
**KAUNO TECHNOLOGIJOS UNIVERSITETAS**

**INFORMATIKOS FAKULTETAS**

**Žygimantas Meškauskas**

# REALAUS LAIKO ĮTERPTINIŲ SISTEMŲ PROGRAMINĖS ĮRANGOS SAUGA

Baigiamasis magistro darbas

**Vadovas**
Prof. dr. Egidijus Kazanavičius

**KAUNAS, 2016**

**KAUNO TECHNOLOGIJOS UNIVERSITETAS**
**INFORMATIKOS FAKULTETAS**
**KOMPIUTERIŲ KATEDRA**

TVIRTINU
Katedros vedėjas
(parašas) Prof. dr. Algimantas Venčkauskas
(data)

# REALAUS LAIKO ĮTERPTINIŲ SISTEMŲ PROGRAMINĖS ĮRANGOS SAUGA

Baigiamasis magistro darbas
**Informacijos ir informacinių technologijų sauga (kodas 621E10003)**

**Vadovas**
(parašas) Prof. dr. Egidijus Kazanavičius
(data)

**Recenzentas**
(parašas) dr. Vygintas Kazanavičius
(data)

**Projektą atliko**
(parašas) Žygimantas Meškauskas
(data)

**KAUNAS, 2016**

KAUNO TECHNOLOGIJOS UNIVERSITETAS

Informatikos fakultetas

(Fakultetas)

Žygimantas Meškauskas

(Studento vardas, pavardė)

Informacijos ir informacinių technologijų sauga (kodas 621E10003)

(Studijų programos pavadinimas, kodas)

Baigiamojo projekto „Realaus laiko įterptinių sistemų programinės įrangos sauga“

**AKADEMINIO SĄŽININGUMO DEKLARACIJA**

20 16 m. gegužės 16 d.

Kaunas

Patvirtinu, kad mano, Žygimanto Meškausko, baigiamasis projektas tema „Realaus laiko įterptinių sistemų programinės įrangos sauga“ yra parašytas visiškai savarankiškai, o 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)*

Meškauskas, Ž. Realaus laiko įterptinių sistemų programinės įrangos sauga. *Magistro* baigiamasis projektas / vadovas Prof. dr. Egidijus Kazanavičius; Kauno technologijos universitetas, informatikos fakultetas, kompiuterių katedra.

Kaunas, 2016. 59 psl.

## SANTRAUKA

Mus supančioje aplinkoje integruota daugybė įterptinių sistemų: mobiliuosiuose telefonuose, televizoriuose, automobiliuose, buitinėje technikoje ir t. t. Kadangi panaudojimo mastas yra toks didelis, iškyla saugumo problema - ne visose įterptinėse sistemose įdiegti saugumo sprendimai, o kai kuriose šie sprendimai gali būti kritiškai svarbūs. Problema, norint įdiegti saugumo sprendimus įterptinėse realaus laiko sistemose, yra ta, kad tokiose sistemose labai riboti resursai, kuriuos praplečiant smarkiai išaugtų bendra sistemos kaina.

Šiame darbe išanalizuojamos realaus laiko įterptinių sistemų saugumo spragos, grėsmės, nustatomi galimi apsisaugojimo būdai ir pasirinktoje platformoje realizuojamas vienas iš saugumo sprendimų – sistemos šifravimas.

Bandymų metu nustatyta, kad realizuotas sprendimas naudoja nemažai procesoriaus galios, kas trumpina sistemos gyvavimo trukmę, jei maitinimo šaltinis yra akumuliatorius, be to ženkliai sumažėja skaitymo ir rašymo operacijų greičiai. Į šiuos tyrimų rezultatus reikia atsižvelgti norint diegti sprendimą realioje sistemoje.

**SUMMARY**

There are many embedded systems integrated everywhere around us: in mobile phones, TV's, cars, household appliances, etc. Because of high degree of utilization, there is security problem – not all embedded systems has installed security solutions and in some systems these solutions may be critically important. When installing real-time embedded systems security solutions the problem is that resources in this type of systems are very limited and if resources are expanded, overall system price would grow up significantly.

This paper researches real-time embedded systems security flaws, threats, identifies possible security solutions and desired solution – system encryption is realized in selected platform.

When performing speed test experiments high processor utilisation and input/output speed decrease is observed. Overall system lifetime is shorter when processor works in higher utilisation if power source is battery. When installing proposed security solution in real system, experiment results need to be taken into account.

**TURINYS**

**LENTELIŲ SĄRAŠAS**

**PAVEIKSLŲ SĄRAŠAS**

## TERMINŲ IR SANTRUMPŲ ŽODYNAS

OSI (angl. Open Systems Interconnection) – abstraktus ryšio protokolų, naudojamų ryšio ir kompiuteriniuose tinkluose, aprašymas;

SD, microSD (angl. Secure Digital) - atminties kortelių tipai;

CAN (angl. Controller Area Network) – duomenų protokolo tipas naudojamas automobiliuose, leidžiantis mikrovaldikliams komunikuoti tarpusavyje nenaudojant pagrindinio kompiuterio;

RAM (angl. Random-access memory) – kompiuterio operatyvioji atmintinė, kurioje duomenys saugomi sistemos veikimo metu, žymiai greitesnė už pastoviąją atmintį;

ARM (angl. Advanced RISC Machine) – procesoriaus architektūra, naudojanti supaprastintą instrukcijų rinkinį, sukonfigūruota skirtingoms aplinkoms;

SoC (angl. System on Chip) – sistema, kurios visi komponentai integruoti viename luste;

RS232 – nuoseklus duomenų perdavimo protokolas;

U-Boot – universali atviro kodo įkrovos programa, plačiai naudojama įterptinėse sistemose;

JTAG (angl. Joint Test Action Group) – sąsaja aparatūriniam testavimui, derinimui;

GPIO (angl. General-purpose input/output) – bendros paskirties procesoriaus išvadas, kurį galima valdyti sistemos veikimo metu;

PWM (angl. Pulse-width modultaion) – moduliavimo būdas, skirtas reguliuoti energijos tiekimą įrenginiams keičiant maitinimo impulsų trukmę;

Ethernet – kompiuterinių tinklų protokolas lokaliems tinklams (LAN);

RJ45 – 8 kontaktų jungtis duomenų perdavimui ethernet protokolu;

USB (angl. Universal Serial Bus) – standartas duomenų perdavimui, jungtims ir protokolams tarp kompiuterio ir įvairių elektroninių įrenginių;

DTB (angl. device tree blob) – duomenų bazė, kurioje aprašyti esamos sistemos aparatūros komponentai;

Rootfs (angl. RootFileSystem) – šakninė failų sistema, branduolio įkeliama į atmintį sistemos krovimosi metu;

Flash atmintis – išliekamąją atmintį turinti duomenų saugykla. Tokio tipo atmintis dažnai naudojama atminties kortelėse;

FAT16 (angl. File Allocation Table) – failų sistema, kurioje specialioje lentelėje talpinami įrašai apie kiekvieną failą (FAT16 – vienas iš pirmųjų FAT variantų);

EXT4 (angl. fourth extended filesystem) - žurnalinė failų sistema naudojama Linux operacinėse sistemose;

PCB (angl. printed circuit board) – spausdinta montažinė plokštė, sistemos techninės įrangos dalis;

OS (angl. operating system) – Operacinė sistema (speciali programinė įranga, užtikrinanti vartotojo sąsają ir kompiuterio techninės įrangos, taikomųjų programų bei duomenų valdymą);

SSH (angl. Secure shell) – tinklo protokolas, aprašantis apsaugotą kliento prisijungimą prie serverio aplinkos (shell) ir komandų vykdymą;

IP (angl. Internet Protocol) – tam tikrame tinkle unikalus skaičius, vienareikšmiam kompiuterio identifikavimui;

LUKS (angl. Linux Unified Key Setup) – nuo platformos nepriklausomas disko šifravimo formatas;

API (angl. Application programming interface) – Aplikacijų programavimo sąsaja (sąsaja, kurią suteikia kompiuterinė sistema, biblioteka ar programa tam, kad programuotojas per kitą programą galėtų pasiekti jos funkcionalumą ar apsikeistų su ja duomenimis);

Shell script – kompiuterio programa, vykdoma operacinės sistemos komandinės eilutės interpretatoriaus;

# ĮVADAS

Pasaulyje sparčiai plinta išmanieji įrenginiai ir vis didėja įterptinių realaus laiko kompiuterinių sistemų integracijos laipsnis mus supančioje aplinkoje. Įterptinės sistemos integruojamos kasdien naudojamuose daiktuose, tokiuose kaip televizoriai, telefonai, laikrodžiai, automobiliai, buitinė technika. Dėl plataus panaudojimo masto ir to, kad realaus laiko įterptinės sistemos paprastai būna skirtos atlikti tik specializuotoms užduotims, tokiose sistemose resursai yra labai riboti, kas nulemia santykinai nedidelę kainą. Tačiau dėl nedidelės kainos dažnai realaus laiko įterptinėse sistemose stinga realizuotų saugumo sprendimų. Nerealizavus saugumo sprendimų ir esant fizinei prieigai, o kai kuriais atvejais net ir nuotolinės prieigos būdu gali būti perimti, pakeisti ar sugadinti realaus laiko įterptinėse sistemose perduodami duomenys, taip sutrikdant sistemos darbą. Atakos prieš realaus laiko įterptines sistemas gali būti vykdomos tais pačiais tikslais kaip ir įprastose kompiuterinėse sistemose – finansinės ar duomenų vagystės, sistemos darbo sutrikdymas, duomenų pakeitimas ir t. t. Tačiau šalia įprastinių atakų atsiranda ir papildomas pavojus, kadangi realaus laiko įterptinės sistemos yra didesnių sistemų sudedamoji dalis ir sutrikdžius įterptinės sistemos veiklą bendra sistema gali pridaryti rimtos fizinės žalos (pvz., automobilio avarija, gamybos brokas, žmogaus sužalojimai). Sutrikdyti realaus laiko įterptinės sistemos darbą dažniausiai yra sunku, kadangi paprastai reikalinga fizinė prieiga prie sistemos ir specializuoti įrankiai, tačiau įgyvendinus šias sąlygas sistemai iškyla gana rimtas pavojus, todėl norint išvengti galimų žalingų padarinių reikalingi saugumo sprendimai.

Renkantis realaus laiko įterptinės sistemos programinės įrangos saugumo sprendimą išanalizuojami galimi programiniai ir aparatūriniai apsisaugojimo būdai bei priemonės. Pasirinkto sprendimo ar sprendimų komplekto realizacijai reikalinga fizinė platforma, kurioje galima atlikti bandymus. Pasirenkama tokia realaus laiko įterptinės sistemos platforma, kuriai galiotų aprašytas saugumo trūkumas – dėl sąlyginai nedidelės kainos nebūtų realizuoti saugumo sprendimai.

Paruošus fizinę platformą ir realizavus siūlomus saugumo sprendimus galima atlikti tyrimą – išmatuoti greitaveiką. Greitaveikos tyrimams reikalinga testinė programa, aplinka, kurioje galima atlikti testus ir skirtingi techninės įrangos komponentai. Greitaveikos tyrimo atveju reikalinga keletas skirtingo tipo realaus laiko įterptinės sistemos atmintinių. Tyrimo rezultatai yra pagrindas, sprendžiant ar esamų resursų pakanka, diegiant įterptinę sistemą su integruotu saugumo sprendimu, ar greitaveika yra per maža ir reikia daugiau resursų (tai iškeltų bendrą sistemos kainą).

**Darbo tikslas ir uždaviniai**

Darbo tikslas: parinkti, pritaikyti programinės įrangos saugumo sprendimą realioje įterptinėje sistemoje ir atlikti realizuoto sprendimo greitaveikos tyrimą. Šiam tikslui pasiekti reikia atlikti šiuos uždavinius:

- atlikti esamų realaus laiko įterptinių sistemų grėsmių, galimų programinių ir aparatūrinių saugumo būdų ir naudojamų priemonių analizę;
- atsižvelgiant į iškeltus kriterijus parinkti geriausiai tinkantį sprendimą ir sudaryti realaus laiko įterptinės sistemos platformą, kurioje galima pritaikyti parinktą sprendimą;
- realizuoti sistemos prototipą, naudojant parinktą saugumo sprendimą realioje įterptinėje sistemoje;
- sudaryti greitaveikos tyrimo aplinką, testavimo programą ir atlikti greitaveikos testus įvertinant gautus rezultatus;

Gauti greitaveikos tyrimo rezultatai yra pagrindas, priimant sprendimą ar saugumo sprendimo realizacija realaus laiko įterptinėje sistemoje nepageidautinai nesumažins našumo ir jei taip, ar didinant resursų kiekį įterptinėje sistemoje ne per daug išaugs bendra sistemos kaina.

**Darbo struktūra**

Dokumentą sudaro keturi pagrindiniai skyriai:

1. Realaus laiko įterptinių sistemų saugumo problemos analizė - analizuojamos realaus laiko įterptinėms sistemoms kylančios grėsmės, galimi programiniai ir aparatūriniai apsisaugojimo būdai, priemonės, sudaromi analizės rezultatai, pateikiamos išvados.
2. Atsižvelgiant į analizės kriterijus, pasiūlomas saugumo sprendimas ir sudaroma reali įterptinės sistemos platforma, kurioje galima realizuoti parinkto saugumo sprendimo prototipą ir atlikti tyrimus.
3. Sudaromas realaus laiko įterptinių sistemų programinės įrangos saugos prototipas, testinė aplinka, testavimo programa ir atliekami greitaveikos tyrimai. Apibendrinami atliktų tyrimų rezultatai.
4. Pateikiamos galutinės išvados, interpretuojami atliko tyrimo rezultatai, pateikiami pasiūlymai ir rekomendacijos tolesniam tyrimui.

# 1. REALAUS LAIKO ĮTERPTINIŲ SISTEMŲ PROGRAMINĖS ĮRANGOS SAUGOS ANALIZĖ

Atliekant realaus laiko įterptinių sistemų programinės įrangos saugos analizę pirmiausia reikia nustatyti, ką norima apsaugoti. Norint apsaugoti programinę įrangą nuo pažeidimų, reikia saugoti failus, failų sistemas, duomenis statiniame (duomenų saugojimas) ir dinaminiame (duomenų mainai) rėžimuose.

Šio darbo tyrimo sritis yra saugos klausimai, susiję su realaus laiko įterptinėmis sistemomis. Analizės dalyje išnagrinėjamos realaus laiko įterptinių sistemų programinės įrangos ir duomenų mainų saugumo problemos, kylančios grėsmės, apžvelgiami ir palyginami saugumą užtikrinantys būdai ir priemonės.

## 1.1. Realaus laiko įterptinių sistemų programinės įrangos saugos problema

Mus supa daugybė daiktų, kuriuose integruotos įterptinės sistemos: mobilieji telefonai, televizoriai, automobiliai, automatinės garažo durys su nuotoliniu valdymu ir t. t. Lyginant kiekiu su kitomis, tokios sistemos sudaro didžiausią dalį - net 98% visų kompiuterinių sistemų. [1] Įterptinės kompiuterinės sistemos dažnai yra didesnių sistemų sudedamoji dalis ir taikomos įvairiose srityse – pramonėje, transporto priemonėse, telekomunikacijose, medicinoje ir kitur. Kadangi panaudojimo mastas yra toks didelis, iškyla saugumo problema. Turint specialią programinę ir techninę įrangą galima įsilaužti į realaus laiko įterptines sistemas, perimti, nuskaityti saugomus ar siunčiamus duomenis, juos pakeisti, suklastoti arba atskleisti sistemos architektūrą. Realaus laiko įterptinių sistemų programinės įrangos apsauga reikalauja resursų, kurie tokiose sistemose labai riboti, todėl tenka ieškoti aukso vidurio tarp efektyvumo ir saugumo. Pvz., įterptinėje realaus laiko automobilio atstumo iki kliūties matavimo sistemoje gali susidaryti situacija, kai dėl didelio apdorojamų duomenų kiekio vienu metu gali nebeužtekti sistemos resursų ir apsaugos užtikrinimui ir užduoties įvykdymui realiame laike. Šią problemą gali išspręsti resursų padidinimas, tačiau iškyla kita problema – nemažai išauganti sistemos kaina. Programinės įrangos sauga realaus laiko įterptinėse sistemose yra aktuali problema, neturinti vieningo sprendimo, todėl reikia veiksmingų būdų ir priemonių, užtikrinančių pakankamą saugumą mažiausiomis galimomis sistemos resursų sąnaudomis.

## 1.2. Realaus laiko įterptinių sistemų duomenų mainų protokolų analizė

Duomenų mainai tam tikroje terpėje vyksta pagal numatytas taisykles, kurios vadinamos protokolu. Tam, kad būtų galima ryšio protokolus klasifikuoti buvo sugalvotas abstraktus aprašymas - OSI modelis, hierarchiškai suskirstantis protokolus į 7 atskirus sluoksnius – funkcijų, naudojamų protokoluose, grupes. Pagal susitarimą OSI modelis vaizduojamas taip, kad pirmas sluoksnis būtų apačioje, o visi sekantys sluoksniai eilės tvarka eitų vienas aukščiau kito. Tai grafiškai atvaizduoja principą, kad kiekvienas sluoksnis bendrauja tik su jam gretimais žemesniu sluoksniu, kurio paslaugomis naudojasi ir aukštesniu sluoksniu, kuriam paslaugas teikia. Nagrinėjant duomenų mainus realaus laiko įterptinėse

sistemose aktualūs tampa „žemo lygio“ duomenų mainų protokolai. Kadangi OSI modelis yra abstraktus ir teorinis, žemo lygio duomenų mainai gali būti priskiriami prie pirmojo OSI sluoksnio (fizinio sluoksnio) – tai aparatūros duomenų mainų protokolai – SPI, $I^2C$, 1-Wire, UART, ir t.t... Šiandieniniame pasaulyje integruota daugybė įterptinių sistemų, kurios vykdydamos savo užduotis komunikuoja žemo lygio duomenų mainų protokolais. Iškyla saugumo problema – kaip apsaugoti tokiais protokolais perduodamus duomenis.

Žemo lygio duomenų mainais galima vadinti ir komunikaciją tarp to paties sistemos lusto komponentų bendraujančių per sisteminę magistralę (pvz. tarp procesoriaus ir atmintinės) ir tarp atskirų tarpusavyje komunikuojančių dalių (pvz. tarp automobilio borto kompiuterio ir signalizacijos). Abiem atvejais duomenys perduodami tais pačiais aparatūriniais duomenų mainų protokolais. Plačiausiai iš jų naudojami $I^2C$, SPI, 1-Wire, UART protokolai. Toliau atskirai apžvelgiamas kiekvienas iš paminėtų protokolų, išanalizuojama protokolo struktūra, apsibrėžiama kokie galimi saugumo kriterijai, nustatoma kas juos įtakoja, pabrėžiamos silpnos saugumo atžvilgiu struktūrinės vietos, pažeidžiamumai.

***$I^2C$ protokolas***

$I^2C$ protokolas - 1982m. „Philips Semiconductor“ firmos sukurtas standartas, plačiai naudojamas įvairiuose prietaisuose iki šių dienų mažo greičio periferinių įrenginių jungimui prie kompiuterio motininės plokštės ar įterptinių sistemų (Philips šią sąsają taip pat naudoja savo televizorių viduje). $I^2C$ protokolas pasižymi tuo, kad vienu metu vienas iš įrenginių yra valdantysis (angl. master) ir duomenis siunčia, o visi kiti įrenginiai duomenis tik priima ir yra valdomieji (angl. slave). Signalai keliauja dvejomis linijomis – SDA (duomenys) ir SCL (sinchronizacijos signalas). Originaliai prie vienos $I^2C$ magistralės iš viso gali būti daugiausiai prijungti $2^7$= 128 įrenginiai (7 bitų adreso erdvė), o didžiausias duomenų perdavimo greitis 100kbits/s, tačiau naujausiose protokolo versijose greitis gali siekti iki 5Mbits/s, o įrenginio adresų erdvė praplėsta iki 16 bitų. $I^2C$ protokolo struktūra atrodo taip:

$I^2C$ = <START, ADDRESS, WR, <ACKNOWLEDGE, DATA, ...>, ACKNOWLEDGE, STOP>, kur

$START = \begin{cases} 0 \\ 1 \end{cases}$ - valdančiojo įrenginio generuojama protokolo vykdymo pradžios sąlyga,

ADDRESS = 0..127 – valdomojo įrenginio, kuriam siunčiami duomenys adresas,

$WR = \begin{cases} 0, rašymas \\ 1, skaitymas \end{cases}$,

$ACKNOWLEDGE = \begin{cases} 0 \\ 1 \end{cases}$ - patvirtinimo bitas, gaunamas iš valdomojo įrenginio,

DATA = 0..255 – duomenų baitas,

$STOP = \begin{cases} 0 \\ 1 \end{cases}$ - valdančiojo įrenginio generuojama protokolo vykdymo pabaigos sąlyga.

Šio protokolo saugumas pasiklausymo ir siunčiamų duomenų pakeitimo atžvilgiu yra silpnas, kadangi duomenys keliauja viena linija ir ta pati linija naudojama tiek ryšio tarp valdančiojo ir valdomojo įrenginių sudarymui, tiek duomenų perdavimui. Kadangi nėra jokio apsaugos mechanizmo,

analizatoriaus pagalba stebint duomenų perdavimo procesą galima nesunkiai pastebėti pasikartojimus tarp skirtingų paketų ir to paties paketo dalių, nes ryšio sudarymui vis naudojama preambulė su protokolo vykdymo pradžios sąlyga, valdomojo įrenginio adresu ir skaitymo/rašymo komandos nustatymu, o perduodant duomenis po kiekvieno duomenų baito seka patvirtinimo bitas. Įžvelgus pasikartojimus galima nustatyti, kurie iš perduodamų bitų yra duomenys ir perimti perduodamą informaciją arba sugeneruoti analogiškus signalus su pakeistais duomenimis ir juos perduoti vietoje tikrųjų.

***SPI protokolas***

Vienas iš dažniausiai šiandien naudojamų žemo lygio protokolų mažų atstumų jungimams, tokiems kaip įvairūs jutikliai, SD atminties kortelės, įterptinės sistemos, yra kompanijos „Motorola“ patentuotas standartas – SPI protokolas. Šis protokolas pasižymi lankstumu ir dideliu duomenų perdavimo greičiu. Panašiai, kaip ir $I^2C$ protokolo atveju, vienu metu vienas įrenginys sukonfigūruojamas būti valdančiuoju (angl. master), o visi kiti įrenginiai valdomaisiais (angl. slave). Vienu metu valdantysis ir valdomieji įrenginiai gali vienas kitam siųsti duomenis atskiromis linijomis – tam SPI protokolui, skirtingai nuo $I^2C$ sąsajos, reikalingos dvi duomenų linijos MISO (valdančiojo įrenginio duomenų gavimas iš valdomojo įrenginio, angl. Master Input Slave Output) ir MOSI (valdomojo įrenginio duomenų gavimas iš valdančiojo įrenginio, angl. Master Output Slave Input). Iš viso SPI magistralė turi keturias linijas – be MISO, MOSI ir sinchronizacijos signalo dar yra įrenginio parinkimo linija (SS – Slave Select), kuri reikalinga nustatyti ryšiui tarp valdančiojo ir atitinkamo valdomojo įrenginių, siunčiant duomenis SPI protokolu.

Šio protokolo struktūra yra tokia, kad valdantysis įrenginys sudaro ryšį su valdomuoju įrenginiu naudodamas SS liniją (nėra adresacijos). Sudarius ryšį, valdantysis įrenginys tiesiog siunčia duomenis bitas po bito valdomajam įrenginiui naudodamas MOSI liniją, o valdomasis įrenginys atsako, taip pat siųsdamas duomenis bitas po bito valdančiajam įrenginiui MISO linija. Transakcijos užbaigimas taipogi kontroliuojamas SS linija. Nėra jokio specialaus siunčiamų duomenų tikrinimo ir priimtų duomenų patvirtinimo mechanizmo.

SPI protokolas neturi jokios apsaugos nuo siunčiamų duomenų pasiklausymo, tereikia prisijungti prie duomenų perdavimo linijos ir analizatoriaus pagalba matomi visi siunčiami duomenys tiek iš valdančiojo įrenginio valdomajam, tiek ir atvirkščiai kita linija. Norint išsiųsti pakeistus duomenis reikia sugeneruoti ir išsiųsti dvejomis linijomis ryšio sudarymo ir perduodamų duomenų signalus.

***1-Wire protokolas***

1-Wire yra įrenginių komunikavimo protokolas, sukurtas „Dallas Semiconductor“ korporacijos. Šis protokolas skirtas mažo greičio komunikacijai didesniais atstumais, dažniausiai naudojamas komunikuoti su mažais nebrangiais įrenginiais, tokiais kaip įvairūs skaitmeniniai jutikliai. Kiekvienas 1-Wire įrenginys turi savo unikalų gamintojo identifikacijos numerį, pagal kurį yra atpažįstamas. Duomenys 1-Wire protokole perduodami viena linija.

1-Wire Protokolo struktūra:

1-WIRE = <RESET, SELECT, OPERATION>,

RESET komanda sinchronizuoja visą magistralę, tam, kad būtų galima sudaryti ryšį su valdomaisiais įrenginiais. Toliau seka SELECT komanda, kuri pagal savo formatą sudaro ryšį su nurodytais įrenginiais – įrenginiai nustatomi pagal minėtą unikalų identifikacijos numerį. Gali būti parenkamas vienas konkretus, visi arba sekantis eilės tvarka esantis įrenginys magistralėje. Sudarius ryšį, visi likę įrenginiai (jei nėra sudaromas ryšys su visais įrenginiais) ignoruoja toliau sekančias komandas iki kito sinchronizavimo. Įrenginiams, su kuriais sudarytas ryšys, valdantysis įrenginys siunčia atitinkamas nuo įrenginio priklausančias operacijų komandas – OPERATION. Kadangi kiekvienas įrenginys, esantis 1-Wire magistralėje, gali būti savitas ir atlikti skirtingas operacijas, sudarius ryšį operacijų komandos siunčiamos pagal unikalų tam įrenginiui reikalingą protokolą. Tačiau nors įrenginiai, prijungti prie 1-Wire magistralės gali skirtis, ryšio sudarymo ir komandų siuntimo procesas naudojant 1-Wire protokolą yra bendras.

Kadangi duomenys siunčiami tik sudarius ryšį ir tik tiems įrenginiams, su kuriais ryšys yra užmegztas, norint pasiklausyti šių duomenų reikia prisijungti prie magistralės vietos, esančios tarp valdančiojo įrenginio ir įrenginio, kuriam siunčiami duomenys sudarius ryšį. Tačiau net ir pasiklausius siunčiamų duomenų jie vis tiek yra labai priklausomi nuo įrenginio tipo, nes jungiant daugelį skirtingų įrenginių prie 1-Wire magistralės, įrenginiai gali turėti labai skirtingas ir specifines funkcijas, todėl ir komandų kodai labai skiriasi. Visgi, pasiklausius galima atkartoti tokį patį komandos kodą ir išsiųsti, koks buvo siunčiamas valdančiojo įrenginio. Kitu atveju, norint išsiųsti pakeistą komandą, reikia žinoti konkrečias detales apie įrenginį, su kuriuo atliekama komunikacija. Taigi, 1-Wire protokole nuo perduodamų duomenų pasiklausymo specialios apsaugos nėra, o nuo duomenų pakeitimo apsauga atsiranda iš dalies dėl protokolo struktūros, apjungiančios skirtingų tipų įrenginius per bendrą sąsają.

***UART protokolas***

Kompiuterinėse sistemose įvairaus tipo sujungimuose dažniausiai naudojamas UART (Universal Asynchronous Receiver/Transmitter) protokolas. Šis protokolas paremtas fiksuotu perduodamų duomenų greičiu, kuris turi būti užtikrintas naudojant konkrečią techninę įrangą. Įprastai UART protokolas neturi galimybės palaikyti skirtingų įrenginių, jungiamų prie vienos magistralės, tačiau

didelis privalumas yra tas, kad galima abipusė komunikacija nenustatant valdančiojo ir valdomojo įrenginių rolių. Šio protokolo struktūra atrodo taip:

UART = <START, DATA, PARITY, STOP>, kur

START - konfigūruojamo ilgio perduodamo duomenų bloko pradžios atskaitos taškas,

DATA = 0..X – konfigūruojamo ilgio duomenų seka,

$\text{PARITY} = \begin{cases} 0 \\ 1 \end{cases}$ - lyginumo bitas perduodamų duomenų tikrinimui (naudojamas pasirinktinai),

STOP – konfigūruojamo ilgio perduodamo duomenų bloko pabaigos atskaitos taškas.

UART protokolo tikslas yra užtikrinti fiksuoto greičio duomenų perdavimą. Papildomai gali būti tikrinamas perduodamų duomenų vientisumas. Tiek perduodamų duomenų pasiklausymui, tiek duomenų pakeitimui jokio specialaus apsaugos mechanizmo nėra. Atakuotojui tereikia prisijungti prie magistralės ir atitinkamai sukonfigūruoti nustatymus (perdavimo dažnį, duomenų bitų, kontrolinių bitų kiekius ir lyginumo tikrinimą), kad galėtų klausytis siunčiamų duomenų ar perduoti pakeistus/suklastotus duomenis.

***kiti protokolai***

Egzistuoja daugybė kitų žemo lygio duomenų mainų protokolų, kurių taikymo sritys įvairios, kaip pvz., CAN protokolas, kuris buvo išrastas specialiai automobiliniams taikymams, tačiau jo naudojimas išsiplėtė beveik į visas industrines šakas. CAN protokolas yra gana sudėtingas, nes naudoja įrenginių adresavimą, tikrina duomenų vientisumą, turi perduodamų duomenų klaidų tikrinimo ir taisymo mechanizmą ir keletą sudėtingesnių papildomų funkcijų. Tačiau šiame darbe apsiribojama protokolais, kurie daugiausiai naudojami kompiuterinėse sistemose, todėl tokių protokolų plačiau nenagrinėsime.

***Realaus laiko įterptinių sistemų duomenų mainų terpių protokolų analizės išvados***

Išanalizavus pasirinktus žemo lygio duomenų mainų protokolus galima glaustai apibendrinant palyginti kiekvieną iš jų nagrinėtais saugumo ir efektyvumo klausimais. Palyginimo rezultatai surašomi į 2.1. lentelę.

**1.1. lentelė. Žemo lygio duomenų mainų protokolų palyginimo saugumo ir efektyvumo atžvilgiais lentelė**

| **Protokolas** | **Duomenų perdavimo saugumas** | | **Duomenų perdavimo efektyvumas** | |
|---|---|---|---|---|
| | **Pasiklausymas** | **Pakeitimas, klastojimas** | **Atstumas** | **greitis** |
| $I^2C$ | mažai saugu | mažai saugu | mažas | mažas |
| SPI | nesaugu | nesaugu | mažas | didelis |
| 1-Wire | mažai saugu | iš dalies saugu | didelis | mažas |
| UART | nesaugu | nesaugu | įvairus | konfigūruojamas |

Iš rezultatų matyti, kad siekiant išgauti kuo didesnį duomenų perdavimo efektyvumą (didesnį greitį, didesnį atstumą arba užtikrinti fiksuotą perdavimo greitį) naudojant žemo lygio duomenų mainų

protokolus, akivaizdžiai nukenčia saugumas. Rimtesnės apsaugos nei nuo pasiklausymo, nei nuo perduodamų duomenų pakeitimo neturi nei vienas iš apžvelgtų protokolų. Kiekvienu atveju prisijungus prie magistralės analizatoriaus pagalba galima stebėti visus siunčiamus duomenis. Nuo pakeitimo ir klastojimo SPI ir UART protokolai neturi visai jokios apsaugos, kadangi duomenys siunčiami tiesiog bitas po bito, tačiau ir $I^2C$ protokolo atveju siunčiant duomenis tiesiog pridedama preambulė ir gavimo patvirtinimai, todėl apsauga taipogi labai menka. Vieninteliu 1-Wire atveju apsauga iš dalies atsiranda dėl to, jog siunčiami duomenys priklauso nuo įrenginio, kuriam jie skirti.

### 1.3. Realaus laiko įterptinių sistemų programinei įrangai kylančių grėsmių analizė

Kadangi šiais laikais įterptinės kompiuterinės sistemos tapo daugelio sistemų sudedamoji dalis, atsiranda rimta grėsmė visos sistemos, su ja susijusių kitų sistemų ir naudotojų saugumui, neužtikrinant įterptinės sistemos saugumo. Šiame skyriuje apžvelgiama, kokia žala ir kaip gali būti padaryta, išnaudojant realaus laiko įterptinių sistemų programinės įrangos silpnas vietas ir pažeidžiamumus, sumodeliuojama sistema pabrėžiant vietas, kurioms kyla grėsmės ir išanalizuojamos apibrėžtos grėsmės.

***Grėsmės sistemoms, vartotojams ir galima žala, naudojant neapsaugotą realaus laiko įterptinių sistemų programinę įrangą***

Laikai, kai kompiuteris buvo vienintelis „protingas“ įrenginys daugelio žmonių namuose jau praėjo. Šiandien jam talkina ne tik įvairūs išmanieji telefonai ir planšetiniai kompiuteriai, bet ir išmanūs televizoriai, laikrodžiai ar net šaldytuvai. Išmanūs įrenginiai, kaip ir įprastas kompiuteris, jungiami į tinklą, sudarant galimybę naudotis įvairiomis paslaugomis. Kai kurios iš paslaugų yra mokamos, o apmokėjimai gali būti atliekami tuo pačiu įrenginiu. Atliekant mokėjimus neapsaugotu įrenginiu atsiranda rimta grėsmė saugumui. Atliekant pinigines transakcijas tarp išmanaus įrenginio ir bankinės sistemos, vyksta ne tik komunikacija tarp šių dviejų dalių perduodant duomenis, bet ir vidinė aparatūrinė komunikacija tarp pačio įrenginio sisteminių dalių. Vidinė komunikacija vyksta minėtais žemo lygio duomenų mainų protokolais, pvz. Philips televizoriuose ši firma naudoja savo pačios sukurtą $I^2C$ protokolą. Kadangi remiantis analizės rezultatais matoma, jog žemo lygio duomenų mainų protokolai yra nesaugūs, atakuotojas, prisijungęs prie sisteminės magistralės, gali pasiklausyti siunčiamų duomenų, taip gaudamas pvz.. kreditinės kortelės informaciją. Tas pats galioja ir kitiems išmaniesiems namų apyvokos daiktams. Kadangi prie sisteminės magistralinės reikalingas tiesioginis fizinis prisijungimas, saugumas šiek tiek padidėja dėl įsilaužėliui reikalingos fizinės prieigos, norint pasiklausyti siunčiamų duomenų, tačiau net ir čia išlieka grėsmė – palikus išmanųjį įrenginį be priežiūros, atidavus taisyti ar net nusipirkus, jame gali būti įmontuota speciali įranga, stebinti sistemine magistrale keliaujantį duomenų srautą ir siųstuvo pagalba siunčianti šią informaciją atakuotojui. Lygiai ta pati duomenų perėmimo, sugadinimo ar pakeitimo grėsmė galioja ir statiniams duomenims, saugomiems atmintinėje. Neapsaugojus statinių realaus laiko įterptinių sistemų duomenų, atakuotojas, gavęs fizinę prieigą, naudodamas specialią aparatinę įrangą gali juos nukopijuoti ar pakeisti.

Išmaniuosiuose įrenginiuose sisteminėmis magistralėmis siunčiamų duomenų pasiklausymo galimybė kelia grėsmę vartotojui, tačiau išnaudojant realaus laiko įterptinių sistemų programinės įrangos saugumo spragas gali kilti grėsmė ir visai sistemai ar net su ja susijusioms kitoms sistemoms. Kitas pavyzdys namų apyvokos ribose yra šilumos apskaitos sistema. Duomenys šilumos apskaitos sistemoje surenkami iš kiekvieno mazgo ir perduodami į centrinį įrenginį, kuris atlieka visą suvartotos šilumos apskaitą. Kyla grėsmė – įsilauželiui prisijungus ir tikrus duomenis pakeitus suklastotais, keičiasi visa suvartotos šilumos apskaita. Vieno vartotojo suvartotos šiluminės energijos kiekis gali būti pakeistas į didesnį, kito į mažesnį, siekiant vieno vartotojo mokamos sumos už suvartotą šiluminę energiją dalį perkelti kitam vartotojui ar išdalinti vartotojų grupei, jiems apie tai nieko nežinant.

Be minėtų, su gyventojų būstu susijusių, yra ir daugybė kitų vartotojus ir sistemas liečiančių grėsmių. Viena iš jų – grėsmės, susijusios su vartotojų automobiliais. Kaip žinia, šiuolaikiniuose automobiliuose yra bent po vieną įterptinę realaus laiko sistemą, kuri atsakinga už variklio darbo stebėjimą, reguliavimą, būsenų iš jutiklių surinkimą ir atvaizdavimą vartotojui. Naujausiuose automobiliuose įterptinių sistemų integruota daug daugiau, kurios atsakingos už pavarų dėžės, oro pagalvių valdymą, autopilotavimą, įvairias slydimą likviduojančias, stabilizuojančias sistemas (ABS, ESP) ir kt. Komunikacija tarp šių įterptinių sistemų, prijungtų jutiklių ir aktuatorių (pvz. ABS sistemoje stabdžių) vyksta žemo lygio duomenų mainų protokolais, daugiausiai apžvelgtu CAN protokolu. Nors šis protokolas turi daugelį versijų ir jo naudojimas ribojamas įstatymais, įsilaužėliui pavykus išnaudoti saugumo spragas galimi nemaži padariniai. Kai kuriuose automobiliuose užrakinimo sistemą, variklio užvedimą kontroliuoja kompiuteris. Žinant tokio automobilio sandarą, sisteminių dalių fizinę vietą, dažnai galima nesunkiai pragręžus kėbulą prisijungti prie sisteminės magistralės. Siunčiant atitinkamas komandas tokį automobilį galima atrakinti ir užvesti. Klastojant iš jutiklių kompiuteriui siunčiamus duomenis dėl netinkamo variklio reguliavimo šis gali būti sugadinamas arba dar daugiau – tokios sistemos, kaip pvz., stabilizavimo (ESP), gavus suklastotus duomenis iš jutiklių gali vietoje stabilizavimo automobilį visiškai sumėtyti taip sukeldama avariją ir žmonių sužalojimą.

Dar viena iš grėsmių, galinčių atnešti rimtos komercinės žalos, yra industrinis špionažas. Pristatant sukurtą technologinį produktą į prekybą dokumentacijoje aprašomos jo atliekamos funkcijos, tačiau neatskleidžiama įrenginio architektūra, saugant jį nuo neteisėto padirbinėjimo. Įvairių mikroschemų apsauga nuo architektūros atskleidimo dar labiau padidinama užliejant jas specialiu mišiniu, tačiau net ir tai nepadeda, kadangi įsilaužėliai kartais vis tiek sugeba pragręžti apsauginį sluoksnį tiksliai reikiamoje vietoje ir nuskaityti sistemine magistrale siunčiamus signalus ar atmintyje saugomą informaciją. Ilgesnį laiką stebint sistemos veikimą, surinkus daugiau duomenų, daugeliu atvejų galima atkurti sistemos architektūrą ir pradėti gaminti neteisėtas kopijas, todėl kyla grėsmė technologiniam saugumui.

***Realaus laiko įterptinių sistemų programinės įrangos grėsmių modelis***

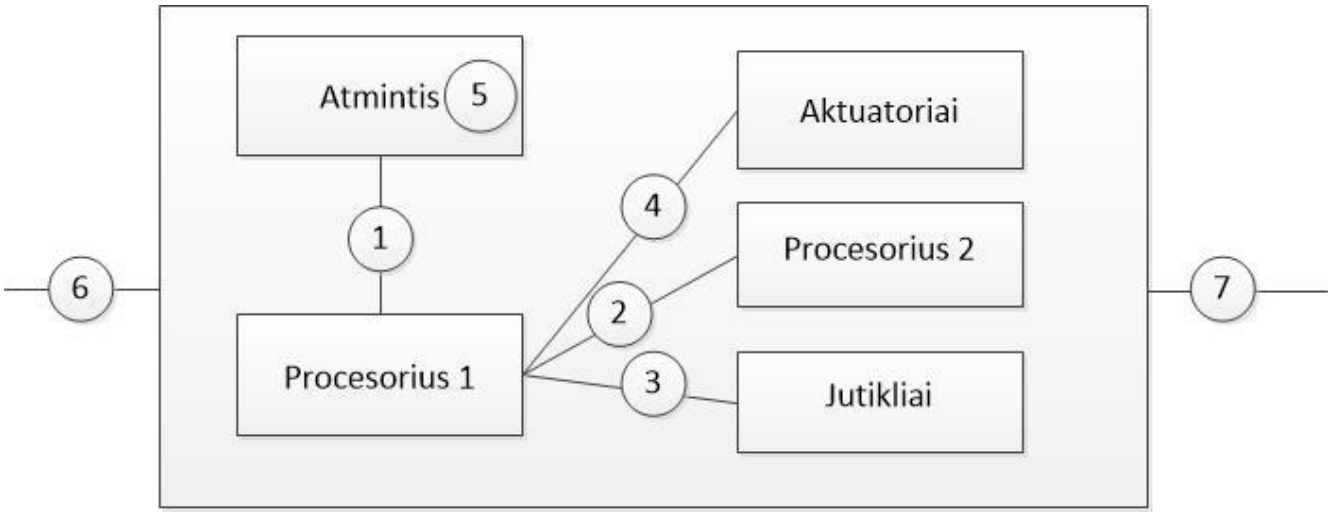
Norint geriau išnagrinėti silpnas sistemos vietas ir atrasti pažeidžiamumus, kylančias grėsmes, naudinga sudaryti teorinį sistemos modelį, kuriame atsispindėtų pagrindiniai komponentai, struktūra, ryšiai tarp komponentų, kuriais komunikuojama ir sąsajos su išore. Sistemos modelyje komponentai yra atmintis, procesorius, registrai, ryšiai tarp komponentų ir sąsajos su išore, atspindinčios galimus sistemos įėjimus ir išėjimus.

Bendru atveju, bet kokią kompiuterinę sistemą sudaro:

- Procesorius
- Atmintis
- Įvesties ir išvesties įrenginiai

Modeliuojant šis apibrėžimas labai svarbus, kadangi reikia nustatyti kokie galimi sistemos komponentai ir kiek jų yra. Sistemoje gali būti ir bendros paskirties kompiuterių ir specialias funkcijas atliekančių mikrovaldiklių, tačiau visi jie turi vienokį ar kitokį proceorių, bei įvairias atmintis. Kadangi šiame modelyje yra aktuali komunikacija tarp vidinių sistemos dalių, o ne architektūra, bet kokio tipo procesorius pažymimas tiesiog procesoriumi ir visokių tipų atmintinės pažymimos tiesiog atmintimi, kadangi nagrinėjant bet kokio lygio atmintinę komunikavimo mechanizmas tarp jos ir procesoriaus, vis tiek atliekamas sisteminėmis magistralėmis žemo lygio duomenų mainų protokolais, o siekiant nustatyti, kurioje vietoje galimos grėsmės aktuali yra tik komunikacija. Įvesties ir išvesties įrenginiai modelyje yra dviejų tipų: pačios sistemos atžvilgiu ir sistemos viduje. Įvestis ir išvestis pačios sistemos atžvilgiu yra komunikacija su išore, o vidiniai įvesties ir išvesties įrenginiai (tai gali būti bet kokios paskirties kompiuterinė sistema) yra įvairūs aplinkos būseną fiksuojantys jutikliai ir veiksmus atliekantys aktuatoriai. Duomenis sauganti atmintis taipogi pažymėta sistemos ribose. Komponentai sutartinai žymimi stačiakampiais, ryšiai tarp jų nubrėžiami linijomis, apibrėžiamos sistemos ribos, nubrėžiamos sąsajos su išore ir apskritimais pažymimos sistemos vietos, kuriose kyla grėsmės.

Sudarytas sistemos grėsmių modelis pateiktas 1.1. pav.




**1.1. pav.** Įterptinės sistemos grėsmių modelis

Grėsmės pažymėtos apskritimais ir sunumeruotos:

1) Grėsmė komunikuojant tarp procesoriaus ir atmintinės. Procesorius skaitymo ir rašymo operacijas su atmintimi atlieka naudodamas aparatūrinių duomenų mainų protokolus (dėl reikalingo didelio greičio dažnai naudojamas SPI protokolas). Procesoriaus komunikacijos su atmintine metu perduodamai informacijai kyla grėsmė;
2) Sistemoje gali būti daug įvairių procesorių, todėl kita grėsmė kyla atliekant jų tarpusavio duomenų mainus sisteminėmis magistralėmis;
3) Procesorius surenka informaciją iš įvairių aplinkos parametrus fiksuojančių jutiklių. Grėsmė iškyla perduodant jutiklių informaciją;
4) Procesorius siunčia duomenis aktuatoriams, kad šie atliktų tam tikrus veiksmus. Grėsmė iškyla perduodant informaciją aktuatoriams;
5) Sistemoje gali būti keletas skirtingo tipo atminčių (registrai, spartinančiosios atmintinės, darbinė, pastovioji atmintis). Paprastumo dėlei modelyje visokio tipo atmintys pažymėtos bendru bloku. Atmintinėms kyla duomenų nuskaitymo grėsmės;
6) Stačiakampiu apibrėžtos sistemos ribos, kurios parodo, jog kyla ir išorinės grėsmės su sistema atliekant įvesties operacijas;
7) Išorinės grėsmė kyla ir išvedant duomenis iš sistemos.

***Realaus laiko įterptinės sistemos programinei įrangai kylančios grėsmės***

Išanalizavus saugumo spragų realaus laiko įterptinėse sistemose išnaudojimo žalą ir silpnas sistemos vietas, dėl kurių kyla grėsmės, lieka išanalizuoti kokios konkrečiai kyla grėsmės. Šiame skyriuje apžvelgiamos konkrečios technikos, kurias naudojant gali būti išnaudojamos išanalizuotos silpnos sistemos vietos.

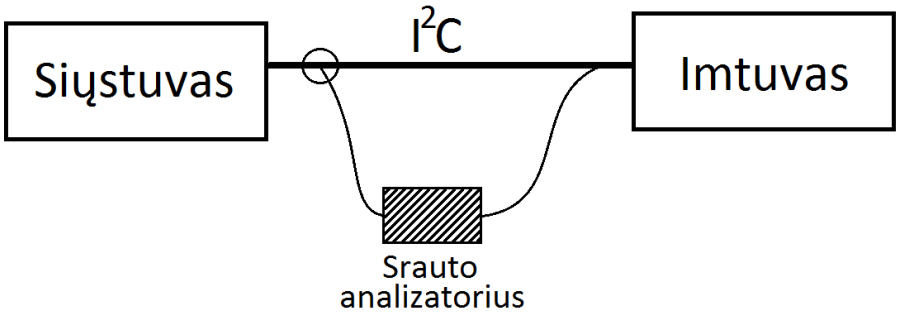
Žvelgiant abstrakčiai, gali būti sukeliamos dviejų tipų grėsmės:

- programinės;
- aparatinės.

Egzistuoja įvairių programinės įrangos variantų, atliekančių skirtingus veiksmus, bet visų jų paskirtis yra ta pati – rinkti ir analizuoti srautu keliaujančius duomenis. Kaip pavyzdys, viena iš tokių programinių grėsmių yra virusas „sniffer‘is“.

Taipogi yra ir aparatinė siunčiamų duomenų perėmimo grėsmė, kada fiziškai prisijungiama prie sisteminės magistralės ir loginio analizatoriaus pagalba renkama informacija. Gali būti naudojami ir kiti aparatiniai metodai, kurių tikslas yra toks pats – išgauti perduodamais duomenimis siunčiamą informaciją ar nuskaityti saugomus duomenis. Pavyzdys – fizinė intervencija, kada nutraukiamos sisteminės grandys ir prijungiamas srautą analizuojantis įrenginys. Taipogi sisteminių grandžių nutraukimas gali būti naudojamas ir kaip priemonė sistemos darbo sutrikdymui ir pvz., stebėti veiksmus, kurie atliekami, siekiant išgauti išimtinių atvejų (angl. exceptions) apdorojimo vykdymo procesą, kad vėliau tai būtų galima panaudoti blogiems tikslams – pvz., atjungti ar pakeisti avarinį rėžimą reguliuojančią posistemę.

Srauto analizatoriaus principas pateikiamas 1.2. pav.




**1.2. pav.** Srauto analizatoriaus veikimo principas

Beveik visada naudojami įrankiai yra specialios aparatūrinės ir programinės įrangos kombinacija. Pavyzdžiui, mygtukų paspaudimo registravimo kenkėjiška programa apjungiama kartu su specialia aparatūrine įranga, siekiant išgauti duomenis. Be perimamų duomenų yra tikrinama, kokio stiprumo sistemos architektūra ir kiek ji atspari vidinių būsenų išgavimui.

***Realaus laiko įterptinių sistemų programinei įrangai kylančių grėsmių analizės išvados***

Atlikus realaus laiko įterptinių sistemų programinei įrangai kylančių grėsmių analizę matyti, jog didžiausios grėsmės atsiranda duomenų saugojimui, perdavimui ir technologiniam saugumui. Reikia veiksmingų būdų ir priemonių, galinčių užtikrinti apsaugą nuo kylančių grėsmių, kadangi silpnų vietų sistemoje nemažai, o padaryta žala gali būti labai didelė.

### 1.4. Realaus laiko įterptinių sistemų programinės įrangos apsaugos būdų analizė

Analizėje tiriamas realaus laiko įterptinių sistemų programinės įrangos ir duomenų saugumas. Tokio tipo sistemose ištekliai maži, o resursai labai riboti, todėl atsižvelgiant į architektūrą, vien programinio saugumo sprendimo dėl resursų trūkumo gali nepakakti. Kaip tik dėl šios priežasties saugumas papildomai gali būti užtikrinamas ne vien programiniais būdais, bet kartu komplektuojamas su aparatūriniais sprendimais, fizine apsauga. Norint dar labiau padidinti saugumą, atsižvelgiant į konkrečios įrangos architektūrines savybes galima net parinkti jai atitinkamus metodus ir būdus. Toliau apžvelgiami galimi realaus laiko įterptinių sistemų programinės įrangos ir duomenų apsaugos būdai.

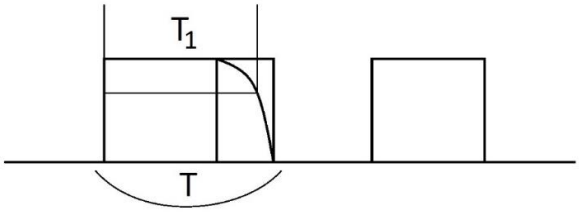
***Šifravimas***

Turbūt vienintelis būdas, dėl kurio naudojimo nekyla jokių abejonių, yra duomenų šifravimas. Šifravimo saugumas paremtas sudėtingais matematiniais algoritmais, todėl užšifruotą informaciją iššifruoti dažniausiai yra labai sunku ar beveik neįmanoma. Galima šifruoti failus, failų sistemas, duomenis. Šifruojant duomenis panaudojamos duomenų mainų protokolų preambulės (pvz., $I^2C$ protokolo atveju preambulėje nustatomas įrenginio, kuriam bus siunčiama informacija, adresas ir komanda, nusakanti, ar bus vykdoma skaitymo ar rašymo operacija), paliekant jas nešifruotas (šifruojant tik duomenis, esančius tarp preambulių). Tokiu atveju įsilaužėlis, net ir galėdamas pasiklausyti viso siunčiamų duomenų srauto ar perėmęs atmintyje saugomus duomenis, juos turės užšifruotus, kurie neturint šifravimo rakto yra beverčiai. Šifravimo raktas gali būti įvairaus ilgio ir tipo, laikinas ir pastovus, saugomas skirtingose vietose. Tai taipogi gali būti raktas, kuris generuojamas pagal biometrines savybes, pvz., piršto antspaudą ar akies rainelę (dar geriau pagal kelis unikalius biometrinius parametrus vienu metu). Jei raktas pastovus, jis turi būti saugomas kokioje nors atmintinėje, tada reikia parinkti specialią atmintinės saugojimo vietą, pvz., apsaugotą slaptažodžiu. Pačioje atmintinėje esantys duomenys irgi gali būti šifruojami įvairiai – tik tam tikros atmintinės vietos, šifruojami tik duomenys, šifruojamas tik programos kodas, atliekamas pilnas šifravimas ir pan. Šifravimas gali būti atliekamas vieną kartą arba būti nuolatos vykstantis procesas. Šifravimo algoritmo (simetrinis (DS, 2DS, 3DS), asimetrinis (RSA)), raktų ilgio ir kitų aptartų parametrų parinkimas labai stipriai priklauso nuo įterptinėje sistemoje esančių resursų kiekio, kadangi šifravimas turi vykti kaip papildomas saugumą užtikrinantis procesas netrukdydamas vykdyti pagrindinių įterptinės sistemos užduočių.

***Griežtai determinuoti protokolų vykdymo laiko intervalai***

Siekiant padidinti saugumą mažesnėmis resursų sąnaudomis beveik visada papildomai taikomi aparatūriniai saugumą padidinantys būdai. Vienas iš tokių būdų yra griežtai apriboti žemo lygio duomenų mainų protokolų vykdymo laikines charakteristikas. Protokolo vykdymo laikinėje charakteristikoje stačiakampiai signalai generuojami griežtai determinuotais intervalais ir yra labai jautrūs netikslumams. Tokiu atveju įsilaužėlis, net ir žinodamas protokolo struktūrą, gali nesugebėti

sugeneruoti reikiamo tikslaus signalo. Aparatūrinis apsaugos būdas griežtai determinuojant laiko intervalus pavaizduotas 1.3. pav.



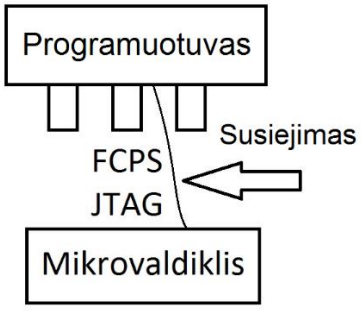

**1.3. pav.** Aparatūrinis protokolų apsaugos būdas griežtai determinuojant laiko intervalus

Naudojant šį metodą praktinės protokolu perduodamų duomenų signalų charakteristikos turi būti pakankamai artimos teorinėms – signalo trukmė turi tilpti į griežtai determinuotą laiko intervalą su labai mažomis paklaidomis, todėl yra jautri kylančio ir krentančio fronto nelygumams, kaip parodyta 1.3. pav. Per laiko tarpą T turėjęs praeiti signalas dėl krentančio fronto nelygumo praeina greičiau, per laiko tarpą $T_1$, ir yra nebepriimamas.

***Aparatūros susiejimas***

Kitas pakankamai plačiai taikomas aparatūrinės apsaugos būdas yra konkrečios aparatūros susiejimas. Pvz., atmintis susiejama su procesoriumi pagal jo slaptą unikalų serijinį identifikacijos numerį ir joks kitas įrenginys, turintis kitokį serijinį numerį negali kreiptis į tą atmintinę, kol ji yra susieta. Susiejimas realizuojamas kriptografiniu būdu užkoduojant atmintinėje esančią informaciją, o slaptas procesoriaus identifikacijos numeris tampa šifravimo raktu.

Programuojant mikrovaldiklius programa įkeliama naudojant programuotuvą. Čia taipogi taikomas susiejimo metodas, programa įdiegiama naudojant specialų programuotuvą, kuris atlieka šifravimą ir joks kitas programuotuvas nebegali mikrovaldiklio perprogramuoti net ir siunčiant tuos pačius duomenis tais pačiais protokolais. Aparatūrinis mikrovaldiklio susiejimo su programuotuvu apsaugos būdas pateiktas 1.4. pav.




**1.4. pav.** Aparatūrinis mikrovaldiklio susiejimo su programuotuvu apsaugos būdas

Naudojant tokį metodą architektūra tampa saugi, kadangi programa įkeliama naudojant specialų programuotuvą, turintį reikalingus šifravimo raktus.

***Atmintinės parinkimas***

Be jau paminėtų aparatūrinės apsaugos būdų, kurie skirti apsaugoti atmintinę ar joje esančius duomenis, galima naudoti ir paprastesnį būdą – parinkti saugumo atžvilgiu tinkamą atmintinę įterptinei sistemai. Kai kurios atmintinės turi rezervuotą vieną kartą programuojamą tik skaitomą (angl. OTP, One Time Programmable) atminties sritį pradžioje, į kurią įrašomi sisteminiai parametrai. Tokioje atmintinės vietoje galima įrašyti šifravimui reikalingus raktus ir kitą svarbią informaciją. Taipogi atmintinės turi tokias funkcijas, realizuotas elektroninių komponentų lygmenyje, kaip apsauga nuo perrašymo (angl. WP, Write Protection) ar galimybę atmintinės vietas apsaugoti slaptažodžiu. Parenkant tinkamiausią atmintinę, atsižvelgiant į sistemos specifikaciją ir saugumo reikalavimus palengvinama realaus laiko įterptinėse sistemose statinių ir dinaminių duomenų apsauga, kartu atlaisvinant dalį resursų.

***Realaus laiko įterptinių sistemų programinės įrangos apsaugos būdų analizės išvados***

Egzistuoja nemažai realaus laiko įterptinių sistemų programinės įrangos apsaugos būdų, tačiau norint užtikrinti kuo didesnį galimą saugumą kuo mažesnėmis sistemos resursų sąnaudomis, dažnai tenka apjungti aparatinius ir programinius saugumo būdus, parenkant geriausią kombinaciją, atsižvelgiant į konkrečios sistemos architektūrą, arba net ir parinkti tokius specialius sistemos komponentus, kaip atmintinė. Taipogi reikia pasirinkti, kokiu lygmeniu ir kokiomis priemonėmis bus užtikrinamas saugumas.

## 1.5. Realaus laiko įterptinių sistemų programinės įrangos saugą užtikrinančių priemonių analizė

Šioje analizės dalyje remiantis išanalizuotais realaus laiko įterptinių sistemų programinės įrangos saugą užtikrinančiais būdais apžvelgiamos saugumui užtikrinti naudojamos priemonės ir pagal ką jas pasirinkti, kokias ir kaip galima nusistatyti realaus laiko įterptinių sistemų programinės įrangos saugumą užtikrinančių priemonių pasirinkimo gaires, ir kokios apskritai yra saugumą užtikrinančios priemonės.

### 1.5.1. Realaus laiko įterptinių sistemų programinės įrangos apsaugos priemonių pasirinkimo gairės

Renkantis įterptinių sistemų programinės įrangos apsaugai naudojamas priemones verta nusibrėžti gaires, kurios padėtų nustatyti svarbiausias vietas, į kurias reikėtų atkreipti dėmesį. Šiuo atveju kai jau atlikta apsaugos būdų analizė, pats svarbiausias klausimas yra, kaip ir kokiomis priemonėmis naudojantis galima įgyvendinti kiekvieną išanalizuotą būdą. Sudarytos realaus laiko įterptinių sistemų programinės įrangos apsaugos priemonių pasirinkimo gairės:

- priemonės tipas – kadangi apsaugos būdai bendru atveju yra dviejų tipų (programinis ir aparatūrinis), tai viena iš apsaugos priemonių pasirinkimo gairių taipogi turėtų būti tipas;
- efektyvumas – kiek efektyviai galima įgyvendinti išanalizuotą apsaugos būdą naudojant pasirinktas priemones;

- tinkamumas – ar tikrai pasirinkta apsaugos priemonė yra tinkamiausia lyginant su galimomis alternatyvomis, norint įgyvendinti išanalizuotą apsaugos būdą;
- saugumas – pačios apsaugą užtikrinančios priemonės saugumas (atsparumas įvairioms atakoms);
- patvarumas – kiek ilgai ir nuo kokio stiprumo atakų galima bus apsaugoti sistemą, naudojant pasirenkamą priemonę;
- kainos ir kokybės santykis – kiek išaugs galutinio produkto vertė panaudojus jame reikiamą saugumo lygį užtikrinančias pasirinktas priemones.

Išanalizavus saugumą užtikrinančias priemones, nustatytus apsaugos būdus, sekant susidarytomis gairėmis, pasirenkamos labiausiai tinkamos numatytą saugumo strategiją vykdyti priemonės.

### 1.5.2. Realaus laiko įterptinių sistemų programinės įrangos saugą užtikrinančios priemonės

Šiame skyrelyje apžvelgiamos realaus laiko įterptinių sistemų programinės įrangos saugą užtikrinančios priemonės, susijusios su išanalizuotais apsaugos būdais.

***Programuotuvas***

Vienas iš aparatūrinio susiejimo būdų yra specialaus programuotuvo naudojimas. Bendru atveju programuotuvas tiesiog įkelia programos kodą į prijungtą mikrovaldiklį, tačiau mus domina kita šio įrenginio savybė – galimybė pridėti specifinį aparatūrinį skaitmeninį parašą įkeliamos programos autentiškumui užtikrinti. Toks programuotuvas turi savyje šifravimo raktus ir programos įdiegimo metu užšifruoja įkeliamą informaciją šiais raktais.

***Lituoklis***

Norint papildyti sistemą specialiomis aparatūrinėmis dalimis dažnai neišsiverčiama be kitos svarbios priemonės – lituoklio, kurį naudojant prie esamų schemų prijungiami papildomi komponentai, pvz., nagrinėjamu atveju vienas iš apžvelgtų apsaugos būdų yra griežtai nustatyti laiko intervalai. Procesorius gali pats generuoti tokius signalus arba specialiose sudarytose schemose tokie signalai gali būti išgaunami iš netobulo pradinio sugeneruoto signalo. Tiek montuojant tokias saugumą užtikrinančias schemas, tiek jungiant jas prie sistemos neapsieinama be šios priemonės.

***Speciali programinė įranga***

Be reikalingų aparatūrinių priemonių, taipogi reikalinga ir programinė įranga. Reikalinga specialiai suprojektuota programinė įranga, įdiegianti saugumo modulius į įterptines sistemas, pvz., specialiai pritaikytų šifravimo algoritmų sudarymas ir įdiegimas. Taipogi speciali programinė įranga gali būti naudojama kaip papildoma sistemos dalis tikrinanti siunčiamų duomenų vientisumą.

Nustačius būdus, kuriais galima apsaugoti realaus laiko įterptinių sistemų programinę įrangą ir duomenis, sudarius gaires apsaugos priemonių pasirinkimui ir apžvelgus galimas apsaugos priemones sudaroma labiausiai tinkanti saugumo strategija konkrečiai sistemai su savita architektūra, resursų kiekiu ir reikiamu apsaugos lygiu. Sudarius tokią saugumo strategiją toliau ją galima bandyti įgyvendinti ir įgyvendinus stebėti rezultatus, susijusius su saugumu, siekiant nustatyti padidinamo saugumo ir išnaudojamų resursų balansą ir jį įvertinti.

## 1.6. Realaus laiko įterptinių sistemų programinės įrangos saugos analizės išvados

Atlikus realaus laiko įterptinių sistemų programinės įrangos saugos analizę išnagrinėti keturi plačiai kompiuterinėse sistemose naudojami duomenų mainų protokolai. Analizės rezultatai rodo, jog jie yra nesaugūs dėl to, kad stengiamasi išgauti kuo didesnį duomenų perdavimo efektyvumą.

Grėsmės, susiję su įterptinių realaus laiko sistemų saugumo spragų panaudojimu yra didelės, galinčios atnešti rimtos piniginės žalos, o kai kuriais atvejais net rimtus fizinius sužalojimus. Įterptinėse kompiuterinėse sistemose grėsmės kyla saugant duomenis, juos perduodant tarp procesoriaus ir atmintinės, tarp skirtingų procesorių, tarp procesoriaus ir kitų įrenginių bei tarp sistemos ir išorės per įvestis ir išvestis. Bendru atveju kyla grėsmės duomenų saugojimo, perdavimo ir technologiniam saugumui.

Norint užkirsti kelią šioms grėsmėms, išanalizuoti aparatūriniai ir programiniai apsaugos būdai – duomenų šifravimas, griežtai determinuoti laiko intervalai žemo lygio protokolais keliaujančių duomenų signalams, aparatūros susiejimas užšifruojant raktu, kuris gali būti aparatūros slaptas unikalus identifikatorius. Šiems būdams įgyvendinti reikalingos įvairios aparatūrinės ir programinės saugumo priemonės (programuotuvas, lituoklis, speciali programinė įranga), kurių pasirinkimui sudarytos gairės.

Toliau pagal įterptinės sistemos architektūrą ir resursų kiekį pasirinkus tinkamus apsaugos būdus ir pagal apsaugos priemonių pasirinkimo gaires parinkus tinkamas priemones šiems būdams įgyvendinti, kuriamas metodas, užtikrinantis realaus laiko įterptinių sistemų programinės įrangos saugumą ir įvertinamas metodo efektyvumas saugumo ir resursų naudojimo atžvilgiu.

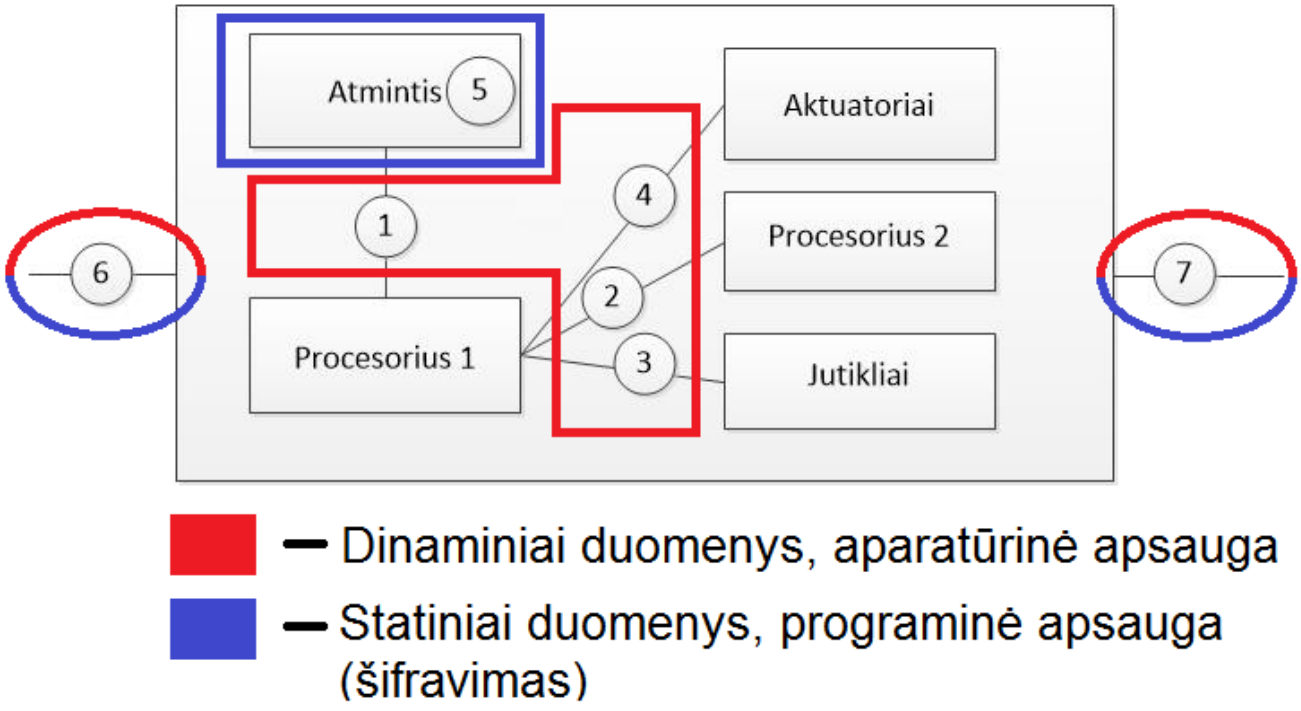
# 2. REALAUS LAIKO ĮTERPTINIŲ SISTEMŲ PROGRAMINĖS ĮRANGOS SAUGOS SPRENDIMAS

Atliktos analizės rezultatai rodo, kad realaus laiko įterptinių sistemų programinės įrangos saugos problemai spręsti yra galimi tiek programiniai, tiek aparatūriniai apsaugos mechanizmai. Galima taikyti kurį nors saugumo sprendimą ar kelių sprendimų kombinaciją, tačiau norint nustatyti, kuris sprendimas geriausiai tinkantis, atsižvelgiant į konkrečias aplinkybes, reikia tiksliai apibrėžti ką ir nuo kokių pavojų norima apsaugoti. Norint apsaugoti konkrečią sistemą nuo tam tikrų potencialių grėsmių, kiekvienu atveju sprendimas priklauso nuo sistemos architektūros ir saugos problemų, kurias norima išspręsti. Šiame skyriuje pasiūlomas konkretus saugos sprendimas, pritaikomas tam tikroje architektūroje, kuris padėtų sustiprinti realaus laiko įterptinių sistemų programinės įrangos apsaugą.

## 2.1. Realaus laiko įterptinių sistemų programinės įrangos saugos sprendimo pagrindimas

Daugelyje realaus laiko įterptinių sistemų, kaip ir kitose kompiuterinėse sistemose, duomenų mainus sisteminėmis magistralėmis (SPI, I2C, UART) valdo operacinė sistema. Kai kuriose įterptinėse sistemose nėra operacinės sistemos, veikia tik įkelta programa – valdymo ciklas, bet tai labiau būdinga mikrovaldikliams, kurie dažniausiai projektuojami atlikti vieną ar kelias tam tikras specialias valdymo užduotis ir turi per daug ribotą resursų kiekį didesnių programų veikimui, todėl atliekant tyrimą apsiribojama tomis sistemomis, kuriose veikia operacinė sistema. Operacinė sistema atlieka tarpinį vaidmenį tarp vartotojo programų ir duomenų mainų sistemoje, inicijuotų priklausomai nuo programose atliekamų veiksmų ir papildomų paslaugų. Tai reiškia, kad pvz., linux operacinėse sistemose „root“ vartotojo įvedamos komandos, kuriomis keičiama sistemos būklė, yra griežtai kontroliuojamos operacinės sistemos ir visi duomenys, kurie turi būti perduoti sistemoje, atliekant operacijas, reikalingas šių komandų įvykdymui taip pat valdomi operacinės sistemos. Lygiai taip pačiai valdomi ir paprasto vartotojo komandų vykdymui reikalingi duomenų mainai, tačiau paprasto vartotojo atliekamų operacijų saugumu papildomai rūpinasi ir pati operacinė sistema, naudodama sudarytus teisių ir apribojimų sąrašus. Kaip tik dėl to ir pateikiamas pavyzdys apie root vartotoją, turintį visišką kompiuterinės sistemos kontrolę, kurią jam suteikia operacinė sistema. Tačiau, nors „root“ vartotojas turi visos sistemos kontrolę ir privilegijas naudoti sisteminius kvietinius, reikalaujančius aukščiausių teisių sistemoje, šių kvietinių įgyvendinimu (reikalingų duomenų mainų atlikimu tarp sistemos blokų, pvz. tarp procesoriaus ir RAM atminties) rūpinasi pati operacinė sistema, vadinasi reikalinga apsauga nuo galimo neteisėto operacinės sistemos valdymo perėmimo, apsaugant visą sistemą nuo galimų pažeidžiamumų. Reikia apsaugoti statinius bei dinaminius duomenis, naudojant išanalizuotus programinius ir aparatūrinius apsaugos būdus.

Iš 1.1. pav. pateikto grėsmių modelio matyti, kad 1 - 4 numeriais pažymėtos grėsmės atsiranda tarpusavyje komunikuojant vidiniams sistemos komponentams (grėsmė dinaminiams duomenims), 5 numeriu pažymėta grėsmė atmintyje saugomiems duomenims (grėsmė statiniams duomenims), o 6 ir 7 numeriais pažymėtos sistemos išorinės grėsmės. Tokiu būdu suskirsčius sistemos grėsmes galima parinkti kiekvienai išskirtai grupei po siūlomą saugumo sprendimą. Siūlomas bendras saugumo sprendimas pateiktas 2.1. pav.




**2.1. pav.** Realaus laiko įterptinės sistemos saugumo sprendimas

Statinių duomenų apsaugai siūloma naudoti programinį apsaugos būdą (šifravimą), o dinaminių duomenų apsaugai siūloma naudoti aparatūrinį apsaugos būdą.

### 2.2. Realaus laiko įterptinių sistemų programinės įrangos saugos sprendimo platforma

Realaus laiko įterptinių sistemų programinės įrangos veikimui rinkoje siūloma daugybė skirtingų gamintojų parduodamų platformų. Platformos skiriasi savo kaina, pajėgumu, architektūra, dydžiu, funkcijomis ir t.t... Kadangi nagrinėjama saugumo problema apima duomenų mainų (perduodamų tarp sistemos dalių sąsajomis su išoriniais įrenginiais) saugos problemą, prieš sudarant sprendimą reikia išsirinkti tinkamą architektūrą. Sisteminių blokų išdėstymo atžvilgiu egzistuoja du architektūros tipai:

- procesorius ir atmintis patalpinti dviejuose atskiruose lustuose, kuriuos jungia atviros sisteminės magistralės;
- procesorius ir atmintis pataplinti tame pačiame, taip vadinamame SoC arba SOC uždarame luste („system on a chip“ arba „system on chip“) – plačiai naudojama įterptinėse sistemose dėl savo kompaktiškumo ir mažų energijos sąnaudų.

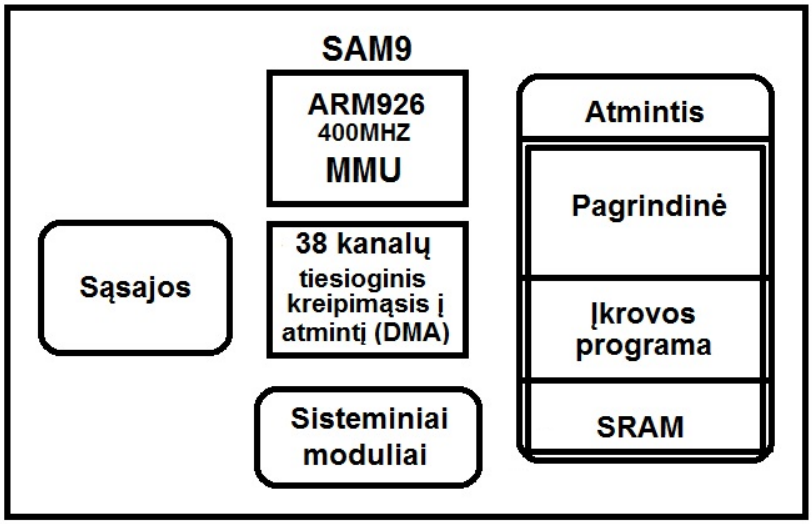
Atsižvelgiant į analizės rezultatus matyti. jog, atviros sisteminės magistralės sukelia grėsmę įterptinės sistemos saugumui, kadangi sistemos veikimo metu atliekant duomenų mainus, į atmintį rašomi ir iš atminties skaitomi duomenys gali būti nuskaityti specialių loginių analizatorių pagalba. Patalpinant procesorių ir atmintį viename luste išsprendžiama jų tarpusavio magistralės saugos problema, kadangi sumažinamas taškų kiekis, kuriuose realaus laiko įterptinėms sistemoms kyla grėsmės. Tyrimui renkantis SOC architektūrą, viena iš grėsmių sumažėja, taip padidinant sistemos saugumą vien dėl platformos parinkimo. Negana to, neretai viename luste talpinant kelis sistemos komponentus šis lustas papildomai apsaugomas padarant jį daugiasluoksniu, todėl fizinio priėjimo galimybė (pvz., pragręžiant ar kitaip fiziškai bandant prieiti prie uždarame luste esančios magistralės) taip pat užkertama. Dėl šių priežasčių renkantis realaus laiko įterptinių sistemų programinės įrangos saugumo platformą nagrinėjamos tik SOC architektūros.

Norint apsaugoti atmintį reikalinga statinė duomenų apsauga – t.y., duomenų (failų, failų sistemų) šifravimas. Tam platforma, kurią renkamasi turi suteikti galimybę aparatūriškai ar programiškai įgyvendinti tokį sprendimą.

Viena populiariausių platformų įterptinių sistemų ir daiktų interneto sprendimų kūrimui ir prototipų realizacijai yra tapęs mikrokompiuteris Raspberry pi. Naujojoje versijoje (Raspberry pi 2, kuriame integruotas Broadcom procesorius) pristatomi iki tol nebuvę saugumo sprendimai. Vienas jų yra duomenų šifravimo galimybė, tačiau už šią paslaugą tenka mokėti. Už papildomą mokestį gamintojas siūlo USB atmintinę, kurioje patalpinami šifravimo raktai, kurių pagalba užtikrinama apsauga, kad tik patikimos šalies pasirašyta programinė įranga gali būti paleista vykdymui mikrokompiuteryje.

Taipogi šiuo metu labai plačiai įvairiems realaus laiko įterptinių sistemų sprendimams yra taikomi ARM architektūros Atmel korporacijos procesoriai. Taip yra dėl nedidelės jų kainos ir siūlomo patrauklaus funkcionalumo, tačiau tokie procesoriai turi pakankamai rimtų saugumo spragų. Skirtingai nuo Raspberry pi, daugelyje šių platformų yra sudaryta šifravimo galimybė, tačiau nerealizuota („pasidaryk pats“ principas“). Joks papildomas mokestis nėra taikomas, o duomenų apsaugojimo sprendimą šifruojant galima padaryti ir remiantis jau esančiais pavyzdžiais ir taipogi papildomai ar unikaliai sudaryti ir savąjį. Paskutiniu metu į prekybą leidžiama nebrangių ir pakankamai galingų Atmel korporacijos ARM926 architektūros procesorių šeima, pavadinta SAM9. Šie procesoriai tinka daugeliui įvairių taikymų, tačiau kaip ir minėta turi pagrindinį trūkumą – nėra įdiegto jokio specialaus saugumą užtikrinančio mechanizmo, tačiau yra šifravimo ir saugios įkrovos galimybės.

SAM9 procesoriaus architektūra pateikta 2.2. pav.




**2.2. pav.** SAM9 procesoriaus architektūra

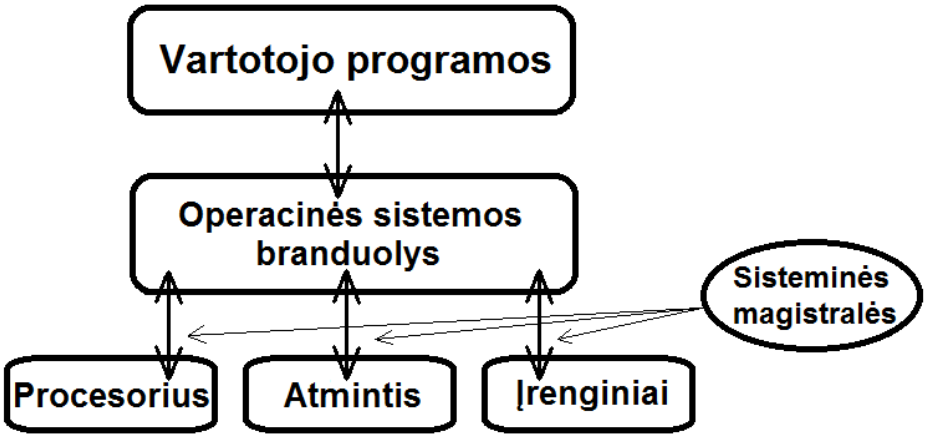
Kaip matyti 2.2. pav. SAM9 šeimos procesoriai yra pakankamai galingi – veikia 400MHZ taktiniu dažniu, pajėgiu atlikti programinį šifravimą, o kai kurie jų turi net kriptografijos modulį, suteikiantį aparatūrines šifravimo galimybes. Dėl nedidelės platformos kainos, santykinai didelio galingumo, ir galimybės pačiam sudaryti saugumo sprendimą, nuspręsta sprendimo kūrimui pasirinkti būtent šią platformą.

## 2.3. Realaus laiko įterptinių sistemų programinės įrangos saugos sprendimo metodo pasirinkimas

Turint platformą, kurioje galima sudaryti saugos sprendimą, reikia parinkti tinkamiausius analizėje apžvelgtus sprendimo metodus. Aparatūrinis saugumas užtikrinamas pasirinktos sprendimo platformos SOC ir keliasluoksne PCB technologijomis, todėl tam, kad būtų išspręsta statinių duomenų saugumo problema ir duota paspirtis tolimesniems tyrimams, susisiaurinama ties programine apsauga, papildomai nenaudojant daugiau aparatinių apsaugos metodų, kurie apsunkintų sprendimo kūrimo procesą ir yra tik papildoma saugumą užtikrinanti, lemiamos įtakos neturinti priemonė. Pagrindinis tikslas yra užtikrinti realaus laiko įterptinių sistemų programinės įrangos duomenų konfidencialumą, integralumą ir vientisumą ir šio tikslo galima pasiekti naudojant programines saugos priemones – duomenų šifravimą, maišos funkcijas ir t. t.

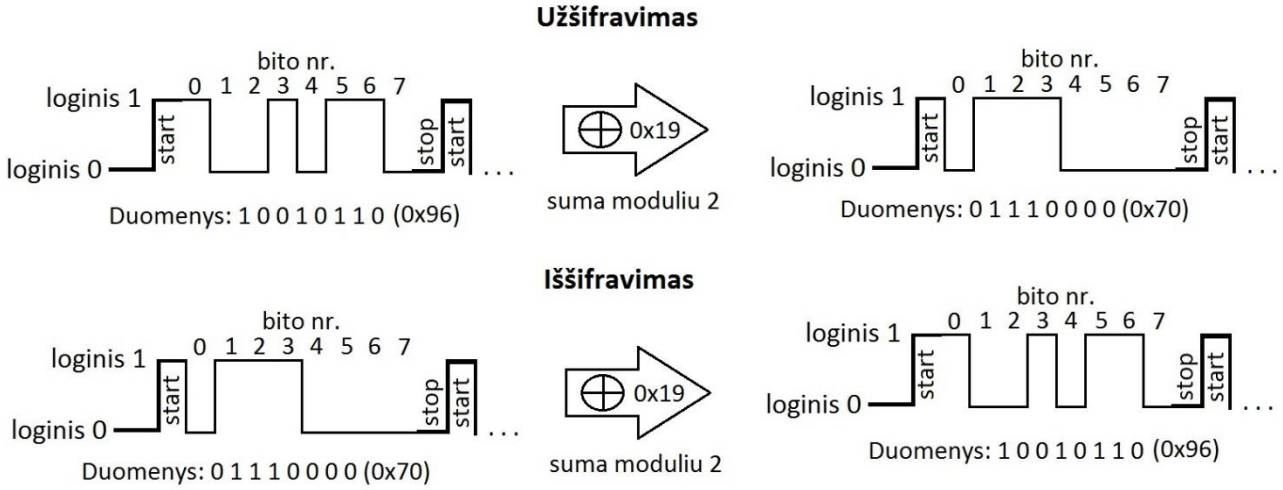
Pasirinktas Atmel SAM9 lustas yra naudojamas įterptinėse sistemose, kuriose veikia operacinė sistema. Kaip minėta, operacinė sistema atlieka tarpininko vaidmenį tarp vartotojo programų ir aparatinės įrangos – šiuo atveju tarpininkauja siunčiant duomenis sisteminėmis magistralėmis. Tarp kiekvienos operacinės sistemos ir sisteminėmis magistralėmis siunčiamų duomenų taip pat egzistuoja tarpininkas – tai operacinės sistemos branduolys (angl. „kernel“). Branduolys yra operacinės sistemos kodo dalis, atsakinga už įvedimo ir išvedimo veiksmų apdorojimą, kai šiuos kviečia vartotojo

programinė įranga. Pvz. Vartotojo programoje paprašius operacinės sistemos išvesti duomenis, esančius atmintyje, šią užduotį vykdyti imasi branduolys paversdamas kvietinį sisteminėmis komandomis. Sisteminės komandos magistralėmis keliauja tarp branduolio ir atmintinės, procesoriaus, įrenginių. Operacinės sistemos duomenų įvedimo ir išvedimo schema pateikta 2.3. pav.




**2.3. pav.** Operacinės sistemos duomenų įvedimo ir išvedimo schema

Žemo lygio duomenų mainų protokolai turi griežtai apibrėžtą fizinę struktūrą, kurios negalima keisti, todėl šifruoti galima tik protokolu keliaujančius duomenis – t.y. negalima pakeisti fizinės struktūros ir laiko diagramų, kad būtų perduodama daugiau/mažiau duomenų bitų ar pakeisti jų perdavimo charakteristikų, turi būti perduodamas griežtai protokolu apibrėžtas bitų kiekis fiksuotais laiko intervalais. Tai reiškia, kad pvz. šifruojant RS232 protokolo duomenis, kurie perduodami 8 bitų sekomis, šie duomenų bitai pagal šifravimo algoritmą turi būti invertuojami taip, kad būtų gauta kita 8 bitų duomenų seka, kuri ir siunčiama sisteminėmis magistrale kaip šifruoti duomenys (kitoje pusėje priimant duomenis jie naudojant tą patį raktą ar pagal atitinkamą algoritmą dešifruojami ir apdorojami). Duomenų šifravimo pavyzdys, naudojant RS-232 protokolą pateiktas 2.4. pav.




**2.4. pav.** RS-232 protokolo duomenų šifravimo pavyzdys

Prieš siunčiant RS-232 protokolu, duomenys užšifruojami atliekant sumą moduliu 2 su nustatyta reikšme (šią reikšmę galima laikyti simetriniu šifravimo raktu). Gavus užšifruotus duomenis, simetrinio šifravimo (tas pats raktas duomenų užšifravimui ir iššifravimui) principu jie vėl sudedami su ta pačia nustatyta reikšme ir gaunama pradinė šifrograma (originalūs duomenys). Šiame pavyzdyje paprastumo dėlei pateiktas simetrinio šifravimo metodas su pačiu primityviausiu algoritmu. Priklausomai nuo turimų resursų ir reikiamo saugumo laipsnio galima šifruoti naudojant įvairius algoritmus: AES, DES, asimetrinio šifravimo (RSA, elipsinių kreivių), bet kaip žinia realaus laiko sistemose labai svarbus aspektas yra laikas – duomenys turi būti perduodami laiku su kuo mažesniais (nustatytais pagal tvarkaraštį didžiausiais leistinais) vėlinimais. Dėl šios priežasties privaloma naudoti tokius šifravimo algoritmus, kurie nesukeltų didesnių, nei toleruotini sistemos vėlinimų.

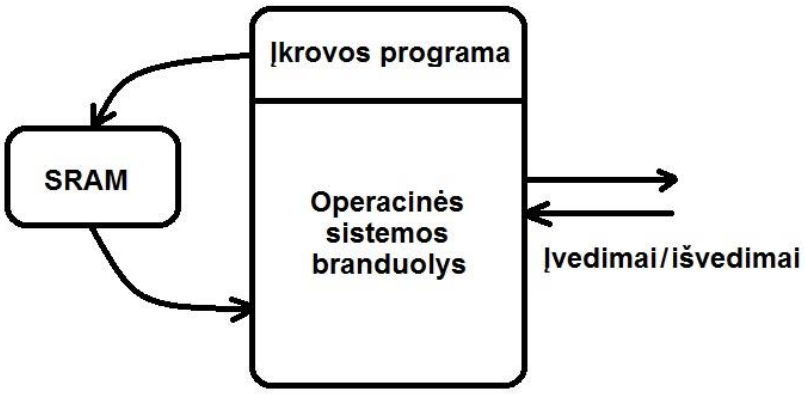
Kitas būdas užšifruoti perduodamus duomenis – vidinis algoritmas, kuris kitaip, nei šifravimo atveju nėra priklausomas nuo šifravimo raktų, o pats savaime tampa raktu. Kodavimo algoritmas įdiegiamas aparatūriškai gamybos metu (pvz. į „tik skaitomą“ atmintinę procesoriaus luste) ar naudojant kitas priemones (pvz. programuotuvą, turintį raktus, leidžiančius perrašyti apsaugotoje procesoriaus flash atmintinėje esantį kodavimo algoritmą). Įdiegtas algoritmas gali atlikti kodavimo vaidmenį arba būti kaip būsenų automatas. Kai kuriose automobilių apsaugos sistemose nuotolinių užrakto komandų siuntimo saugumui užtikrinti naudojamas „KeeLog“ kodavimas, kuris paremtas blokinio šifro rėžimo algoritmu. Tai vienas iš minėtų aparatūrinių kodavimo algoritmų, kada net pasiklausius perduodamų duomenų negalima įsilaužti į sistemą jų atkartojus, kadangi jie kiekvieną kartą keičiami priklausomai nuo sistemos būsenos pagal aparatūriškai užkoduotą algoritmą. Esant tokiai apsaugai ataka įmanoma tik atakuojant patį įrenginį.

Be paminėtų šifravimo, kodavimo algoritmų, visada galima naudoti jų kombinacijas, papildymus ar sukurti savo unikalų duomenų apsaugojimo algoritmą.

Kadangi tyrimas orientuojasi į realaus laiko įterptinių sistemų programinės įrangos saugą, o duomenų mainų saugą nuspręsta užtikrinti operacinės sistemos lygmeniu, siūloma nedaug resursų reikalaujančiu algoritmu užšifruoti visą OS failų sistemą, įtraukiant branduolį, kad sistemą įkrauti ir vykdyti užduotis būtų įmanoma tik turint šifravimo raktą.

Norint paleisti operacinės sistemos branduolį reikalinga tam tikra branduolio įkrovos programa. Paleidžiant kompiuterinę sistemą prieš pradedant krautis operacinei sistemai (branduoliui) pirmiausiai paleidžiama įkrovos programa (angl. „bootloader“). Ši programa tai instukcijų rinkinys, skirtas paleisti operacinės sistemos branduolį. Įkrovos programa yra visiškai priklausoma nuo sistemos aparatinės įrangos (procesoriaus) ir kiekviena motininė plokštė turi savo unikalią įkrovos programą. Įterptinėse sistemose instrukcijų, operacinės sistemos branduolio paleidimui, sudarymui, paprastai naudojama atvirojo kodo įkrovos programa U-BOOT (Universal Bootloader). U-BOOT yra atvirojo kodo produktas, pagrinde naudojamas įterptinėse sistemose komplektuoti instrukcijas operacinės sistemos

branduolio paleidimui (angl. boot kernel). Operacinės sistemos branduolio paleidimo schema pateikta 2.5. pav.




**2.5. pav.** Operacinės sistemos branduolio paleidimo schema

Operacinės sistemos branduolys paleidžiamas per įkrovos programą. Įterptinėse sistemose įkrovos programa arba būna talpinama atskirame bendros atminties skirsnyje arba šalia procesoriaus esančioje „flash“ atmintinėje, naudojant programuotuvą fizinei prieigai. Kartais įkrovos programa būna užrakinta, siekiant uždrausti branduolio perrašymus. Užrakinimas – tai įdiegtas aparatūrinis saugos mechanizmas (hardware DRM). Pvz. kai kurie Intel procesoriai parduodami su „užrakintomis“ savybėmis, kurios už papildomą kainą „atrakinamos“ (kai kurie kompanijos IBM System 370 kompiuteriai turėdavo papildomą aparatūrinę įrangą, kuri nusipirkus produktą būdavo atjungta ir tik sumokėjus papildomą kainą kompanija atsiųsdavo serviso inžinierių, kuris fiziškai pašalindamas atitinkamas varžas „įjungdavo“ papildomą aparatūrą).

Kai kurie įrenginiai palaiko taip vadinamos „saugios įkrovos“ (secure boot) galimybę, kuri kaip jau minėta Raspberry pi 2 atveju leidžia vykdyti tik patikimu parašu pasirašytą programinę įrangą.

Esant užrakintai įkrovos programai, operacinės sistemos branduolio įdiegimo procedūra susideda iš keturių dalių:

1. Įkrovos programos atrakinimas;
2. Saugiklių (angl. „fuses“ nustatymas“);
3. Operacinės sistemos branduolio perrašymas;
4. Įkrovos programos užrakinimas.

Įdiegus šifruotą branduolį, reikia išmatuoti greitaveiką ir palyginti su greitaveika, kai branduolys nebuvo užšifruotas, įvertinti vėlinimą. Įvertinus vėlinimą nustatyti ar jis yra priimtinas – t.y. nustatyti sudaryto saugumo sprendimo efektyvumą. Šis modeliavimas atliekamas prototipo testavimo dalyje.

### 2.4. Realaus laiko įterptinių sistemų programinės įrangos saugos sprendimo išvados

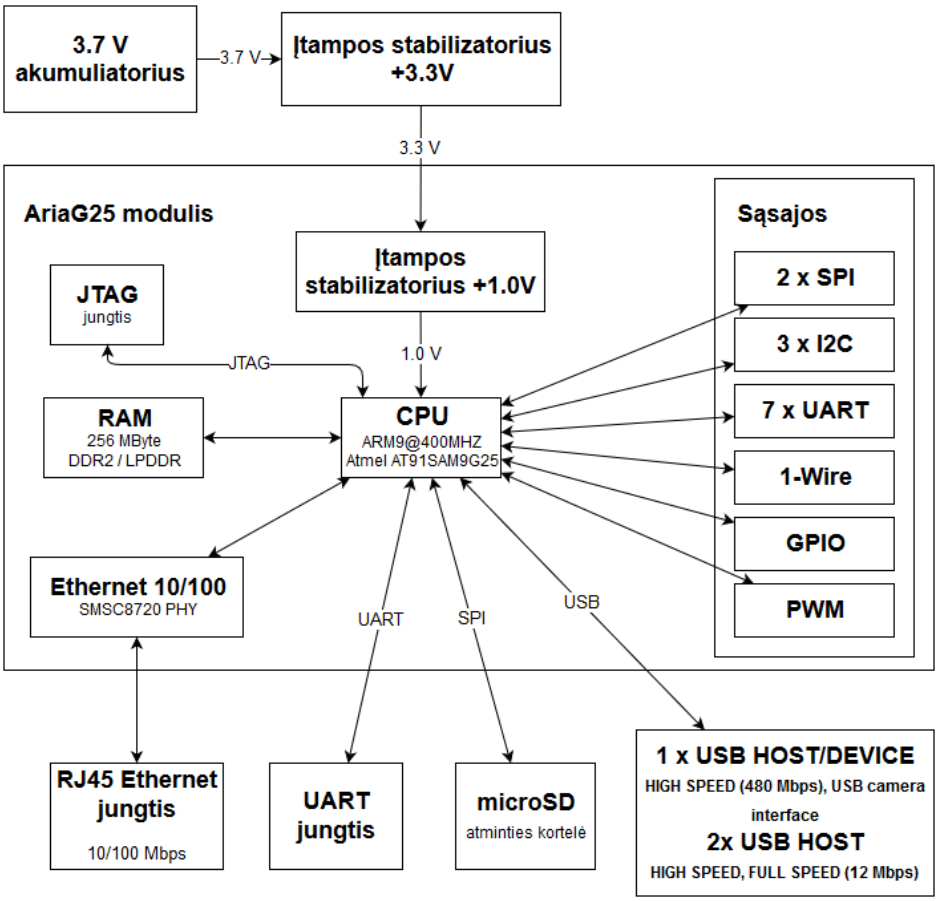
Apžvelgus galimus variantus, realaus laiko įterptinių sistemų programinės įrangos saugos sprendimo plaforma pasirinktas Atmel korporacijos SAM9 lustas dėl savo saugumo problemų, kurias kaip tik ir siekiama išspręsti, nemažo pajėgumo ir šifravimo galimybės. Išsikeltam tikslui – apsaugoti realaus laiko įterptinių sistemų programinės įrangos, užtikrinant jų konfidencialumą, vientisumą ir integralumą, reikalingos programinės priemonės, o aparatinės priemonės galėtų būti kaip papildomas saugumo mechanizmas, tačiau šiame darbe jų toliau nuspręsta nebenagrinėti ir palikti tai tolimesniems tyrimams. Siūlomas saugumo sprendimas pilnai užšifruoti operacinę sistemą, paliekant tik atvirą įkrovos programą. Modeliavimo dalyje laukia svarbus akcentas – išmatuoti šifruotos ir nešifruotos sistemos greitaveikas, jas palyginti ir įvertinti vėlinimus, t.y. nustatyti ar greitaveika priimtina veikiant užšifruotai OS.

# 3. REALAUS LAIKO ĮTERPTINIŲ SISTEMŲ PROGRAMINĖS ĮRANGOS SAUGOS SPRENDIMO PROTOTIPAS

Atliktos analizės rezultatų pagrindu sudarytas realaus laiko įterptinių sistemų programinės įrangos saugumo sprendimas. Šioje dalyje aprašoma sudaryto sprendimo prototipo realizacija, atlikti bandomieji tyrimai, gauti tyrimų rezultatai ir gautų tyrimo rezultatų analizės išvados.

## 3.1. Realaus laiko įterptinių sistemų programinės įrangos saugos sprendimo prototipo aparatūrinė įranga

Realaus laiko įterptinių sistemų programinės įrangos saugos sprendimo prototipą parinkta realizuoti AriaG25 moduliui, turinčiam Atmel AT91SAM9G25 mikroprocesorių. Tokį pasirinkimą nulėmė tai, jog realaus laiko įterptinėse sistemose plačiai naudojami Atmel procesoriai, kurie siūlo patrauklų funkcionalumą už santykinai nedidelę kainą, tačiau nėra realizuoti saugumo sprendimai. AriaG25 modulis yra SoC („system on chip“) tipo – visi komponentai išdėstyti viename luste (procesorius, RAM atmintis, ethernet valdiklis ir kt.). Pasirinktam moduliui pritaikyta pagrindo plokštė, kurioje esamų sąsajų pagalba išvestos reikiamos jungtys komunikacijai. AriaG25 modulio kartu su pritaikyta pagrindo plokšte struktūra pateikta 3.1. pav.



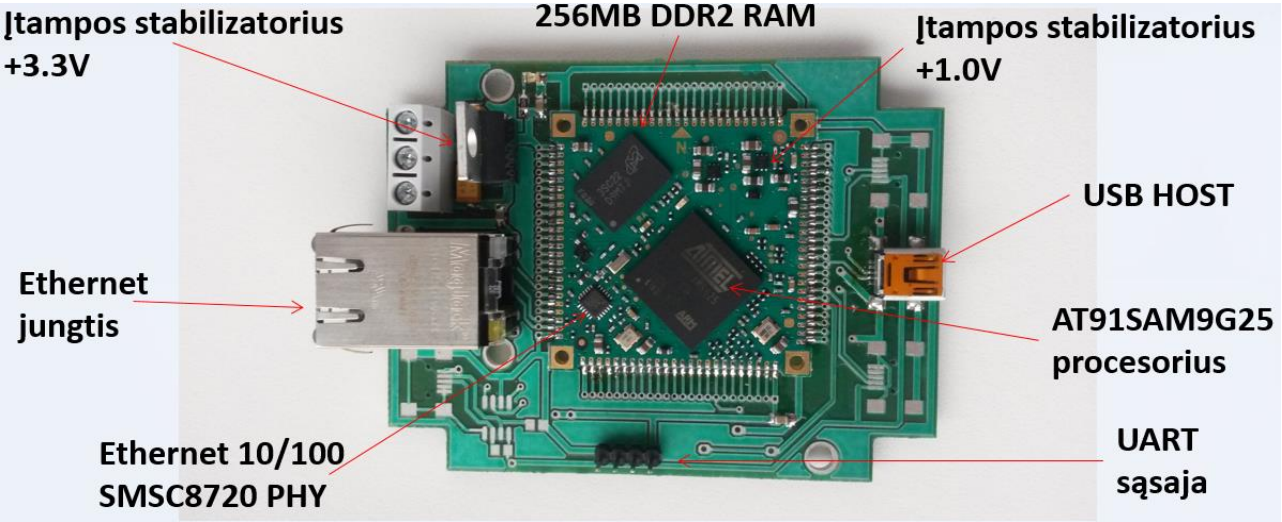

**3.1. pav.** AriaG25 modulio ir jo išvestų sąsajų blokinė diagrama

Modulio paviršiuje išdėstyti pagrindiniai sisteminiai komponentai – ARM architektūros 400MHZ procesorius, DDR2 tipo 256MB operatyvioji (RAM) atmintis ir Ethernet valdiklis.

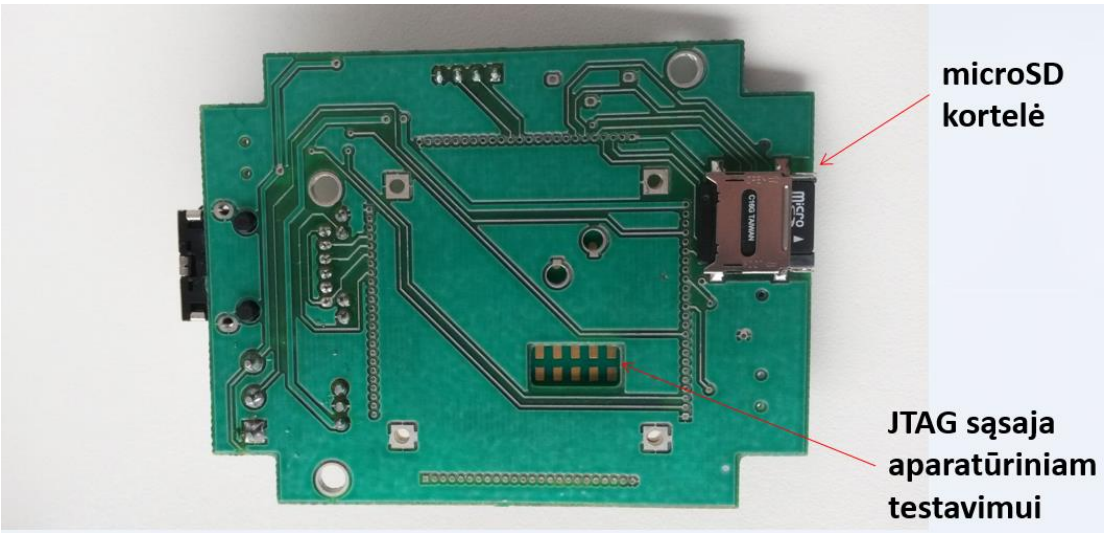
Modulis turi įtampos stabilizatorių, tam, kad procesoriui būtų tiekiama pastovi 1.0V įtampa ir yra išvesta JTAG jungtis aparatūriniam testavimui.

AriaG25 suteikia nemažą kiekį visų plačiai naudojamų sąsajų – SPI, I2C, UART, galimybę prijungti 1-wire protokolu komunikuojančius įrenginius, pagal poreikius konfigūruojamus bendros paskirties išvadus - GPIO (angl. general purpose input/output) ir programiškai emuliuojamus analoginius išvadus – PWM (angl. pulse width modulation). Pritaikytoje pagrindo plokštėje panaudotos esamos sąsajos ir išvesta standartinė „RJ45“ tipo Ethernet jungtis, UART sąsaja, USB jungtys, microSD kortelės jungtis ir sąsaja maitinimo šaltiniui – 3.7V akumuliatoriui, kartu su stabilizatoriumi, išlyginančiu maitinimo įtampą iki stabiliai tiekiamų 3.3V.

Realūs AriaG25 modulio ir pritaikytos pagrindo plokštės vaizdai (nuotraukos su komponentų paaiškinimais) pateikti 3.2. ir 3.3. pav.




**3.2. pav.** AriaG25 modulio ir pritaikytos pagrindo plokštės vaizdas iš viršaus

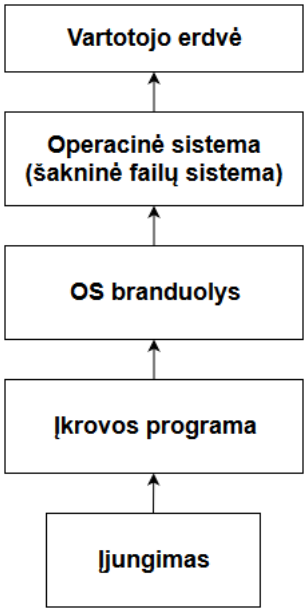



**3.3. pav.** AriaG25 modulio ir pritaikytos pagrindo plokštės vaizdas iš apačios

Pateiktos abiejų plokštės pusių nuotraukos, nurodant, kuriose vietose realiai išdėstyti elementai, atvaizduoti blokinėje diagramoje (3.1. pav.).

### 3.2. Realaus laiko įterptinių sistemų programinės įrangos saugos sprendimo prototipo operacinė sistema

Kadangi pasirinkta realaus laiko įterptinių sistemų programinės įrangos saugos sprendimo prototipo platforma pagal savo galimybes ir architektūrą skirta veikti operacinės sistemos pagrindu, reikalinga operacinė sistema, suderinta su parinktos sprendimo prototipo platformos įranga. Bendruoju atveju operacinės sistemos įkrovos dalys pavaizduotos 3.4. pav.




**3.4. pav.** Operacinės sistemos įkrovos dalys bendruoju atveju

Žemiausiame lygmenyje yra techninė įranga (3.1. pav.). Pirmas dalykas, kuris įvyksta įjungus sistemą – pradeda veikti įkrovos programa. Įkrovos programa labai glaudžiai susijusi su esama aparatine įranga ir yra reikalinga tam, kad būtų galima paleisti tą patį operacinės sistemos branduolį, esant skirtingiems aparatinės įrangos komponentams. Įkrovos programa paruošia reikalingą konfigūraciją ir įkelia į RAM atmintį operacinės sistemos branduolį. Operacinės sistemos branduolys paleidžia reikiamus modulius ir taip sudaro sąlygas startuoti pačiai operacinei sistemai (šakninei failų sistemai), kurios paleidimo metu įkeliama ir vartotojo erdvė. Vartotojo erdvė – tai operacinės sistemos dalis, kurioje vykdomos vartotojo programos. Ši erdvė yra atskirta nuo OS branduolio.

Iš 4.4. pav. matyti, kad operacinės sistemos sudarymui reikalingi trys komponentai:

1. Įkrovos programa;
2. OS branduolys;
3. Operacinė sistema.

Kiekvienas iš šių elementų sudaromas pagal tą pačią seką, kurios schema pateikta 3.5. pav.


**3.5. pav.** Operacinės sistemos komponentų sudarymas

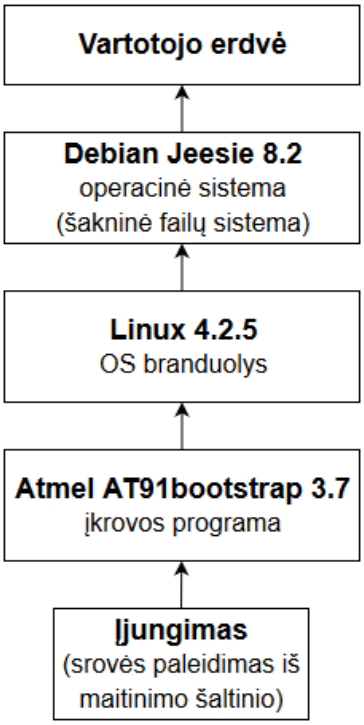
Iš oficialių šaltinių atsisiunčiamas komponentų programinis kodas. Kartu su programiniu kodu (arba papildomai) atsiunčiami ir pritaikomi pataisymai pradiniam kodui. Turint pataisytą kodą atliekama sistemos konfigūracija esamai įrangai. Paskutinis žingsnis – kompiliavimas. Sukompiliavus programinį kodą gaunami operacinės sistemos komponentų failai.

Linux branduolio 3.11 ir naujesnėms versijoms AriaG25 moduliui yra sudaryta „Atmel AT91boostrap“ įkrovos programa, kurios programinį kodą galima nemokamai atsisiųsti ir sukompiliuoti. Sukompiliavus gaunamas failas, pavadinimu „boot.bin“. Naudojant „AT91bootstrap“ galima paleisti ir antrojo lygmens įkrovos programą, pavyzdžiui, plačiai naudojamą „U-Boot“.

Analogiškai atsisiunčiamas linux branduolio programinis kodas, pritaikomi pataisymai, nustatoma konfigūracija ir atliekama kompiliacija. Sukompiliavus gaunami du failai - įrenginių medžio failas (Device Tree Blob file - .dtb) ir branduolio programinis kodas, kuris talpinamas dvejetainiame suspaustame atvaizdžio faile, pavadinimu „zImage“.

Šakninės failinės sistemos sukūrimui reikalingas papildomas kompiuteris su įdiegta linux „Ubuntu“ operacine sistema. Prieš generuojant atvaizdį atliekama konfigūracija, kurios metu nustatomas šakninio „root“ vartotojo slaptažodis ir SSH prisijungimo galimybė. Panaudojamas „multistrap“ įrankis šakninės failų sistemos turinio generavimui. Atsisiunčiamas modulio konfigūracijos failas ir sugeneruojama šakninė failinė sistema iš operacinės sistemos programinio kodo. Pasirinkta Debian Jeesie operacinė sistema dėl savo plačių panaudojimo galimybių ir suteikiamo lankstumo.

Turint visus tris reikiamus operacinės sistemos komponentus galima patikslinti 3.4. pav. pateiktą diagramą, pateikiant konkrečius naudojamus elementus. AriaG25 operacinės sistemos įkrovos dalys pateiktos 3.6. pav.




**3.6. pav.** AriaG25 operacinės sistemos įkrovos dalys

Tam, kad AriaG25 modulis galėtų paleisti sudarytą operacinę sistemą, sukompiliuoti failai talpinami moduliui prieinamoje pastoviojoje atmintinėje. Įprasta, kad įkrovos programa saugoma šalia procesoriaus esančioje „flash“ tipo atmintinėje, tačiau iš 3.1. pav. matyti, kad pasirinktas modulis šios atmintinės neturi. Vienintelė pastovioji atmintinė – microSD kortelė, įstatyta į pagrindo plokštėje išvestą jungtį. Sudarytus tris operacinės sistemos komponentus reikia įkelti į microSD kortelę pagal tam tikrą tvarką:

- reikalingi bent du skirsniai:
    - FAT16 formato 128MB skirsnis kortelės pradžioje, pavadinimu „boot“;
    - EXT4 formato bent 800MB antrasis skirsnis, pavadinimu „rootfs“;
- įkrovos programa ir operacinės sistemos branduolys keliami į pirmąjį skirsnį („boot“);
- Operacinės sistemos failai keliami į antrąjį skirsnį („rootfs“);

Atsižvelgiant į reikalavimus, matoma, kad norint patalpinti operacinės sistemos veikimo failus atminties kortelėje reikia 1GB ar daugiau vietos. Turint didesnės talpos kortelę, be šių skirsnių, reikalingų operacinės sistemos veikimui, likusią kortelės vietą galima padalinti, sudarant daugiau papildomų skirsnių. Siekiant padidinti šifravimo galimybes, išskiriamas dar vienas papildomas skirsnis.

Operacinės sistemos patalpinimui panaudota 16Gb microSD kortelė. Disko dalijimą į skirsnius ir formatavimą atlieka nemokamas įrankis, veikiantis linux aplinkoje - „GParted“. Sukurtų skirsnių struktūros vaizdas, matomas „GParted“ programoje, pateiktas 3.7. pav.


**3.7. pav.** GParted microSD skirsniai

Sukurti du skirsniai operacinės sistemos failams ir trečiasis vartotojo reikmėms. Kuriant antrąjį kortelės skirsnį („rootfs“), palikta 4GB neišskirtos vietos. Ši vieta panaudojama kuriant trečiąjį EXT4 formato kortelės skirsnį vartotojo reikmėms ir šifravimui, pavadinimu „enc“.

Sudarius ir suformatavus skirsnius bei įkėlus reikiamus failus į atminties kortelę, ji įstatoma į išvestą jungtį AriaG25 modulio pagrindo plokštėje ir prijungiamas akumuliatorius kaip maitinimo šaltinis. Prijungus akumuliatorių prasideda linux sistemos krovimas, tai indikuoja ir žalias mirksintis LED diodas specialiai suprojektuotas AriaG25 modulyje.

### 3.3. Realaus laiko įterptinių sistemų programinės įrangos saugos sprendimo prototipo sukūrimas

Realaus laiko įterptinių sistemų programinės įrangos saugos sprendimo prototipo kūrimas atliekamas sudarytoje platformoje. Sprendžiama problema reikalauja programinės įrangos duomenų apsaugojimo. Programinės įrangos apsaugai aparatūriniai sprendimai netaikomi, nes iš dalies šis klausimas išspręstas pačios AriaG25 modulio architektūros – 8 PCB (angl. printed circuit board) sluoksnių technologijos. Programiškai saugumo klausimą pasirinkta spręsti šifruojant diską (atminties kortelės turinį). Disko šifravimo technologija suteikia duomenų apsaugą atminties kortelės perėmimo atveju. Atminties kortelę galima prijungti ir atjungti nuo sistemos, o aparatūriškai uždraudus atminties kortelės išėmimo ar pakeitimo galimybę prarandamas lankstumas ir sutrumpėja sistemos ilgaamžiškumas dėl riboto „flash“ tipo atmintinės skaitymų/rašymų ciklų skaičiaus, po kurio ji tampa nebetinkama naudoti. Pasibaigus atminties kortelės gyvavimo laikui, jei duomenys kortelėje yra užšifruoti ir nėra pridėta aparatūrinė kortelės pakeitimo apsauga, ją galima pakeisti nauja, o sistema sėkmingai gali veikti toliau. Priešingu atveju, tektų pašalinti aparatūrinę kortelės pakeitimo apsaugą, o jeigu to nepavyksta padaryti, keisti visą įterptinę sistemą.

### 3.3.1. Prisijungimas prie realaus laiko įterptinės sistemos OS

Paleidus AriaG25 modulyje sudarytą linux operacinę sistemą, sekantis žingsnis yra prisijungimas. Pagrindo plokštėje išvestos dvi sąsajos – UART ir ethernet (3.1. pav.). Kuriant operacinės sistemos atvaizdį sukonfigūruotas SSH prisijungimas ir sistemos šakninio („root“) vartotojo paskyros slaptažodis. Prie realaus laiko įterptinės sistemos OS galima prisijungti per abi išvestas sąsajas. Jungiantis UART sąsaja terminale nustatomas sukonfigūruotas komunikacijos dažnis (115200), o jungiantis SSH nurodomas AriaG25 modulio IP adresas (IP adresą galima sužinoti „ping“ komandos pagalba, maršrutizatoriaus įrenginių lentelėje ar tinklą skenuojančių programų pagalba) ir SSH prievadas (22). Paleidus sistemą prašoma įvesti vartotojo vardą ir slaptažodį. Prisijungiama su operacinės sistemos atvaizdžio konfigūracijos metu sukurtais „root“ vartotojo duomenimis.

### 3.3.2. Realaus laiko įterptinės sistemos šifravimo galimybės

AriaG25 modulyje integruotas AT91SAM9G25 procesorius. Tos pačios linijos AT91SAM9G46 modelio procesorius turi aparatūrinio AES/DES šifravimo galimybę, kurios AriaG25 modulis neturi, tačiau yra žymiai brangesnis, be to, sudarytas programinis šifravimo sprendimas tinka visoms sistemoms, o sudaryto aparatūrinio šifravimo sprendimo pritaikymas būtų galimas tik tokiose sistemose, kurios turi šią galimybę. Dėl šios priežasties sudaromas programinis disko šifravimo sprendimas. Egzistuoja ne vienas programinio šifravimo variantas, veikiantis linux aplinkoje:

- TrueCrypt – realaus laiko šifravimas;
- EncFS – vartotojo erdvės šifravimas;
- eCryptfs – organizacijų kriptografinė failų sistema;
- LUKS – „The Linux Unified key Setup“ ir „dm-crypt“ – nuo platformos nepriklausomas standartas;
- ...

Kadangi LUKS šifravimo ir dm-crypt palaikymas jau įkompiliuotas daugumoje linux operacinių sistemų branduolių, yra pilno sistemos šifravimo galimybė ir santykinai nemažas šifravimo ir dešifravimo operacijų greitis, naudojimui pasirinkta ši programinio šifravimo technologija.

dm-crypt yra linux branduolio „įrenginių žemėlapio“ (device-mapper) kriptografijos modulis. Įrenginių žemėlapis yra linux infrastruktūra, suteikianti būdą sukurti virtualius sluoksnius blokiniams įrenginiams. Įrenginių žemėlapio kriptografijos modulis suteikia blokinių įrenginių realaus laiko šifravimo galimybę naudojant linux branduolio kriptografijos API (angl. application programming interface). Naudojant dm-crypt galima rinktis vieną iš kelių suteikiamų simetrinių šifravimo algoritmų, šifravimo rėžimą, rakto ilgį (galima rinktis vieną iš leidžiamų ilgių), generatoriaus rėžimą ir kt.. Vartotojui suteikiama galimybė sukurti naują blokinį įrenginį „/dev“ kataloge. Rašymo operacijos į šį įrenginį yra užšifruojamos, o skaitymo dešifruojamos. Šis įrenginys prijungiamas prie failų sistemos kaip ir bet koks kitas įrenginys.

„Cryptsetup“ yra komandinės eilutės „dm-crypt“ sąsaja šifruotų įrenginių kūrimui, prieigai ir valdymui. „Cryptsetup“ turi LUKS plėtinį, kuris visą dm-crypt nustatymų informaciją išsaugo diske automatiškai ir suabstraktina skirsnių bei šifravimo raktų valdymą, suteikdamas naudojimui patogumo. Cryptsetup paketo, naudojamoje Debian operacinėje sistemoje, įdiegimui panaudojama komanda:

```
# apt-get install cryptsetup
```

„Cryptsetup“ skirtas naujo „dm-crypt“ įrenginio LUKS šifravimo rėžimu sudarymui „luksFormat“ formate. Nors pavadinimo dalyje figūruoja žodis „format“, tačiau įrenginys nėra formatuojamas. Nustatoma tik LUKS įrenginio antraštė ir atsižvelgiant į nustatymus įrenginys užšifruojamas. Numatytuoju rėžimu užšifravimo veiksmas atliekamas naudojant komandą:

```
# cryptsetup -v luksFormat <device>
```

Parametras <device> nurodo šifruojamą blokinį įrenginį. Vykdant komandą galima pasirinkti nustatymus:

- šifravimo algoritmą:
    - AES - Advanced Encryption Standard - FIPS PUB 197;
    - twofish - Twofish: A 128-Bit Block Cipher;
    - serpent;
    - cast5;
    - cast6;
- tipą:
    - LUKS;
    - plain – naudoti „dm_crypt“ plain rėžimu;
    - loopaes – loopaes legacy rėžimas;
    - tcrypt – suderinamas su truecrypt rėžimas;
- maišos funkciją:
    - hash-spec;
    - sha1;
    - sha256;
    - sha512;
    - ripemd160;
- rakto ilgį (šifruotas blokinis įrenginys apsaugomas raktu, kuriuo gali būti slaptažodis (iki 512 simbolių) arba rakto failas (iki 8192kB));
- kt.

Šifravimo komandos pavyzdys, nustatant šifravimo parametrus:

```
# cryptsetup -v --cipher aes-xts-plain64 --key-size 256 --hash sha256 luksFormat
<device>
```

### 3.3.3. Realaus laiko įterptinės sistemos atminties kortelės skirsnio užšifravimas

Prototipo sudarymui šifruojamas specialiai vartotojo reikmėms sudarytas microSD kortelės skirsnis, pavadinimu „enc“ (3.7. pav.). Prijungtų įrenginių peržiūrai linux sistemoje naudojama komanda:

```
# fdisk –l
```

Taipogi prijungtų įrenginių peržiūrai galima naudoti ir programinius įrankius (pvz. GParted). Šifravimui paskirtas „/dev/sdb3“ skirsnis. Šio skirsnio užšifravimui pagal numatytuosius „cryptsetup“ nustatymus panaudojama komanda:

```
# cryptsetup -y -v luksFormat /dev/sdb3
```

Šifravimo komandos rezultatai:

```
WARNING!
========
This will overwrite data on /dev/sdb3 irrevocably.

Are you sure? (Type uppercase yes): YES
Enter LUKS passphrase:
Verify passphrase:
Command successful.
```

Perspėjama, kad visi duomenys skirsnyje bus negrįžtamai ištrinti. Paprašoma įvesti, pakartoti šifravimo slaptažodį ir skirsnis užšifruojamas. Įrenginio prijungimas sukuriamas naudojant komandą:

```
# cryptsetup luksOpen /dev/sdb3 enc
```

enc – prijungimo vardas. Įvykdžius komandą paprašoma įvesti sudarytą slaptažodį ir sukuriamas įrenginio prijungimas „/dev/mapper/enc“. Sukurto įrenginio prijungimo peržiūrai naudojama komandą:

```
# ls -l /dev/mapper/enc
```

Prijungto įrenginio peržiūros komandos rezultatai:

```
lrwxrwxrwx 1 root root 7 Jan 12 14:11 /dev/mapper/enc -> ../dm-0
```

Įrenginio prijungimo rezultatų atvaizdavimui naudojama komanda:

```
# cryptsetup -v status enc
```

Įrenginio prijungimo rezultatai:

```
/dev/mapper/enc is active.
  type:    LUKS1
  cipher:  aes-xts-plain64
  keysize: 256 bits
  device:  /dev/enc
  offset:  4096 sectors
  size:    8187904 sectors
  mode:    read/write
Command successful.
```

LUKS antraštės informacijos atvaizdavimui naudojama komanda:

```
# cryptsetup luksDump /dev/enc
```

LUKS antraštės atvaizdavimo komandos rezultatas:

```
LUKS header information for /dev/sda3

Version:        1
Cipher name:    aes
Cipher mode:    xts-plain64
Hash spec:      sha1
Payload offset: 4096
MK bits:        256
MK digest:      aa 48 06 c7 fa 51 ff 23 a5 5e 30 48 d4 f6 58 75 b3 27 a3 da
MK salt:        86 fe a3 8b 5e 63 3e 69 3c c7 71 f4 7c f4 5c 55
                c6 f5 45 d6 3e c4 e0 d0 6d 48 f3 33 59 c0 cc 30
MK iterations:  105875
UUID:           07030a47-08a1-4850-badf-83435c1402f5

Key Slot 0: ENABLED
        Iterations:             423840
        Salt:                   d8 6d 38 b0 31 43 b8 f2 d1 c5 c6 fb 83 8a 98 4c
                                05 a3 9b 41 d3 55 c7 9a d2 8d 93 50 e1 5e 00 78
        Key material offset:    8
        AF stripes:             4000
Key Slot 1: DISABLED
Key Slot 2: DISABLED
Key Slot 3: DISABLED
Key Slot 4: DISABLED
Key Slot 5: DISABLED
Key Slot 6: DISABLED
Key Slot 7: DISABLED
```

Pradžioje visa užšifruoto įrenginio atmintis turi būti užpildyta nuliais. Tai užtikrina, kad iš išorės duomenys matomi tik kaip atsitiktinė seka ir nepaliekama galimybė atkurti duomenų pagal pasikartojančius šablonus. Užpildymas nuliais atliekamas naudojant komandą:

```
# dd if=/dev/zero of=/dev/mapper/enc
```

Komandos, užpildančios atmintį nuliais, rezultatas:

```
8187905+0 records in
8187904+0 records out
4192206848 bytes (4.2 GB) copied, 3933.76 s, 1.1 MB/s
```

EXT4 failų sistemos sukūrimui šifruotame įrenginyje naudojama komanda:

```
# mkfs.ext4 /dev/mapper/backup2
```

Failų sistemos kūrimo komandos rezultatas:

```
mke2fs 1.42.12 (29-Aug-2014)
Creating filesystem with 1023488 4k blocks and 256000 inodes
Filesystem UUID: 9dbd51ee-31f9-4129-acd6-3ca35cd0d9fa
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912, 819200, 884736

Allocating group tables: done
Writing inode tables: done
Creating journal (16384 blocks): done
Writing superblocks and filesystem accounting information: done
```

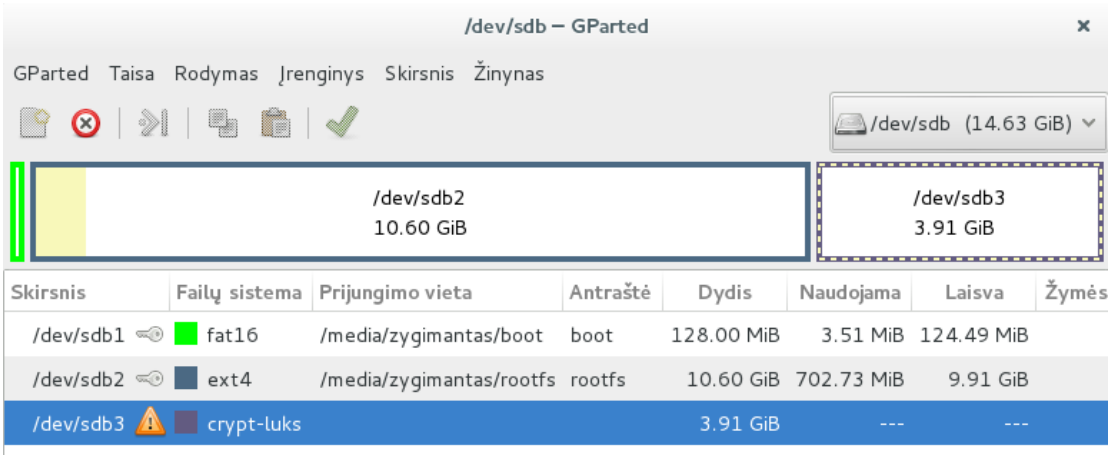
Sukurta failų sistema prijungiama prie OS šakninės failų sistemos. Sukuriamas katalogas „/enc“ ir naudojant komandą „mount“ atliekamas failų sistemos prijungimas:

```
# mount /dev/mapper/enc /enc
```

Prijungimo patikrinimui naudojama komanda „df –H“, pateikianti informaciją apie failų sistemas. Įvykdžius komandą matomas rezultatas:

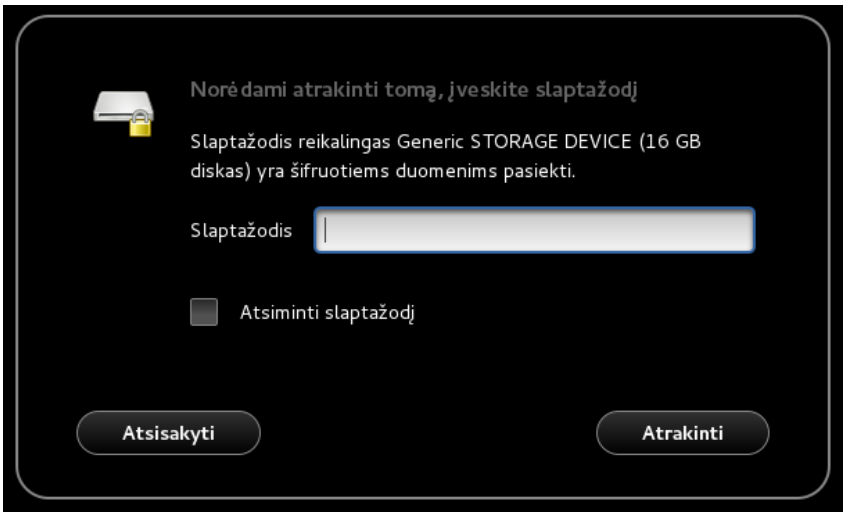
```
Filesystem      Size  Used Avail Use% Mounted on
/dev/mapper/enc  4.1G  8.2M  3.9G   1% /enc
```

Užšifruoto skirsnio prijungimui prie sistemos sudaromas „shell script“, kurį nustatoma vykdyti OS įkrovos metu, kad nebereikėtų kiekvieną kartą naudoti tas pačias komandas. Užšifruoto skirsnio vaizdas GParted programoje pateiktas 3.8. pav.




**3.8. pav.** šifruotas skirsnis GParted programoje

Grafinėje Linux aplinkoje norint skaityti duomenis iš prijungtos atminties kortelės pateikiamas langas, kuriame prašoma įvesti slaptažodį duomenų atrakinimui (dešifravimui) (3.9. pav.).




**3.9. pav.** šifruoto skirsnio atrakinimas grafinėje linux aplinkoje

Užšifruotame skirsnyje esantys statiniai duomenys apsaugomi, užtikrinamas jų konfidencialumas, vientisumas ir integralumas.

### 3.3.4. Realaus laiko įterptinės sistemos šakninės failų sistemos užšifravimas

Užšifravus vartotojo reikmėms sudarytą microSD kortelės „enc“ skirsnį, analogiškai užšifruojanas ir šakninės failų sistemos skirsnis „rootfs“, tokiu būdu paliekant neužšifruotą tik įkrovos programą ir operacinės sistemos branduolį (esančius atskirame skirsnyje, atminties kortelės pradžioje), per kurį šakninė failų sistema atrakinama. Užšifravus šakninę failų sistemą ir netaikant papildomų modifikacijų operacinė sistema įkrovos metu nesugebės perskaityti šifruotų failų ir įkrovos procesas sustos. Tam, kad operacinė sistema galėtų įsikrauti, naudojant šifruotą šakninę failų sistemą, reikia atlikti „initramfs“ modifikaciją (įdėti atrakinimą). „initramfs“ (trumpinys nuo „initial RAM file system“) yra suarchyvuota pradinė failų sistema, kuri nukopijuojama į atmintį vykstant Linux operacinės sistemos krovimosi procesui. Linux OS branduolys nukopijuoja archyvą į laikiną šakninę failų sistemą ir bando paleisti „/init“ programą, prieš atliekant įprastinius įkrovos procesus. „init“ programa atlieka užduotis prieš prijungiant tikrąją šakninę failų sistemą. Modifikavus „initramfs“ galima nustatyti „rootfs“ atrakinimą prieš prijungimą. Tam reikia atlikti pakeitimus konfigūraciniuose sistemos failuose ir panaudojus naujai sudarytus failus pergeneruoti „initramfs“ archyvą. Initramfs archyvas generuojamas naudojant komandą:

```
mkinitramfs -o /boot/initramfs.gz
```

„initramfs“ archyvas patalpintas „/boot“ kataloge. Pirmiausiai atliekamas pakeitimas tame pačiame kataloge esančiam „config.txt“ failui, jį papildant eilute:

```
initramfs initramfs.gz followkernel
```

Ši eilutė nurodo, kad įkrovos programa sistemos paleidimo metu įkeltų „initramfs“ į atmintį. Kitas konfigūracinis failas, kurį reikia pakeisti yra „/tmp/pi_boot/cmdline.txt“. Šiame faile reikia pakeisti „root=/dev/mmcblk0p2“ į „root=/dev/mapper/užšifruoto_skirsnio_prijungimo_vardas“ ir pridėti nustatymą „cryptdevice=/dev/mmcblk0p2:užšifruoto_skirsnio_prijungimo_vardas“ taip nurodant operacinės sistemos branduoliui, kas yra naujas „root“ skirsnis ir kaip jį pasiekti. Galiausiai reikia pranešti likusiai sistemai apie naują skirsnį. Tam atliekamas pakeitimas „/etc/fstab“ faile perrašant „/dev/mmcblk0p2“ į „/dev/mapper/užšifruoto_skirsnio_prijungimo_vardas“. Šis failas skirtas nurodyti, kaip diskų skirsniai, blokiniai įrenginiai ar kitokio tipo failų sistemos turi būti prijungiamos prie sistemos. Labai panašus į „fstab“ yra „crypttab“ failas („/etc/crypttab“), kuriame saugomas šifruotų įrenginių, atrakinamų sistemos krovimosi metu, sąrašas. Norint atrakinti užšifruotą šakninę failų sistemą krovimosi metu, reikia papildyti „crypttab“ sąrašą, failo pabaigoje prirašant eilutę:

```
užšifruoto_skirsnio_prijungimo_vardas /dev/mmcblk0p2  none    luks
```

Pirmosios įkrovos metu sistema negali atidaryti užšifruoto „root“ skirsnio. Tam reikia turėti tiesioginį priėjimą prie terminalo (nenaudojant SSH prisijungimo), kad būtų galima dirbti su pasirodžiusiu „Busybox recovery shell“. Iš 3.2. ir 3.3. pav. matyti, kad tiesioginis priėjimas prie

terminalo AriaG25 modulyje galimas UART sąsajos pagalba. Užšifruoto „root“ skirsnio atrakinimui tereikia prijungti šakninę failų sistemą ir išeiti iš „Busybox recovery shell“. Terminale pasirodžius „initramfs“ žodžiui įvedamos dvi komandos:

```
cryptsetup luksOpen /dev/mmcblk0p2 užšifruoto_skirsnio_prijungimo_vardas
exit
```

Šakninė failų sistema atrakinama ir OS paleidžiama. Tam, kad kito krovimosi metu būtų rodomas tik pranešimas, prašantis įvesti slaptažodį „rootfs“ atrakinimui, reikia dar kartą pergeneruoti „initramfs“ archyvą.

Aprašytu metodu atlikti šakninės failų sistemos šifravimą tenka papildomu kompiuteriu (analogiškai „enc“ skirsniui nukopijuojant, užšifruojant ir perkeliant atgal šifruotą skirsnį), o jungiantis kiekvieną kartą reikalingas tiesioginis priėjimas prie terminalo slaptažodžio įvedimui (suvedus slaptažodį atrakinama sistema). Tokiu būdu prarandamas lankstumas, nes nėra galimybės atrakinti sistemos nuotoliniu būdu, tačiau tai galima įdiegti papildomai. Norint pridėti galimybę atrakinti sistemą jungiantis „SSH“ protokolu iš išorės, reikalingi dar keli pakeitimai ir prisijungimo raktas. Papildomai reikia įdiegti „dropbear“ – labai mažą SSH serverį, kuris automatiškai integruojasi į „initramfs“, jei aptinka šifruotą sistemos skirsnį. Įdiegus „dropbear“ reikia pergeneruoti „initramfs“ archyvą, kad būtų sugeneruotas „SSH“ raktas „/etc/initramfs-tools/root/.ssh/“ kataloge pavadinimu „id_rsa“. Šį raktą reikia parsisiųsti į kompiuterį, iš kurio jungiamasi nuotoliniu būdu. Pakeitimai reikalingi „initramfs“ archyvui, kad būtų rodomas slaptažodžio įvedimo prašymas, kai tik prisijungiama „SSH“ protokolu prie paleidžiamos sistemos. Failas „/etc/initramfs-tools/root/.ssh/authorized_keys“ papildomas, pradžioje prirašant:

```
command="/scripts/local-top/cryptroot && kill -9 `ps | grep -m 1 'cryptroot' | cut
-d ' ' -f 3`"
```

Ši komanda paleidžia atrakinimo skriptą, kuriam suveikus nutraukiamas skripto vykdymas, kad įkrovos procesas galėtų tęstis toliau. Atlikus pakeitimus reikia dar kartą pergeneruoti „initramfs“ archyvą. Taip sudaroma nuotolinio „SSH“ prisijungimo galimybė iš išorės sistemos atrakinimui. Jungiamasi panaudojant turimą raktą ir paprašius suvedant atrakinimo slaptažodį. Teisingai įvedus slaptažodį atrakinama šakninė failų sistema ir OS paleidžiama.

Aprašytas užšifravimo metodas paremtas laikinosios šakninės failų sistemos įkėlimu į atmintį prieš prijungiant tikrąją. Atrakinimas galimas tiesiogiai jungiantis prie terminalo arba papildomai įdiegtu „SSH“ nuotolinio atrakinimo būdu, turint atrakinimo raktą. Išplečiant atrakinimo galimybes, šifravimo raktas gali būti patalpintas išorinėje USB atmintinėje, tačiau tai tėra alternatyva lokaliam atrakinimui, kada jungiamasi tiesiai prie terminalo. Norint užtikrinti didžiausią saugumą, priklausomai nuo įterptinės sistemos taikymo, fizinės vietos ir prieigos galimybių, galima rinktis kurį nors vieną atrakinimo būdą ar kelių atrakinimo būdų komplektą.

Užšifruotų „enc“ ir šakninės failų sistemos skirsnių greitaveikos patikrinimui sudaroma testinę programą, tikrinanti įvedimo ir išvedimo operacijų greičius.

### 3.3.5. Realaus laiko įterptinės sistemos programinės įrangos saugos sprendimo prototipo testinė programa

Šifravimo dalyje naudota komanda „dd: puikiai tinka matuoti rašymo ir skaitymo operacijų greitį. Šiuo pagrindu galima sudaryti testinę programą, matuojančią sistemos greitaveiką.

Rašymo operacijų greičio patikrinimui galima įrašinėti pasirinkto dydžio nulinius blokus į atmintį ir matuoti laiką, kiek užtrunka operacija. Svarbu, jog bendras įrašomo failo dydis būtų didesnis, negu sistemoje yra RAM atminties, kad būtų išvengiama perkėlimo į šią atmintį ir negautume neadekvačių rezultatų. Sistemoje esančios RAM atminties kiekiui sužinoti linux operacinėje sistemoje naudojama komandą „free“. Kadangi AriaG25 modulis turi 256MB RAM atminties, bendras rašomo failo dydis turi būti didesnis, nei 256MB.

Komandos „dd“ vykdymo laikui išmatuoti galima naudoti kitą linux komandą „date“, pateikiančią esamą sistemos laiką nustatytu formatu. Patikrinus sistemos laiką milisekundėmis prieš vykdant komandą „dd“ ir iškart po komandos įvykdymo galima išmatuoti proceso veikimo trukmę nuo jo pradėjimo iki užbaigimo. Bendra komandos „dd“ struktūra:

```
# dd if=/dev/zero of=[PATH] bs=[BLOCK_SIZE]k count=[LOOPS] && sync
```

**if** – įvesties failas. Kadangi nustatoma „/dev/zero“, tai reiškia, kad vietoje failo kopijuojama nulinių bitų seka;

**of** – išvesties failas (failas, į kurį kopijuojama). Atliekant testą turi būti parinktas failas testuojamąjame skirsnyje;

**bs** – kopijuojamo failo bloko dydis;

**count** – kopijuojamų blokų kiekis;

**sync** – vėliavėlė, nurodanti, kad procesas pirma turi įrašyti visą failą į diską, prieš užbaigdamas savo darbą. Nepanaudojus šios vėliavėlės „dd“ procesas baigtųsi anksčiau (nebaigęs įrašinėti į diską) ir laiko matavimo rezultatai būtų neteisingi (neįtrauktas pilnas sinchronizavimo diske laikas).

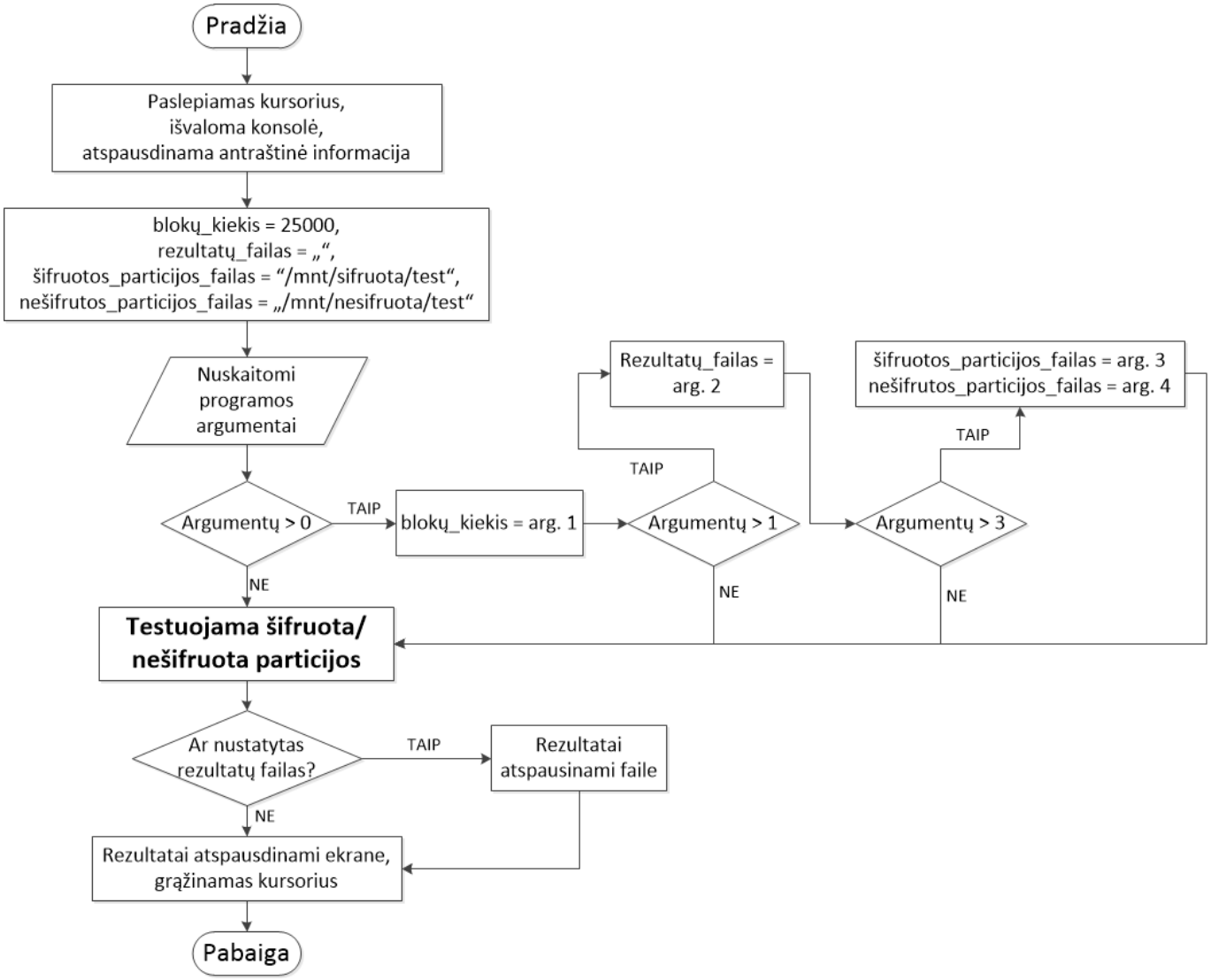
Išmatavus rašymo operacijos greitaveiką, turimas įrašyto failo dydis ir išmatuota rašymo proceso trukmė. Greičio apskaičiavimui galima taikyti paprasčiausią formulę įrašytų_baitų_kiekis / įrašymo_trukmė ir gautą rezultatą paversti KB/s (padalinti iš 1024), MB/s (dar kartą padalinti iš 1024) ir t... Taip gaunamas vidutinis rašymo greitis, gautas testo atlikimo metu.

Tą pačią „dd“ komandą galima naudoti ir skaitymo operacijos greičio matavimui. Kadangi testuojant rašymo operacijos greitį įrašytas failas, didesnis, nei sistemoje esančios RAM atminties dydis, tą patį įrašytą failą testo metu galima ir perskaityti (failas bus pilnai skaitomas iš disko, išvengiama skaitymo iš RAM atminties). Skaitymo atveju komandos „dd“ struktūra trumpesnė:

```
# dd if=[PATH] of=/dev/null bs=[BLOCK_SIZE]
```

Pasirenkama perrašyti tą patį failą, kuris buvo įrašytas į nulinį įrenginį (ekvivalentu skaitymo operacijai). Skaitymo operacijos greitis pagal gautus rezultatus apskaičiuojamas analogiškai kaip ir rašymo atveju.

Skaitymo ir rašymo operacijų greičio testavimui sudaroma programa („shell script“). Programos struktūrinės schemos vaizdas pateiktas 3.10. pav.

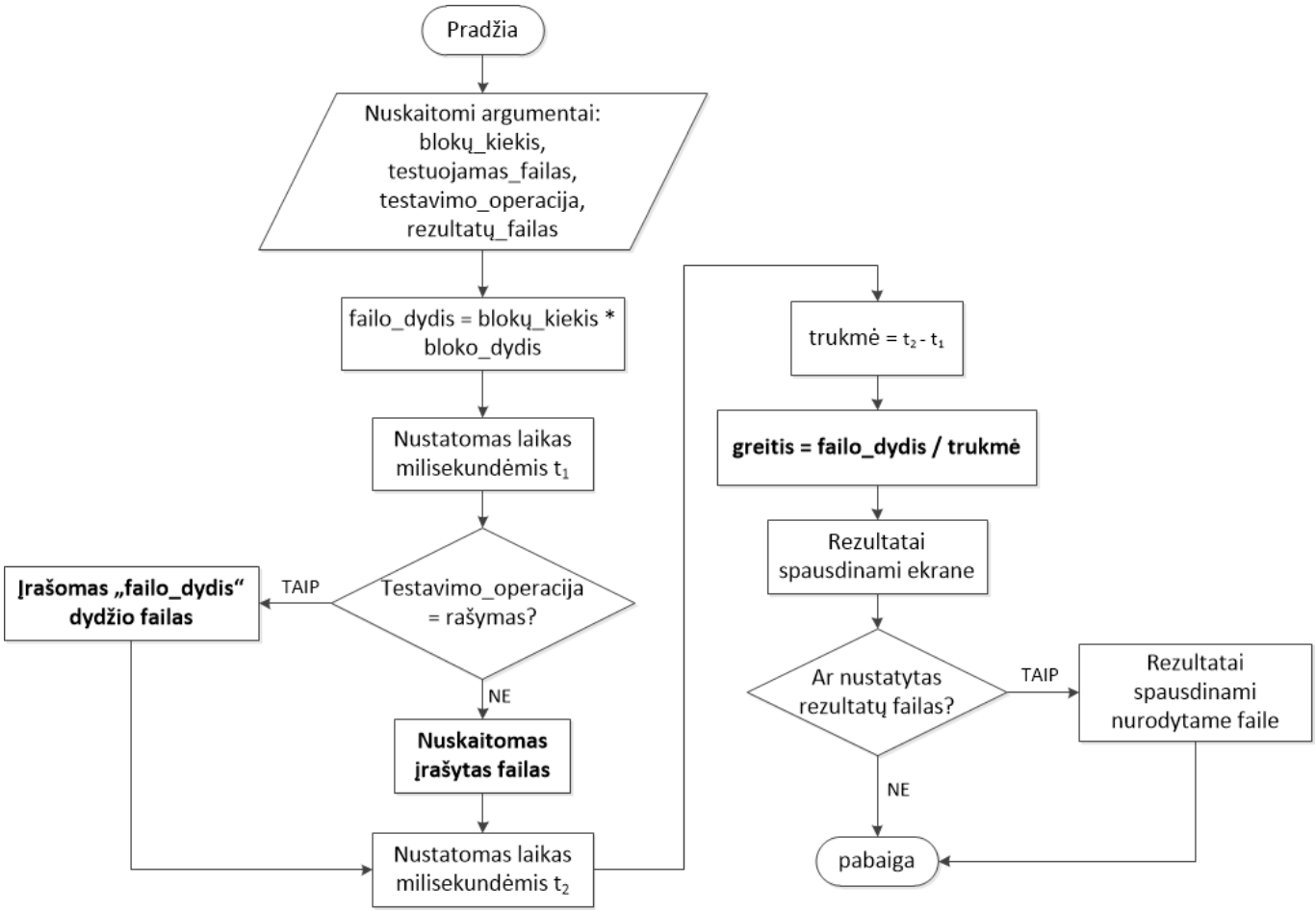



**3.10. pav.** Skaitymo ir rašymo operacijų greitaveikos matavimo testinės programos struktūrinė schema

Programą galima pasirinkti vykdyti nenurodant argumentų, tada pagal nutylėjimą nustatomas blokų kiekis, šifruotos ir nešifruotos particijų testiniai failai ir rezultatų išvedimas tik į ekraną. Vykdant programą su vienu argumentu nurodomas blokų kiekis, nuo kurio priklauso testuojamo failo dydis. Vykdant programą su dviem argumentais, antrasis nustato rezultatus išvesti į nurodytą failą. Vykdant programą su visais keturiais argumentais, paskutiniai du nurodo šifruotos particijos failą, kuris bus testuojamas (3 argumentas) ir nešifruotos particijos failą (4 argumentas).

Programa išjungia kursorių, išvalo konsolę ir pagal nustatytus argumentus atlieka greitaveikos testavimo procedūrą. Baigusi testavimą programa ekrane atspausdina rezultatus, grąžina kursorių ir jei yra nustatyta, lygiagrečiai rezultatai išvedami nurodytame faile. Vykdant programą iš karto testuojamos dvi particijos. Testą galima leisti atskirai vienai particijai naudojant sumažintą testavimo programos versiją.

3.10. pav. pavaizduotoje programos schemoje paryškintas testavimo procesas, kadangi jis surašytas atskirame skripte. Skaitymo ir rašymo greitaveikos matavimo skripto struktūrinė schema pateikta 3.11. pav.




**3.11. pav.** Skaitymo ir rašymo greitaveikos matavimo skripto struktūrinė schema

Testavimo skriptui perduodami keturi argumentai: testuojamo failo blokų kiekis, testuojamo failo pavadinimas ir kelias, testavimo operacija ir rezultatų failas. Pagal blokų kiekį ir fiksuotą vieno bloko dydį nustatomas viso testuojamo failo dydis baitais. Prieš pradedant testą nustatomas sistemos laikas $t_1$ milisekundėmis ir pagal tai, kokia testavimo operacija parinkta – rašymo ar skaitymo, testuojamas arba rašymas į nurodytą failą arba nurodyto failo skaitymas. Baigus operaciją, dar kartą nustatomas laikas $t_2$ ir apskaičiuojama testavimo proceso trukmė $t_2 - t_1$. Turint nustatytą failo dydį ir rašymo ar skaitymo operacijos trukmę galima apskaičiuoti bendrą vidutinį greitį. Gauti rezultatai atspausdinami ekrane ir jei yra nustatytas rezultatų failas, lygiagrečiai išvedami į nurodytą failą.

### 3.4. Realaus laiko įterptinių sistemų programinės įrangos saugos sprendimo prototipo tyrimas

Atliekant tyrimą realiai išbandomos užšifruoto ir nešifruoto skirsnių greitaveikos skirtingose microSD kortelėse įterptinėje realaus laiko sistemoje. Panaudojant aprašytą greitaveikos tyrimo algoritmą (testinę programą) analizuojami realūs duomenys atliekant skaitymo ir rašymo operacijas. Kadangi microSD kortelėje sudaryti trys skirsniai, iš kurių du nepriklauso įkrovos programos ir branduolio skirsniui, galima lyginti skirsnių greitaveikas testuojant su tais pačiais duomenimis. Testuojami „rootfs“ ir „enc“ skirsnių skaitymo ir rašymo operacijų greičiai (šifruotos ir nešifruotos failų sistemos atvejais). Testuojamas atminties kortelės „enc“ skirsnis užšifruotas numatytuoju „aes-xts-plain64“ algoritmu, kurį palaiko dauguma linux operacinių sistemų branduolio versijų. Į užšifruotą ir nešifruotą skirsnius įkeliami 1GB talpos nuliais užpildyti failai (250000 blokų po 4Kb). 3.12. pav. pateikti programos į ekraną ir į rezultatų failą išvesti testavimo rezultatai.

```
/dev/mapper/enc is active and is in use.
  type:    LUKS1
  cipher:  aes-cbc-essiv:sha256
  keysize: 256 bits
  device:  /dev/sdb3
  offset:  4096 sectors
  size:    5728256 sectors
  mode:    read/write
Command successful.

+======================+
| Greitaveikos testas |
+======================+
Testuojama ŠIFRUOTA particija
RAŠYMAS:
Rašoma 1000000000 baitų (~1000 Mb)
Užtruko sekundžių: 245.205 s
================================
 Rašymo greitis: 4.78 Mb/s
================================
SKAITYMAS:
Skaitoma įrašyti 1000000000 baitų (~1000 Mb)
Užtruko sekundžių: 226.436 s
================================
 Skaitymo greitis: 4.416 Mb/s
================================


================================================


Testuojama NEŠIFRUOTA particija
RAŠYMAS:
Rašoma 1000000000 baitų (~1000 Mb)
Užtruko sekundžių: 193.747 s
================================
 Rašymo greitis: 5.161 Mb/s
================================
Testuojama NEŠIFRUOTA particija
SKAITYMAS:
Skaitoma įrašyti 1000000000 baitų (~1000 Mb)
Užtruko sekundžių: 63.260 s
================================
 Skaitymo greitis: 15.807 Mb/s
================================
================================================
Rezultatai atspausdinti faile Rezultatai.txt
```

Rezultatai.txt

```
1  ================================================
2  Šifruota particija
3  Rašymo greitis: 4.78 Mb/s
4  Skaitymo greitis: 4.416 Mb/s
5  ================================================
6  Nešifruota particija
7  Rašymo greitis: 5.161 Mb/s
8  Skaitymo greitis: 15.807 Mb/s
9  ================================================
```

**3.12. pav.** Greitaveikos testavimo programos išvesti rezultatai

Testas atliekamas naudojant tą patį sistemos atvaizdį trijose skirtingos klasės atminties kortelėse.

Galutiniai testo rezultatai pateikti 3.1. lentelėje.

**3.1. lentelė.** Skaitymo ir rašymo operacijų greitaveikos tyrimo rezultatai

| Atmintinės tipas | **Rašymo greitis (Mb/s)** | | **Skaitymo greitis (Mb/s)** | |
|---|---|---|---|---|
| | **Šifruotas skirsnis** | **Nešifruotas skirsnis** | **Šifruotas skirsnis** | **Nešifruotas skirsnis** |
| 4 klasė | 4.0 | 4.4 | 5.2 | 14.5 |
| 6 klasė | 4.78 | 5.161 | 4.416 | 15.807 |
| 10 klasė | 4.64 | 11.5 | 4.95 | 16.3 |

Rezultatai rodo, jog nešifruotuose skirsniuose skaitymo ir rašymo operacijų greitis priklauso nuo atmintinės klasės, o šifruotų skirsnių skaitymo ir rašymo operacijų greitis nukrenta iki 4-5 Mb/s. Mažo greičio atminties kortelėse rašymo greičio skirtumas dar gana nežymus (~10%), tačiau skaitymo greitis sumažėja beveik tris kartus. Didesnio greičio atminties kortelėse (geresnės klasės), skirtumas tarp šifruoto ir nešifruoto skirsnio rašymo operacijų greičio gana didelis. Analogiški rezultatai gaunami ir testuojant užšifruoto ir nešifruoto „root“ skirsnio greitaveiką, be to užšifravus šakninę failų sistemą įkrovos laikas taip pat kelis kartus prailgėja (nuo nepilnos minutės iki dviejų).

Testas atliktas skirsnius užšifravus paprastu AES šifravimo algoritmu. Šifravimą ir dešifravimą galimai pagreitintų aparatūrinė AES šifravimo aksceleracija (papildomas šifravimo modulis) ar paprastesnis šifravimo algoritmas, mažesnis šifravimo rakto ilgis, tačiau pagerėjimas nebūtų esminis arba neproporcingai sumažėtų saugumas.

### 3.5. Realaus laiko įterptinių sistemų programinės įrangos saugos sprendimo prototipo realizacijos ir tyrimo išvados

Sudarytas pasirinktos platformos operacinės sistemos atvaizdis. Atsižvelgiant į reikiamas taisykles atvaizdžio failai patalpinti atminties kortelėje. Išskirtas papildomas atminties kortelės skirsnis, kuris naudojant esamus programinius šifravimo įrankius užšifruotas numatytuoju AES algoritmu. Atliktas tyrimas, testuojant šifruoto ir nešifruoto skirsnių greitaveikas pagal sudarytą metodiką. Tyrimo rezultatai parodė, jog rašymo ir skaitymo operacijos šifruotame skirsnyje sulėtėja iki 4-5Mb/s (nuo 10% iki kelių kartų, priklausomai nuo atminties kortelės klasės). Tai reiškia, jog atsiranda vėlinimas, kurį reikia įvertinti, norint integruoti saugumo sprendimą atitinkamoje realaus laiko sistemoje, be to procesorius viso šifravimo ar dešifravimo proceso metu dirba beveik pilnu pajėgumu, dėl ko pakyla sistemos temperatūra ir suvartojamos galios kiekis. Norint sumažinti vėlinimą ir padidinti procesoriaus našumą, reikalinga didesnė įtampa, kas sutrumpina realaus laiko įterptinės sistemos akumuliatoriaus gyvavimo laiką.

## 4. IŠVADOS

Parinktas ir ralioje platformoje pritaikytas realaus laiko įterptinių sistemų programinės įrangos saugumo sprendimas. Atliktas realizuoto sprendimo prototipo greitaveikos tyrimas parodė, jog integruotas saugumo sprendimas ganėtinai stipriai įtakoja bendrą sistemos našumą.

Esant priimtinam vėlinimui atitinkamoje realaus laiko sistemoje analogiškai išskirto „enc“ skirsnio ir šakninės failų sistemos pagrindu galima užšifruoti ir OS branduolį, kuris patalpintas „boot“ skirsnyje atminties kortelės pradžioje (atlikti pilną sistemos šifravimą), paliekant atvirą tik įkrovos programą, per kurią sistema būtų atrakinama. Tai galima pasiekti patalpinant šakninę failų sistemą ir „boot“ į vieną skirsnį, naudojant kelių lygmenų įkrovos programą su pritaikytomis reikiamomis pataisomis. Kelių lygmenų įkrovos programa reiškia, jog sistemoje paleidžiama universalioji įkrovos programa U-BOOT, kuri įdiegus reikalingus pataisymus pati savaime palaiko LUKS šifravimą. U-BOOT, paleidus įkrovos programą, randa LUKS particiją ir prašo įvesti slaptažodį. Teisingai suvedus slaptažodį sistema atrakinama ir nuo šios vietos toliau įkrovos procesas vyksta įprastai. Tokiu būdu nebelieka nešifruoto OS branduolio ir „initramfs“, nešifruoti lieka tik keli baitai atminties pradžioje, kuriuose yra failų sistemos lentelės įrašai ir įkrovos programos kodas, tačiau saugumo atžvilgiu iš to papildomai nieko negaunama, tik du atskiri skirsniai apjungiami į vieną (nebelieka „boot“ skirsnio). Tai gali būti naudinga nebent tuo atveju, jei norima sistemą paleisti naudojant mažą USB raktą (priešingu atveju, norint paleisti sistemą, naudojant USB raktą, jame turi būti patalpintas visas „boot“ skirsnis), kurį galima pašalinti po sistemos paleidimo.

Prieigos ir atrakinimo galimybės įdiegiamos pagal poreikį, svarbiausia problema išlieka greitaveikos klausimas. Norint diegti įterptinę realaus laiko sistemą, naudojant pasiūlytą saugumo sprendimą, būtina įvertinti vėlinimus ir energijos suvartojimą, nustatant ar tai priimtina diegiamoje sistemoje, nes failinės sistemos šifravimas ir dešifravimas yra nemažai resursų naudojanti užduotis, o resursai realaus laiko įterptinėse sistemose yra kritiškai riboti.

## 5. LITERATŪROS SĄRAŠAS


[1] I. Crnkovic, „Component-based Software Engineering for Embedded,“ įtraukta *Ninth International Workshop on Component-Oriented Programming*, Oslo, 2004.

[2] „ACME systems,“ [Tinkle]. Available: http://www.acmesystems.it/. [Kreiptasi 27 01 2016].

[3] „Atmel AT91SAM9G25 parameters,“ [Tinkle]. Available: http://www.atmel.com/devices/SAM9G25.aspx?tab=parameters. [Kreiptasi 27 01 2016].

[4] „dm-crypt,“ [Tinkle]. Available: https://wiki.archlinux.org/index.php/Dm-crypt. [Kreiptasi 29 01 2016].

[5] „dm-crypt,“ [Tinkle]. Available: https://gitlab.com/cryptsetup/cryptsetup/wikis/DMCrypt#iv-generators. [Kreiptasi 29 01 2016].

[6] „File encryption software in open source,“ [Tinkle]. Available: http://www.linuxuser.co.uk/reviews/the-best-file-encryption-software-in-open-source. [Kreiptasi 29 01 2016].

[7] „dm-crypt Device encryption,“ [Tinkle]. Available: https://wiki.archlinux.org/index.php/Dm-crypt/Device_encryption. [Kreiptasi 29 01 2016].

[8] „Linux Hard Disk Encryption With LUKS,“ [Tinkle]. Available: http://www.cyberciti.biz/hardware/howto-linux-hard-disk-encryption-with-luks-cryptsetup-command/. [Kreiptasi 29 01 2016].

[9] „LUKS On-Disk Format Specification,“ [Tinkle]. Available: http://wiki.cryptsetup.googlecode.com/git/LUKS-standard/on-disk-format.pdf. [Kreiptasi 29 01 2016].

[10] „Benchmark disk IO with DD,“ [Tinkle]. Available: http://www.jamescoyle.net/how-to/599-benchmark-disk-io-with-dd-and-bonnie. [Kreiptasi 29 01 2016].

[11] „Engscope,“ [Tinkle]. Available: http://www.engscope.com. [Kreiptasi 07 11 2014].

[12] „Serial Peripheral Interface (SPI),“ [Tinkle]. Available: https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi. [Kreiptasi 16 11 2014].

[13] „1-Wire Overview,“ [Tinkle]. Available: https://www.maximintegrated.com/en/products/1-wire/flash/overview/. [Kreiptasi 19 11 2014].

[14] „I2C Alternatives,“ [Tinkle]. Available: http://www.i2c-bus.org/alternatives/. [Kreiptasi 19 11 2014].

[15] „1-Wire Protocol,“ [Tinkle]. Available: http://coecsl.ece.illinois.edu/ge423/sensorprojects/1-wire_full.doc. [Kreiptasi 20 11 2014].

[16] „U-Boot,“ [Tinkle]. Available: http://www.stlinux.com/u-boot. [Kreiptasi 12 05 2015].

[17] „Raspberry pi secure boot,“ [Tinkle]. Available: http://arstechnica.com/information-technology/2015/03/now-theres-a-secure-boot-add-on-for-raspberry-pi-oh-yay/. [Kreiptasi 16 05 2015].

[18] „Booting Linux kernel using U-Boot,“ [Tinkle]. Available: http://processors.wiki.ti.com/index.php/Booting_Linux_kernel_using_U-Boot. [Kreiptasi 20 05 2015].

[19] „Raspberry pi U-Boot,“ [Tinkle]. Available: http://elinux.org/RPi_U-Boot. [Kreiptasi 20 05 2015].

[20] „SAM9 boot strategies,“ [Tinkle]. Available: https://dkc1.digikey.com/fi/en/tod/Atmel/AT91SAMBootStrategies_NoAudio/AT91SAMBootStrategies_NoAudio.html. [Kreiptasi 20 05 2015].

[21] „ARM926EJ datasheet,“ [Tinkle]. Available: http://www.atmel.com/Images/Atmel_11053_32-bit-ARM926EJ-S-Microcontroller_SAM9G35_Datasheet.pdf. [Kreiptasi 20 05 2015].

[22] „Using an Encrypted Root Partition with Raspbian,“ [Tinkle]. Available: http://paxswill.com/blog/2013/11/04/encrypted-raspberry-pi/. [Kreiptasi 25 03 2016].

[23] „Full system encryption for Linux,“ [Tinkle]. Available: http://xercestech.com/full-system-encryption-for-linux.geek. [Kreiptasi 04 04 2016].

[24] „initramfs,“ [Tinkle]. Available: https://www.kernel.org/doc/Documentation/filesystems/ramfs-rootfs-initramfs.txt. [Kreiptasi 21 03 2016].

[25] „initrd,“ [Tinkle]. Available: https://www.kernel.org/doc/Documentation/initrd.txt. [Kreiptasi 21 03 2016].

[26] W. Badawy ir G. A. e. Jullien, „System-on-Chip for Real-Time Applications,“ įtraukta *Kluwer international series in engineering and computer science*, SECS 711. Boston, Kluwer Academic Publishers, 2003, p. 465.

[27] F. Clemens, „New Methods in Hard Disk Encryption,“ įtraukta *Institute for Computer Languages: Theory and Logic Group*, Vienna University of Technology, 2005.

[28] H. Stuart, H. Bill, P. Joe, S. Tomas ir E. T. Robert, „If Piracy is the Problem, Is DRM the Answer?,“ HP Laboratories, Cambridge, 2003.

# 6. PRIEDAI

## 6.1. priedas. Shell skriptai

### testas.sh

```
#!/bin/bash

# kursorius slepiamas kol veikia skriptas
kursoriausPaslepimas()
{
   tput civis # paslepia kursorių
   while kill -0 $PPID 2> /dev/null; do
      sleep 1
   done
   tput cnorm # grąžina kursoriaus matomumą
}

skyriklioSpausdinimas()
{
   echo "======================================================"
}

kursoriausPaslepimas &

clear       # išvalo konsolę
# spausdina informaciją apie šifrą
echo Šifras:
cryptsetup -v status usb
echo

echo "\033[32m+====================+\033[m"
echo "\033[32m| Greitaveikos testas |\033[m"
echo "\033[32m+====================+\033[m"

sifruotosParticijosFailas="/mnt/usb/labas"
nesifruotosParticijosFailas="/mnt/nesifruota/labas"

if [ $# -gt 0 ]; then
  count=$1
  if [ $# -gt 1 ]; then
    outputFile=$2
    if [ $# -gt 4 ]; then
      sifruotosParticijosFailas=$3
      nesifruotosParticijosFailas=$4
    fi
  else
    outputFile='0'
  fi
else
  count=25000
  outputFile='0'
fi

if [ $outputFile != '0' ]; then
  skyriklioSpausdinimas > $outputFile
  echo "Šifruota particija" >> $outputFile
fi
```

```
echo Testuojama ŠIFRUOTA particija
echo RAŠYMAS:
sh ./rasymasSkaitymas.sh $sifruotosParticijosFailas $count '0' $outputFile
echo SKAITYMAS:
sh ./rasymasSkaitymas.sh $sifruotosParticijosFailas $count '1' $outputFile
echo
echo "\033[32m==========================================\033[m"

if [ $outputFile != '0' ]; then
  skyriklioSpausdinimas >> $outputFile
  echo "Nešifruota particija" >> $outputFile
fi

echo
echo Testuojama NEŠIFRUOTA particija
echo RAŠYMAS:
sh ./rasymasSkaitymas.sh $nesifruotosParticijosFailas $count '0' $outputFile
echo Testuojama NEŠIFRUOTA particija
echo SKAITYMAS:
sh ./rasymasSkaitymas.sh $nesifruotosParticijosFailas $count '1' $outputFile
echo "\033[32m==========================================\033[m"
if [ $outputFile != '0' ]; then
  skyriklioSpausdinimas >> $outputFile
  echo "Rezultatai atspausdinti faile ${outputFile}"
fi
echo
```

**rasymasSkaitymas.sh**

```
#!/bin/bash

spinner()
{
   while kill -0 $PPID 2> /dev/null; do
      printf ".  \r"
      sleep 0.5
      printf ".. \r"
      sleep 0.5
      printf "...\r"
      sleep 0.5
   done
}

count=$2
kilo=1024
bs=$((40 *  1000))
dydis=$((bs * count))
of=$1
mb=$(($dydis / 1000 / 1000))
T=$(date +%s%N)
if [ $3 = '0' ];
then
   opcija="Rašymo"
   echo "\033[33mRašoma ${dydis} baitų (~${mb} Mb)\033[m"
   spinner &
   (dd if=/dev/zero of=$of bs=$bs count=$count && sync) > /dev/null 2>&1
else
   opcija="Skaitymo"
   echo "\033[33mSkaitoma įrašyti ${dydis} baitų (~${mb} Mb)\033[m"
   spinner &
   dd if=$of of=/dev/null bs=$bs > /dev/null 2>&1
fi
kill "$!"
T=$(($(date +%s%N)-T))
T=$((T / 1000 / 1000))
greitaveika=$((dydis / T ))
grH=$((greitaveika / 1000))
kiekK=$((grH * 1000))
grL=$((greitaveika - kiekK))
sec=$(($T / 1000))
millis=$(($T - sec * 1000))
echo "\033[33m\rUžtruko sekundžių: ${sec}.${millis} s\033[m"
echo "\033[31m================================\033[m"
echo "\033[31m ${opcija} greitis: ${grH}.${grL} Mb/s \033[m"
echo "\033[31m================================\033[m"

if [ $4 != '0' ]; then
   echo "${opcija} greitis: ${grH}.${grL} Mb/s" >> $4
fi
```