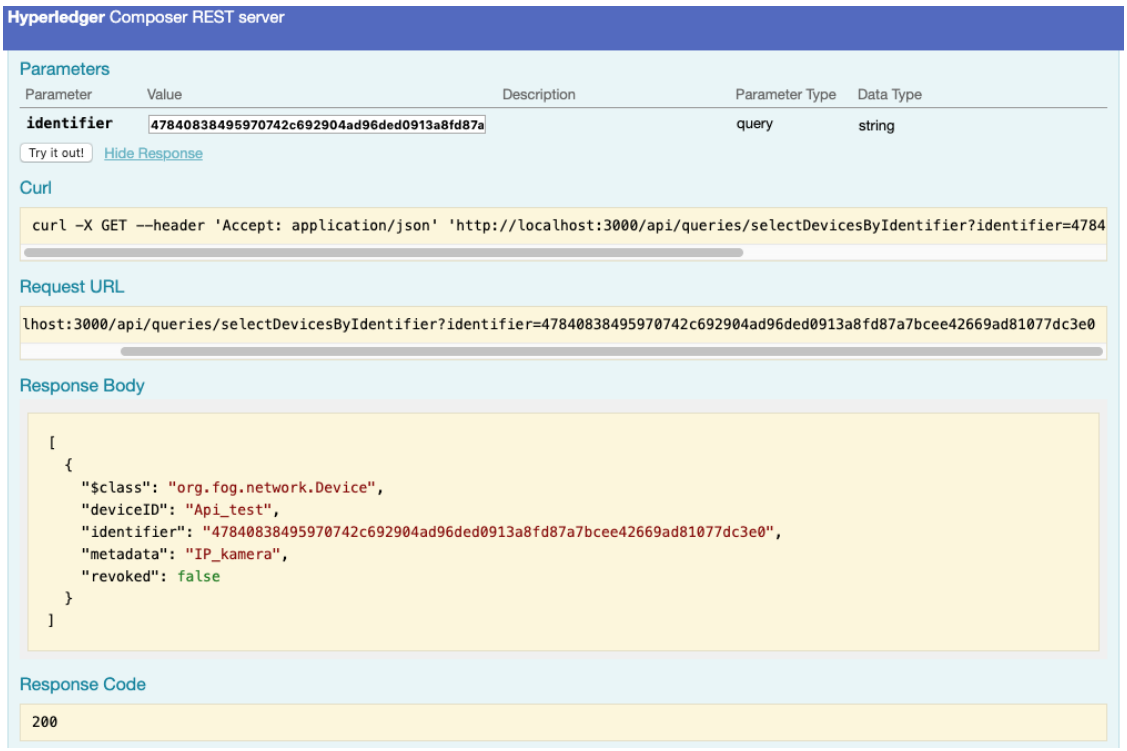
3.4 pav. *Composer* grafinė vartotojo sąsaja ir galiniai įrenginiai

REST API serveris atlieka duomenų perdavimo į blokų grandinių sistemą funkciją, užklausos yra priimamos iš žinučių dorojimo serverio. Įrenginio registravimo POST API užklausos pavyzdys yra pateikiamas 3.2 lentelėje, formuojant užklausą būtina nurodyti visus reikiamus galinio įrenginio parametrus.

3.2 lentelė. API kreipinys įrenginio registracijai

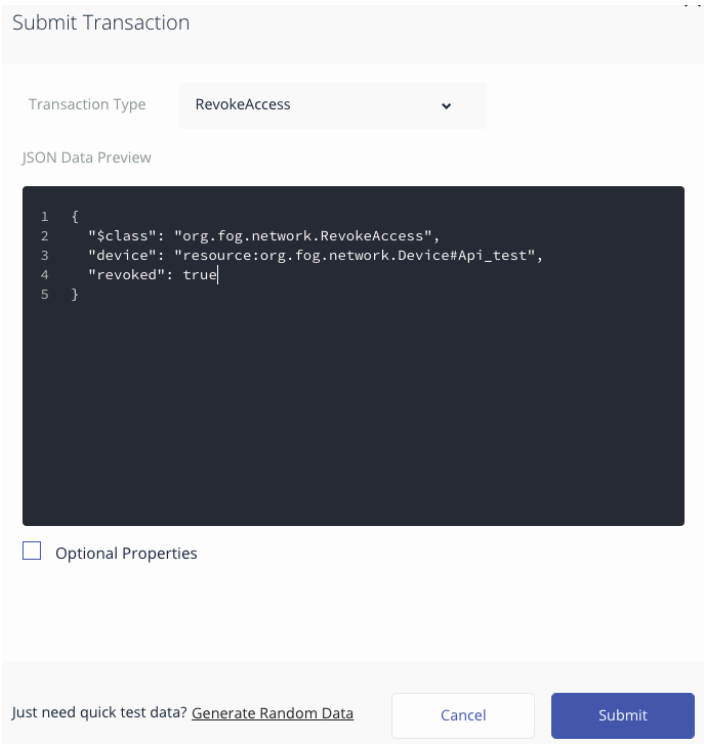
```
curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' -d '{ "$class": "org.fog.network.Device", "deviceID": "test_api", "identifier": "46548ec760ecf7795dd5bfbaf425280f9538149e676e9385f7c3674abc20ccd6", "metadata": "termometras", "revoked": false }' 'http://localhost:3000/api/Device'
```

Galinio įrenginio autentifikavimo metu, žinučių apdorojimo serveris kreipiasi į REST API serverį, naudodamas GET tipo užklausą, kurioje nurodoma galinio įrenginio identifikatoriaus reikšmė. Gavus atsakymą iš REST API serverio nusprendžiama ar žinutė yra sėkmingai apdorojama ir įrenginys yra autentifikuotas. Kitu atveju įrenginys išsiuntęs žinutę yra neužsiregistravęs sistemoje, šiuo atveju žinutė yra atmetama. Galinio įrenginio paieška pateikiant identifikatoriaus reikšmę demonstruojama *Composer* REST serverio grafinėje vartotojo sąsajoje, GET užklausos siuntimas ir atsakymas pavaizduotas 3.5 pav.



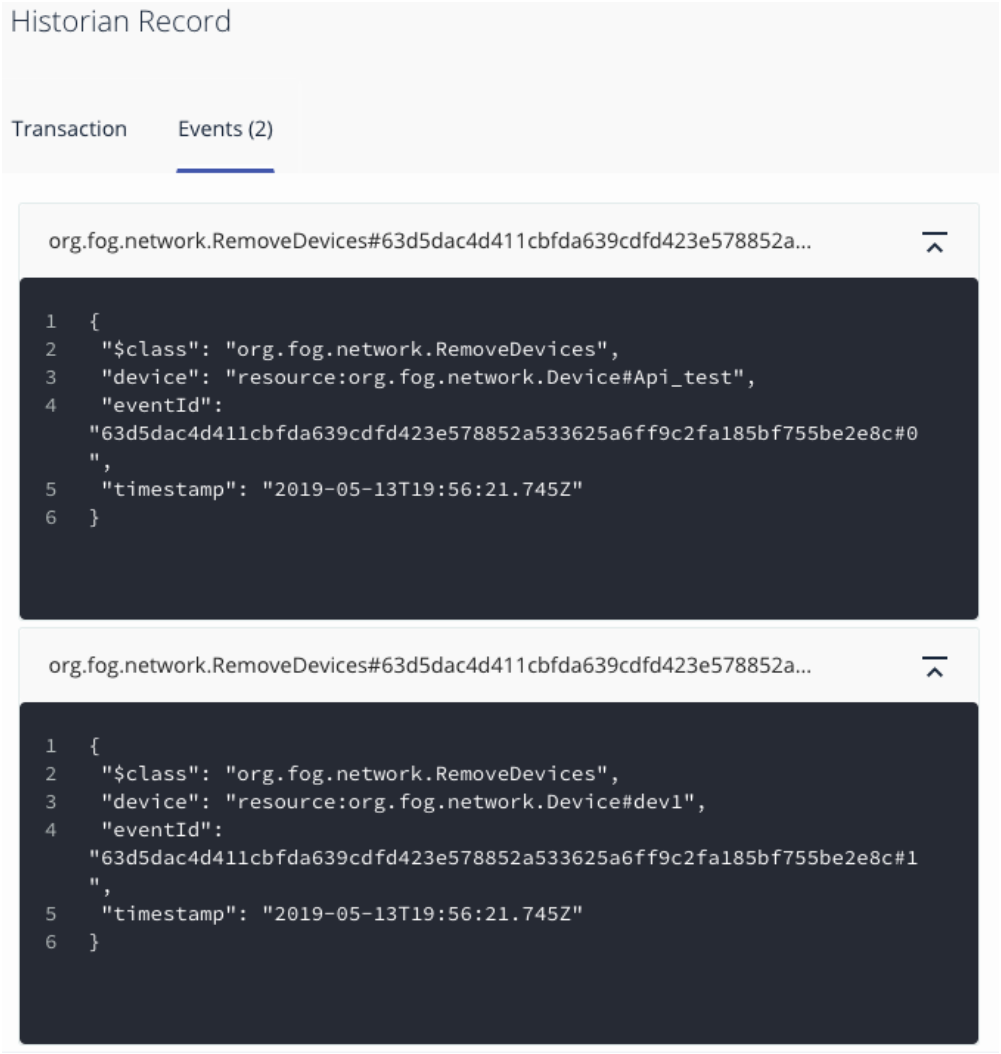
3.5 pav. Galinio įrenginio paieška naudojant identifikatorių

Įrenginių narystės pašalinimui iš blokų grandinės sistemos, gali būti naudojami tiesioginiai DELETE API kreipiniai arba gali būti naudojamos blokų grandinės transakcijos, kurios pakeičia įrenginio narystės būseną į negaliojančią (angl. revoked). Visi šią žymą turintys galiniai įrenginiai gali būti pašalinti iš blokų grandinės naudojant papildomą transakciją *RemoveRevokedDevices*. Įrenginio ypatybių pakeitimas gali būti atliekamas naudojant API arba grafinę vartotojo sąsają, įrenginio ypatybių keitimas vartotojo sąsajoje pavaizduotas 3.6 pav.



3.6 pav. Įrenginio ypatybių keitimas vartotojo sąsajoje

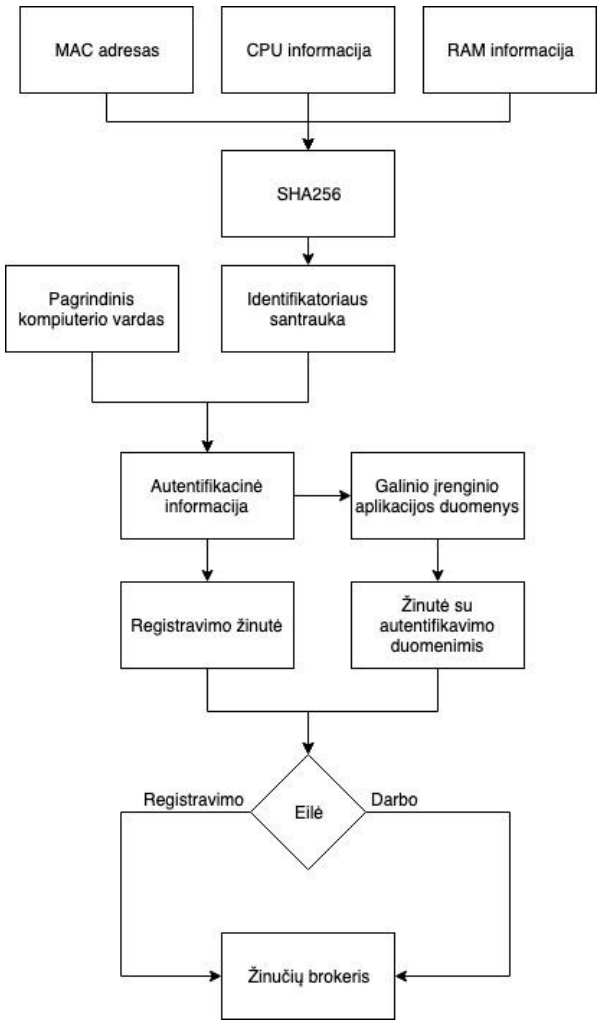
Galinių įrenginių narystės atšaukimo funkcijos ir atšaukiamų dalyvių paieškos užklausos leidžia išplėsti sistemos autonomiškumą ir pritaikius papildomą logiką žinučių apdorojimo serveriuose galima palengvinti infrastruktūros administravimo darbus. Šios funkcijos gali būti vykdomos nustatytu paros metu ar panaudojamos rankiniu būdu sistemos administratoriaus. Funkcijų veikimo demonstravimui, įrenginiai *Api_test* ir *dev1* buvo pažymėti narystės nutraukimui. Įvykdžius transakciją pašalinti visus atšauktus prietaisus, naudojama užklausa rasti visus įrenginius kurie turi atšaukimo žymą, rasti įrenginiai yra pašalinami iš sistemos. Kiekviena transakcija, įvykdyta blokų grandinėje, turi priskirtą transakcijos *ID*, audito tikslais galima matyti, kokie veiksmai blokų grandinėje buvo įvykdyti panaudojus transakciją, 3.7 pav. matomos operacijos kurios buvo įvykdytos blokų grandinėje panaudojus transakciją panaikinti visus atšauktus įrenginius.



3.7 pav. Blokų grandinių transakcijų audito įrašai

Prototipe, prietaiso identifikatoriaus generavimui ir AMQP žinučių siuntimui (angl. publish) naudojama *Python* programavimo kalba parašyti scenarijai, naudojantys standartines sistemines ir *aioamqp* bibliotekas. MAC adreso informacija gaunama nurodžius pasirinktą interneto plokštės vardą. Bendra prietaiso procesoriaus informacija surenkama panaudojant *platform* biblioteką. RAM informacijai surinkti naudojama *psutil* biblioteka. Pagrindinio kompiuterio vardo gavimui naudoja *socket* biblioteka. Sistemos prototipo kūrimo metu PUF funkcijų naudojimas nėra įgyvendinamas, kaip alternatyva naudojamas įrenginio MAC adresas, RAM ir CPU informacija. Galinio įrenginio identifikatoriaus vienkryptei funkcijai generuoti naudojama *hashlib* biblioteka ir SHA256 algoritmas.

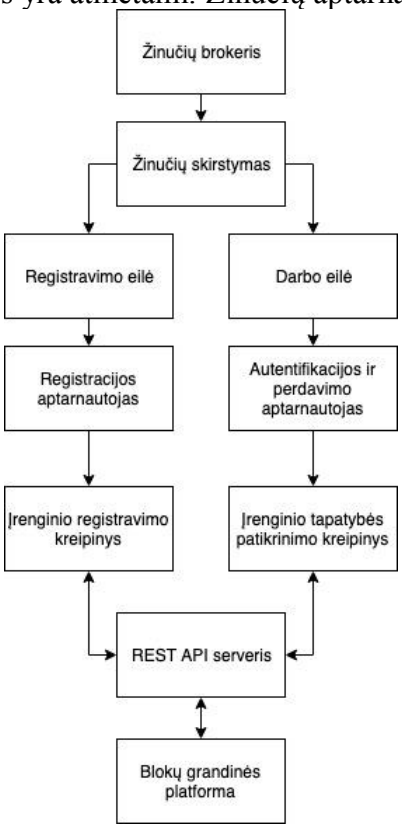
AMQP žinutės siunčiamos naudojant *aioamqp* biblioteką, žinutės siunčiamos į *RabbitMQ* serverį, perduodant žinutę kurią sudaro pagrindinis galinio įrenginio kompiuterio vardas (angl. hostname) ir galinio įrenginio identifikatoriaus kriptografinė santrauka, šios reikšmės atskirtos kabliataškiu. Įrenginio registracija ir autentifikavimas atliekamas dviem skirtingais žinučių siuntimo scenarijais. Įrenginio registravimui naudojamas scenarijus, kuriame naudojama identifikatoriaus generavimo funkcija, tuomet galinio įrenginio identifikatoriaus ir pagrindinis kompiuterio vardas siunčiami į *RabbitMQ* brokerio registravimo eilę. Galinio įrenginio autentifikavimui perduodant duomenis į ūko kompiuterijos sluoksnį naudojamas atskiras *Python* scenarijus, kurio vykdymo metu taip pat naudojama identifikatoriaus generavimo funkcija. Autentifikavimo procesas yra integruojamas į įprastą galinių įrenginių duomenų perdavimo eigą, šios žinutės siunčiamos į *RabbitMQ* brokerio darbo eilę. Žinučių perdavimo ir identifikatoriaus generavimo eiga pavaizduota 3.8 pav.



3.8 pav. Žinučių perdavimo ir identifikatoriaus generavimo eiga

Žinutės, išsiųstos į *RabbitMQ* serverį, yra persiunčiamos ir apdorojamos žinučių doroklių (angl. consumer). Žinučių doroklės prenumeruoja ir gauna žinutes pagal eilės pavadinimą iš žinučių brokerių, doroklės taip pat realizuojamos naudojant Python scenarijus. Registravimo ir darbo eilių aptarnavimui kuriami du atskiri scenarijai. Registravimo aptarnavimo scenarijus, laukia žinučių ir žinučių brokerio, kurios siunčiamos į registravimo eilę, kada žinutė yra gaunama, registravimo apdorojimo scenarijus, gautą žinutę apdoroja. Registravimo žinutės dorojimo procese, scenarijus gautą informaciją žinutės aplikacijos duomenų lauke interpretuoja atskirdamas pagrindinį galinio įrenginio kompiuterio vardą ir identifikatorių. Naudojant šiuos duomenis yra suformuojama API POST tipo užklausa ir siunčiama REST API serveriui, kuris kreipiasi į blokų grandinę. Blokų

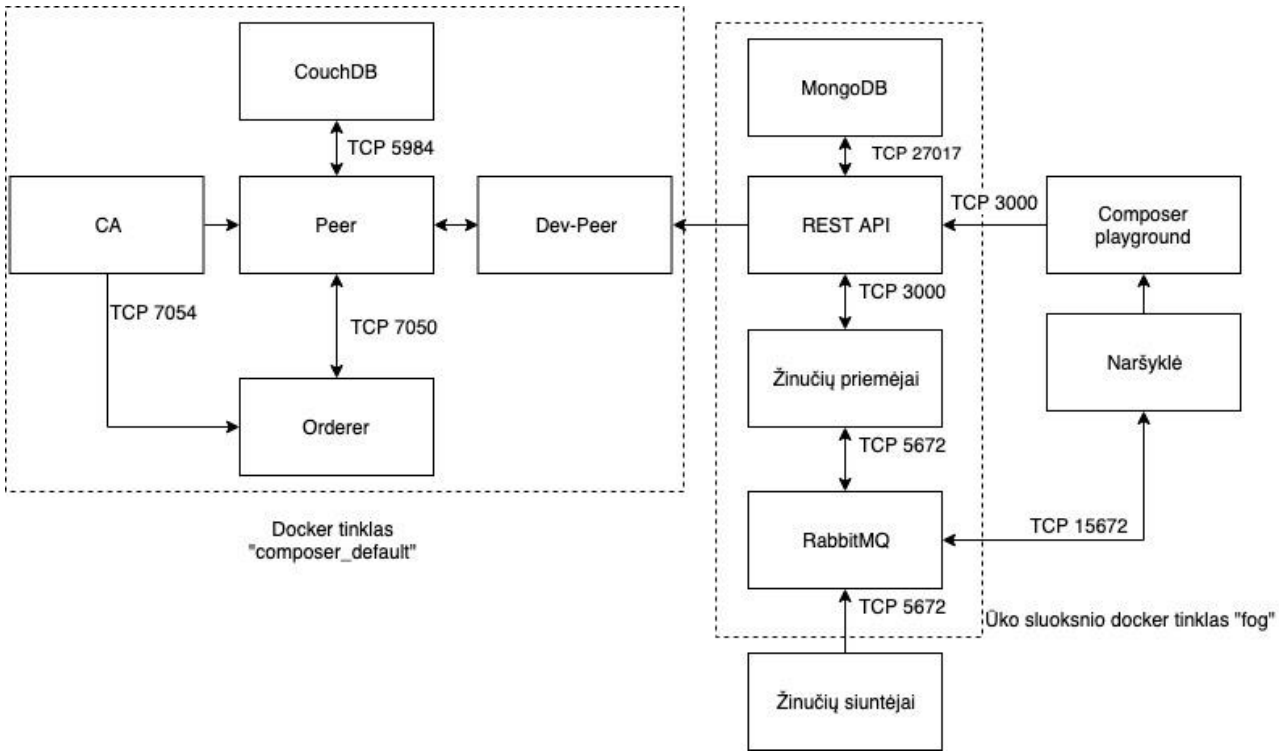
grandinės sistemoje API užklausa yra įvykdoma naudojant grandinių kodo funkcijas naujam dalyviui sukurti. Darbo eilės aptarnavimo scenarijus, priima žinutes iš darbo eilės, gavus žinutę, scenarijus interpretuoja žinutėje pateikiamus duomenis, suformuoja GET tipo API užklausą ir kreipiasi į REST API serverį autentifikuoti galinį įrenginį. REST API serveris įvykdo gautą užklausą ir kreipiasi į blokų grandinę surasti įrenginį su nurodytu identifikatoriumi. Jeigu sistemoje yra sukurtas įrenginio įrašas, kurio identifikatoriaus reikšmė atitinka pateiktą reikšmę, įrenginio autentifikavimas atliktas sėkmingai ir žinutė yra apdorojama, neatitikus įrenginio autentifikacinei informacijai žinutė ir jos duomenys yra atmetami. Žinučių aptarnavimo eiga pavaizduota 3.x pav.



3.9 pav. Žinučių aptarnavimo procesai

Sukurtas blokų grandinėmis grįstas galinių įrenginių autentifikavimo metodas ūko kompiuterijoje prototipas leidžia toliau tobulinti ir plėsti sistemos funkcionalumą, blokų grandinės veikimo logika veikia *Hyperledger Fabric* sistemoje *Composer* pagalba. Žinučių siuntimas ir identifikatoriaus generavimas atliekamas Python scenarijais ir AMQP bibliotekomis skirtomis *Python* scenarijų programavimo kalbai. Visi ūko sluoksnio serveriai ir daiktus imituojantys konteineriai sukuriami ir valdomi *Docker* ir *docker-compose* konteinerių valdymo ir virtualizavimo sistemomis. Schemoje pateikiamoje 3.10 pav. vaizduojamas sistemos komponentų veikimas *Docker* konteineriuose. Konteinerių valdymui yra naudojamos *Docker* sistema ir *Docker*-compose įrankis konteinerių loginiam sujungimui. Šių programų pagalba, galima valdyti atskirus konteinerius ar jų visumą, sustabdyti visos sistemos virtualizaciją ir toliau tęsti darbą. Imituoti galinių prietaisų ir ūko sluoksnio atskyrimui yra naudojami du virtualūs konteinerių tinklai, šių tinklų ribos pavaizduotos žemiau esančiame paveikslėlyje. Tinkle *fog* veikia *RabbitMQ* žinučių brokerio konteineris, kuris priima siunčiamas žinutes iš galinių įrenginių ir žinutes paskirsto į žinučių eiles, taip pat konteineryje veikia *RabbitMQ* administracinė vartotojo sąsaja, kuria galima pasiekti naudojantis interneto naršykle. Žinučių priėmėjų konteineriuose veikia *Python* scenarijų programavimo kalba suprogramuoti žinučių priėmimo scenarijai, kurie aptarnauja darbo ir registravimo eiles. Žinučių aptarnavimo scenarijai pagal aptarnaujamos eilės tipą siunčia užklausas į REST API serverio konteinerį. REST API serveris

ir jo funkcijos yra sugeneruojamos naudojant *Hyperledger Composer* įrankį, REST API serveris naudojamas atlikti skaitymo ir rašymo operacijoms blokų grandinėje. MongoDB konteineryje veikia *MongoDB* duomenų bazė, ši duomenų bazė veikia kaip išliekamosios atminties saugykla, nutrūkus REST API serverio darbui yra išsaugoma informacija, kurią naudoja REST API serveris. *Docker* tinkle *composer_default* veikia blokų grandinių sistema *Hyperledger Fabric*, blokų grandinių sistema ir jos tinklas yra atskirtas nuo kitų ūko kompiuterijos konteinerių ir susisiekimas su sistema galimas naudojant tik REST API arba *Composer playground* grafinę vartotojo sąsaja, taip didinamas blokų grandinių sistemos atsparumas atakoms iš vidaus ir iš išorės. Blokų grandinių sistemoje turi veikti mažiausiai penki konteineriai, norint užtikrinti sistemos funkcionalumą. *CA* konteineriai atlieka sertifikatų kūrimo ir išdavimo funkciją kitiems blokų grandinių sistemos serveriams. *Orderer* konteineriai atlieka darbų skirstymo funkciją, gautos transakcijų ar skaitymo operacijų užklausos yra perduodamos atitinkamiems *Peer* konteineriams. *Dev-peer* konteineriai yra automatiškai sukuriami transakcijų metu ir atlieka grandinės kodo vykdymą ir patvirtinimą, vėliau *peer* konteineriams perduodamas nurodymų rinkinys konteineriams tolimesnėms operacijoms atlikti. *Peer* konteineriuose saugoma pagrindinės knygos kopija, atliekamos operacijos blokų grandinėje ir vykdomas duomenų skaitymas ir rašymas į duomenų būsenos duomenų bazę. *CouchDB* konteineryje saugoma duomenų būsenos duomenų bazė, kurią naudoja *peer* tipo konteineriai.



3.10 pav. *Docker* tinklo infrastruktūra

3.2. Sprendimo prototipo realizavimo išvados

Remiantis sistemos projektavimo metu apibrėžtomis gairėmis, sukurtas galinių įrenginių autentifikavimo prototipas ūko kompiuterijoje, naudojant blokų grandinių technologijas. Prototipui kurti pasitelktos virtualizacijos priemonės sukurti ūko kompiuterijos sluoksnį bei užtikrinti greitą prototipo vystymą. Autentifikavimo sistemos prototipo kūrimui pasirinktos atvirojo kodo sistemos ir įrankiai, taip pat, sukurtas prototipas gali būti perkeltas į debesijos ar virtualizacijos platformas ir gali būti naudojamas tolimesniam funkcionalumo plėtojimui. Sistemos architektūra yra modulinė ir pilnai

palaiko sistemos plečiamumo galimybes pagal poreikį. Sukurtas sistemos prototipas užtikrina sistemos funkcijų plečiamumo galimybes, sistemos architektūra leidžia ūko sluoksnį papildyti naujais specializuotais serveriais ar konteineriais. Žinučių apdorojimo metu, gauta informacija iš galinių įrenginių gali būti perduoda agregavimo ar kito pobūdžio serveriams, sistemoms ar aplikacijoms atlikti tolimesnes operacijas su gautais duomenimis. REST API serverio panaudojimas sistemoje leidžia atlikti operacijas blokų grandinėje, todėl nauji sistemos komponentai gali naudotis apibrėžtais API kreipiniais papildomam funkcionalumui užtikrinti. Prototipe įgyvendinta tik AMQP žinučių priėmimo galimybė *RabbitMQ* brokeryje, tačiau šis brokeris taip pat gali priimti žinutes siunčiamas MQTT ir STOMP protokolais. Sukurtas galinių įrenginių autentifikavimo metodo prototipas ir sistema nėra adaptyvi galinių įrenginių atžvilgiu, dėl to reikalingas rankinis galinių įrenginių konfigūravimas.

Sukurtas sistemos prototipas atlieka galinių duomenų autentifikavimo uždavinį, pasitelkdamas blokų grandinių technologijas, žinučių brokerius ir lengvasvorius IoT prietaisams skirtus protokolus. Sukurtas sistemos prototipas atitinka kriterijus ir išpildo funkcijas apibrėžtus analizės metu. Sukurtas prototipas ir jo greitaveika, kokybiniai ir kiekybiniai rodikliai tiriami sekančiame skyriuje.

4. Blokų grandinėmis grįsto galinių įrenginių autentifikavimo ūko kompiuterijoje metodo tyrimas

Tyrimo dalies subjektas yra sistemos prototipas, tyrimo metu tiriami sistemos prototipo kokybiniai ir kiekybiniai rodikliai. Kokybiniai rodikliai apima patikimumą, integralumą ir auditą. Taip pat bus tiriami šie kiekybiniai rodikliai: greitaveika, infrastruktūra, tinklo pralaidumas.

Tyrimas buvo pradėtas nuo galinių įrenginių autentifikavimo ūko kompiuterijoje prototipo sukūrimo. Parengus sistemą ir atlikus konfigūravimą, sistema gali autentifikuoti galinius įrenginius ūko sluoksnyje autonomiškai, be papildomo sistemos administravimo. Įprastu atveju, galinių įrenginių autentikavimui yra naudojamas vartotojo vardas ir slaptažodis, siunčiant žinutes į žinučių brokerius. Sistemos prototipe išlaikomas toks pat panaudojimo atvejis, galiniame įrenginyje naudojama taikomoji programa ar scenarijus siunčia AMQP žinutę į ūko kompiuterijos sluoksnyje esantį žinučių brokerio serverį, naudojant prisijungimo vardą ir slaptažodį. Prisijungimo vardo atitikmuo yra įrenginio pagrindinis kompiuterio vardas, tačiau slaptažodžiui pakeisti naudojamas kas kartą generuojamas identifikatorius, kurį sudaro RAM, CPU ir MAC adreso sumos vienkryptė funkcija.

4.1. Kokybinių rodiklių tyrimas ir rezultatai

Patikimumas: sistemos prototipas sprendžia duomenų patikimumo problemą, duomenys gali būti siunčiami iš nepatikimų šaltinių tačiau, šie duomenys bus įrašomi į blokų grandinę, todėl bet kuriuo metu galima peržiūrėti blokų grandinės įrašus ir atsekti, kurie prietaisai šiuos duomenis pateikė. Taip pat, kuomet sistemoje veikia atskiri organizaciniai vienetai, blokų grandinės įrašai yra replikuojami visų organizacijų blokų grandinėse, todėl viena organizacija negalės suklastoti duomenų ar kitaip manipuliuoti duomenimis.

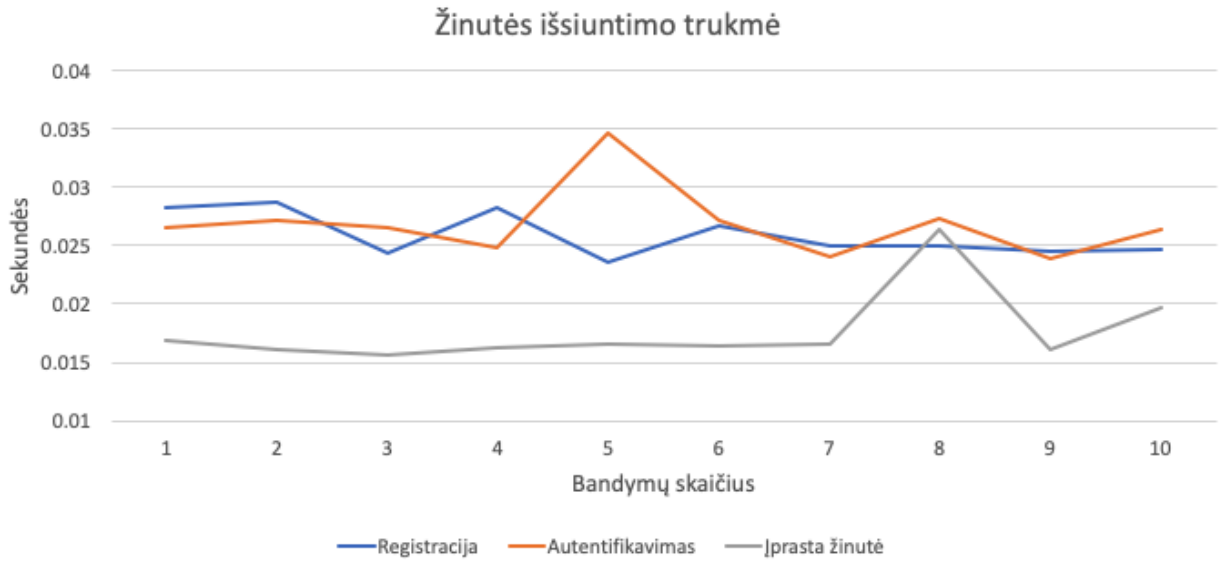
Integralumas: duomenų integralumas sistemos prototipe yra sprendžiamas remiantis blokų grandinių veikimo principu. Duomenys yra neprieštaraujantys, kadangi kiekvienas įrašas yra įrašomas blokų grandinėje skirtingose organizaciniuose vienetuose. Duomenys yra užbaigti, kadangi įrašius įrašą blokų grandinėje įrašo duomenų manipuliuoti neįmanoma. Kiekvienas blokų grandinėje esantis įrašas yra patikrinamas bendradarbiaujančių serverių, kuriuose įrašai yra kaupiami. Duomenys pasižymi pertekliškumu, kadangi blokų grandinės įrašai yra replikuojami tarpusavyje susijusiuose, bet nepriklausomuose serveriuose.

Auditas: sukurtame sistemos prototipe galimas įvairaus lygio audito įgalinimas. Numatytuoju atveju, audito funkciją atlieka blokų grandinėje sąveikaujantys prietaisai. Kai galinis įrenginys prisijungia prie ūko sluoksnio, šis veiksmas įrašomas blokų grandinėje. Taip pat, yra galimybė audituoti žinučių siuntimo atvejus, kuomet galime matyti, kuris prietaisas kokiu metu siuntė žinutes į ūko sluoksnį ir kuris ūko sluoksnio serveris perdavė žinutes debesų sluoksniui. Tačiau minėtas funkcionalumas duomenims perduoti į debesų sluoksnį nėra įgyvendintas sistemos prototipe.

4.2. Kiekybinių rodiklių tyrimas ir rezultatai

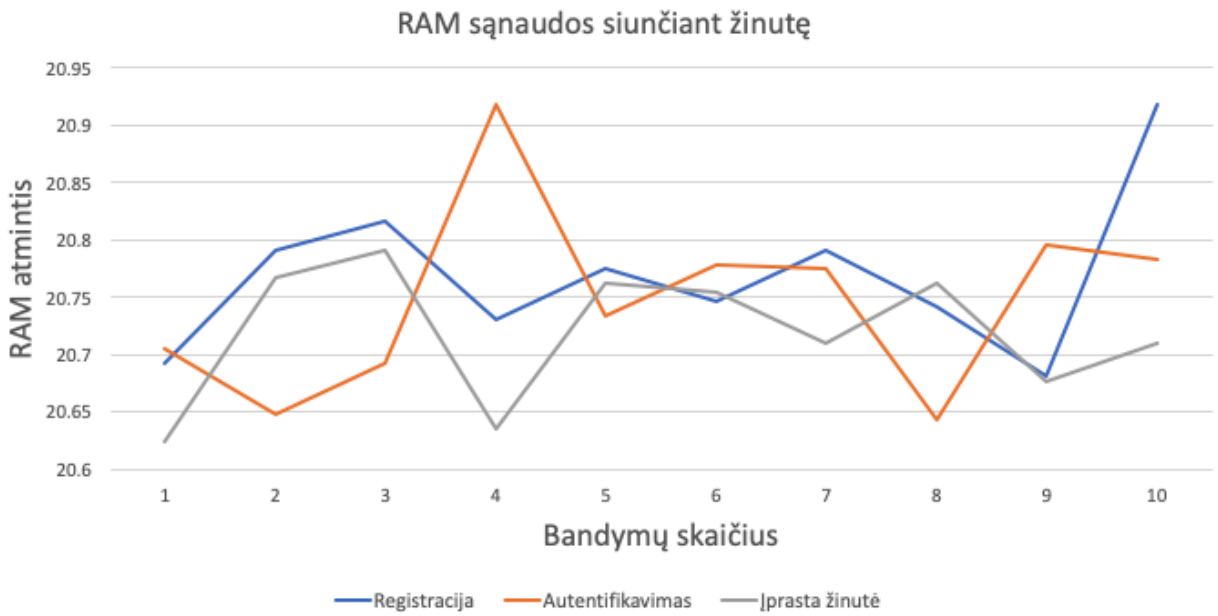
Greitaveika: tiriant galinio įrenginio greitaveiką yra matuojami trys žinučių išsiuntimo ir dorojimo atvejai sistemos prototipe. Kadangi sistemos prototipas yra kuriamas naudojant virtualizaciją vietiniame kompiuteryje, tirti CPU sąnaudų nepavyko, kadangi *Python psutil* biblioteka pateikia tik procentinę proceso CPU apkrovą [33], atlikus bandymus buvo nustatyta, kad scenarijus naudoja 0,0% CPU atminties. Scenarijai ir žinučių brokerių sistema, veikianti prototipo aplinkoje, nenaudoja

programinio kodo ar sistemos optimizavimo algoritmų ir sprendimų. Registravimo žinutės siuntimo metu atliekamas identifikatoriaus generavimas ir šifravimas, šių veiksmų atlikimas naudoja vidutiniškai iki 21 MB RAM atminties. Registravimo žinutės išsiuntimas į vietinį ūko sluoksnyje veikiantį žinučių brokerį, beveik tobulomis sąlygomis užtrunka apie 26 ms ir naudoja 20,76 MB RAM atminties. Kadangi autentifikavimo žinutės siuntimo scenarijus atlieka tuos pačius veiksmus, kaip ir registracijos žinutės siuntimo scenarijus, žinutės dydis nekinta, todėl gauname panašius rezultatus. Autentifikavimo žinutės siuntimas užtrunka taip pat vidutiniškai 26 ms ir naudoja 20,74 MB RAM atminties. Įprastos žinutės siuntimo scenarijuje pašalintos visos identifikatoriaus generavimo funkcijos, dėl to sumažėja siunčiamos žinutės dydis, žinutės perdavimas be autentifikacijos informacijos generavimo ir duomenų užtrunka 18 ms ir naudoja 20,72 MB RAM atminties.



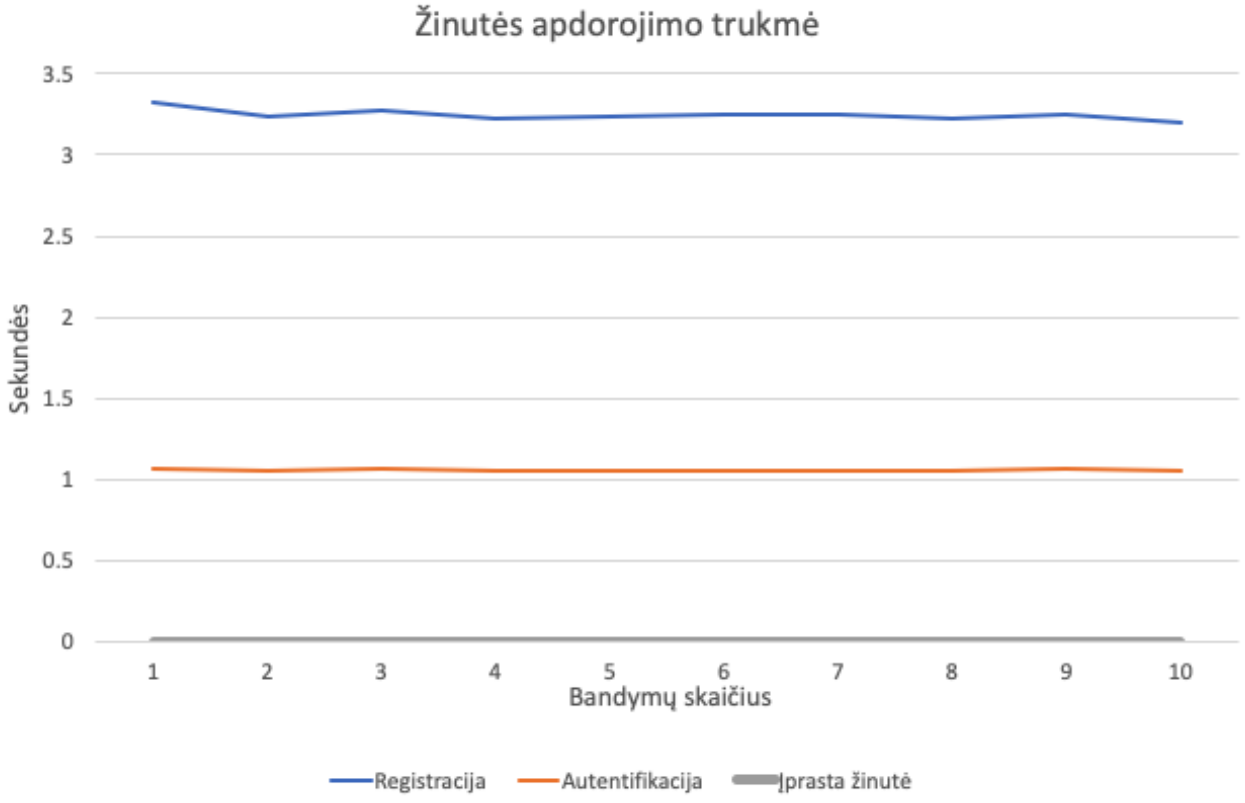
4.1 pav. Žinučių išsiuntimo trukmės

Atlikus žinučių išsiuntimo greitaveikos tyrimus skirtingais atvejais pastebėta, kad galinio įrenginio identifikatoriaus generavimo funkcijos nesukelia ženklių sąnaudų tiek RAM atminties, tiek žinutės perdavimo trukmės sąnaudų atžvilgiu. Identifikatoriaus generavimas vidutiniškai papildomai naudoja 0,04 MB RAM atminties. Tačiau lyginant žinučių išsiuntimo trukmes, galima pastebėti, kad žinutės, nenaudojančios identifikatoriaus generavimo funkcijų, žinutę suformuoja ir išsiunčia 6 ms greičiau, nei scenarijuose, kuriuose naudojamos identifikatoriaus generavimo funkcijos. Kritinėse sistemose toks laiko pokytis gali įtakoti sistemos patikimumą ir neatitikti apibrėžtų standartų duomenų perdavimo spartai. Žinučių siuntimo RAM atminties sąnaudų ir žinutės išsiuntimo trukmės greitaveikos tyrimo rezultatai ir statistikos pateikiama 4.1 pav. ir 4.2 pav.



4.2 pav. RAM atminties sąnaudos žinutės siuntimo metu

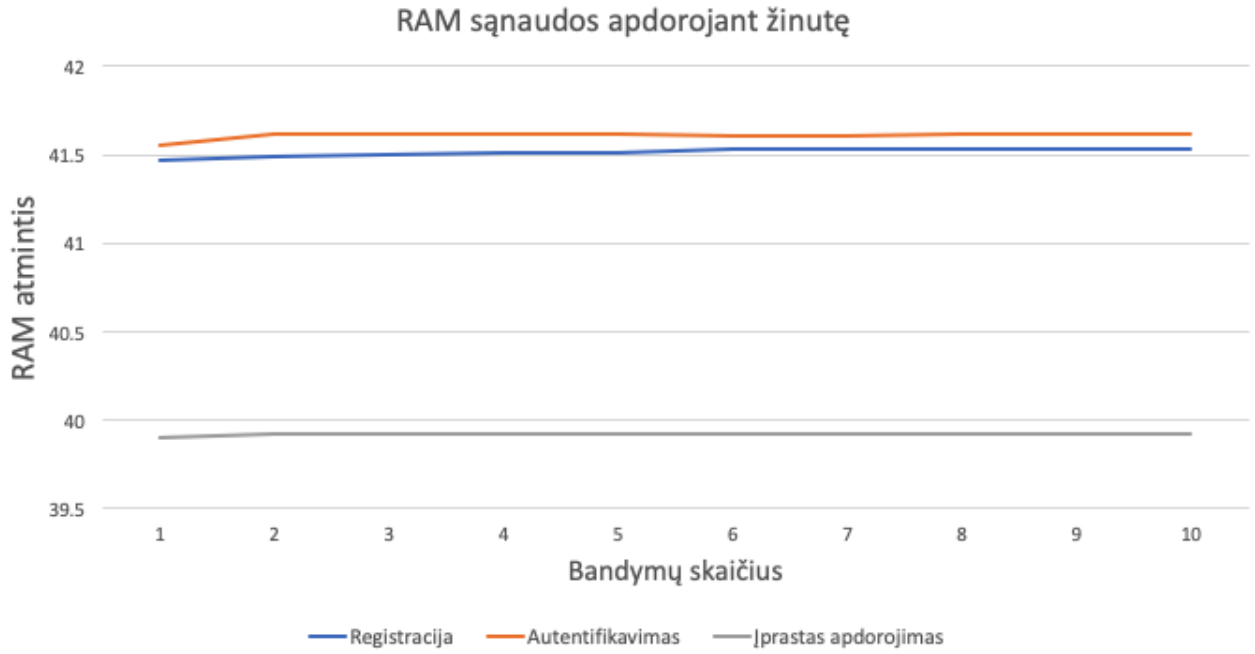
Naujo galinio įrenginio registravimas blokų grandinėje vidutiniškai trunka 3,25 sekundes, kurios metu atliekamos transakcijos blokų grandinėje ir perduodamas atsakymas žinučių dorojimo konteineriams. Tačiau galinio įrenginio autentifikavimas blokų grandinėje trunka vidutiniškai 1,06 sekundę. Įprastų žinučių be autentifikacijos ar registracijos procesų apdorojimas vidutiniškai trunka 0,25 ms.



4.3 pav. Žinučių apdorojimo trukmės

Registravimo žinutės apdorojimo metu, naudojami du kreipiniai į REST API serverį, žinutės apdorojimas vidutiniškai sunaudoja 41,51 MB RAM atminties ūko serverių konteineriuose.

Autentifikavimo žinutės atveju, naudojamas vienas API kreipinys, tačiau vidutinės RAM sąnaudos autentifikavimui atlikti yra 41,61 MB. Įprastu žinutės apdorojimo atveju, kuomet nėra naudojamos blokų grandinės funkcijos, žinutės apdorojimas vidutiniškai sunaudoja 39,92 MB RAM atminties. Žinučių apdorojimo rezultatai ir statistikos pateikiamos 4.3 pav. ir 4.4 pav.



4.4 pav. RAM atminties sąnaudos žinučių apdorojimo metu

Atlikus žinučių apdorojimo greitaveikos tyrimus, pastebėta, kad sukurtas autentifikavimo metodas taip pat nesukelia žymių papildomų RAM atminties sąnaudų žinučių apdorojimo metu. Tačiau palyginus žinutės apdorojimo trukmes, pastebėta, kad registracija gali trukti iki 3,5 sekundės, o autentifikacija trunka apie 1 sekundę. Lyginant šias trukmes su įprastiniu žinutės apdorojimo atveju, laiko sąnaudos ženkliai padidėja, tačiau šios trukmės taip pat atitinka ūko kompiuterijos siūlomus greitaveikos kriterijus. Remiantis analizės metu pateiktais galinių įrenginių duomenų apdorojimo kriterijais, galima teigti, kad sukurtas sistemos prototipas siūlo standartus atitinkančius greitaveikos rodiklius.

Infrastruktūra: sistemos prototipe naudojami elementai yra atsparūs trikdžiams ar vieno iš komponentų trumpalaikiam sunaikinimui ar praradimui. Ūko sluoksnyje veikiantis REST API serveris gali būti sukonfigūruotas keliuose serveriuose, o kreipimasis į API gali būti pasiekiamas per apkrovos skirstytoją, po kuriuo gali būti balansuojami N serverių. Žinučių gavėjai gali veikti bet kuriame ūko sluoksnio serveryje, tuomet žinučių gavėjai kreipiasi į artimiausią REST API serverį ūko sluoksnyje, geografinio atstumo nustatymą sprendžia apkrovos skirstytojas. Todėl laikinai ar visai sutrikus REST API serveriams žinutės bus saugomos *RabbitMQ* eilėse iki tol kol pradės veikti bent vienas REST API serveris. REST API serveriai naudoja *MongoDB* arba *CouchDB* duomenų bazes spartinančiajai ir išliekamajai informacijai kaupti, todėl sutrikus veiklai ir serveriams vėl pradėjus funkcionuoti bus išlaikytas duomenų integralumas. Sistemos prototipas yra įgyvendintas naudojant vieną vykdymo ir duomenų serverį, tačiau vystomi komerciniai sprendimai turėtų naudoti bent tris vykdymo ir duomenų serverius blokų grandinės pagrindinės knygos kopijų saugojimui ir atlikti grandinių kodo vykdymą.

Tinklo pralaidumas: Įprastu atveju, kuomet galiniai įrenginiai siunčia duomenis tiesiogiai į debesų paslaugas, tinklo pralaidumas nėra našus, kadangi kiekvienas prietaisas siunčia žinutes tiesiogiai į geografiškai nutolusius serverius duomenų centruose. Ūko kompiuterijos atveju duomenys pirmiausia yra siunčiami į ūko sluoksnyje esančius serverius, kurie yra kiek įmanoma arčiau galinių įrenginių, todėl duomenys pirmiausia yra apdorojami ūko sluoksnyje. Ūko sluoksnyje veikiantys serveriai interpretuoja gautus duomenis ir apsprendžia ar duomenis reikia skubiai perduoti į debesis ar duomenis gali būti kaupiami, agreguojami ir persiunčiami į debesis po tam tikro laiko tarpo, kuomet tinklo apkrova yra mažesnė. Todėl tinklo pralaidumas, naudojant ūko kompiuteriją, yra didinamas, atsižvelgiant į nustatytas taisykles ir konfigūracijas.

4.3. Tyrimo išvados

Atlikus sistemos prototipo tyrimą galima įžvelgti, kad sukurtas prototipas atitinka galinių įrenginių duomenų perdavimo standartus, apibrėžtus analitinėje dalyje. Kokybiniai sistemos prototipo rodikliai įrodo, kad sistema atitinka pagrindinius saugumo standartus ir atsižvelgia į patikimumo, integralumo ir audito reikalavimus. Ištyrus kiekybinius sistemos prototipo rodiklius pastebėta, kad sistemos greitaveika atitinka galinių įrenginių duomenų perdavimo spartos standartus apibrėžtus analitinėje dalyje. Taip pat, sukurtas metodas nesukelia didelių, papildomų resursų sąnaudų galiniuose įrenginiuose. Sukurtos sistemos infrastruktūra yra modulinė, todėl užtikrinamos sistemos plėtojimo galimybės. Taip pat, naudojant blokų grandinėmis grįstą galinių įrenginių autentifikavimo metodą, didinamas ir optimizuojamas sistemos tinklo pralaidumas.

Išvados

Šio darbo tikslas buvo sukurti blokų grandinėmis grįstą galinių įrenginių autentifikavimo ūko kompiuterijoje metodą. Spartus IoT prietaisų populiarumo augimas sukelia naujus iššūkius sprendžiant informacinių sistemų saugumo ir infrastruktūrinius uždavinius. Darbo analizės metu buvo iškelti tikslai – išsiaiškinti panaudos atvejus ir ūko kompiuterijos architektūros galimybes bei pranašumus, juos lyginant su tradiciniais debesų kompiuterijos sprendimais. Pagrindinė darbo problema – galinių įrenginių autentifikacija. Esami sprendimai ir metodai autentifikuoti galinius įrenginius nėra našūs ir tinkamai panaudojami galinių įrenginių autentifikavimui, todėl reikalinga nauja sistema, kuri saugiai ir efektyviai autentifikuoja galinius įrenginius ūko kompiuterijoje, panaudojant blokų grandinių technologijas.

Apibrėžus kuriamos sistemos reikalavimus ir gaires, galinių įrenginių autentifikavimo metodas ir sistema buvo projektuojama, atsižvelgiant į ūko kompiuterijos architektūrą ir sistemos plečiamumą bei suderinamumą su plačiu galinių įrenginių spektru. Sistemos prototipo kūrimo metu naudojami sistemos komponentai ir veikimo principai, apibrėžti projektavimo dalyje. Autentifikacijos metodo realizavimo metu pateikiamas sistemos prototipo kūrimo procesas ir techninės detalės.

Tyrimo metu sistemos prototipas yra lyginamas su esamais autentifikavimo sprendimais ir tiriami sistemos kiekybiniai bei kokybiniai rodikliai. Atliktas sistemos prototipo tyrimas rodo, kad sukurtas blokų grandinėmis grįstas galinių įrenginių autentifikavimo metodas ūko kompiuterijoje atitinka visus apibrėžtus reikalavimus ir standartus autentifikavimo uždaviniui spręsti.

1. Atlikus ūko kompiuterijos architektūros analizę buvo išsiaiškinti standartai, keliami ūko kompiuterijos architektūriniam modeliui, trūkumai, pranašumai bei sprendžiamos problemos;
2. analizuojant esamus galinių įrenginių autentifikavimo sprendimus literatūroje, pastebėta, kad ūko kompiuterijoje naudojami tradiciniai autentifikavimo metodai, kaip LDAP, VRI infrastruktūra ar statinės vartotojo vardo ir slaptažodžio reikšmės;
3. blokų grandinių technologijų ir esamų sistemų analizės metu pastebėta, kad šiuo metu nėra plačiai paplitusių komercinių, specializuotų sistemų spręsti galinių įrenginių autentifikacijai užtikrinti, tačiau esamos sistemos gali būti naudojamos eksperimentams ir prototipams kurti;
4. remiantis literatūroje apibrėžtais galinių įrenginių duomenų perdavimo spartos rodikliais ir techniniais apribojimais, daroma prielaida, kad duomenų perdavimui reikia naudoti specializuotus lengvasvorius IoT protokolus;
5. projektuojant ir kuriant sistemos prototipą pastebėta, kad tradicinis blokų grandinių technologijos panaudojimas nėra našus skaitymo ir rašymo operacijų atžvilgiu, tačiau naudojant spartinančią atmintį, skaitymo ir rašymo operacijos blokų grandinėje tampa priimtinos naudoti įvairiose sistemose;
6. sistemos projektavimo metu sukurtas galinių įrenginių autentifikavimo sistemos modelis ūko kompiuterijoje, naudojantis blokų grandinių technologijas, projektavimo metu apibrėžtos sistemos gairės projekto realizavimui;
7. sukūrus sistemos prototipą pastebėta, jog sistemos plečiamumo ir integravimo galimybės gali būti labai plačios ir neapsiriboti tik galinių įrenginių autentifikavimo uždaviniui spręsti;
8. atlikus sistemos greitaveikos tyrimą, nustatyta, kad galinių įrenginių autentifikavimas, naudojant blokų grandines, atitinka ūko kompiuterijos laiko standartus;
9. sukurtas autentifikavimo metodas duomenų perdavimo atžvilgiu nesudaro papildomos apkrovos galiniuose įrenginiuose.

Plėtojant blokų grandinių taikymą galinių įrenginių autentifikavimui ūko kompiuterijoje būtų reikšminga atlikti tyrimus, naudojant fizinę ūko kompiuterijos platformą, kadangi tyrimo rezultatai yra eksperimentiniai ir gauti esant beveik idealioms tinklo sąlygoms. Dėl šios priežasties, praktiniai tyrimai ir rezultatai, esant nepalankioms ar sunkioms tinklo sąlygoms, gali kardinaliai skirtis nuo eksperimentų. Dėl gausaus IoT įrenginių kiekio informacinėse sistemose, siekiamas visiškai pilnas galinių įrenginių administravimo autonomiškumas nėra lengvai užtikrinamas, dėl daug nenumatytų faktorių ir žmogiškųjų klaidų.

Literatūros sąrašas

[1] D. Yu, Y. Jin, Y. Zhang ir X. Zheng, „A survey on security issues in services communication of Microservices-enabled fog applications," *Concurrency and Computation: Practice and Experience,* 2018.

[2] F. Bonomi, R. Milito, J. Zhu ir S. Addepalli, „Fog Computing and Its Role in the Internet of Things," įtraukta *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, New York, ACM, 2012, pp. 13-16.

[3] C. Kolias, G. Kambourakis, A. Stavrou ir J. Voas, „DDoS in the IoT: Mirai and Other Botnets," *Computer,* t. 50, nr. 7, pp. 80-84, 2017.

[4] M.-Q. Tran, D. T. Nguyen, V. A. Le, D. H. Nguyen ir T. V. Pham, „Task Placement on Fog Computing Made Efficient for IoT Application Provision," *Wireless Communications and Mobile Computing,* t. 2019, p. 17, 2019.

[5] Cisco, „Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are," 2015.

[6] S. Nakamoto, „Bitcoin: A Peer-to-Peer Electronic Cash System," 2009. [Tinkle]. Available: : https://bitcoin.org/bitcoin.pdf. [žiūrėta 2019-01-19].

[7] Z. Zibin, S. Xie, H.-N. Dai, X. Chen ir H. Wang, „Blockchain challenges and opportunities: a survey," *International Journal of Web and Grid Services,* t. 14, nr. 4, pp. 352-375, 2018.

[8] C. Cachin ir M. Vukolic, „Blockchain Consensus Protocols in the Wild," *arXiv preprint,* 2017.

[9] X. Li, P. Jiang, T. Chen, X. Luo ir Q. Wen, „A survey on the security of blockchain systems," *Future Generation Computer Systems,* 2017.

[10] H. Watanabe, S. Fujimura, A. Nakadaira, Y. Miyazaki, A. Akutsu ir J. Kishigami, „Blockchain contract: Securing a blockchain applied to smart contracts," įtraukta *2016 IEEE International Conference on Consumer Electronics (ICCE)*, Las Vegas, 2016.

[11] L. V. D. Horst, K. R. Choo ir N. Le-Khac, „Process Memory Investigation of the Bitcoin Clients Electrum and Bitcoin Core," *IEEE Access,* t. 5, pp. 22385-22398, 2017.

[12] M. Wohrer ir U. Zdun, „Smart contracts: security patterns in the ethereum ecosystem and solidity," įtraukta *2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*, Campobasso, 2018.

[13] Ethereum, „Solidity Documentation," 2019. [Tinkle]. Available: https://solidity.readthedocs.io/en/v0.5.8/. [žiūrėta 2018-12-02].

[14] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukoli, S. W. Cocco ir J. Yellick, „Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains," įtraukta *Thirteenth EuroSys Conference*, Porto, 2018.

[15] A. H. Lone ir R. N. Mir, „Forensic-chain: Blockchain based digital forensics chain of custody with PoC in Hyperledger Composer,“ *Digital Investigation*, t. 28, pp. 44-55, 2019.

[16] M. Samaniego ir R. Deters, „Internet of Smart Things - IoST: Using Blockchain and CLIPS to Make Things Autonomous,“ įtraukta *2017 IEEE International Conference on Cognitive Computing (ICCC)*, Honolulu, 2017.

[17] MultiChain, „MultiChain for Developers,“ 2019. [Tinkle]. Available: https://www.multichain.com/developers/. [žiūrėta 2019-01-15].

[18] N. Naik, „Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP,“ įtraukta *2017 IEEE International Systems Engineering Symposium (ISSE)* , Vienna, 2017.

[19] M. Singh, M. A. Rajan, V. L. Shivraj ir P. Balamuralidhar, „Secure MQTT for Internet of Things (IoT),“ įtraukta *2015 Fifth International Conference on Communication Systems and Network Technologies*, Gwalior, 2015.

[20] N. Glicoric, T. Dimcic, D. Drajic ir S. Krco, „CoAP over SMS: Performance evaluation for machine to machine communication,“ įtraukta *2012 20th Telecommunications Forum (TELFOR)*, Belgrade, 2012.

[21] J. Kreps, „Linkedin,“ 27 April 2014. [Tinkle]. Available: https://engineering.linkedin.com/kafka/benchmarking-apache-kafka-2-million-writes-second-three-cheap-machines. [žiūrėta 2019-05-01].

[22] J. Kreps, N. Narkhede ir R. Jun, „Kafka: a Distributed Messaging System for Log Processing,“ 2011.

[23] M. Rostanski, K. Grochla ir A. Seman, „Evaluation of highly available and fault-tolerant middleware clustered architectures using RabbitMQ,“ įtraukta *2014 Federated Conference on Computer Science and Information Systems*, Warsaw, 2014.

[24] T. A. S. Foundation, „Apache ActiveMQ,“ Apache ActiveMQ, [Tinkle]. Available: https://activemq.apache.org/performance.html. [žiūrėta 2019-04-14].

[25] G. E. Suh ir S. Devadas, „Physical Unclonable Functions for Device Authentication and Secret Key Generation,“ įtraukta *2007 44th ACM/IEEE Design Automation Conference*, San Diego, 2007.

[26] L. Bolotnyy ir G. Robins, „Physically Unclonable Function-Based Security and Privacy in RFID Systems,“ įtraukta *Fifth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom'07)*, White Plains, 2007.

[27] Hyperledger, „Hyperledger,“ 2018. [Tinkle]. Available: https://www.hyperledger.org/wp-content/uploads/2018/10/HL_Whitepaper_Metrics_PDF_V1.01.pdf. [Žiūrėta 5 Vasario 2019].

[28] Pivotal, „RabbitMQ Clustering Guide,“ 2019. [Tinkle]. Available: https://www.rabbitmq.com/clustering.html. [žiūrėta 2019-02-07].

[29] Hyperledger, „Hyperledger: Ledger documentation,“ 2019. [Tinkle]. Available: https://hyperledger-fabric.readthedocs.io/en/release-1.4/ledger.html. [žiūrėta 2019-02-08].

[30] Pivotal, „AMQP 0-9-1 Model Explained,“ 2019. [Tinkle]. Available: https://www.rabbitmq.com/tutorials/amqp-concepts.html. [žiūrėta 2019-03-16].

[31] Hyperledger, „Welcome to Hyperledger Composer,“ 2019. [Tinkle]. Available: https://hyperledger.github.io/composer/v0.19/introduction/introduction. [žiūrėta 2018-12-18].

[32] Hyperledger, „Developer tutorial for creating a Hyperledger Composer solution,“ 2019. [Tinkle]. Available: https://hyperledger.github.io/composer/v0.19/tutorials/developer-tutorial. [žiūrėta 2019-05-15].

[33] G. Rodola, „Psutil documentation,“ 2019. [Tinkle]. Available: https://psutil.readthedocs.io/en/latest/index.html. [žiūrėta 2019-03-15].

Priedai

1 Priedas. Programinis kodas

```
# org.fog.network.cto
/**
 * Galinių įrenginių autentifikavimo sistema
 */

namespace org.fog.network

// Dalyvio objektas, aprašantis saugojamus įrenginio duomenis
participant Device identified by deviceID {
  o String deviceID // Įrenginio pagrindinis kompiuterio vardas
  o String identifier // Įrenginio identifikatorius
  o String metadata // Įrenginio metaduomenys
  o Boolean revoked // true - irenginio narystė atšaukta
}

// Įrenginio prieigos tesių atšaukimo transakcija
transaction RevokeAccess {
  --> Device device // Įrenginys, kurio duomenys bus keičiami transakcijos metu
  o Boolean revoked // Įrenginio narystės būsena
}

//Įrenginio pranešimo funckija
event DeviceNotification {
    --> Device device
}
```

```
//Transakcicja skirta pašalinti visus atšaukiamus įrenginius
transaction RemoveRevokedDevices {
}

// Įvykis skirtas pašalinti įrenginį
event RemoveDevices {
   --> Device device
}
```

```
#lib/logic.js
'use strict';
/**
* Transakcija atnaujina įrenginio narystės būseną
* @param {org.fog.network.RevokeAccess} RevokeAccess - atšaukiama įrenginio narystė
* @transaction
*/
async function updateAccess(revokeAccess) {
   let participantRegistry = await getParticipantRegistry('org.fog.network.Device');
   var factory = getFactory();
   revokeAccess.device.revoked = true;
   await participantRegistry.update(revokeAccess.device);
}

/**
 * Pašalinti visus atšauktus įrenginius
 * @param {org.fog.network.RemoveRevokedDevices} RemoveRevokedDevices - pašalinami įrenginiai kurių narystė yra atšaukta
```

```
 * @transaction
 */
async function removeRevokedDevices(remove) {
    let participantRegistry = await getParticipantRegistry('org.fog.network.Device');
    let results = await query('selectRevokedDevices');

    for (let n = 0; n < results.length; n++) {
        let revokedDevice = results[n];

        // išsiųsti pranešimą apie įrenginio pašalinimą
        let removeDevices = getFactory().newEvent('org.fog.network','RemoveDevices');
        removeDevices.device = revokedDevice;
        emit(removeDevices);
        await participantRegistry.remove(revokedDevice);
    }
}
```

```
#permissions.acl
rule NetworkAdminUser {
    description: "Grant business network administrators full access to user resources"
    participant: "org.hyperledger.composer.system.NetworkAdmin"
    operation: ALL
    resource: "org.fog.network.*"
    action: ALLOW
}
```

```
rule NetworkAdminSystem {
    description: "Grant business network administrators full access to system resources"
    participant: "org.hyperledger.composer.system.NetworkAdmin"
    operation: ALL
    resource: "org.hyperledger.composer.system.**"
    action: ALLOW
}
```

```
#queries.qry
/*
* Blokų grandinės užklausos
*/

query selectDevices {
  description: "Rasti visus įrenginius"
  statement:
      SELECT org.fog.network.Device
}

query selectDevicesById {
  description: "Rasti įrenginį pagal ID"
  statement:
      SELECT org.fog.network.Device
          WHERE (deviceID==_$id)
}

query selectDevicesByIdentifier {
```

```
  description: "Rasti įrenginius pagal identifikatorių"
  statement:
      SELECT org.fog.network.Device
          WHERE (identifier == _$identifier)
}

query selectRevokedDevices {
  description: "Rasti atšaukiamus įrenginius"
  statement:
      SELECT  org.fog.network.Device
          WHERE (revoked == true)
}
```

```
#pub_register.py
#!/usr/bin/env python

import asyncio
import aioamqp
import hashlib
import sys
from psutil import virtual_memory
import netifaces
import platform
import socket
import os

# Funkcija gauti prietaiso MAC adresa
def getMac():
```

```
    mac = netifaces.ifaddresses('en1')[netifaces.AF_LINK]
    return mac


#Funkcija generuoti santraukai
def encrypt_string(hash_string):
    sha_signature = \
        hashlib.sha256(hash_string.encode()).hexdigest()
    return sha_signature


#Funkcija gauti procesoriaus informacijai
def getCPU():
     cpu = platform.processor()
     return cpu


#Gaunama RAM informacija
getRAM = virtual_memory().total


#Gaunamas prietaiso domeno vardas
hostname = socket.gethostname()


#Apjungiama MAC, CPU, RAM
string = str(getMac()) + str(getCPU()) + str(getRAM)


#Kvieciama santraukos funkcija
dev_pass = encrypt_string(string)


print(string)
#### AMQP
```

```
async def new_task():
    try:
        transport, protocol = await aioamqp.connect(
         host='localhost',
         port=5672)
    except aioamqp.AmqpClosedConnection:
        print("closed connections")
        return


    channel = await protocol.channel()


    await channel.queue('register_queue', durable=True)


    message = "dev1" + ";" + dev_pass


    await channel.basic_publish(
        payload=message,
        exchange_name='',
        routing_key='register_queue',
        properties={
            'delivery_mode': 2,
        },
    )
    print(" [x] Isiusta %r" % message,)


    await protocol.close()
    transport.close()
```

```
asyncio.get_event_loop().run_until_complete(new_task())
```

```
#pub_data.py
#!/usr/bin/env python

import asyncio
import aioamqp
import hashlib
import sys
from psutil import virtual_memory
import netifaces
import platform
import socket
import os

# Funkcija gauti prietaiso MAC adresas
def getMac():
    mac = netifaces.ifaddresses('en1')[netifaces.AF_LINK]
    return mac

#Funkcija generuoti santraukai
def encrypt_string(hash_string):
    sha_signature = \
        hashlib.sha256(hash_string.encode()).hexdigest()
    return sha_signature

#Funkcija gauti procesoriaus informacijai
```

```
def getCPU():
    cpu = platform.processor()
    return cpu

#Gaunama RAM informacija
getRAM = virtual_memory().total

#Gaunamas prietaiso domeno vardas
hostname = socket.gethostname()

#Apjungiama MAC, CPU, RAM informacija
string = str(getMac()) + str(getCPU()) + str(getRAM)

#Kvieciama santraukos funkcija
dev_pass = encrypt_string(string)

print(string)
#### AMQP
async def new_task():
   try:
       transport, protocol = await aioamqp.connect(
        host='localhost',
        port=5672)
   except aioamqp.AmqpClosedConnection:
       print("closed connections")
       return

   channel = await protocol.channel()
```

```
    await channel.queue('data_queue', durable=True)


    message = "dev2" + ";" + dev_pass + ";" + "metaduomenys"


    await channel.basic_publish(
        payload=message,
        exchange_name='',
        routing_key='data_queue',
        properties={
            'delivery_mode': 2,
        },
    )
    print(" [x] Isiusta %r" % message,)


    await protocol.close()
    transport.close()


asyncio.get_event_loop().run_until_complete(new_task())
```

```
# con_register.py
#!/usr/bin/env python

import asyncio

import aioamqp

import requests

import sys
```

```
#funckija patikrinti ar prietaisas su tokiu slaptazodziu yra bloku grandineje
def check(password):
    headers = {'Accept': 'application/json'}
    url = "http://localhost:3000/api/queries/selectDevicesByIdentifier?identifier="+password
    r = requests.get(url, headers=headers)
    response = str(r.content)
    return response


async def callback(channel, body, envelope, properties):
    b = body.decode()
    b_split =  b.split(';')
    hostname = b_split[0]
    password = b_split[1]
    await asyncio.sleep(body.count(b'.'))
    headers = {'Accept': 'application/json'}
    payload = {'$class': 'org.fog.network.Device', 'deviceID': hostname, 'identifier': password, 'metadata': " ", 'revoked': 'false'}
    #Tikrinama ar egzistuoja irasas bloku grandineje su nurodytu identifikatoriumi
    if check(password) == "b'[]'":
        r = requests.post("http://localhost:3000/api/Device/", headers=headers, json=payload)
        print ("Irenginys sekmingai uzregistruotas bloku grandineje")
    else:
        print("Irenginius su identifikatoriumi: "+password+" jau registruotas")
    print("Atlikta")
    await channel.basic_client_ack(delivery_tag=envelope.delivery_tag)



async def worker():
```

```
    try:
        transport, protocol = await aioamqp.connect('localhost', 5672)
    except aioamqp.AmqpClosedConnection:
        print("closed connections")
        return



    channel = await protocol.channel()


    await channel.queue(queue_name='register_queue', durable=True)
    await channel.basic_qos(prefetch_count=1, prefetch_size=0, connection_global=False)
    await channel.basic_consume(callback, queue_name='register_queue')



event_loop = asyncio.get_event_loop()
event_loop.run_until_complete(worker())
event_loop.run_forever()
```

```
# con_data.py
#!/usr/bin/env python
import asyncio
import aioamqp
import requests
import sys


#funckija patvirtinti autentifikacinius duomenis
def check(password):
```

```
    headers = {'Accept': 'application/json'}
    url = "http://localhost:3000/api/queries/selectDevicesByIdentifier?identifier="+password
    r = requests.get(url, headers=headers)
    response = str(r.content)
    return response


async def callback(channel, body, envelope, properties):
    b = body.decode()
    b_split =  b.split(';')
    hostname = b_split[0]
    password = b_split[1]
    data = b_split[2]
    await asyncio.sleep(body.count(b'.'))
    #Tikrinama ar egzistuoja irasas bloku grandineje su nurodytu identifikatoriumi
    if check(password) == "b'[]'":
        print ("Prietaisas neregistruotas")
    else:
        print("Prietaisas kurio identifikatorius: "+password+" registruotas bloku grandineje")
    print("Zinute apdorota")
    await channel.basic_client_ack(delivery_tag=envelope.delivery_tag)



async def worker():
    try:
        transport, protocol = await aioamqp.connect('localhost', 5672)
    except aioamqp.AmqpClosedConnection:
        print("closed connections")
        return
```

```
    channel = await protocol.channel()

    await channel.queue(queue_name='data_queue', durable=True)
    await channel.basic_qos(prefetch_count=1, prefetch_size=0, connection_global=False)
    await channel.basic_consume(callback, queue_name='data_queue')


event_loop = asyncio.get_event_loop()
event_loop.run_until_complete(worker())
event_loop.run_forever()
```

```
#Dockerfile
FROM python:3.5

WORKDIR /usr/src/app

COPY . .

RUN pip install -r requirements_dev.txt
```

```
#docker-compose.yaml
version: '3'
services:
  rabbitmq:
    hostname: rabbitmq
```

```
  image: rabbitmq:3-management
  environment:
    - RABBITMQ_NODENAME=my-rabbit
  ports:
    - "15672:15672"
    - "5672:5672"
  networks:
    - composer_default

amqp-reg-consumer:
  build: .
  command: ["python3", "con_register.py"]
  depends_on:
    - rabbitmq
  environment:
    - AMQP_HOST=rabbitmq
  restart: on-failure
  links:
    - rabbitmq

amqp-reg-publisher:
  build: .
  command: ["python3", "pub_register.py"]
  depends_on:
    - amqp-reg-consumer
    - rabbitmq
  environment:
    - AMQP_HOST=rabbitmq
```

```
  restart: on-failure
  links:
    - rabbitmq


amqp-data-consumer:
  build: .
  command: ["python3", "con_data.py"]
  depends_on:
    - rabbitmq
  environment:
    - AMQP_HOST=rabbitmq
  restart: on-failure
  links:
    - rabbitmq


amqp-data-publisher:
  build: .
  command: ["python3", "con_data.py"]
  depends_on:
    - amqp-data-consumer
    - rabbitmq
  environment:
    - AMQP_HOST=rabbitmq
  restart: on-failure
  links:
    - rabbitmq


rest:
```

```
    build: rest
    ports:
      - "3000:3000"
    env_file:
      - "rest/envvars.env"
    networks:
      - composer_default
    volumes:
      - "~/.composer:/home/composer/.composer"
    depends_on:
      - mongo
    networks:
      - composer_default

  mongo:
    image: mongo
    networks:
      - composer_default
    ports:
      - "27017:27017"

networks:
  composer_default:
    external: true
  local:
```