KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
KOMPIUTERIŲ KATEDRA

Hubertas Válbasas

Aparatinės programų apsaugos metodų tyrimas ir paskirstytų skaičiavimų modelio panaudojimas apsaugos rakto realizacijai

Magistro baigiamasis darbas

Darbo vadovas
lekt. dr. A. Liutkevičius

Kaunas
2011

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
KOMPIUTERIŲ KATEDRA

Hubertas Válbasas

Aparatinės programų apsaugos metodų tyrimas ir paskirstytų skaičiavimų modelio panaudojimas apsaugos rakto realizacijai

Magistro baigiamasis darbas

Recenzentas
doc. dr. J. Toldinas
2011-05-

Darbo vadovas
lekt. dr. A. Liutkevičius
2011-05-

Darbą atliko:
Hubertas Válbasas
2011-05-

Kaunas
2011

T U R I N Y S

ĮVADAS 5
1. APARATINIŲ APSAUGOS METODŲ ANALIZĖ 8
1.1. Optiniai diskai, praplėsties plokštės, kompiuterinių stočių aparatinių dalių apsaugos metodai 8
1.2. Apsaugos raktai – aparatinis apsaugos metodas 9
1.2.1. Apsaugos raktų įrenginiai 10
1.2.2. Apsaugos raktas kompiuteriniame tinkle 11
1.2.3. Taikomosios programos ir apsaugos rakto komunikavimas 12
1.2.4. Taikomųjų programų ir jų komunikavimo su apsaugos raktu sauga 12
1.2.4.1. Apsaugos raktų skirstymas 13
1.2.4.2. Apsaugos raktas su laiko matuokliu 16
1.2.4.3. Vientisumo ir abipusio autentiškumo patikrinimas 16
1.2.4.4. Kriptografiniai algoritmai ir slapti raktai 18
1.2.4.5. Daugiablokinė santraukos schema 19
1.2.4.6. Išmanaus kliento sistemos panaudojimas aparatinėje apsaugoje pavyzdžiai 19
1.2.4.7. Patikimo aparatinio apsaugos metodo sudarymas 22
1.2.5. Atakų metodai prieš taikomųjų programų apsaugą 22
1.2.5.1 Apgrąžos inžinerijos ataka 23
1.2.5.2. Vertimo (statinio metodo) ataka 23
1.2.5.3. Derinimo (aktyvaus metodo) ataka 24
1.2.5.4 Nuotolinės prieigos kodo įterpimo ataka 25
1.2.5.5 Programos klonavimo ataka 25
1.2.5.6 Darbinės atminties nuskaitymas ir apsauga 27
1.2.5.7. Kiti taikomųjų programų su apsaugos raktu pažeidimai 28
1.3. Programų apsaugos aparatiniais metodais analizės rezultatai 28
1.4. Papildomi apsaugos metodai, kurie gali būti naudojami sustiprinti aparatinius apsaugos metodus 30
1.5. Aparatinių ir papildomų apsaugos metodų, skirtų programų apsaugai, palyginimas 31
1.6. Išvados 34
2 SIŪLOMAS IŠMANAUS APARATINIO APSAUGOS RAKTO MODELIS 35
2.1. Išmanaus kliento principas aparatinėje programų apsaugoje 35
2.2. Siūlomas dalies programos vykdymas apsaugos rakte 36
2.3. Aparatinės apsaugos modelio funkciniai ir nefunkciniai reikalavimai 36
2.4. Siūlomo išmanaus aparatinės apsaugos rakto modelis 38
2.5. Išvados 45

3. APSAUGOS RAKTO METODU PAGRĮSTO PROGRAMŲ APSAUGOS LYGIO ĮVERTINIMO TYRIMAS 46
3.1. Tyrimo programiniai įrankiai, aplinkos ir programavimo kalbos 46
3.2. I tyrimo dalis. Eksperimentinės konsolės tipo programos apsaugos su komerciniu aparatiniu raktu efektyvumo tyrimas 47
3.2.1. Komercinio apsaugos rakto ir eksperimentinės programos be papildomos apsaugos tyrimas. 48
3.2.2. Komercinio apsaugos rakto ir eksperimentinės programos su maskavimo apsauga tyrimas 49
3.2.3. Komercinio apsaugos rakto ir eksperimentinės programos su suspaudimo ir nederinimo apsaugomis tyrimas 50
3.2.4. Komercinio apsaugos rakto ir eksperimentinės programos tyrimo rezultatai 51
3.3. II tyrimo dalis. Eksperimentinės programos apsaugos su apsaugos raktu, realizuotu siūlomo modelio pagrindu, tyrimas 51
3.3.1. Eksperimentinio *Tmote sky* apsaugos rakto ir langų tipo programos be papildomos apsaugos tyrimas 53
3.3.2. Eksperimentinio Tmote sky apsaugos rakto ir komandinės eilutės tipo programos be papildomos apsaugos tyrimas 53
3.4. I ir II tyrimo dalių rezultatų palyginimas 54
3.5. Išvados 55
IŠVADOS 56
LITERATŪRA 57
SUMMARY 62
PRIEDAI 63
1. Priedas Nr. 1. Programos dalis, vykdoma eksperimentiniame *Tmote sky* apsaugos rakte 63
2. Priedas Nr. 2. Eksperimentinė programa, bandyta su komerciniu apsaugos raktu 65
3. Priedas Nr. 3. Eksperimentinė langų tipo programa. Form1.h byla 65
4. Priedas Nr. 4. Eksperimentinė komandinės eilutės tipo programa. DongleTest.cpp byla 67

ĮVADAS

Programinės įrangos nelegalus naudojimas centrinėje ir rytų Europoje išlieka didžiausias pasaulyje, 64 iš 100 taikomųjų programų yra naudojamos nelegaliai [1]. 2009 metais pasaulio mastu 43% taikomųjų programų buvo piratinės, 34-44% naudojamos legaliai, o 12-22% programų atviro kodo, kai 2007 metais pasaulyje piratinės programos sudarė 38%, o 2008 metais – 41% programų rinkos [1]. Metinį piratavimo lygio didėjimą lėmė auganti Azijos rinka [1]. Mažiausias piratavimo lygis 2007, 2008 ir 2009 metais išliko Šiaurės Amerikos regione – 21%. Vakarų Europoje 2007-2009 metais išliko 33% piratavimo lygis. Vidurio rytuose ir Afrikoje užfiksuotas 59-60% piratavimo lygis, Pietų Amerikoje 65%, Azijoje 2008 metais – 59%, o 2009 metais – 61% piratavimo lygis. Europos Sąjungoje 2007-2009 metais užfiksuotas 35% piratavimo lygis. Centrinėje ir Rytų Europoje, į kurios sudėtį įeina ir dalis rytinių Europos Sąjungos valstybių, 2007 metais užfiksuotas 68%, o 2008 metais – 66%, o 2009 metais – 64% piratavimo lygis. Plečiantis kompiuterijos rinkai didėja ir nelegalus programų naudojimas Lietuvoje. Lietuvoje 2008 metais nustatytas 54% [1] (su 2% paklaida [2]) piratavimo lygis. Pagrindiniai veiksniai, kurie lėmė didėjantį nelegalų programų naudojimą yra didėjantis kompiuterių naudotojų skaičius, laisva nelicencijuotų programų prieiga per internetą ir išaugęs programišių išsilavinimo lygis [1]. Pakankamai aukštą piratavimo lygį Lietuvoje lemia sparčiai didėjantis kompiuterijos naudojimas įvairiose srityse [3]. Taip pat piratavimo lygį didina ekonominė krizė, kuri stabdo antipiratavimo politiką [1].

Visuotinai nelegaliai naudojamas programas bandoma mažinti įvairiais metodais: valstybių švietimo sistemos pertvarkomis, programų pardavėjų susitarimais su originalios įrangos gamintojais, valdžios ir industrijos susitarimais, technologijų pažanga [1]. Lietuvoje piratavimo lygį bando mažinti 2006 metais įsteigta antipiratinės veiklos asociacija LANVA, kurios vienas iš pagrindinių tikslų – vykdyti antipiratines akcijas elektroninėje erdvėje – internete, kadangi teisių turėtojai patiria didžiulius nuostolius dėl internetinio piratavimo, teigia, kad „piratavimas“ yra kūrinių vagystė, tolygi automobilio ar kito daikto vagystei [4]. Tačiau taikomąją programą su kompiuterine įranga galima dublikuoti, ko negalima pasakyti apie automobilį, todėl kompiuterinė programa turi turėti apsaugą, kurios įveikimas reikalautų kompiuterijos ir programavimo žinių. Piratavimo probelemą didina nelegalus įmonių konkuravimas, kai vienos įmonės licencijuotą programinę įrangą naudoja nelegaliai, o kitos įmonės legaliai. Nuo nelegalaus kopijavimo ir panašių problemų bandoma apsisaugoti programinėmis ir aparatinėmis priemonėmis. Prie aparatinių priemonių priskiriami apsaugos raktai (angl. *dongle*).

Apsaugos raktai – USB atmintinių dydžio aparatiniai įrenginiai, kurie jungiami prie kompiuterio ir be kurių atitinkama programa neveiks [5] arba praras dalį funkcijų. Teigiama, kad suderinus aparatinį apsaugos metodą su programiniais metodais gaunama optimali programų apsauga [6]. Vieni tyrėjai yra skeptiškai nusiteikę prieš taikomųjų programų apsaugą aparatiniu ar programiniu metodu, nes apsauga nepasiteisindavo, tačiau kiti tyrėjai (Conner, Rumelt, Givon, Prasad, Mahajan, Shy ir Thisse) mano, kad taikomąsias programas galima apsaugoti įvairiais metodais [7].

Programų leidėjai sprendžia, kokį apsaugos lygį naudoti programos kodui, nes nuo to priklauso programavimo laikas ir kaina, todėl apsaugos raktai, pagal apsaugos lygį, skirstomi į atminties modulius ir modulius, vykdančius papildomus skaičiavimus.

Taikomosios programos apsauga su apsaugos raktu dažniausiai remiasi modulio identifikavimu ir slaptos vertės patikrinimu. Tačiau įsilaužėliai pasinaudodami apgrąžos inžinerijos metodais gali apeiti identifikavimo patikrinimą ir naudoti programą be apsaugos rakto. Padidintą apsaugos lygį nuo apgrąžos inžinerijos suteikia užmaskuotas (angl. *obfuscated*) programos kodas, kuriame kreipiniai į apsaugos raktą yra paslėpti. Nuo apgrąžos inžinerijos metodų galima apsisaugoti panaudojus išmaniojo kliento modelį (angl. *Fat client*). Išmaniojo kliento modeliu naudojasi interaktyvūs internetiniai tinklapiai, kuriuose vartotojo atliekami veiksmai nereikalauja nuolatinio prisijungimo prie žiniatinklio serverio. Panaudojus išmaniojo kliento principą dalis programos skaičiavimų atliekami apsaugos rakte, kuriame įmontuotas procesorius, darbinė ir nuolatinė atmintinės, o grąžinama vertė toliau naudojama programos funkcijų procesuose. Taip įsilaužėlis nežinotų kaip gaunama vertė modulyje, nes apsaugos rakto atmintis fiziškai ir programiškai apsaugota nuo įsilaužimų.

Šio darbo tikslas yra atlikti taikomųjų programų aparatinės apsaugos metodų tyrimą ir pasiūlyti aparatinį apsaugos metodą, kuris remiasi paskirstytų skaičiavimų modeliu ir užtikrina programų apsaugą nuo apgrąžos inžinerijos atakų, bei įvertinti siūlomo metodo efektyvumą lyginant su tradiciniais aparatiniais apsaugos metodais. Šiam tikslui pasiekti reikia atlikti tokius uždavinius:

1.2.3. atlikti aparatinės apsaugos metodų analizę;
1.2.3. atlikti papildomų apsaugos metodų, kurie gali pagerinti aparatinius apsaugos metodus, analizę;
1.2.3. sukurti išmanaus aparatinio apsaugos rakto modelį, kuris pagrįstas paskirstytuoju skaičiavimo modeliu;
1.2.3. sukurti išmanaus apsaugos rakto prototipą, remiantis aukščiau sudarytu modeliu;

1.2.3. atlikti išmanaus apsaugos rakto prototipu apsaugotų programų apsaugos lygio įvertinimo tyrimą ir palyginti su tradiciniais aparatiniais apsaugos metodais.

1. APARATINIŲ APSAUGOS METODŲ ANALIZĖ

Aparatiniai apsaugos metodai, kurie taikomi programų apsaugai, yra kompiuterinių sistemų apsaugos objektai, kurie priskiriami techninėms apsaugos priemonėms ir apriboja prieigą prie taikomosios programos [3]. Kompiuterinės sistemos subjektas, norėdamas gauti pilną prieigos teisę prie apsaugotos programos, turi turėti apsaugai priskirtą aparatinį įrenginį. Pagal surinktus duomenis aparatiniai apsaugos įrenginiai skirstomi į keturias grupes:

1. Optinius diskus.
2. Praplėsties plokštes.
3. Darbo ar tarnybinių stočių aparatines dalis.
4. Apsaugos raktus.

1.1. Optiniai diskai, praplėsties plokštės, kompiuterinių stočių aparatinių dalių apsaugos metodai

Programų apsaugai nuo neteisėto naudojimo priskiriama aparatinių apsaugos metodų grupė: optinių diskų skaitytuvai (angl. *optical disk drive*), praplėsties plokštės (angl. *expansion cards*), kompiuterinių stočių aparatinės dalys ir *dongle* moduliai (apsaugos raktai plačiau aptariami sekančiuose skyriuose).

Optiniai diskai (CD, DVD, BD) naudojami kaip aparatiniai apsaugos moduliai, kuriuose dažniausiai įrašoma legali programos kopija ir serijos numeris, kuris identifikuoja kompiuteryje įrašytą programą ir suteikia vartotojui vykdymo teises. Tačiau paplitus optinių diskų įrašomiesiems įrenginiams ir optinių diskų dublikavimo programoms namų vartotojai įgavo teisę identiškai nukopijuoti optinio disko turinį į naują diską. Optinių diskų kainos 1-20 LT, todėl prieinamos daugeliui namų vartotojų.

Praplėsties plokštės – plokštės, jungiamos į kompiuterio pagrindinės plokštės PCI jungtį kaip apsaugos moduliai (pvz. IBM 4758 [8]). Šių modulių kainos siekia apie 4500 LT (2000$) [8]. Dėl sąlyginai didelės kainos PCI moduliai prieinami organizacijoms, kurios apsaugai skiria didžiausią dėmesį, o moduliai skirti specializuotoms programoms, kurios namų vartotojams yra nenaudingos.

Kompiuterinių stočių aparatinių dalių moduliai yra kompiuterių elementų: procesoriaus, atminties modulių, vaizdo plokštės, kietojo disko, garso plokštės serijos numerių rinkinių kombinacijos, kurios susiejamos su viena programos kopija, todėl pakeitus kurį nors kompiuterio elementą pasikeis serijos numerių kombinacija ir programa nustos veikusi ar praras dalį funkcijų. Kompiuterinių stočių elementų kainos priklausomai nuo pardavėjų ir techninių parametrų vidutiniškai svyruoja nuo 400 iki 4000 LT. Tačiau

saugomoje programoje radus serijos numerių kombinaciją ir perkėlus į tekstinę bylą, programą galima platinti į kitus kompiuterius.

Apsaugos raktai – aparatiniai įrenginiai, kurie jungiami prie kompiuterio nuoseklaus ar lygiagretaus prievado [9]. Apsaugos raktų kainos siekia nuo 57LT iki 230LT [10, 11, 12]. Aparatinių apsaugos modulių kainų palyginimas pateiktas lentelėje Nr. 1.

Lentelė Nr. 1. Aparatinių apsaugos modulių kainos

Aparatinio įrenginio tipas	Kaina	Specializuota diegimo įranga	Apsauga nuo namų vartotojų
Optiniai diskai	1-20 LT	Nereikalinga	–
PCI plokštės	~4500 LT	Reikalinga	+
Kompiuterių elementų identifikaciniai numeriai	400-4000 LT	Nereikalinga	+
Apsaugos raktai	57-230 LT	Reikalinga	+

Atsižvelgus į modulių kainas ir specializuotos gamybos įrangos naudojimą, apsaugos raktai atitinka daugiausia programų leidėjų poreikių, nes yra apsaugoti nuo namų vartotojų ir kainuoja dešimtis kartų mažiau nei praplėsties plokštės.

1.2. Apsaugos raktai – aparatinis apsaugos metodas

Taikomąsias programas nuo nelegalaus kopijavimo galima apsaugoti pritaikius aparatinį metodą, pagrįstą apsaugos rakto naudojimu. Apsaugos raktas – aparatinis įrenginys, kuris jungiamas prie kompiuterio, ir be kurio atitinkama taikomoji programa neveiks [7, 13] arba praras dalį funkcijų. Dalis programų apsaugotos apsaugos raktais, kuriuose įrašytas identifikacinis numeris, todėl programa patikrina programoje įrašyta ID su rakte įrašytu ID. Dalis apsaugos raktų susideda iš mikroprocesoriaus ir atminties, taip raktas atlieka funkcijų skaičiavimus ir kriptografinius šifravimus, todėl raktą sunkiau dublikuoti ar generuoti [9]. Programa siunčia duomenų kiekį į modulį, kuriame duomenys apdorojami ir gautas rezultatas siunčiamas atgal programai [14]. Jei rezultatas tinkamas – programa veiks. Tai leidžia apsaugoti programą nuo neautorizuoto naudojimo.

Apsaugos raktai skiriasi keliais parametrais: prie kompiuterio jungiamu prievadu, taikomosios programos ir įrenginio komunikavimo algoritmu, šifravimo sudėtingumu, pritaikymo galimybėmis ir kaina. Modulių skaičius dažniausiai apsprendžiamas kai apsprendžiamas programų kopijų skaičius [15].

1.2.1. Apsaugos raktų įrenginiai

Universalios nuoseklios magistralės (angl. *USB – Universal Serial Bus*) modulis 1 pav. USB prievado moduliai yra populiariausi, nes kiekvienas kompiuteris turi kelis USB prievadus. Kompiuteryje įdiegtos kelios taikomosios programos gali būti apsaugomos atskirais USB moduliais. Pasinaudojus USB komutatoriumi prie vieno kompiuterio prievado galima prijungti kelis USB modulius [10].

1 pav. USB modulis [16]

Lygiagretaus (angl. *LPT – Line Print terminal*) 2 pav. ir nuoseklaus (*COM*) 3 pav. prievado moduliai. Senesnio modelio kompiuteriuose buvo populiarūs LPT ir nuoseklūs (pvz. RS232) prievadai. Taikomoji programa, kuri nereikalauja spartaus procesoriaus ir atminties, gali būti įdiegta į kompiuterį, turintį LPT arba RS232 prievadą.

2 pav. LPT modulis [17]

3 pav. RS232(DB25)modulis [17]

4 pav. PCMCIA modulis [17]

PCMCIA prievado modulis 4 pav. Nešiojamuose kompiuteriuose paplitęs prievadas, skirtas prijungti bevielio tinklo modulį. Bevielio tinklo modulis lygiagrečiai galėtų dirbti ir kaip taikomųjų programų apsaugos nuo nelegalaus kopijavimo įrenginys.

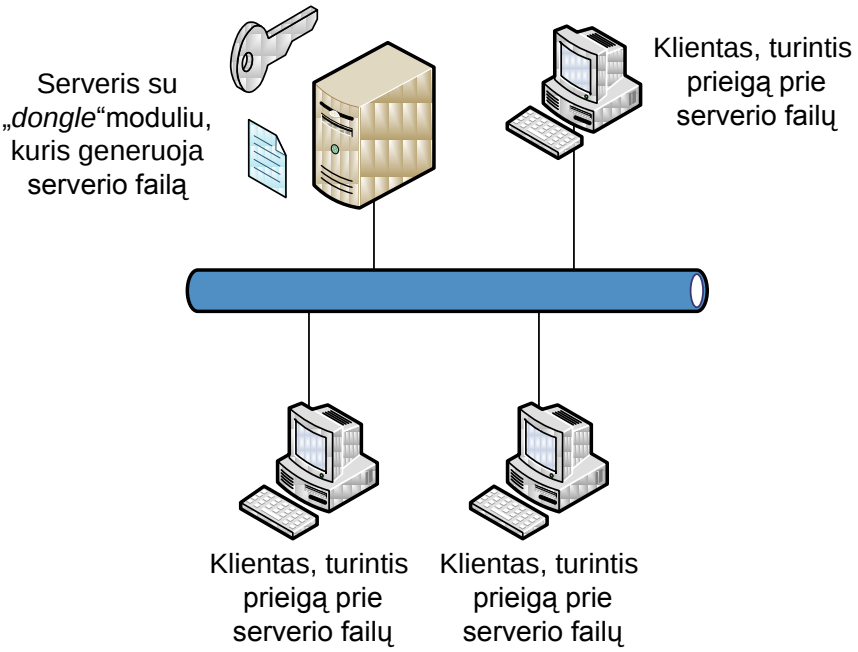
Komercinės kompanijos rinkoje siūlo ir kitų prievadų ({E}ISA, NEC-PC{*serial*}, NEC-PC98-System) modulius, tačiau jie rečiau naudojami, yra brangesni ir sunkiau prijungiami nei USB, LPT, RS232, PCMCIA prievadų moduliai. Rinkoje siūlomi 3,5 colių lanksčiųjų diskelių (angl. *floppy*) moduliai [18].

Dongle moduliais laikomos ir sumaniosios kortelės (angl. *smart card*), kuriose įdiegti mikroprocesoriai ir atminties moduliai. Tačiau sumaniosios kortelės programų apsaugai nėra populiarūs apsaugos raktai, nes joms nuskaityti reikalingi specialūs kortelių skaitytuvai ir milimetrų eilės mikroprocesoriai ir atminties moduliai [19].

1.2.2. Apsaugos raktas kompiuteriniame tinkle

Vienas apsaugos raktas gali būti naudojamas prie kelių darbo stočių (kompiuterių), jei darbo stotys sujungtos į tinklą [20]. Modulis jungiamas prie bet kurios darbo stoties. 5 pav. atveju modulis prijungtas prie serverio. Serverio programa sukuria serverio failą, taip darbo stotys, turinčios prieigą prie serverio, naudojasi serverio failu tarsi apsaugos raktas būtų prijungtas prie darbo stoties [20].

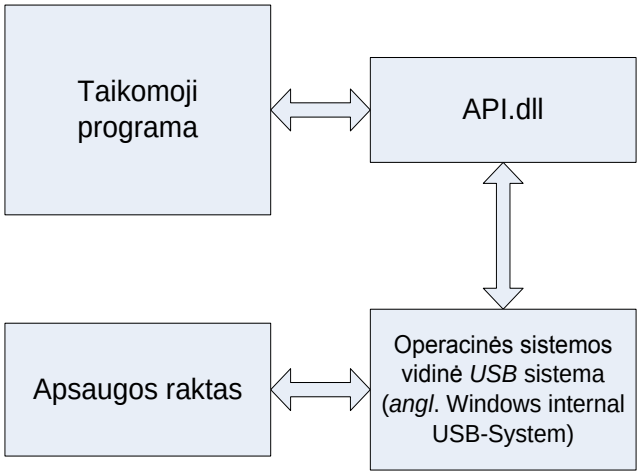
Taikomųjų programų apsauga tinklu nereikalauja interneto protokolų, todėl gali būti naudojama bet kokio tipo tinkle [20]. Serverio failas yra šifruojamas ir atnaujinamas numatytais laiko periodais [20]. Šifravimo algoritmas keičiasi kiekvieno atnaujinimo metu [20]. Tinkle naudojamas modulis turi pranašumą prieš modulį, kuris saugo vieną taikomąją programą, nes gali aptarnauti kelias programas. Tačiau įvykus tinklo gedimui sutrinka prieiga prie apsaugos rakto ir visos saugomos programos neveiks.



5 pav. Apsaugos rakto panaudojimas tinkle [20]

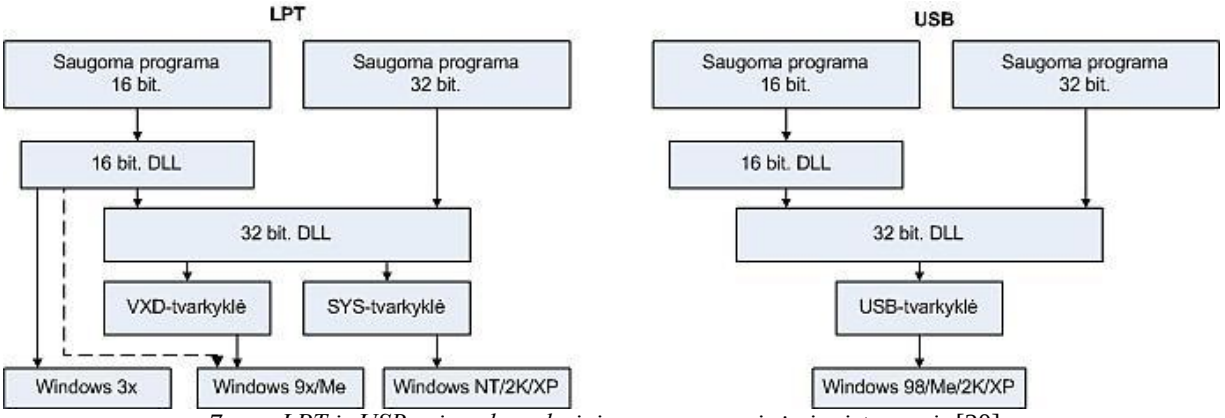
1.2.3. Taikomosios programos ir apsaugos rakto komunikavimas

Dažniausiai visų apsaugos raktų veikimas remiasi tuo, kad taikomoji programa siunčia funkcijos užklausą į modulį, o modulis siunčia funkcijos atsaką į taikomąją programą [9]. Jei priimtas atsakas tenkina funkcijos algoritmą – programa veiks, jei netenkina – programa neveiks.



6 pav. Programos ir modulio komunikavimas[21]

Vienam apsaugos raktui galima



7 pav. LPT ir USB prievadų suderinimas su operacinėmis sistemomis [20]

priskirti ribotą arba neribotą kiekį programų [11]. Saugoma taikomoji programa gali būti įrašyta optiniame (CD, DVD, BD) diske, ir prieinama tik su apsaugos raktu [12].

6 pav. pavaizduotas apsaugotos programos komunikavimas su apsaugos raktu, kuris vyksta per API (angl. *application programming interface*) dinaminių bibliotekų failus [10]. 7 pav. pavaizduota lygiagrečių ir nuoseklių jungčių diegimas operacinėje sistemoje. Jei naudojamos 16 bitų programos, tai tarp taikomosios programos ir *API* .dll failo reikia naudoti papildomą 16 bitų .dll failą. „*Cross-platform*“ metodu paremta taikomosios programos apsauga nereikalauja įdiegti USB tvarkylių operacinėje sistemoje, šios tvarkyklės įrašomos apsaugos rakte [10].

1.2.4. Taikomųjų programų ir jų komunikavimo su apsaugos raktu sauga

Renkantis programos apsaugą su apsaugos raktu įvertinami trys pagrindiniai veiksniai:

1. Apsaugos lygis.
2. Apsaugos diegimo laikas.
3. Kaina.

Trys veiksniai tarpusavyje susiję tiesiogiai, jei didinamas apsaugos lygis kyla apsaugos diegimo laikas ir kaina, jei kaina mažinama – krenta apsaugos lygis, tačiau mažėja ir diegimo laikas.

Apsaugos nuo nelegalaus kopijavimo aparatiniu metodu remiasi dviejų argumentų autentifikavimo metodu, angliškai vadinamu „*Two factor authentication*“ metodu. Tai reiškia, kažkas ką žinai ir kažkas ką turi [22]. Vartotojo slaptažodis, kurį vartotojas įveda į apsaugos raktą, kad jį aktyvuotų, reiškia tai ką žinai. Tačiau dažniausiai slaptažodžio, kitaip vadinamo kriptorakto, įvedimas yra vienkartinis. Apsaugos raktas, reiškia tai ką turi.

Kompanijos siūlo įvairius apsaugos metodus. Pavyzdžiui, taikomoji programa turi pati identifikuotis apsaugos raktui, po to modulis identifikuojasi taikomajai programai, taip apsisaugoma nuo neautorizuoto prisijungimo [12]. Kitas pavyzdys, dalį taikomosios programos galima įrašyti į apsaugos raktą, taip įsilaužėlis neturės galimybės įtakoti visos taikomosios programos [23].

1.2.4.1. Apsaugos raktų skirstymas

Pagal taikomosios programos funkcionalumą ir naudojimo laiką apsaugos raktai skirstomi į šias grupes [24]:

a) neribotas naudojimas; po modulio suprogramavimo, taikomoji programa gali būti saugoma neribotą kiekį kartų ir neribotą laiko periodą;

b) ribotas naudojimų skaičius; taikomosios programos paleidimų skaičius yra fiksuojamas, pasiekus nustatytą skaičių programa nebeveikia;

c) ribotas tam tikram laiko periodui (angl. *shareversion*); taikomoji programa veiks nustatytu laikotarpiu;

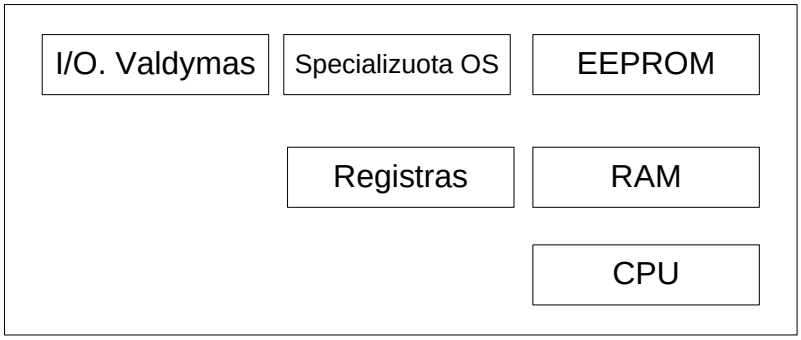
d) priklausomai nuo vartotojo poreikių ir finansinių galimybių; apsaugos raktas gali leisti naudotis kaikuriomis taikomosios programos funkcijomis;

e) vieno modulio naudojimas tinklu; vienas modulis fiziškai prijungtas prie vieno kompiuterio aptarnauja keletą taikomųjų programų, įdiegtų skirtinguose tinklo kompiuteriuose.

Taip pat pagal programų apsaugą apsaugos raktai skirstomi į kitas dvi grupes: atminties modulius ir modulius, kurie atlieka papildomus skaičiavimus [20]. Atminties moduliai saugo reikalingus duomenis nuolatinėje atmintyje, o saugoma programa, norėdama veikti, duomenis nuskaito iš modulio. Šio tipo apsaugos raktas lengvai diegiamas ir konfigūruojamas. Tačiau saugumo požiūriu įsilaužėlis gali lengvai nuskaityti modulio atminties turinį ir emuliuoti modulio atsaką į taikomosios programos kreipinius. Moduliuose, kurie atlieką dalį skaičiavimų, įmontuotas procesorius, RAM (angl. *Ramdom Access Memory*)

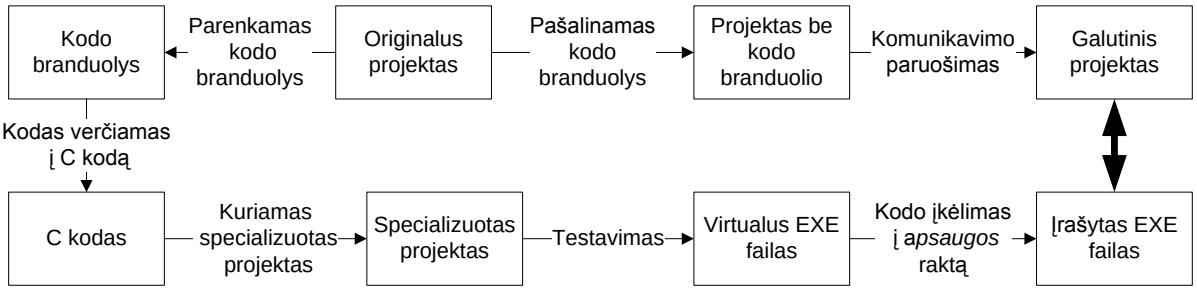
darbinė atmintis, *FLASH* ar EEPROM (angl. *Electrical erasable programable read only memory*) nuolatinė atmintis. Šiuo atveju taikomoji programa gali komunikuoti su apsaugos raktu kriptografiniais algoritmais *DES, AES, IDEA, ECC, RSA, MD5, SHA-1* [18]. Sumaniojo modulio diegimas yra sudėtingesnis, taikomosios programos kode reikia įdiegti papildomas funkcijas. Tačiau saugumo požiūriu komunikavimo duomenys perduodami šifruoti ir įsilaužėliui bus sudėtingiau juos nuskaityti.

Taikomosios programos saugumas padidinamas, kai apsaugos rakte saugoma ir vykdoma dalis taikomosios programos funkcijų. Taip programa bus apsaugota nuo kopijavimo, nes svarbios funkcijos saugomos apsaugos rakto atmintyje, o ne kompiuteryje, kuriame įdiegta programa [25] [26]. Taip dalis programos vykdymo apsaugoma nuo derinimo ir stebėjimo atakų [26]. Struktūrinė apsaugos rakto, kuriame diegiama ir vykdoma dalis programos funkcijų, struktūra pavaizduota 8 pav.



8 pav. Apsaugos rakto, kuriame vykdoma dalis skaičiavimų, struktūra [26]

Apsaugos rakto, kurio blokinė struktūra pateikta 8 pav., gali būti panaudota taikant apsaugos metodiką, kurios etapai pateikti 9 pav. Šioje metodikoje programa suskaidoma į dvi dalis: viena diegiama į apsaugos raktą, kita lieka kompiuteryje.



9 pav. Dalies programos diegimas į apsaugos raktą ir komunikavimas su programa [25]

Pagal programų apsaugą saugumo metodai skirstomi į dvi grupes: automatinę apsaugą, kai programų apsaugai naudojami aumatiniai programiniai įrankiai ir fizinę apsaugą, kai dalis programos kodo yra perrašoma ar pridedama kodo dalis. Su fizine apsauga taip pat naudojami automatiniai programiniai įrankiai [20]. Taikant automatinę apsaugą nereikia keisti programos šaltinio kodo. Taip sutaupomas laikas ir palengvinamas apsaugos įdiegimas.

Tačiau saugumo prasme šis būdas nėra patikimas. Taikant fizinę apsaugą reikia keisti programos šaltinio kodą. Šio metodo įdiegimas užtrunka daugiau laiko, tačiau saugumo požiūriu metodas yra geresnis.

Taikomosios programos ir apsaugos rakto komunikavimo algoritmas parenkamas toks, kad pašalinei programai nebūtų galima imituoti komunikavimo algoritmo [15]. Paprasčiausias būdas taikomosios programos apsaugai yra apsaugoti .exe arba .com failą (vadinama automatinė apsauga), tam reikalingas vykdomosios programos kodas, bet nereikalingas šaltinio (angl. *source*) kodas [15]. Sudėtingesni metodai reikalauja įterpti šifravimo kodą į šaltinio kodą [15].

Papildomai šifruodami taikomąją programą galime apsisaugoti nuo atakų, kurios aprašytos 1.2.5. skyriuje. Maskavimo (angl. *obfuscator*) metodas transformuoja programos kodą į kitą programos kodą, kurio funkcijos tos pačios, bet patį kodą suprasti yra sunkiau [14].

Maskavimo metodas aprašomas taip: algoritmas **O** tinka kiekvienai programai **P,** todėl išraiška **O(P)** tenkina šias sąlygas [28]:

- O(P) turi tokias pačias savybes kaip ir P;
- O(P) yra neįveikiamas analizės ar apgrąžos inžinerijos (angl. *reverse-engineer*) metodams.

Tačiau užmaskuota programa negali būti lėtesnė nei programos vykdymas per polinomį laiką [28]:

- Funkcionalumas **O(P)** ~ funkcionalumui **P**;
- Polinominis sulėtinimas **,** kur **p()** polinominė funkcija.

Maskavimo kokybė matuojama keturiais parametrais: stiprumu (angl. *potency*), kuris nurodo kaip stipriai užmaskuota programa; atsparumu (angl. *resilience*), kuris nurodo kiek užmaskuota programa atspari prieš apgrąžos inžinerijos atakas; slaptumu (angl. *stealth*), kuris nurodo kaip maskuota kodo dalis sąveikauja su nemaskuotu kodu; savikaina (angl. *cost*), kuris nurodo kokią įtaką maskavimas turi programos vykdymo spartai [14]. Eksperimentais įrodyta, kad apgrąžos inžinerijos atakų įrankiai greičiau pakeičia nemaskuotos programos kodą, nei maskuotos programos kodą [14]. Maskuojamas kodas turėtų „paslėpti“ programos kodo funkcijas, kurios kreipiasi į apsaugos raktą. Taip įsilaužėliui būtų sunkiau atsekti „paslėptas“ funkcijas.

Programos maskavimą galima atlikti pasitelkus kintantį duomenų tipo maskavimą (angl. *Changing Data Type Obfuscation*), kuris keičia kintamųjų tipus iš ilgalaikių (angl. *long-term*) į trumpalaikius (angl. *short-term*) arba atvirkščiai [29].

Taikomąsias programas galima apsaugoti filigranavimo (angl. *watermarking*) metodu, kurio idėja panaši į daugialypėje terpėje (angl. *multimedia*) naudojamą metodą. Į garso, vaizdo, vaizdavimo duomenis yra įterpiamas unikalus identifikatorius, kurio žmogaus akis nemato. Taikomojoje programoje yra įterpiamas programos fragmentas, kuris netrukdo visos programos veikimo ir sunkiai aptinkamas (arba neaptinkamas) derinimo programų. Jei nenaudojamas programos fragmentas, tai dažniausiai naudojamas vartotojo identifikacinis numeris [14]. Filigranavimo apibrėžimas [30]:
Duota programa *P*, filigrinas *w* ir raktas *k*. Programos filigrinavimo sistema susideda iš dviejų funkcijų: **įterpimo**(*P*, *w*, *k*)→*P'* ir **atpažinimo**(*P'*, *k*)→*w*.

Taikomųjų programų filigranavimą galima atlikti su *Sandmark* programa, kuria galima tyrinėti programų apsaugą, maskuoti ir skaitmeniškai pasirašinėti kodą [30].

1.2.4.2. Apsaugos raktas su laiko matuokliu

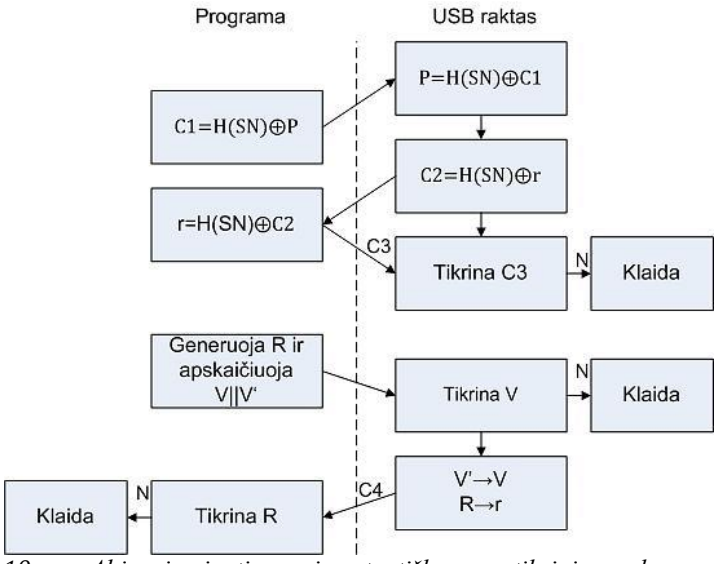
Apsaugos raktas su laiko matuokliu (angl. *real time clock*) fiksuoja tikslų laiką ir datą [27]. Tam reikalingas vidinis energijos šaltinis. Tokie moduliai skirti automatiškai fiksuoti bandomosios programos naudojimo laiką, kuriam pasibaigus programa nebeveiks arba praras dalį funkcijų. Moduliai su laiko matuokliais ir vidiniu energijos šaltiniu vidutiniškai veikia 4 metus [27], tai svarbu apsprendžiant programos veikimo laiką.

1.2.4.3. Vientisumo ir abipusio autentiškumo patikrinimas

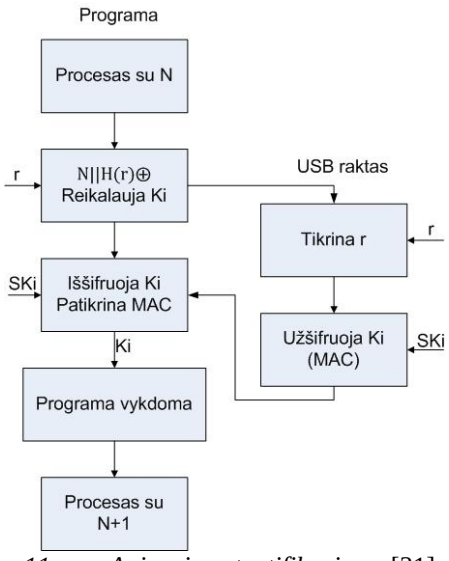
MeiHong ir JiQiang siūlo abipusio autentifikavimosi metodą. Taikomoji programa ir apsaugos raktas pradėdami komunikavimą turi vienas kitam autentifikuotis. Tam naudojamas abipusio autentiškumo ir vientisumo patikrinimas, kurios struktūra pavaizduota 10 pav. Pagrindiniai vientisumo patikrinimo etapai:
1) Vartotojas taikomojoje programoje įveda įdentifikacinį numerį (P), kurį programa užšifruoja ir šifrogramą C1 siunčia į apsaugos raktą,

$C1=H(SN) \oplus P$,

H – santraukos funkcija, kurios parametras SN yra unikalus skaičius, patalpintas programoje ir apsaugos rakte.



10 pav. Abipusio vientisumo ir autentiškumo patikrinimo schema [31]



11 pav. Apipusis autentifikavimas [31]

2) Apsaugos raktas iššifruoja P iš C1 ir užšifravęs siunčia atgal šifrogramą C2,

$C2=H(SN) \oplus r$,

r – atsitiktinis sugeneruotas skaičius. Programa iššifruoja r iš C2 ir siunčia USB raktui C3,

$C3=H(H(f_1)||H(f_2)H(f_i)||\ldots)$

f_i-oji funkcija reiškia programos saugomą vietą, t.y. programos dalis susideda iš i kiekio f funkcijų. Apsaugos raktas aspkaičiuoja savo C3, taip pat kaip ir programa, funkcijos f_i yra įrašytos rakto atmintinėje, ir sulygina su gauta C3 reikšme. Jei abi C3 reikšmės sutampa tai vientisumo patikrinimas sėkmingas.

Po to seka abipusio autentifikavimosi patikrinimas, pavaizduotas 11 pav. Programa sugeneruoja atsitiktinį skaičių R ir apskaičiuotas dvi reikšmes V ir V' ir siunčia apsaugos raktui.

$V=H(r||P)$, $V'=H(R||P)$

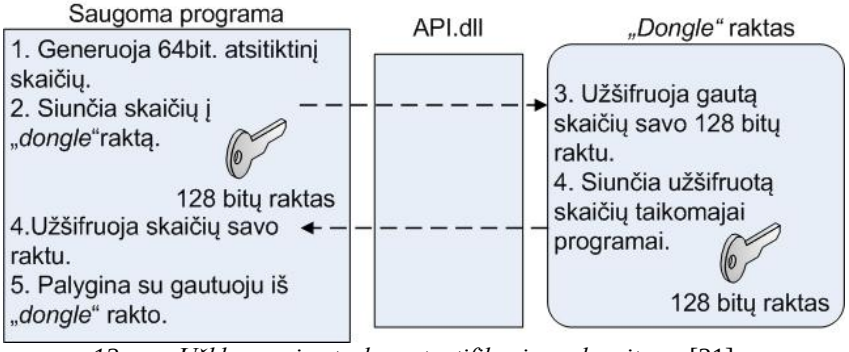
Apsaugos raktas patikrina V reikšmę su atmintyje saugoma V reiškme. Jei gauta V sutampa su saugomas V reikšme programa sėkmingai autentifikavosi. Rakto atmintyje saugomos V ir r reikšmės pakeičiamos naujomis V' ir R reikšmėmis, taip įgalinamas dinamis patikrinimas. Apsaugos raktas užšifruoja R reikšmę ir siunčia C4 programai.

$C4=En(K,R)$,

kur $K=H(r) \oplus R$. Programa iššifruoja C4 ir autentifikuoja apsaugos raktą [31].

1.2.4.4. Kriptografiniai algoritmai ir slapti raktai

Taikomosios programos ir apsaugos rakto komunikavimas turi būti saugomas, kitu atveju įsilaužėlis galės emuliuoti modulį ir nelegaliai naudotis programa. Saugumui užtikrinti naudojami kriptografiniai algoritmai ir slapti raktai. Algoritmo pritaikymas taikomojoje programoje ir apsaugos rakte parodytas 12 pav. Šiuo atveju programa siunčia nešifruotą skaičių, o modulis grąžina šifruotą skaičių.



12 pav. Užklausos ir atsako autentifikavimo algoritmas [21]

Saugumui užtikrinti reikia duomenų srautą viena ir kita kryptimi šifruoti. Dešifravimas vyksta taikomojoje programoje ir apsaugos rakte. 12 pav. pateiktame algoritme naudojamas 128 bitų slaptas raktas, tai vienas iš dviejų autentifikavimo argumentų, tai ką žinau. Slaptas raktas yra kriptografinės funkcijos (algoritmo) kintamasis, kurio pagalba užšifruoti duomenys gali būti saugiai perduodami į apsaugos raktą. Slaptų raktų ilgių būną kitokių, pvz. 56, 112, 168, 1024, 2048 bitų. Slaptą raktą sudaro skaičių, simbolių arba skaičių ir simbolių rinkinys, todėl kuo rinkinys ilgesnis tuo sunkiau jį atspėti. Naudojama ir dviejų raktų, šakninio ir vartotojo, kombinacija [10]. Vartotojo raktą žino vienos įmonės darbuotojai, o šakninį raktą žino vienas įmonės atstovas, kurio pareiga, esant reikalui, pakeisti vartotojo raktą [10]. Keliems apsaugos raktams gali būti naudojamas vienas slaptas raktas, taip vienos įmonės darbuotojai gali lengviau jais naudotis.

Vien raktas duomenų neapsaugos, tam reikia patikimo algoritmo, kuris slapto rakto pagalba užšifruoja siunčiamus duomenis. Naudojami AES (angl. *Advanced Encryption Algorythm*), DES (angl. *Digital Encryption Algorythm*), TEA (angl. *Tiny Encryption Algorythm*), MD5 (angl. *Message-Digest algorithm 5*), SHA-1(angl. *Secure Hash Algorithm*), 3DES, RSA1024/2048 (angl. *1024/2048 bit Rivest, Shamir, Adleman*), HMAC (angl. *Hash Message Authentication Code*) algoritmai.

Algoritmams naudojami atsitiktinių skaičių generatoriai. Jie generuoja pvz. 64 bitų (12 pav.) ir aukštesnės eilės atsitiktinius skaičius, kurie naudojami kaip funkcijos argumentai.

Į algoritmus, kaip argumentai, įtraukiami taikomųjų programų ir apsaugos raktų identifikaciniai numeriai (ID), kurie sustiprina algoritmą [32].

Kaip funkcijos argumentas vartotojui gali būti asmeniškai suteiktas 6-8 baitų ilgio slaptažodis [33]. Vartotojas turi įvesti slaptažodį kiekvieną kartą paleidžiant taikomąją programą.

1.2.4.5. Daugiablokinė santraukos schema

Programos kodas verčiamas dvejetainiu pavidalu ir sudalinamas į skirtingų dydžių blokus. Kiekvienas blokas neturi sąryšio su kitu bloku. Kiekvienas blokas šifruojamas ankstesnio bloko santraukos (*hash*) verte. Paskutinis blokas šifruojamas priešpaskutinio bloko santraukos verte, priešpaskutinis blokas šifruojamas trečio nuo pabaigos bloko santraukos verte ir t.t. Pirmas blokas lieka nešifruotas ir vadinamas baziniu bloku. Programos valdiklis, skirtas dešifravimui, patalpinamas programos pabaigoje. Kiekvienas blokas turi rodyklę į programos valdiklį, kad galėtų iššifruoti sekantį bloką. Programa pradedama vykdyti nuo bazinio bloko santraukos vertės siuntimo į programos valdiklį, kuris užšifruoja sekantį bloką ir kuris savo santraukos funkciją siunčia atgal į programos valdiklį ir taip iki programos pabaigos. Santraukos vertė yra apskaičiuojama dinamiškai pradėjus programos vykdymą [14].

Šiuo atveju įsilaužėlio tikslas yra programos valdiklio pakeitimas arba valdiklio siunčiamų ir gaunamų verčių perėmimas. Todėl taikomosios programos valdiklį galima patalpinti apsaugos rakte, kuriame būtų apsaugotas nuo pakeitimo. Taip pasunkėtų įsilaužėlio užduotis pakeisti programos valdiklį. Tačiau taip nebūtų apsaugotas duomenų perdavimas tarp programos valdiklio ir blokų, todėl įsilaužėlis galėtų juos nuskaityti.

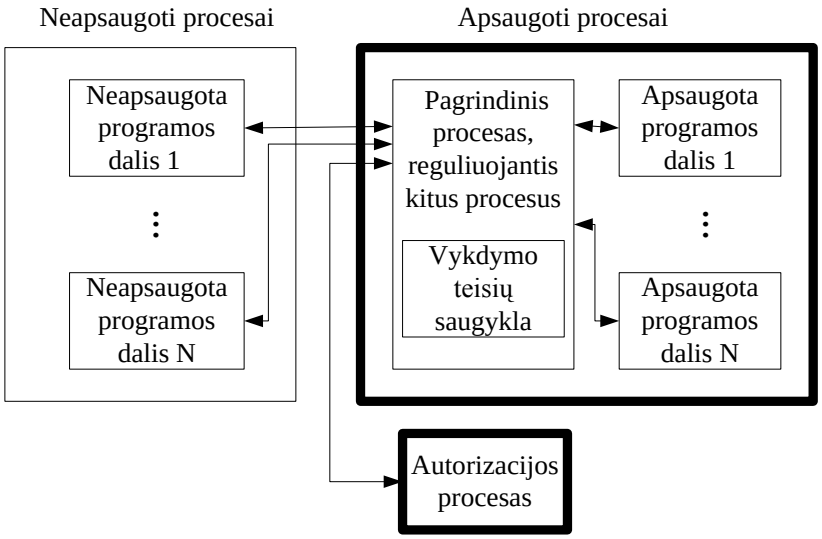
1.2.4.6. Išmanaus kliento sistemos panaudojimas aparatinėje apsaugoje pavyzdžiai

Taikomoji programa, kurios dalis įdiegta kliento įrenginyje, o dalis įdiegta serveryje vadinama išmaniaisiais klientais (angl. *Fat clients*) [34]. Taip vartotojas gali atlikti dalį funkcijų kliento įrenginyje nesijungdamas prie serverio [35]. Išmanusis klientas atlieka dalį duomenų apdorojimo ir nėra visada priklausomas nuo serverio [36] [37] [38], nes klientinė sistema apima vaizduojamąjį ir taikomąjį lygmenį [34] [36]. Tokios sistemos pavyzdys yra pašto klientinė programa [35].

Pirmieji taikomųjų programų apsaugos aparatiniais įrenginiais, kuriuose buvo vykdoma dalis arba visa programa, taikymai buvo atliekami 1990 metais. Steve R. White ir Liam Comerford 1990 metų straipsnyje [39] pristatė taikomųjų programų apsaugos architektūrą ABYSS, kuri skirta vykdomų programų apsaugai nutolusiuose kompiuteriuose. Vykdoma programa suskirstoma į neapsaugotus procesus ir apsaugotus procesus. Apsaugoti

procesai vykdomi saugioje kompiuterinėje aplinkoje, apsaugotame procesoriuje [37]. Apsaugoto proceso blokinė architektūra pavaizduota 13 pav.

Apsaugotas procesorius yra logiškai, fiziškai ir procedūriškai apsaugotas, nes programa negali tiesiogiai kreiptis į pagrindinį procesą ar apsaugotas programos dalis. Procesorius yra atsparus pažeidimams, o procesai, kurie persiunčia informaciją, negali pažeisti apsaugos. Apsaugotos programų dalys procesoriaus išorėje yra šifruotos, o jas vykdant iššifruojamos. Neapsaugotos programos dalys yra nešifruojamos. Vykdomų teisių saugyklų procesas yra atskirtas nuo programos, nes jame saugoma svarbi informacija: šifravimo ir dešifravimo raktai, apsaugotos programos dalies vykdymas ir pagrindinio proceso suteikiamų teisių autorizacijos procesui taisyklės. Autorizacijos procesas vykdomas aparatiniame įrenginyje (sumaniojoje kortelėje ar apsaugos rakte), kuris suteikia vykdymo teises pagrindiniam procesui [39].



13 pav. Apsaugoto procesoriaus sistemos architektūra [6]

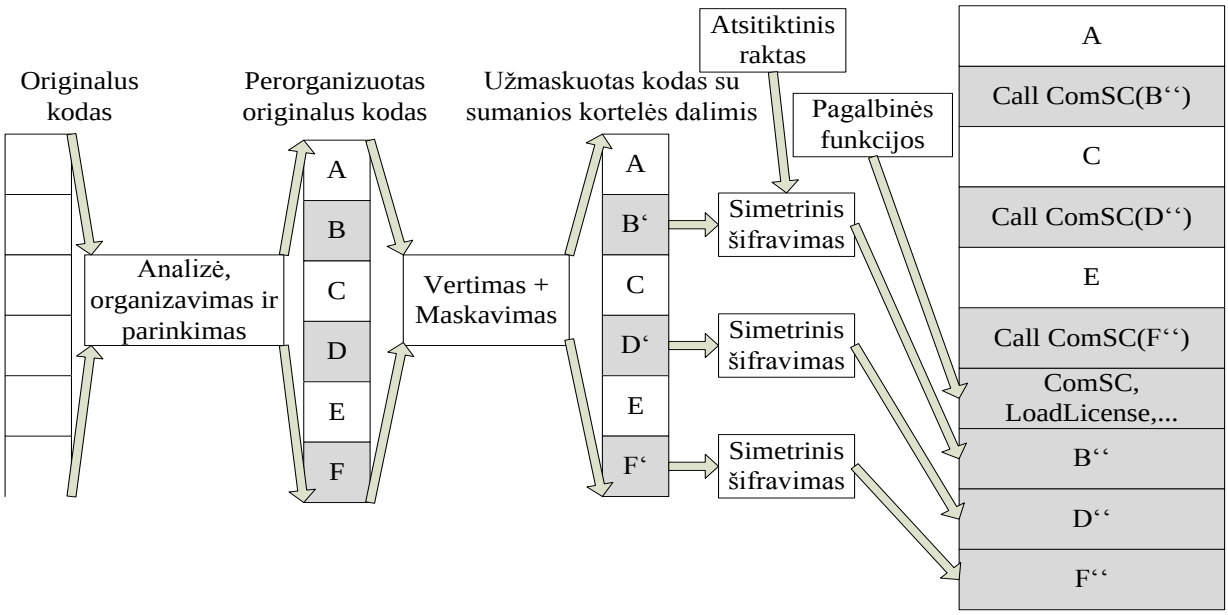
Programos dalinimas remiasi semantiniu arba kombinatoriniu sudėtingumu. Semantinis sudėtingumas aprašo sudėtingą apsaugotų programos dalių radimą iš neapsaugotų programos dalių, kai stebima jų tarpusavio sąveika. Kombinatorinis sudėtingumas aprašo sudėtingą apsaugotų programos dalių charakteristikų radimą, kai stebimas programos dalių veikimas [39].

Kaip teigia Antonio Mana su bendraautoriais pagrindiniai ABYSS sistemos trūkumai yra papildomų aparatinių įrenginių, kurie skirti autorizacijai, naudojimas su apsaugotu procesoriumi ir galimybė įsilaužėliams sukurti netikrus apsaugotus procesus [40]. Šių trūkumų sprendimui autoriai pasiūlė SmartProt sistemą, kuri programą transformuoją taip, kad būtų apsaugoma nuo apgrąžos inžinerijos analizės ir neautorizuoto vykdymo. Vietoje

atskiro apsaugoto procesoriaus ir autorizacijos kortelės buvo pasirinkta viena sumanioji kortelė, kuri veikia kaip apsaugotas procesorius ir autorizacijos procesas.

Programos apsauga vykdoma pirmiausia iš programos kodo parinkus specifines funkcijas. Kodas reorganizuojamas taip, kad parinktos funkcijos galėtų būti vykdomos sumaniojoje kortelėje. Parinktos funkcijos iš Java objektinio kodo verčiamos į SmartProt virtualų mašininį kodą. Papildomai apsaugai funkcijoms pritaikomas maskavimo metodas, todėl kodas verčiamas į sunkiau suprantamą kodą. Po vertimo ir maskavimo etapų pasirinktos funkcijos šifruojamos pasinaudojus simetrinę kriptosistemą su atsitiktinai sugeneruotu raktu. Pasirinktos funkcijos sukeičiamos su funkcijų kreipiniais, kurie kreipiasi į sumaniąją kortelę [40]. Galutiniame kode šifruotos funkcijos talpinamos į kodo pabaigą. Programos apsaugos eiga pateikta 14 pav.

Papildomai autorizacijai naudojama programos leidėjo sukurta licencija, kurioje kliento viešuoju raktu užšifruojami atsitiktinis raktas, programos ID, licencijos ID.



14 pav. Programos apsauga su SmartProt [40]

Programa vykdoma patikrinus kliento licenciją. Kreipiantis į užšifruotas funkcijos dalis programa siunčia užklausą į sumaniąją kortelę, kuri iš programos parsisiunčia reikiamą programos dalį ir ją įvykdo. Kortelės atmintyje išsaugoma tam tikrą funkcijos vertė, kad butų apsisaugoma nuo funkcijų analizės atakų [40].

Ištobulėjus puslaidininkinėms technologijoms galima sukurti mažesnių matmenų, tačiau talpesnius atminties modulius, į kuriuos galima įrašyti didesnes programos dalis, užimančias daugiau vietos atmintinėje. Taip pat, papildomos užklausos siuntimas iš kortelės į programą ir dalies programos atsiuntimas vykdymui užtrunka laiko. Kuo programos dalis didesnė tuo ilgiau trunka siuntimas. Todėl programos dalį galima įrašyti į apsaugos rakto

atmintinę. Tada programos dalis bus vykdoma kai bus gaunama užklausa iš programos. Taip bus sutaupomas siuntimo laikas. Taip pat, programos dalys bus fiziškai neprieinamos įsilaužėliams, todėl jas galima apsaugos rakto atmintyje saugoti nešifruotas. Asimetriškai arba simetriškai šifruojami tik perduodami duomenys.

1.2.4.7. Patikimo aparatinio apsaugos metodo sudarymas

Įrenginys, kuris šifruoja duomenis arba saugo atmintinėje svarbius programos elementus yra sunkiau emuliuojamas [6], todėl norint sukurti gerai saugomos taikomosios programos metodą reikia remtis šiais veiksniais [32]:

a) taikomosios programos apsaugos metodus reikia taikyti jau kuriant programą [3];

b) programos ir apsaugos rakto komunikavimo funkcijos turi būti stiprios. Reikia naudoti AES algoritmą. Ir kaip galima sutrumpinti slapto rakto išlaikymą RAM (angl. *random access memory*) atmintyje;

c) turi būti patikimas asmuo, kuriantis programų apsaugą arba programos kodą atskirai dalimis rašo skirtingi programuotojai;

d) slaptieji raktai įrašomi į apsaugos raktą ne modulių gamintojų, o programų kūrėjų. Modulių gamintojas pateikia komutavimo apsaugos algoritmą, o programų kūrėjas įrašo slaptą raktą;

e) Komunikavimo apsaugos funkcijų algoritmas ir slapto rakto generavimo algoritmas turi būti patalpinti atskirai. Į algoritmus turi įeiti apsaugos rakto identifikacinis numeris. Algoritmų funkcijos turi būti patalpinamos į dinaminių bibliotekų failus, kuriuos yra sunku dekompiliuoti.

f) reikia naudoti klastingus ir painius programinius kodus, kurie pvz. imituotų netikrą prisijungimą prie apsaugos rakto;

g) kaip galima dažniau naudoti slaptą raktą apsaugos algoritme;

h) planuotai atnaujinti taikomąją programą.

Aparatinis taikomųjų programų apsaugos metodas turi būti derinamas su programiniu apsaugos metodu, nes nei vieno jų geriausios savybės nepralenks jų savybių junginio [6]. Silpną taikomosios programos apsaugą sudarytų toks algoritmas, kuris apsaugos rakto autentifikavimui naudotų tik jo identifikacinį numerį [6].

1.2.5. Atakų metodai prieš taikomųjų programų apsaugą

Daugelis apsaugos raktų gamintojų ir testuotojų teigia, kad pažeidžiamiausia taikomosios programos apsaugos metodo vieta yra komunikavimas tarp taikomosios programos ir apsaugos rakto [5, 11, 27, 32]. Taikomoji programa siunčia užklausą per API.dll

failą į USB raktą, todėl įsilaužėlis, aptikęs šį kreipinį, gali imituoti USB rakto atsaką arba apeiti užklausą. Apsaugos nulaužimo metodai skirstomi į dvi grupes: statinę ir dinaminę. Statiniai metodai atlieka programos kodo išeities teksto analizę, o dinaminiai metodai vykdo kodo sekimą specialiomis apsaugos nulaužimo priemonėmis [41].

1.2.5.1 Apgrąžos inžinerijos ataka

Apgrąžos inžinerijos (angl. *Reverse-engineering*) greičiausiai bus pirmas žingsnis, kurį naudos įsilaužėliai, norėdami įveikti taikomosios programos apsaugą [27]. Įsilaužėlis neturėdamas programos šaltinio kodo gali išsiaiškinti jos veikimo principą [14, 20, 42, 43]. Pritaikius gilų šio metodo panaudojimą galima pakeisti programos kodą ir pritaikyti veikti kitaip [42]. Ši ataka brangiai kainuoja programos kūrėjui, nes pažeidžiamą kodą reikia iš naujo tobulinti ir testuoti [14]. Įsilaužėliai, norėdami įveikti programų apsaugas, naudoja dviejų tipų universalius įrankius: vertimo (angl. *disassemblers*) derinimo (angl. *debuggers*) ir dekompiliavimo (angl. *decompile*) programas.

1.2.5.2. Vertimo (statinio metodo) ataka

Vertimo programa konvertuoja programos kodą į asemblerio kodą [27, 44]. Įsilaužėliai gali turėti tik asemblerio programavimo žinias, kurių užtenka suprasti taikomosios programos veikimą [27]. Programų analizei naudojama asemblerio kalba, kurioje nėra apribojimų darbui su steku, procesoriaus registrais, atmintimi, įvedimo ir išvedimo prievadais [41]. Išverčiamo kodo kokybė priklauso nuo vertimo programos kokybės [27]. Vertėjas prasčiau išverčia taikomosios programos kodą, kuris buvo parašytas aukšto lygio kalba (pvz. C++ [45, 32], *Delphi* [32]), tačiau geriau atkuria šaltinio kodą, parašytą *FoxPro* [32] kalba. Vertimo programa pati automatiškai nekeičia programos šaltinio kodo [44]. Naudojamos vertimo programos yra *OllyDbg*, *IDA-Pro* [46], *Pic-Simulator_DE* [47].

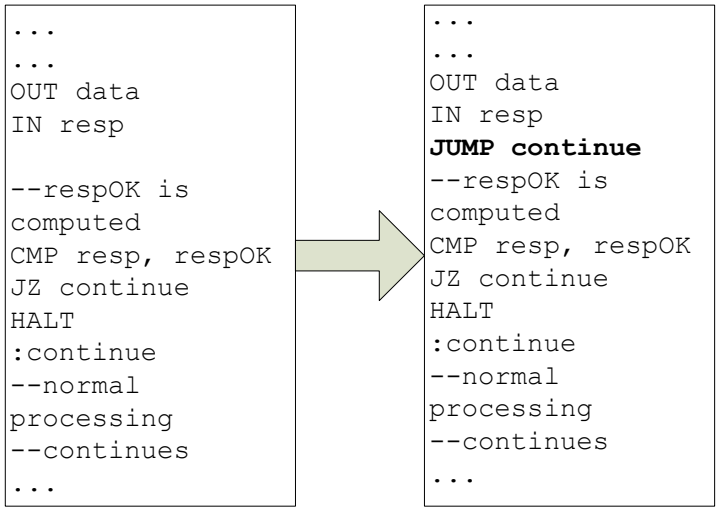
Vertimą yra sudėtingiau jei šaltinio kodo fragmentai yra dinamiškai koduojami, taip kad visas kodas vienu metu nebūtų talpinamas atmintyje [44]. Jei tokios apsaugos neįmanoma įgyvendinti tada reikia šaltinio kodą išplėsti taip, kad vertimo analizės atlikimas užtruktų kuo daugiau laiko, tam siūlomi keli metodai [44]:

- Įdiegti antraštes, kurios nebūtų susijusios su kodu.
- Naudoti daug besąlygiškų šuolių.
- Tarp komandų įterpti atsitiktinius kodo fragmentus, kurie neturėtų įtakos kodo eigai.
- Naudoti savaime kintantį kodą (angl. *self modifying code*). Kodas, kuris automatiškai keičiasi, jei pradedamas vykdyti neteisėtai [48].

- Taikomosios programos funkcijas perkelti į dinaminių bibliotekų (angl. *dll*) failus, nes *dll* failai lengvai kompiliuojami, bet sunkiai dekompiliuojami [32].

1.2.5.3. Derinimo (aktyvaus metodo) ataka

Derinimo (angl. *debug*), priešingai nei vertimo, programos yra aktyvūs įrankiai, kurie gali stebėti vykdomo kodo etapus[44] ir pakeisti kodo dalį, kad programa negalėtų normaliai funkcionuoti arba apskritai veikti [49]. *Dongle* modulio patikrinimą galima apeiti panaudojus derinimo ataką ir įterpus JUMP komandą. JUMP komandos įterpimo pavyzdys pateiktas 15 pav.



Originalus kodas Patikrinimo apėjimas

15 pav. Aparatinės apsaugos apėjimas panaudojus JUMP komandą **[50]**

Išskiriami trys derinimo programų tipai: vartotojo lygio, branduolio lygio ir emuliavimo [44].

- Vartotojo lygio derinimo programos dažnai būna kartu įdiegtos su operacine sistema, tam kad leistų vartotojui pačiam derinti nesaugomas programas programinio gedimo metu [44].
- Branduolio lygio derinimo programa, įdiegta į operacinės sistemos šerdinius katalogus, leidžia procesų valdymą tik iš branduolio lygmens [44].
- Aparatinį apsaugos raktą identiškai imituoja emuliavimo programos. Jas kuria ir patys apsaugos raktų gamintojai, tam jei klientas pamestų, sugadintų ar kitaip negalėtų naudotis saugoma programa. Šiomis programomis pasinaudoja įsilaužėliai.

Taikomąsias programas galima apsaugoti nuo derinimo atakų testuodami programos kodą ir analizuodami padarinius. Taip pat reikia naudoti derinimo programas, kurios atlieka programos kodo testavimus [49]. Derinimo atakos-analizės atliekamos su WinDbg [51], OllyDbg [52].

Vertimo ir derinimo programiniai įrankiai nėra skirti vien nelegaliai veiklai. Juos programinio kodo analizavimui naudoja programų kūrėjai ir testuotojai [53]. Egzistuoja

specializuotos taikomosios programos, skirtos prieš derinimo atakas, tokios kaip *Themida*, *EXECryptor*, *ACProtect standard*, *AntiCrack Software Protector Basic* [54] ir kt.

1.2.5.4 Nuotolinės prieigos kodo įterpimo ataka

Nuotolinės (angl. *remote*) prieigos kodo įterpimo („*process dump*“) ataka naudojama programos kodo įterpimui ar vykdomo proceso pakeitimui. Jei nelegaliai naudojama programa ar piktybinė programa (virusas, kirminas ir kt.) įdiegta kietajame diske tai ją sąlyginai lengva aptikti ir analizuoti. Tačiau kodo įterpimo per nuotolinę prieigą ataką galima priskirti įsiterpusio žmogaus (angl. *man in the middle*) grupei. Ši ataka atliekama tinklu, todėl nereikia diegti papildomos piktybinės programos į aukos darbo stotį. Įsilaužėlis pirmiausia įsitikina, kad piktybinė programa sėkmingai apėjo apsaugos priemones (pvz. užkardą, tarpinį serverį ar kt.). Apsaugos priemonės apeinamos panaudojus pakartotinio sujungimo panaudojimo (angl. *connection reusing*) arba protokolo sutraukimo (angl. *protocol encapsulation*) metodą. Piktybinė programėlė, apėjusi apsaugas, pradeda veikti kompiuterio atmintyje, tačiau nepalikdama įrašų kietajame diske (angl. *anti forensic technique*). Šios atakos metu saugoma taikomoji programa gali būti išjungiama arba veikti ne pagal instrukcijas (angl. *process mapping*) [55]. Ataką galima įvykdyti su „*Process Dumper*“ [56] ir „*Netcat*“ programa [57]. „*Process Dumper*“ nepalieka įrašų kietajame diske. Jei programos proceso dalis yra pažeidžiama, ją reikia analizuoti ir radus klaidą tobulinti programos kodą. Analizę galima atlikti su „*Memory Parser*“ programa [58].

„*Process Dumper*“, „*Netcat*“ ir „*Memory Parser*“ yra legaliai platinamos taikomosios programos, kurios skirtos legaliems taikomųjų programų procesų patikrinimams ir derinimams. Tačiau įsilaužėlis jas gali pritaikyti ir nelegaliai veiklai, pvz. nuotoliniu būdu jungtis prie aukos kompiuterio ir jame analizuoti taikomosios programos ir apsaugos rakto komunikavimą.

1.2.5.5 Programos klonavimo ataka

Programos klonavimas (angl. *software cloning*) vyksta kai įsilaužėlis nukopijuoja taikomąją programą ir pritaiko savo reikmėms. Klonavimo metodai skirstomi į 4 grupes [59]:

- I grupė. Kodo kopijos fragmentas yra toks pats kaip originalo (kodas gali skirtis tuščių ar naujų eilučių skaičiumi, lygiavimu);
- II grupė. Kodo kopijos fragmentas sutampa su originaliu, tačiau gali skirtis kintamųjų pavadinimai, komentarai;
- III grupė. Kodo kopijos fragmentas yra pakeistas, sąlygose pridėti arba atimti veiksmai;

- IV grupė. Kodo kopijos fragmentas yra semantiškai panašus į originalų kodą, tačiau jų sintaksės struktūra skiriasi. Fragmentai gali būti parašyti skirtingų programuotojų, tačiau atliekamų funkcijų rezultatai yra vienodi.

Panašių ar identiškų kodų aptikimas yra naudingas kovai su nelegaliai naudojama programine įranga [59]. Tai aktualu tarpusavyje konkuruojančioms įstaigoms, kurios naudoja panašią, tačiau skirtingų leidėjų, programinę įrangą. Viena iš taikomosios programos autentifikavimo pakopų yra patikrinti ar tam tikra programos kodo dalis yra identiška apsaugos rakte saugomam kodui. Tam naudojami klonuotų programų aptikimo įrankiai: *Dup, Jplag, CloneDr, DupLoc, CCFinder, CP-miner, Sim, DiLucca Pro., eMetrics* ir kt., kurie lyginami pagal jų sąvybes [59]:

- Portatyvumą: įrankis yra portatyvus jei sugeba analizuoti tūkstančiu programavimo kalbų ir keletą kiekvienos kalbos dialektų.
- Tikslumą: įrankis turi tiksliai skirti klonuotus kodus ir neklonuotus.
- Grįžtamumą: įrankis turi kelis kartus patikrinti ir įsitikinti, kad kodas yra klonuotas arba ne.
- Keičiamą dydį: įrankis turi sugebėti patikrinti didelius kodus, nes didelės kodų sistemos yra sunkiau patikrinamos nei mažesnio dydžio sistemos.
- Tvirtumą: su didesniu tikslumu ir grįžtamumu turi aptikti skirtingus klonuotų programų pakeitimus.

Tiksliai klonuotą kodą atpažįstanti programa ne visada aptiks visus klonuotus kodo fragmentus, todėl lentelėje Nr. 2 palyginti klonuotų programų aptikimo įrankiai pagal tikslumą ir grįžtamumą.

Lentelė Nr. 2. Tikslumo ir grįžtamumo palyginimas pagal Burd ir Bailey's eksperimentą [59]

	CCFinder	*CloneDr*	*Covet*	*JPlag*	*Moss*
Tikslumas	72%	100%	63%	82%	73%
Grįžtamumas	72%	9%	19%	12%	10%

Pagal lentelę *CloneDr* tiksliai nustato ar kodo dalis yra klonuota ar ne, tačiau pakartotinai atlikus kodo patikrinimą nevisada randama klonuota dalis. *CCFinder* optimaliausiai atpažįsta ir suranda klonuotą kodo dalį.

Įsilaužėlis nežinodamas programos ir apsaugos rakto komunikavimo slapto rakto negalės atlikti modulio emuliavimo. Taigi, įsilaužėlio užduotis nelegaliai klonuoti programą pasunkėja jei dalis taikomosios programos kodo ir slapti raktai yra saugomi apsaugos rakto atmintyje.

1.2.5.6 Darbinės atminties nuskaitymas ir apsauga

Įsilaužėliai, norėdami perimti slaptažodžius ar kriptografinius raktus, dažnai naudoja pilno perrinkimo arba žodyno atakas, tačiau jos neveiksmingos prieš stiprius slaptažodžius ir ilgus raktus, todėl skaitmeninę informaciją bandoma perimti iš darbinės atminties, kurioje duomenys dažnai būna atviro teksto [47]. Maartmann C. M., Thorkildsen S. E. ir Årnes A., darbe [60] pristatė modeliavimo rezultatus, kaip iš darbinės atminties nuskaitomi ir analizuojami AES algoritmo, Serpent ir Twofish kriptografiniai raktai. Straipsnyje teigiama, kad AES algoritmo pagrindinis ir raundo raktai atmintyje saugomi atvirame bitų masyve. Įsilaužėlis žinodamas masyvo pirmos eilutės 16B (pagrindinį raktą) gali sugeneruoti likusius 112 B. raundo raktus, taip nuspėdamas visą AES kriptoraktą [60].

Kitas metodas, naudojamas darbinės atminties nuskaitymui, yra RAM skaitytuvai (angl. *RAM scrapers*). Jie naudojami kaip piktybinės programos, kuriomis įsilaužėlis nuotoliniu būdu gauna informaciją iš RAM atminties. Tačiau turėdami patikimą interneto prieigos apsaugą ir laiku pastebėdami kintančias darbo ar tarnybinės stoties eksploatacines savybes, galima apsisaugoti nuo RAM skaitytuvų [61].

Kurdami programas su kriptografiniais algoritmais atsižvelgiama į tris punktus [60]:

- Kriptografiniai algoritmai turi ištrinti kriptoraktus iš atminties kai raktai nebereikalingi.
- Programa su kriptografiniu algoritmu turi visada ištrinti raktus kai baigiamas programos naudojimas.
- Programa turi užtikrinti, kad dėl darbinės atminties trūkumo raktai nebus rašomi į nuolatinę atmintį.

Todėl programos su kriptografiniais algoritmais skirstomos į tris grupes:

- **Viso disko šifravimas**. Programos turi atlikti autentifikaciją ir niekada neįkelti kriptoraktų į darbinę atmintį, jei sėkmingai neatlikta autentifikacija.
- **Virtualaus disko šifravimas**. Šioms programoms reikia įkelti raktus į darbinę atmintį, bet uždaroma programa turi juos ištrinti.
- **Sesija paremtas šifravimas**. Šios programos turi ištrinti raktus iš darbinės atiminties kai baigiasi sesijos galiojimas arba naudojamas vienkartinis raktas.

Taigi, patikimos kriptografinės programos turi aptikti programos išjungimą, darbalaukio užsklandos aktyvavimą ir atminties perkėlimą į nuolatinę atmintį (angl. *hibernation*) [9].

1.2.5.7. Kiti taikomųjų programų su apsaugos raktu pažeidimai

Apsaugos raktai gali būti pažeisti fiziškai. Dažnai jie kaltinami dėl kitų programų neveikimo, kompiuterio operacinės sistemos užkrovos ir išsijungimo problemų. Kompiuterio antivirusinė programa gali pripažinti modulį kaip „blogą programą“ (angl. *malware*), nes modulio programa įrašoma į šakninius operacinės sistemos katalogus. Taikomoji programa gali likti neapsaugota jei vienu kompiuteriu naudosis keli vartotojai.

1.3. Programų apsaugos aparatiniais metodais analizės rezultatai

Atlikus aparatinių apsaugos metodų, naudojamų taikomosioms programoms apsaugoti nuo nelegalaus naudojimo, analizę išskyriau keturias apratinių metodų rūšis, kurių savybės palyginamos lentelėje Nr. 3.

Lentelė Nr. 3. Aparatinių apsaugos metodų palyginimas

Aparatinis įrenginys	Karšto jungimo (angl. *hot plug-in*) palaikymas	Suderinimas su daugiau nei viena taikomąja programa	Reikalinga specializuota diegimo įranga	Apsauga nuo namų vartotojų	Kaina
Optinis diskas	+	–(1)	–	–	1 – 2 LT
Praplėsties plokštė	–	+	+	+	~4500 LT
Apsaugos raktas	+	+	+	+	57 – 230 LT
Kompiuterio elementai		+	+		400 – 4000 LT(2)

(1) Teoriškai optinį diską galima suderinti su keliomis taikomosiomis programomis, tačiau gamintojai ir programų leidėjai optinį diską derina su viena programa.
(2) Apsaugos kaina priklausys nuo to kiek kompiuterio elementų sudarys darbo stotį.

Daugiausia programų leidėjų lūkesčių atitinka apsaugos raktas, nes optinius diskus galima dublikuoti su namų įranga, o praplėsties plokščių ir kompiuterio elementų kaina, palyginus su apsaugos raktais, kelis kartų didesnė. Lentelėje Nr. 4 apsaugos raktai suskirstyti pagal jų parametrus.

Lentelė Nr. 4. Apsaugos raktų parametrai

Prievadas				Apsaugos pritaikymas					Fizinės savybės	
USB	LPT	RS232	PCMCIA	Neribotas naudojimas	Ribotas naudojimų skaičius	Ribotas laiko periodui	Priklausomai nuo vartotojo poreikių	Vieno modulio naudojimas tinklu	Atminties modulis	Modulis, atliekantis papildomus skaičiavimus

Kompiuteriuose plačiausiai naudojamas USB prievadas, tačiau prieš dešimt metų daugiau paplitę buvo LPT ir RS232 prievadai. Apsaugos pritaikymas patogus komercijoje, kai parduodamą programą galima papildomai apriboti paruošus bandomąją versiją. Lentelėje Nr. 5 lyginamos apsaugos raktų fizinės savybės.

Lentelė Nr. 5. Modulių fizinės savybės

Įrenginio tipas	Naudoja vidinę atmintį	Naudoja kriptoprocesorių	Šifruoja/dešifruoja siunčiamus duomenis	Nereikia keisti programos šaltinio kodo	Mažiau laiko užimantis diegimas ir valdymas
Atminties modulis	+	–	–	+	+
Modulis, atliekantis papildomus skaičiavimus	+	+	+	–	–

Jei naudojamas kriptoprocesorius, tai apsaugos rakte galima šifruoti ir dešifruoti siunčiamus duomenis, tačiau papildomiems skaičiavimams kriptoprocesoriuje reikalingas papildomas apsaugos rakto konfiguravimas. 6 lentelėje lyginami pagrindiniai 3 atakų metodai, nukreipti prieš aparatinius apsaugos meotdus, pagal jų savybes.

Lentelė Nr. 6. Atakų prieš aparatinius apsaugos metodus palyginimas

Metodas	Nuotolinė prieiga prie programos	Nereikia apeiti papildomų išorinių apsaugų(1)	Didesnis įrankių pasirinkimas	Reikalingos gilesnės informatikos žinios
Apgrąžos inžinerija	–	+	+	+
Nuotolinės prieigos kodo įterpimo „*process dump*“ ataka	+	–	–	+
Programos klonavimas	–	+	–	–

(1) Ugniasienės, IDS (angl. *Intrusion detection system*).

Programos klonavimas atliekamas su automatiniais programiniais įrankiais, tačiau, jei identiškai nukopijuojama programa, tai nukopijuojama ir programos apsauga, todėl vėliau reiklingi apgrąžos inženerijos įrankiai apsaugos apėjimui.

1.4. Papildomi apsaugos metodai, kurie gali būti naudojami sustiprinti aparatinius apsaugos metodus

Su aparatiniais apsaugos metodais naudojami papildomi programiniai apsaugos metodai sustiprina bendrą programų apsaugos lygi, nes papildoma apsauga pailgina bendros apsaugos įveikimo laiką. Programiniai apsaugos metodai skirstomi į tris grupes:

- Programų aktyvavimą.
- Licencijos raktai.
- Kodo maskavimas, keitimas, suspaudimas.

Produkto aktyvavimas (angl. *product activation*) – skaičių ir simbolių seka, kuri susieta su konkrečia taikomaja programa, o programa susieta su CPU identifikaciniu numeriu ar tinklo plokštės MAC adresu. Tas pats aktyvavimo kodas negali būti panaudotas diegiant kitą programą, o taikomoji programa, įdiegta viename kompiuteryje, negali būti įdiegta kitame **[7]**.

Licencijos raktas (angl. *license key*) – skaičių ir simbolių seka, kurią reikia įvesti diegiant programą. Licencijos rakto apsaugą užtikrina sudėtingi algoritmai, todėl įsilaužėlis, norėdamas atspėti raktą, turi žinoti algoritmą.

Kodo maskavimas, keitimas – programos šaltinio kodo pakeitimas į sunkiau sintaksiškai suprantamą kodą, kuris apsunkina įsilaužėliui surasti svarbių kodo dalių. Kodo maskavimas ir keitimas atliekamas automatiniais programiniais įrankiais, dalis jų pateikta lentelėje 7.

Kodo suspaudimas (angl. *executable compression*) – vykdomosios bylos programinis suspaudimas ir sujungimas su išskleidimo (angl. *decompression*) kodu. Kai vykdoma suspausta byla, išskleidimo kodas išskleidžia suspaustą bylą prieš pradedant ją vykdyti. Suspaustame kode gali būti talpinami kreipiniai į apsaugos raktą. Įsilaužėlis apgrąžos inžinerijos metodu bandydamas išsiaiškinti kodo veikimą pirmiausia turi rasti išskleidimo kodo pabaigą ir suspausto kodo pradžią, kurios suradimui reikalingas papildomas laikas. Vykdomosios bylos programinį suspaudimą galima atlikti automatiniais programiniais įrankiais: Armadillo Packer, EXECryptor, SmartKey GSS ir kitais įrankiais. Pagrindinis kodo suspaudimo trūkumas yra tai, kad kiekvienam suspaudimo įrankiui yra sukurtas išskleidimo įrankis, todėl programuotojas turi sukurti savitą kodo suspaudimo metodą konkrečiai programos apsaugai.

1.5. Aparatinių ir papildomų apsaugos metodų, skirtų programų apsaugai, palyginimas

Atlikus aparatinių ir programinių apsaugos metodų, naudojamų taikomosioms programoms apsaugoti nuo nelegalaus naudojimo, analizę išskiriamos dvi kompiuterizuotos apsaugos grupės – aparatiniai ir programiniai metodai. Grupės pateiktos 7 lentelėje.

Lentelė Nr. 7. Programiniai ir aparatiniai apsaugos metodai

Programinis apsaugos metodas			Aparatinis apsaugos metodas			
Licenziniai raktai, bylos	Programų aktyvavimas	Kodo maskavimas, keitimas, suspaudimas	Optinių diskų skaitytuvai (ODS)	Praplėsties plokštės (PP)	Apsaugos raktai (AR)	Kompiuterių techniniai komponentai (KTK)

Aparatiniai apsaugos metodai turi svarbų pranašumą prieš programinius apsaugos metodus, nes naudoja principą tai ką tu turi (apsaugos raktą, optinį diską, praplėsties plokštę). Be to aparatiniam apsaugos metodui galima pritaikyti ir programinius apsaugos metodus.

8 lentelėje pateikti trys pagrindiniai atakų metodai prieš programų apsaugą aparatiniais ir programiniais metodais. Egzistuoja ir daugiau atakų metodų, tačiau juos galima įvykdyti kombinuojant šiuos tris metodus.

Lentelė Nr. 8. Atakos ir prieš jas nukreipti aparatiniai ir programiniai apsaugos metodai

<table>
<tr><th colspan="3" rowspan="2">Atakos</th><th colspan="2">Apsaugos</th></tr>
<tr><th>Metodai</th><th>Įrankiai</th><th>Metodai</th><th>Įrankiai</th></tr>
<tr><td rowspan="8">Apgrąžos inžinerija</td><td colspan="2" rowspan="4">Vertimas (pasyvus)</td><td rowspan="4">IDA-Pro
Pic-Simulator_DE</td><td>Dalies programos slėpimas</td><td></td></tr>
<tr><td>Funkcijų perkėlimas į dll failus</td><td>Microsoft Visual Studio</td></tr>
<tr><td>Filigranavimas</td><td>Sandmark</td></tr>
<tr><td>ODS, PP, AR, KTK</td><td></td></tr>
<tr><td rowspan="4">Derinimas (aktyvus)</td><td rowspan="2">Vartotojo lygio</td><td rowspan="2">IDA-Pro, WinDgb, OllyDbg, W32Dasm, SoftIce</td><td>Maskavimas, Suspaudimas</td><td>Themida, EXECryptor, AntiCrack</td></tr>
<tr><td>ODS, PP, AR, KTK</td><td></td></tr>
<tr><td rowspan="2">Branduolio lygio</td><td rowspan="2">IDA-Pro, WinDbg</td><td>Aukšto lygio programavimo kalba</td><td>C++, Java,...</td></tr>
<tr><td rowspan="2">ODS, PP, AR, KTK</td><td rowspan="2"></td></tr>
<tr><td rowspan="2">Emuliavimo</td><td rowspan="2">[62](1)</td></tr>
<tr><td>Aukšto lygio programavimo kalba</td><td>C++, Java,...</td></tr>
<tr><td colspan="3" rowspan="2">Nuotolinės prieigos kodo įterpimo „process dump“ ataka (įsiterpusio žmogaus atakų grupė)</td><td rowspan="2">Process Dumper su Netstat</td><td>Ugniasienės ir IDS sistemos,</td><td>NIDS, OSSEC HIDS..</td></tr>
<tr><td>Kodo analizė(2)</td><td>Memory Parser</td></tr>
</table>

		Dalies programos slėpimas	
		Maskavimas	
		ODS, PP, AR, KTK	
Programos klonavimas	*„Copy-Paste“, Nero Burning ROM,*	Maskavimas, Programos dalies pakeitimas po kopijavimo,	
		Kopijas aptinkančios sistemos	*Dup, Jplag, CloneDr, ...*
		PP, AR, KTK	
		Dalies programos slėpimas	

(1) – apsaugos raktų emuliavimu užsiima ir jų gamintojai, nes prarasto ar sugadinto modulio programinės kopijos gamyba užtrunka mažiau laiko nei naujo aparatinio modulio paruošimas.

(2) – kodo analizė yra pasyvi apsauga, programos kūrėjų atliekama periodiškai įvertinant programos saugumo spragas ir jas ištaisant.

8 lentelės paminėti apsaugos metodai turi privalumų ir trūkumų, todėl 9 lentelėje pateikti apsaugos metodų palyginimai pagal jų savybes.

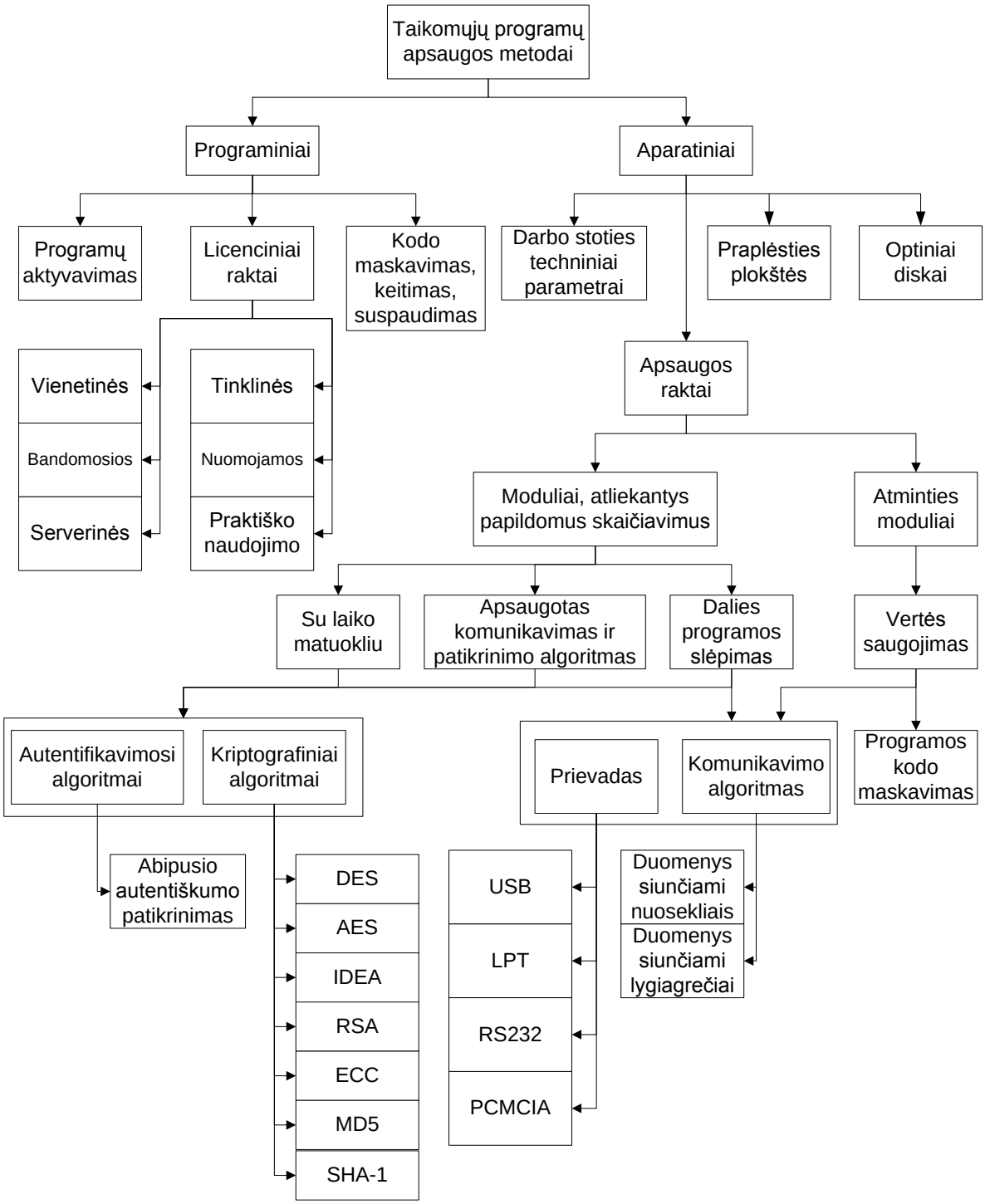
Lentelė Nr. 9. Apsaugos metodų, kurie gali būti taikomi aparatiniuose apsaugos metoduose, palyginimas

Metodas	Reikalingos gilesnės informatikos žinios	Platesnis programinių įrankių pasirinkimas	Reikalingas programos šaltinio kodo keitimas	Atsparesnė apgrąžos inžinerijos atakai	Atsparesnė kodo įterpimo „*process dump*“ atakai	Atsparesnė programos klonavimo atakai
Maskavimas	+	–	+	–	+	–
Filigranavimas	+	–	+	–	–	–
Dalies programos slėpimas(1)	+	–	+	+	+	+
Kopijas aptinkančios sistemos	–	+	–	+	–	+
Kodo suspaudimas	–	+	+	–	–	–

(1) Dalis programos įdiegta į išorinę atmintinę, pvz. apsaugos rakto nuolatinę atmintį.

9 Lentelės rezultatai gauti atlikus literatūros [5, 11, 14, 20, 27, 28, 29, 30, 32, 42, 43, 44, 46, 59] analizę, pagal kurią matyti, jog dalies programos slėpimas ir vykdymas yra atsparesnis išvardytoms atakoms, nei likusieji apsaugos metodai. Tyrimo dalyje pateikiu atliktus eksperimentus, kurie įrodo analizės rezultatus.

Atlikus aparatinių ir programinių apsaugos metodų, skirtų taikomųjų programų apsaugai nuo nelegalaus naudojimo, analizę 16 pav. pateikiu apsaugos metodų klasifikacija.



16 pav. Programiniai ir aparatiniai programų apsaugos nuo nelegalaus naudojimo metodai

16 pav. pateikti programiniai ir aparatiniai apsaugos metodai gali būti naudojami kartu, todėl apsaugos raktams galima pritaikyti licencijos raktų metodą, kai apsaugos raktas yra naudojamas tinklu arba užtikrinantis bandomosios programos naudojimą, arba riboja dalį programos funkcijų. Aparatinį apsaugos metodą papildžius programiniu apsaugos metodu galima atsisakyti šifravimo, nes, jei naudojamas vienas apsaugos raktas prijungtas prie vieno kompiuterio, tai duomenys siunčiami per vieną kompiuterio prievadą ir į tinklą nepatenka. Taip randami sprendimai, tenkinantys įvairius poreikius ir lūkesčius.

1.6. Išvados

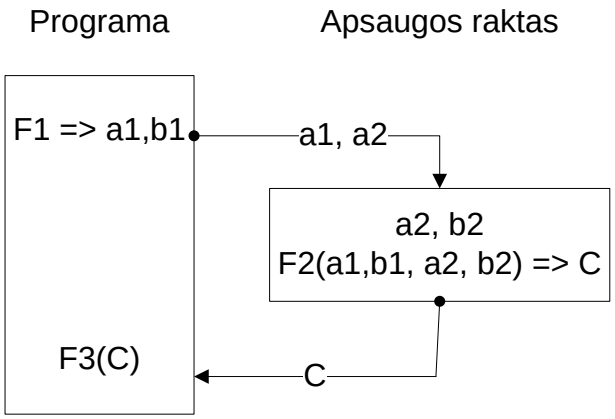
1. Atlikus aparatinių apsaugos metodų analizę išsiaiškinta, kad apsaugos lygio, diegimo laiko ir kainos santykiu apsaugos raktai yra pranašesni prieš optinius diskus, praplėsties plokštes ir kompiuterio aparatinius elementus.
2. Atlikus programų apsaugos aparatiniais ir programiniais metodais analizę išsiaiškinta, kad dalies programos slėpimas ir vykdymas apsaugos rakte geriau apsaugo nuo apgrąžos inžinerijos, nuotolinės kodo įterpimo ir „klonavimo“ atakų, nei maskavimas, filigranavimas, kopijas aptinkančios sistemos ir kodo suspaudimas.
3. Atlikus taikomųjų programų apsaugos aparatiniu raktu analizę išsiaiškinta, kad pažeidžiamiausia apsaugos vieta yra programos ir apsaugos rakto komunikavimas, o galimas sprendimas yra perkelti dalį skaičiavimų į apsaugos raktą, todėl būtina ištirti kiek siūlomas sprendimas atsparus atakoms ir palyginti su tradiciniais apsaugos metodais, kurie aptarti analizėje.

2 SIŪLOMAS IŠMANAUS APARATINIO APSAUGOS RAKTO MODELIS

Įvertinus apsaugos metodus, kurie taikomi programų apsaugai, nustatyta, kad programos šifravimas, komunikavimo šifravimas, programos maskavimas po atitinkamo laiko yra įveikiamos apsaugos, todėl siūlomas išmanaus aparatinio apsaugos rakto modelis dalį programos saugo ir vykdo apsaugos rakto viduje.

2.1. Išmanaus kliento principas aparatinėje programų apsaugoje

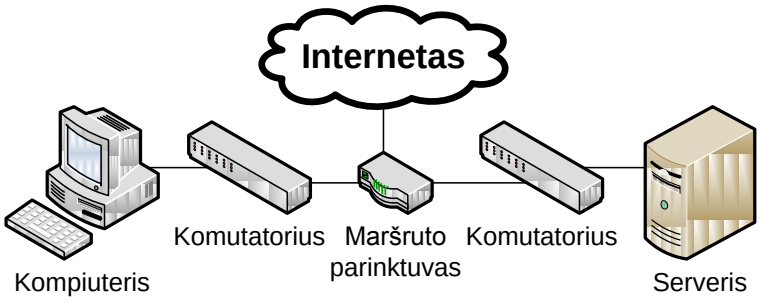
Norėdami apsaugos rakte priimti duomenis, atlikti papildomus skaičiavimus ir išsiųsti duomenis taikomajai programai, apsaugos raktas turi susidėti iš procesoriaus, operatyviosios atminties ir nuolatinės atminties. Šiuo atveju apsaugos rakto struktūra panaši į kompiuterio struktūrą. 17 pav. pateikiu programos ir apsaugos rakto blokinę schemą, pagal kurią sukūriau aparatinės apsaugos modelį. Schemoje pavaizduota programos ir apsaugos rakto sąveika. Programoje funkcija 1 apskaičiuoja dvi vertes, kurios siunčiamos į apsaugos raktą. Jame saugomos dvi vertės a2 ir b2. Gautos ir saugomos vertės naudojamos kaip funkcijos 2 argumentai, taip apskaičiuojama vertė C, kuri toliau siunčiama į programą ir naudojama kaip funkcijos 3 argumentas.



17 pav. Kuriamo modelio schema

Taikomosios programos ir apsaugos rakto sąveika palyginama su kompiuterio ir serverio sąveika, kuri pavaizduota 18 pav. Šiuo atveju apsaugos raktas veiktų kaip serveris, o kompiuteryje būtų įrašyta saugoma programa. Atsisakius tarpinių įrenginių ir sujungus kompiuterį su serveriu tiesiogiai gaunamas taikomosios programos ir apsaugos rakto junginys.

Apsaugos raktas kaip serveris gali atlikti papildomus skaičiavimus. Tokia sistema, kai programos klientinė dalis veikia kompiuteryje, o serverio dalis serveryje ir programos



18 pav. Kompiuterio ir serverio sąveika

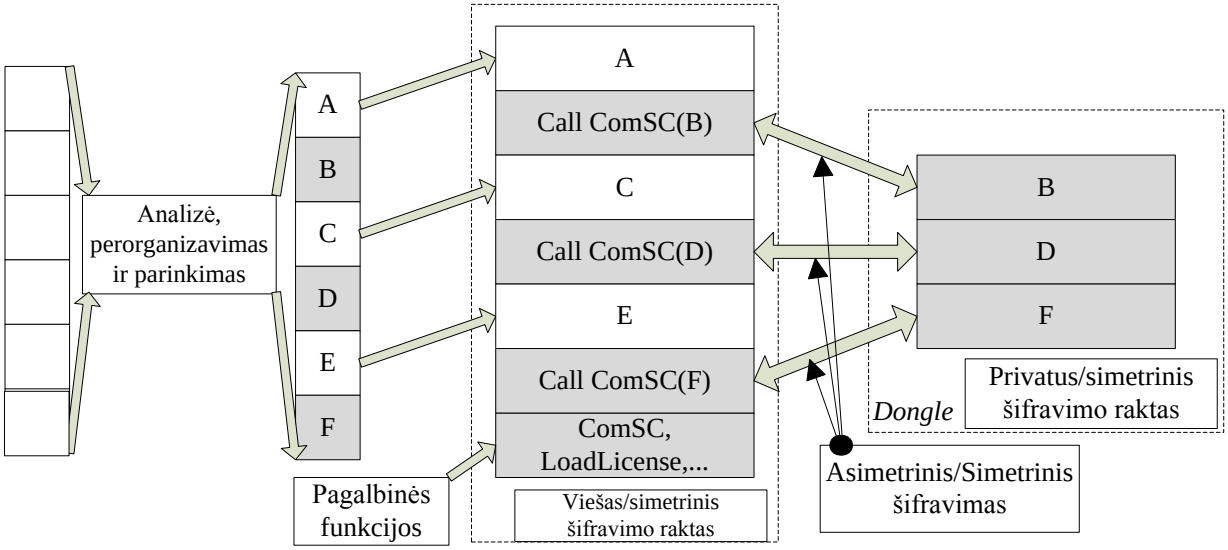
dalys asinchroniniu būdu apsikeičia duomenimis, vadinama išmaniojo kliento (angl. *fat client*) sistema.

Taigi, saugoma taikomoji programa ir apsaugos raktas veikia kaip išmaniojo kliento ir serverio sistema. Taikomosios programos pagrindinė dalis įdiegta įrenginyje, kuris turi vartotojo periferinius įrenginius (vaizduoklį, pelę ir klaviatūrą). Todėl laikysime, kad vartotojas tiesiogiai naudojasi klientine programos dalimi, įdiegta kompiuteryje, o prie apsaugos rakto serverinės dalies vartotojas prieigos neturi.

Taigi, simuliavimo modelis susideda iš dviejų pagrindinių dalių: taikomosios programos (klientinės dalies) ir apsaugos rakto (serverinės dalies).

2.2. Siūlomas dalies programos vykdymas apsaugos rakte

Palyginus A. Mana ir bendraautorių siūlomą programos apsaugos spredimą su dalies programos vykdymu apsaugos rakte, pagal 14 pav. siūloma iš programos apsaugos etapų atsisakyti vertimo ir maskavimo etapo ir funkcijų šifravimo etapo, nes funkcijos būtų saugomos apsaugos rakte nešifruotos. Taip sutrumpėja programos dalies vykdymas. Apsaugos metodas, kai dalis programos vykdoma apsaugos rakte, pateikiamas 19 pav.



19 pav. Paskirstyta programos apsauga

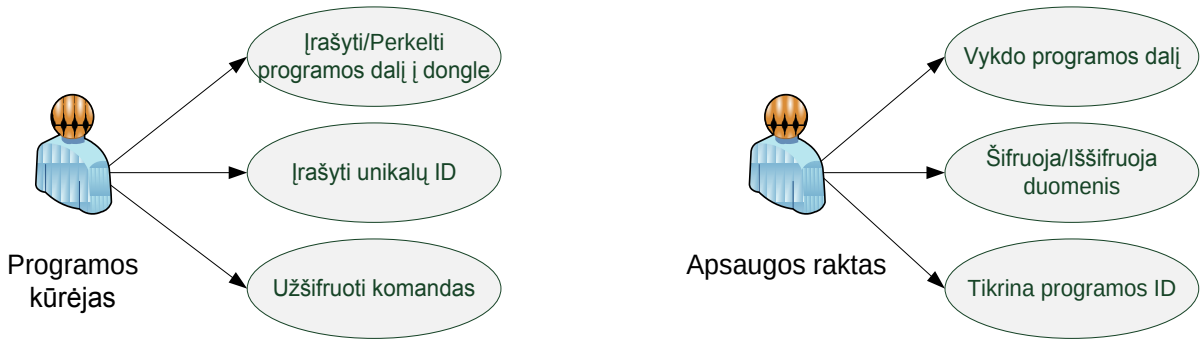
2.3. Aparatinės apsaugos modelio funkciniai ir nefunkciniai reikalavimai

Taikomosios programos ir aparatinės apsaugos raktai remiasi klientinės programos ir serverio modeliu. Saugomą taikomąją programą laikome klientine programa, o apsaugos raktą serveriu, kuriame įdiegta programos dalis. Dažniausiai klientinė programa yra nutolusi nuo serverio, tačiau šiuo atveju apsaugos raktas yra tiesiogiai prijungtas prie kompiuterio, kuriame įdiegta programa. Iš to seka, kad du įrenginiai turi būti tarpusavyje suderinti, todėl apsaugos

rakto kūrėjas turi užtikrinti programos kūrėjui ir vartotojui programos funkcionalumą ir apsaugą už priimtiną kainą. Tam išskiriami aparatinio apsaugos modelio funkciniai ir nefunkciniai reikalavimai, funkciniai reikalavimai pavaizduoti 20 pav.

Funkciniai reikalavimai:

1. **Tikrinti programos ID.** Programoje saugomas unikalus serijinis numeris, kuris siunčiamas patikrinimui į apsaugos raktą. Apsaugos raktas patikrina gautą serijinį numerį su atmintyje saugomu serijiniu numeriu. Jei vertės sutampa, tai leidžiama vykdyti vartotojo autentifikavimą ir tolesnį programos vykdymą.
2. **Vykdo programos dalį.** Apsaugos raktas priima užklausas iš programos, jei gaunama funkcijos užklausa, tai vykdo užklaustą funkciją ar funkcijas. Gautą rezultatą grąžina programai.
3. **Šifruoja/Iššifruoja duomenis**. Apsaugos raktas patikrina ar programa palaiko komunikavimo šifravimą. Jei taip šifruoja išsiunčiamus duomenis ir iššifruoja gautus duomenis.
4. **Įrašyti/perkelti programos dalį į apsaugos raktą**. Programos kūrėjas pasirenka, kurios programos funkcijos bus saugomos ir perkelia jas į apsaugos raktą.
5. **Įrašyti unikalų ID**. Programos kūrėjas sugeneruoja unikalų programos ID, su kuriuo identifikuosis apsaugos raktui.
6. **Užšifruoti komandas**. Programos kurėjas užšifruodamas vykdomas komandas padidina apsaugos laiką, kurio reikia išsiaiškinti ką vykdo konkreti komanda.



20 pav. Aparatinės apsaugos funkciniai reikalavimai

Nefunkciniai reikalavimai:

1. **Antivirusinių ir užkardų suderinamumas su apsauga**. Apsaugos rakto ir programos kūrėjai turi užtikrinti, kad kompiuterio antivirusinės programos ir užkardos pripažintų apsaugos raktą ir vykdomą programą kaip saugius, nes aparatinė apsauga paliečia šaknines operacinės sistemos dinamines bibliotekas. Programos kodo dalys, kurios komunikuos su apsaugos raktu, turi būti suderinamos su apsaugos rakto kūrėju.

2. **Programos dalies įdiegimas į modulį.** Apsaugos rakto kūrėjas turi parūpinti atminties dydį nemažesnį nei 100 KB, į kurį taikomosios programos kūrėjas galės įrašyti programos dalį. Didesnis atminties dydis leis vykdyti daugiau nei vieną programos funkciją. Programos kūrėjas atsakingas, kad programos dalis neviršytų duoto atminties dydžio.
3. **Apsaugos rakto skaičiavimo resursai.** Programos kūrėjas ir apsaugos rakto kūrėjas turi atsižvelgti, kad apsaugos raktas yra ribotų matmenų ir į jį galima įtaisyti riboto dydžio ir galingumo techninę įrangą, todėl įrenginiui negalima užduoti skaičiavimų trunkančių ilgiau nei numatyta programos funkcijų vykdymui.
4. **Šifruotas komunikavimas**. Programa, įdiegta kompiuteryje, ir programos dalis, įdiegta apsaugos rakte, yra nešifruojama, nes apsaugos raktas retai būna įsilaužimo objektas [8], o programa be funkcionuojančios dalies kodo ir nešifruota bus retas įsilaužimo objektas, nes tai semantiškai sudėtingas uždavinys. Tačiau duomenys, perduodami tarp programos ir apsaugos rakto, turi būti šifruojami, nes iš perduodamos informacijos įsilaužėlis gali atkurti reikiamus duomenis, todėl programos kūrėjas turi užtikrinti, kad duomenys priimami iš modulio bus saugiai dešifruojami ir siunčiami duomenys į modulį bus užšifruojami, o apsaugos rakto kūrėjas – kad duomenys priimami iš programos būtų saugiai dešifruojami ir siunčiami duomenys patikimai užšifruojami.
5. **USB sąsaja**. USB sąsaja plačiausiai paplitusi personaliniuose kompiuteriuose, todėl apsaugos rakto kūrėjas turi paruošti modulį su USB jungtimi.
6. **Apsaugos rakto techninė įranga.** Apsaugos rakto kūrėjas turi parinkti optimalius techninius komponentus: procesorių, darbinę atmintį, nuolatinę atmintį, kurie veiktų su 5V įtampa, nes USB sąsaja suderinta su 5V įtampa. Techniniai komponentai turi būti tokio dydžio, kad tilptų į apsaugos rakto korpusą.
7. **Kaina**. Vartotojas suinteresuotas, kad apsauga nekainuotų brangiau nei pati programa, o apsaugos rakto kūrėjas turi užtikrinti, kad modulio gamyba nebūtų brangi ir neviršytų 250LT. Tokios kainos pakanka apsaugos raktui su procesoriumi, darbine atmintimi ir nuolatinėmis atmintinėmis.

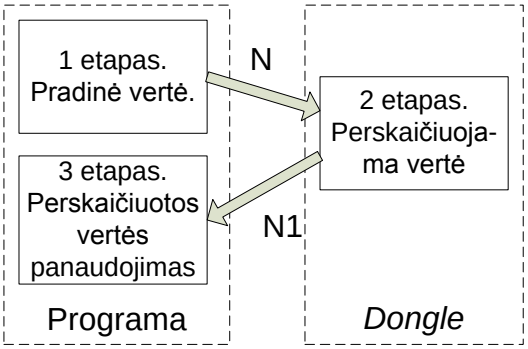
2.4. Siūlomo išmanaus aparatinės apsaugos rakto modelis

Siūlomi du aparatiniai apsaugos modeliai, kuriuose pritaikytas išmaniojo kliento principas.

Pirmas taikomosios programos apsaugos su apsaugos raktu modelis kuriamas pagal MeiHong L. ir JiQiang L. vientisumo ir abipusio autentifikavimo patikrinimo straipsnį [31],

nes jame aprašyta identifikavimo, autentifikavimo ir vientisumo patikrinimo eiga, tai pagrindinės informacijos saugos taisyklės [3]. Apsauga susideda iš trijų etapų.

1. Programoje ir apsaugos rakte yra įrašytas unikalus serijinis numeris, kuris automatiškai tikrinamas prijungus apsaugos raktą prie kompiuterio. Jei serijiniai numeriai sutampa su programoje ir apsaugos rakte saugomais ID numeriais, programa pereis prie antro autentiškumo patikrinimo etapo.
2. Vartotojas įveda slaptažodį arba asmeninį identifikacinį numerį (angl. *PIN*), kuris patikrinamas su apsaugos rakte saugomu slaptažodžiu arba PIN. Jei reikšmės sutampa programa pereina prie pilnaverčio funkcionavimo ir vientisumo tikrinimo.
3. Taikomosios programos visos originalios funkcijų santraukų (angl. *hash*) vertės yra įrašytos į apsaugos raktą. Vartotojas naudodamas programą pasirenka vieną iš funkcijų. Pirmiausia apskaičiuojama pasirinktos funkcijos santraukos vertė ir patikrinama su apsaugos rakte saugoma verte. Jei vertės sutampa, tai vientisumas patikrintas teigiamai. Patikrinimo ciklas atliekamas su visomis pasirenkamomis funkcijomis.



21 pav. 4-tas apsaugos etapas

Apgrąžos inžinerijos metodais įsilaužėlis gali apeiti šiuos apsaugos etapus, todėl jie naudojami apsaugai nuo eilinių vartotojų. Apsaugai nuo įsilaužėlių siūlomas ketvirtas apsaugos etapas – dalies duomenų siuntimas į apsaugos raktą, duomenų apdorojimas ir siuntimas atgal taikomajai programai, kuri duomenis panaudoja funkcijų vykdymui. Ketvirtas apsaugos etapas yra padalinamas į tris smulkesniu etapus, kurie pavaizduoti 21 pav. Programa būtų vykdoma sėkmingai jei visi trys etapai būtų vykdomi programoje, tačiau taip įsilaužėlis matytų visą proceso eigą ir galėtų ją dublikuoti arba keisti. Apsaugos atveju įsilaužėlis negali iššifruoti siunčiamos šifruotos vertės N ir gaunamos apskaičiuotos šifruotos vertės N1, taip pat nežino kaip N1 vertė apskaičiuojama, todėl įsilaužėliui nebūtų prasmės keisti kodo. N1 vertės pakeitimas į kitą vertę reikštų programos proceso nutraukimą.

Modeliavimo darbai buvo atliekami su Matlab programinio paketo moduliu Simulink, kuriame buvo realizuojami keturi apsaugos etapai. Simulink modulio signalai gali būti tik skaitinės vertės, tačiau visi apsaugos etapai reikalauja skaičiuoti santraukos vertes, kurios susideda iš skaitinių ir raidinių simbolių, todėl norėdami apskaičiuoti ir palyginti santraukos vertes, turime jas persikelti į Matlab darbo langą (angl. *To workspace*). Darbo lange atlikus

skaičiavimus ir palyginimus į Simulink modelį siunčiamos (angl. *From workspace*) vertės, kurios įgalina vykdyti sekantį apsaugos etapą.

Taikomosios programos ir apsaugos rakto modelis pavaizduotas 22 pav. Jame išskirtos 4 grupės, kurios atitinka 4 apsaugos etapus. Kairė modelio pusė atitinka taikomąją programą, o dešinė pusė – apsaugos raktą. Tamsesnės spalvos modelio blokai reiškia, kad tai vertės atitinkamai saugomos programos atmintyje ir apsaugos rakto atmintyje.

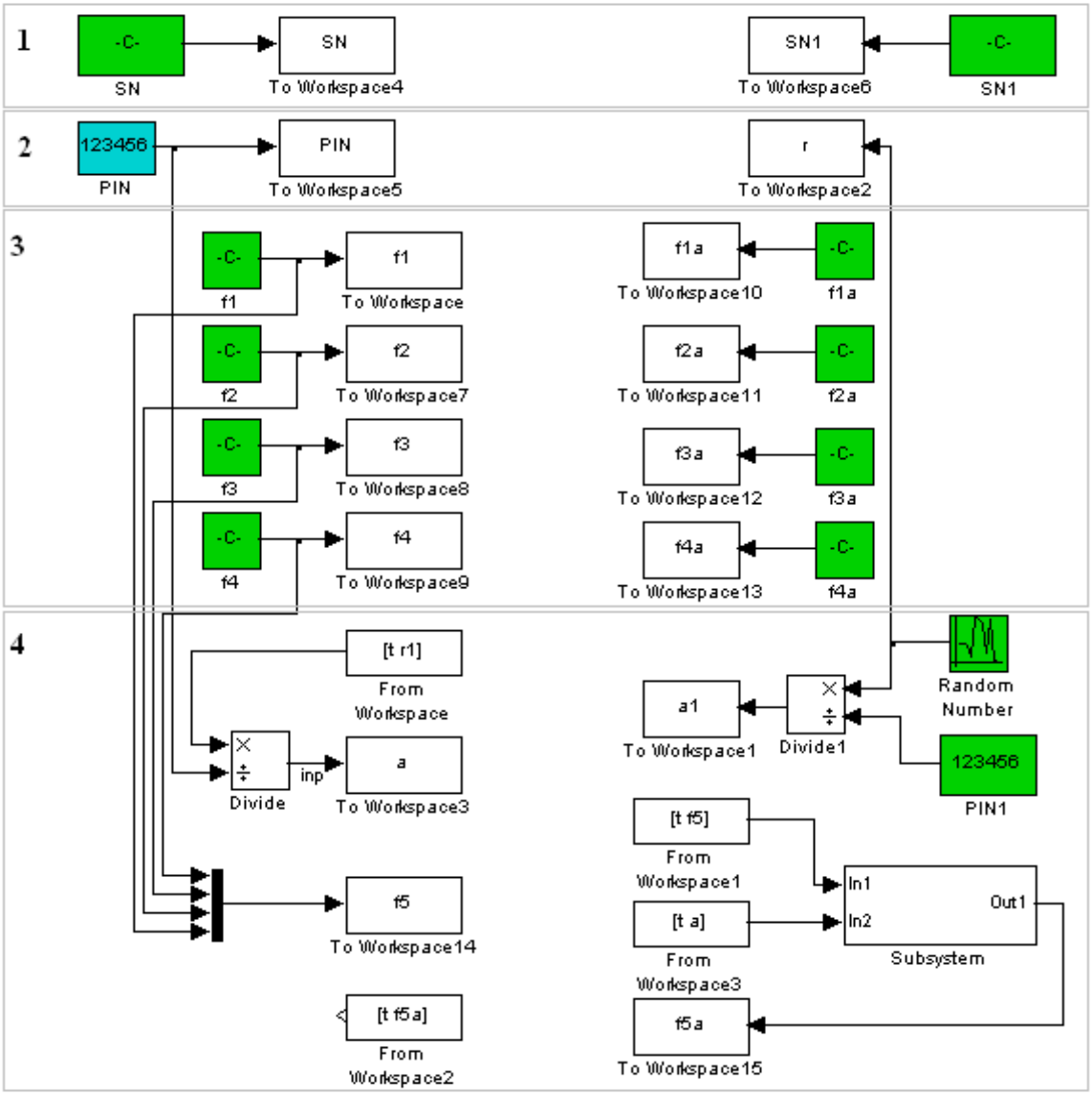
1 etapas. **Apsaugos rakto autentifikavimas**. Iš bloko SN į Matlab darbo langą siunčiamas unikalus serijinis numeris ir apskaičiuojama jo santraukos vertė H(SN), kuri patikrinama su bloko SN1 į darbo langą atsiųstu serijiniu numeriu SN1 santraukos verte H(SN1). Jei H(SN) sutampa su H(SN1) tai galima vykdyti sekantį etapą.

2 etapas. **Vartotojo autentifikavimas**. Apsaugos raktas sugeneruoja atsitiktinį skaičių r kuris siunčiamas iš atsitiktinio skaičiaus generatoriaus (angl. *random number generator*) bloko. Skaičius r padalinamas iš atmintyje saugomo PIN1 numerio, PIN1 blokas. Gauta dalmens vertė D siunčiama į darbo langą. Vartotojas įrašo savo PIN numerį, blokas PIN. Jei vartotojo įvestas numeris teisingas iš dalmens vertės D vartotojas gali išskaičiuoti teisingą atsitiktinį skaičių r. Jei r vertė išskaičiuojama teisingai, tai bus sėkmingai vykdomas trečias etapas.

3 etapas. Funkcijų vientisumas. Šiame etape patikrinamas programos vientisumas. Konstantos f1, f2, f3 ir f4 reiškia programos funkcijas (realioje programoje funkcijų skaičius didesnis), kurių identiškos kopijos saugomos apsaugos rakto atmintyje f1a, f2a, f3a ir f4a blokuose. Konstantų vertės siunčiamos į darbo langą. Iš kiekvienos funkcijos apskaičiuojama santraukos vertė H(f1), H(f2), H(f3) ir H(f4). Visos santraukos vertės sudauginamos ir apskaičiuojama santraukos vertė $H(f_{1\text{-}4})$= H(f1)×H(f2)×H(f3)×H(f4). $H(f_{1\text{-}4})$ vertė padauginama iš atsitiktinio skaičiaus r ir gaunama vertė C= $H(f_{1\text{-}4})$×r. Apsaugos raktas atmintyje laiko kiekvienos programos funkcijos santraukos vertes, todėl C1 vertė apskaičiuojama atskirai. f1a, f2a, f3a ir f4a vertės siunčiamos į Matlab darbo langą, kuriame atliekmi identiški veiksmai su apsaugos rakto atmintyje saugomoma atsitiktine reikšme r. Gauta santraukos vertė C1 patikrinama su pradine C verte. Jei abi vertės sutampa tai programos vientisumas patikrinamas teisingai.

4 etapas. Išmaniojo kliento modelis. Šiame etape vyksta programos pagrindinės funkcijos vertės siuntimas į apsaugos raktą, kuriame vertė perskaičiuojama ir naujas rezultatas siunčiamas į programą. Vykdomos funkcijos vertė (f1, f2, f3 arba f4) ir dalmens vertė D siunčiama į apsaugos raktą, kuriame vertės siunčiamos į posistemės (angl. *Subsystem*) bloką. Posistemės bloke saugoma programos dalis. Gautas rezultatas f5a siunčiamas į programą (angl. *To Workspace 15*). F5a vykdoma kaip funkcijos dalis, o vykdomos funkcijos

santraukos vertė patikrinama su apsaugos rakte saugoma atitinkamos funkcijos santraukos verte. Jei patikrinimas sėkmingas, grįžtama prie 3 etapo arba kartojamas 4 etapas.



22 pav. Taikomosios programos ir apsaugos rakto struktūra

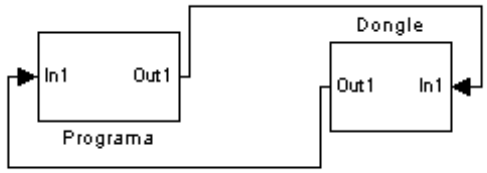
Kairėje 22 pav. dalyje parašyti skaičiai reiškia apsaugos etapus.

2.2 poskyryje minėta, kad duomenų perdavimui tarp programos ir apsaugos rakto bus naudojamas asimetrinis šifravimas, o privatus raktas saugomas apsaugos rakte. Todėl simuliavimo modeliu galima tikrinti asimetrinių šifravimo algoritmų spartas, kaip perduodamų duomenų sparta priklauso nuo šifravimo algoritmo.

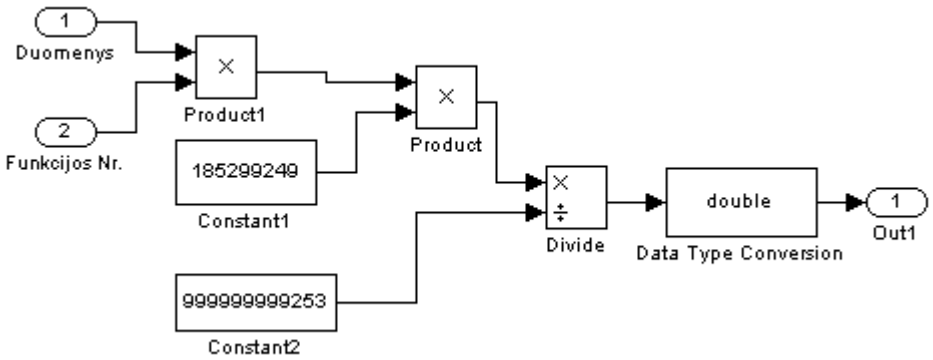
Taigi, keturi apsaugos etapai skirti taikomosios programos apsaugai nuo neteisėto naudojimo. Jei įsilaužėlis apgrąžos inžinerijos metodu sugeba apeiti pirmus tris apsaugos etapus tai 4 etapas atsparus apgrąžos inžinerijos atakai. Sekantis įsilaužėlio žingsnis galėtų būti šifruotos N1 vertės nuskaitymas iš kompiuterio darbinės atminties, todėl sekantis apsaugos užtikrinimo žingsnis turi būti laiko, kai darbinėje atmintyje saugoma N1 vertė, trumpinimas. Atsitiktinių funkcijų perkėlimas į apsaugos raktą apsaugo modulį nuo

klonavimo. Jei parenkama n žinomų funkcijų kombinacijų, tai tikimybė, kad įsilaužėlis pasirinks reikiamą kombinaciją bus lygi 1/n.

Antrame modelyje programos ir apsaugos rakto komunikavimo simuliavimas atliekamas Simulink programoje, kurioje kintamieji išreiškiami signalų skaitinėmis vertėmis, todėl simbolinės vertės simuliavime nenaudojamos. Modelyje kintamieji yra sveiki dešimtainiai skaičiai, nes Simulink dešimtąsias ir žemesnes vertes simuliavimo eigoje suapvalina ar sumaišo. Modelis susideda iš dviejų pagrindinių dalių: taikomosios programos ir apsaugos rakto, kurie tarpusavyje komunikuoja nuosekliu prievadu. 23 pav. pavaizduotos rodyklės skirtos duomenų siuntimui ir priėmimui nuosekliu prievadu. Komunikavimas pradedamas programoje, kai siuntimui paruošiami duomenys ir funkcijos numeris, kuris nurodo į kurią funkciją bus siunčiami duomenys. Duomenys paruošiami siuntimui, kai suformuojamas paketas, kuris susideda iš penkių dalių: paketo pradžios, duomenų, funkcijos numerio, kontrolinės sumos ir paketo pabaigos. Paketo pradžia lygi 6464, kuri, pagal ASCII lentelę [63], atitinka @@, taip užtikrinama unikali paketo pradžia. Duomenys – dešimtainis skaičius, kintantis nuo 1 iki 10^{12}. Dydžio apribojimą lemia modelyje pasirinktos kontrolinės sumos funkcija, tačiau pakeitus funkciją duomenų dydį galima padidinti. Funkcijos numeris – dešimtainis skaičius, kintantis



23 pav. Programos ir apsaugos rakto komunikavimas

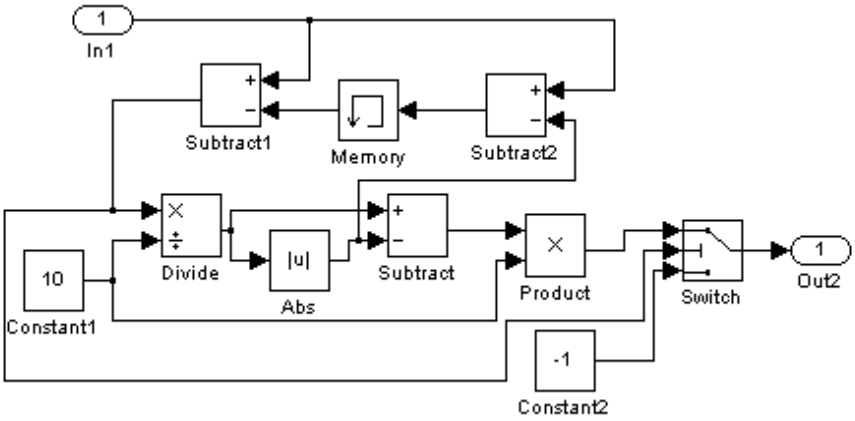


24 pav. Santraukos funkcijos modelis

nuo 1 iki n, kur n funkcijų skaičius.

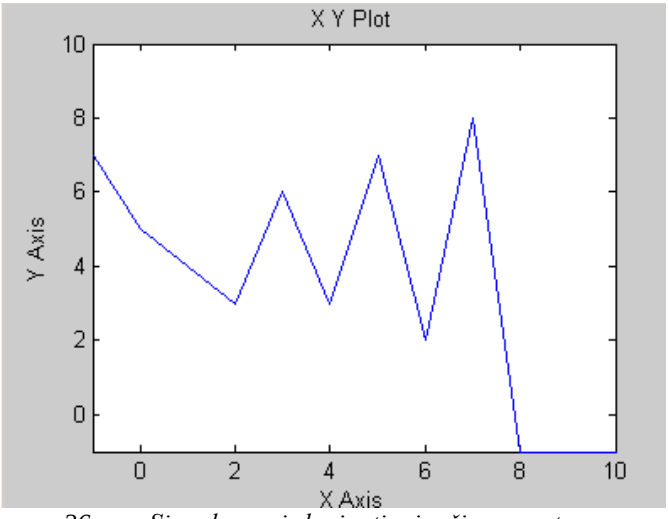
Modeliavimo metu naudojamos trys funkcijos, nes didesnis funkcijų skaičius modeliavimo rezultatų nekeičia. Kontrolinė suma yra santraukos funkcija, kuri iš duomenų ir funkcijos vertės paskaičiuoja santraukos vertę. Santraukos funkcijos algoritmas [64] pavaizduotas 24 pav., kuriame 0 < įvestis < constant2, 0 < išvestis < constant1, constant1 < constant2.

Simulink modelyje turi būti naudojami vienodo formato signalai, t.y. integer, double, int16 arba int32. Jei du signalai, išreikšti dešimtainiais skaičiais, tačiau skirtingų formatų bus gaunama simuliacijos klaida. Skaičių apvalinimui naudojami skirtingi signalų formatai, todėl dalybos (*angl.* divide) bloko išvesties signalo formato pakeitimui reikalingas signalo tipo keitimo (*angl.* data type conversion) blokas. Paketo pabaiga lygi 5959, kuri atitinka ;; , taip užtikrinama unikali paketo pabaiga.



25 pav. Skaičių atskyrimas, imituojant siuntimą bitas po bito

Nuosekliu prievadu paketai siunčiami bitas po bito, todėl suformuotas paketas talpinamas į masyvą ir skaičius po skaičiaus siunčiamas į apsaugos raktą, taip simuliuojamas nuoseklus duomenų siuntimas. Skaičių atskyrimui naudojmas 25 pav. pavaizduotas algoritmas, kuriame ciklišk ai kartojamas algoritmas tol kol įvesties skaičius tampa mažesnis už 10. Taip gaunamas signalas, kuris periodiškai keičia savo vertę tarp 0 ir 10. Duomenų vertė išreikšta signalu pavaizduota 26 pav. Vertė išreiškiama atbuline tvarka, t.y. vienetai siunčiami pradžioje, taip imituojamas FIFO (*angl.* First In First Out) masyvas. Masyvas yra statinis, todėl likusi masyvo vieta užpildoma -1 skaičiais, taip žymima vertės pabaiga.

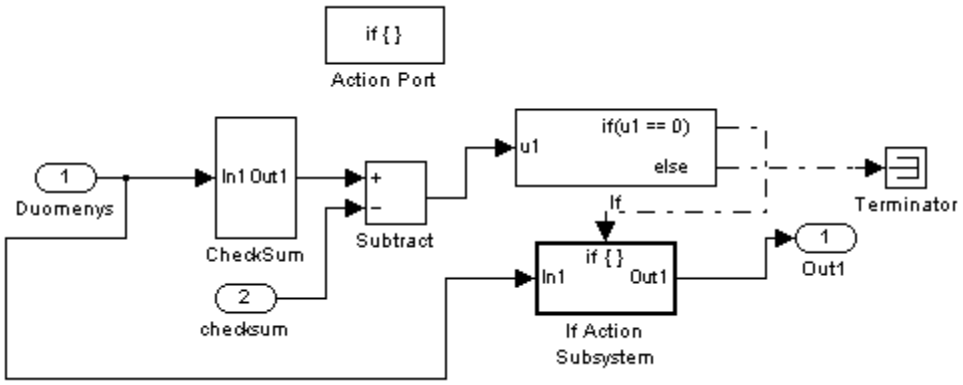


26 pav. Signalas, vaizduojantis siunčiamą vertę

Į apsaugos raktą atsiųstas paketas yra išskaidomas į penkias paketo dalis, kurios tikrinamos. Jei pradžios vertė sutampa su apsaugos rakte saugoma pradžios verte tai tinkrinamos sekančios paketo dalys. Apsaugos rakte patikrinamas duomenų vientisumas perskaičiavus santraukos vertę, jei atsiųsta ir apskaičiuota vertės sutampa, tai duomenys atsiųsti sėkmingai.

Sinchronizuotame Simulink modelyje būtinas etapų tikrinimas, nes modelyje atskiri blokai vykdomi skirtingu laiku, todėl vienetų atskirimo ir vienetų sudėjimo blokuose tarp skaitinių verčių įsiterpia nulinės vertės, kurios sugadina tolimesnius rezultatus. Sinchronizuodami modelį programos dalies blokams naudojamas vienas impulsų generatorius, o apsaugos rakto dalies blokams naudojamas antras impulsų generatorius, nes Simulink modelyje bendras sinchronizacijos šaltinis veikia kaip atskiri sinchronizacijos šaltiniai, kurių fazės yra vienodos, todėl bendras impulsų generatorius nebūtinas. Sinchronizuotame modelyje signalo vertė kinta priklausomai nuo laiko, todėl reikiama vertė išskiriama su signalo vėlinimo (angl. *transport delay*) bloku, kuris ignoruoja signalo dalį tam tikrą laiką, taip ignoruojami nereikalingi signalo nuokrypiai ir gaunama stabili signalo vertė.

Sėkmingai patikrinus duomenų vientisumą ir įsitikinus, kad paketo pabaiga atitinka apsaugos rakte saugomos pabaigos vertę, duomenys siunčiami į funkciją, kurios numeris buvo įrašytas pakete. Atlikus skaičiavimus funkcijoje, duomenys paruošiami siųsti atgal į programą. Pasinaudojus 24 pav. pavaizduotu algoritmu apskaičiuojama duomenų santraukos vertė, kuri pakuojama į paketą. Paketo struktūra susideda iš keturių dalių, nes nebesiunčiamas funkcijos numeris. Paketo vertė, skaičius po skaičiaus, išsiunčiama į programą, kurioje atstatoma ir tikrinamas jos vientisumas. Jei paketo pradžios ir pabaigos vertės sutampa su programos atmintyje saugomomis vertėmis ir apskaičiuota santraukos vertė, kurios patikrinimas pavaizduotas 27 pav., sutampa su atsiųsta santraukos verte, tai duomenys primti sėkmingai.



27 pav. Santraukos vertės patikrinimas

Apgrąžos inžinerijos atakai imituoti buvo panaudotas imitacinis apsaugos rakto apėjimas, kai pagal 23 pav. bloko Programa išvesties prievadas Out1 buvo tiesiogiai sujungtas su įvesties prievadu In1, taip apeinamas apsaugos raktas. Tačiau programa priima tą patį rezultatą, kurį išsiuntė, todėl toks rezultatas neturė prasmės tolesniam programos vykdymui ir gaunamas klaidos pranešimas.

2.5. Išvados

1. Sukurtas išmaniojo aparatinio apsaugos rakto modelis, kuris pagrįstas paskirstytų skaičiavimų principais.
2. Atliktas modelio imitacinis modeliavimas ir nustatyta, kad pritaikius paskirstytų skaičiavimų principą modelis užtikrina apsaugą nuo apgrąžos inžinerijos atakos.
3. Pažeidus paketo, kuris siunčiamas iš programos į apsaugos raktą arba atvirkščiai, vientisumą modeliavimo eiga sustabdoma, taip užtikrinimas teisingas komunikavimas ir apsauga nuo paketo pakeitimo.
4. Atskirtos programos funkcijos, kurios įrašytos apsaugos rakte gali būti vykdomos kelis kartus, tik po to bendras rezultatas siunčiamas į programą, taip padidinama apsauga, nes padidėja funkcijų rezultatų kombinacijų skaičius.

3. APSAUGOS RAKTO METODU PAGRĮSTO PROGRAMŲ APSAUGOS LYGIO ĮVERTINIMO TYRIMAS

Tyrimo metodika remiasi subjektyviais tyrėjo sugebėjimais, kurie imituoja taikomųjų programų įsilaužėlio veiksmus. Tyrimo programiniai įrankiai parinkti pagal atliktos analizės rezultatus ir išvadas. Tyrimo metu naudojami įsilaužimo metodai skirti tik moksliniui tyrimui. Remiantis sukurtu ir simuliacijoje panaudotu išmanaus apsaugos rakto modeliu sukurtas eksperimentinis apsaugos rakto prototipas, panaudojus Tmote Sky programuojamą USB modulį, kuris pavaizduotas 28 pav. Modulis programuojamas panaudojus Contiki operacinę sistemą ir virtualizacijos platformos VMware Instant Contiki programinį paketą [65]. Pagal pateiktas instrukcijas [66] atskirta programos dalis įrašyta į Tmote Sky modulį. Apsaugos rakto programos kodas pateikti priede Nr. 1. Tyrimo rezultatai yra laikas, kuris reikalingas programos apsaugos įveikimui.

28 pav. Tmote Sky programuojamas modulis

3.1. Tyrimo programiniai įrankiai, aplinkos ir programavimo kalbos

Tyrimas atliktas panaudojus šiuos programinius įrankius:

- *OllyDbg* v.1.10 – laisvai platinamas apgrąžos inžinerijos įrankis, kuris atvaizduoja programos kodą į asemblerio kodą ir leidžia kodą derinti. Tai pagrindinis tyrimo įrankis, kuriuo tirtos visos programos. Pasinaudojus OllyDbg programiniais priedais galima „apgauti“ (*angl.* to dump) programos apsaugos dalis, tačiau taip iškraipomi programos kreipiniai (*angl.* imports) į dll bibliotekas ir programa nebeveikia. Kreipinių atstatymui į .dll bylas naudojamos kreipinių atstatymo programos.
- *PeiD* v0.95 – laisvai platinamas programų įvertinimo įrankis, kuris vartotojui pateikia informaciją apie tiriamos programos programavimo kalbą, žinomus kodo suspaudimo įrankius, programos apsaugai naudojamus šifravimo algoritmus.
- *eXeScope* v6.50 – laisvai platinamas programos struktūros vaizdavimo įrankis, kuris skirtas Visual Basic programų tyrimui.
- *Resource Hacker* v3.5.2.84 – laisvai platinamas programos struktūros vaizdavimo įrankis, kuris skirtas Visual Basic programų tyrimui.

- *Import REConstructor* v.1.7 – laisvai platinama programa, kuri atstato tiriamos programos kreipinius į nepasiekiamas .dll bylas ir įrašo programai naują kreipinių adresų lentelę (*angl.* imports address table).
- *SoftwarePassport* v.8 – Armadillo branduoliu paremta programa, skirta programos kodo suspaudimui.
- *Dotfuscator Community Edition* – maskavimo programa, skirta C# kalba suprogramuotų programų maskavimui.
- *Crypto Obfuscator* – maskavimo programa, skirta C# kalba suprogramuotų programų maskavimui.
- *ProGuard* – maskavimo programa, skirta Java kalba suprogramuotų programų maskavimui.
- *.NET Reflector* v.6.6 – dekompiliavimo programa, skirta .NET suprogramuotos programos dekompiliavimui.
- *Dis# .NET Decompiler* v.3.1.4 – dekompiliavimo programa, skirta .NET suprogramuotos programos dekompiliavimui.
- *JAD* – dekompiliavimo programa, skirta Java suprogramuotos programos dekompiliavimui.
- *ArmStriper* v.0.1 – išskleidimo programa, skirta išskleisti Armadillo pagrindu suspaustai programai.
- *IDA Pro Free* v.5.0 – laisvai platinamas apgrąžos inžinerijos įrankis, kuris atvaizduoja programos kodą į asemblerio kodą ir leidžia kodą derinti.

Tyrimas buvo atliekamas Windows XP SP3 ir Windows Server 2003 R2 aplinkose. Aplinkose nebuvo naudojamos antivirusinės programos. Taip buvo apsisaugota nuo to, kad tyrimo įrankis ar tiriama programa bus palaikyta piktybine programa.

3.2. I tyrimo dalis. Eksperimentinės konsolės tipo programos apsaugos su komerciniu aparatiniu raktu efektyvumo tyrimas

Pirmos dalies pirmam tyrimo etapui buvo naudojamas komercinis DLP-D apsaugos raktas, kuris jungiamas per USB. Apsaugos raktas susideda iš unikalaus ID numerio ir 128B. EEPROM atminties, kuri skirta vartotojo duomenims. Iš gamintojo puslapio (http://www.ftdichip.com/FTSupport.htm) buvo atsiųstos API bylos, kurios skirtos programų komunikavimui su apsaugos raktu. API bylos skirtos Java, .NET C# ir C++. Taip pat

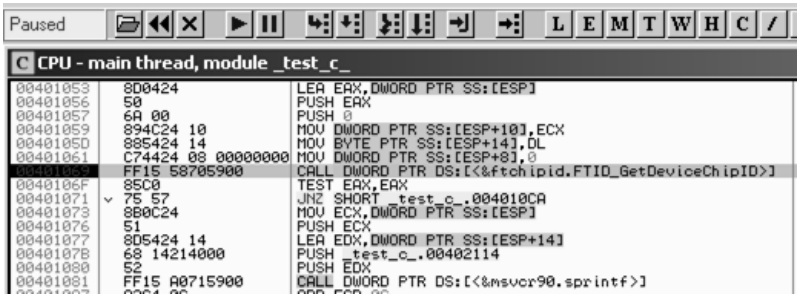
panaudotos įmonės pateiktos instrukcijos, kurios nurodo per kurias API bibliotekas programa kreipiasi į apsaugos raktą. Programos kodas pateiktas priede Nr. 2.

3.2.1. Komercinio apsaugos rakto ir eksperimentinės programos be papildomos apsaugos tyrimas

Buvo parašyta eksperimentinė komandinėje eilutėje vykdoma programa, kuri kreipiasi į prijungtą apsaugos raktą per API bibliotekas ir nuskaito jo ID numerį. Jei prijungtas reikiamas modulis, tai nuskaityta ID vertė palyginama su programoje saugomu ID numeriu. Jei ID numeris netinkamas arba apsaugos raktas neprijungtas, tai programa neveikia.

Šiam tyrimo etapui naudojamas apsaugos rakto identifikavimas be papildomų programos apsaugos metodų. Šio tyrimo etapo tikslas yra įvertinti apsaugs raktą kaip vienintelį programos autorizuotos prieigos įrankį.

Pirmiausia buvo tikrinama C++ kalba parašyta programa. Panaudotas OllyDbg apgrąžos inžinerijos įrankis, kuris eksperimentinę programą išvertė į asemblerio kodą. Pagrindinė užduotis buvo surasti programos kreipinius per API bibliotekas į apsaugos raktą. Kadangi apsaugos rakto gamintojas pateikia visus naudojamus API kreipinių paaiškinimus, tai OllyDbg aplinkoje paleidus kreipinių paiešką, per maždaug 10min. laikotarpį, buvo rastas kreipinys, kuris pavaizduotas 29 pav., į apsaugos raktą. Sužinojus kreipinio pavadinimą, per kelias papildomas minutes, randami likę kreipiniai į apsaugos raktą. Panaudojus JMP komandą buvo apeiti kreipiniai ir programa toliau veikė be apsaugos rakto.



29 pav. Kreipinio į apsaugos raktą radimas

Jei kreipinių pavadinimai nežinomi, tai panaudojus MSDN Windows API kreipinių aprašymus, galima rasti kreipinių per API bibliotekas pavadinimus į išorinius įrenginius.

JAVA ir .NET C# kalbomis parašytų programų tyrimas skiriasi nuo C++ programos, nes virtualūs interpretatoriai neatvaizduoja vykdomo kodo vartotojo lygyje. Tam būtų galima panaudoti specializuotus OllyDbg priedus, tačiau pasinaudojus dekompiliatoriais gauname pradinius programos kodus. .NET C# programai panaudojus .NET Reflector dekompiliatorių, o Java programai panaudojus JAD dekompiliatorių gaunami pradiniai programų kodai. Toliau

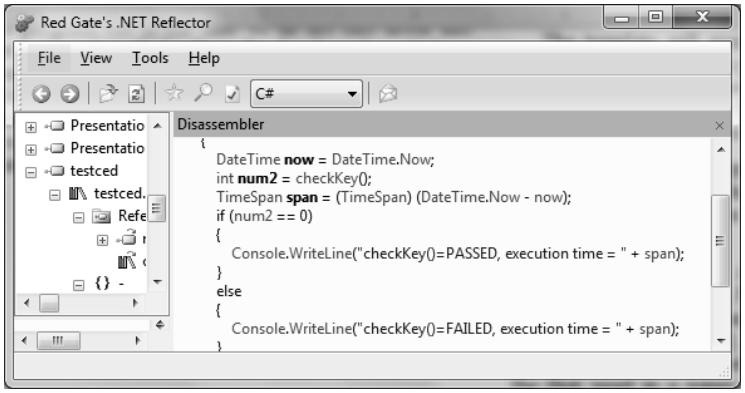
kreipinių į apsaugos raktą paieška vykdoma taip pat kaip C++ atveju. Kreipiniai randami per kelias minutes.

Pirmos dalies eksperimentai parodė, kad turėdami apsaugos raktą ir OS gamintojo pateiktas kreipinių dokumentacijas galime „nulaužti“ programą, kuri apsaugota vien apsaugos raktu, užtrukdami nuo kelių minučių iki 1-2val.

3.2.2. Komercinio apsaugos rakto ir eksperimentinės programos su maskavimo apsauga tyrimas

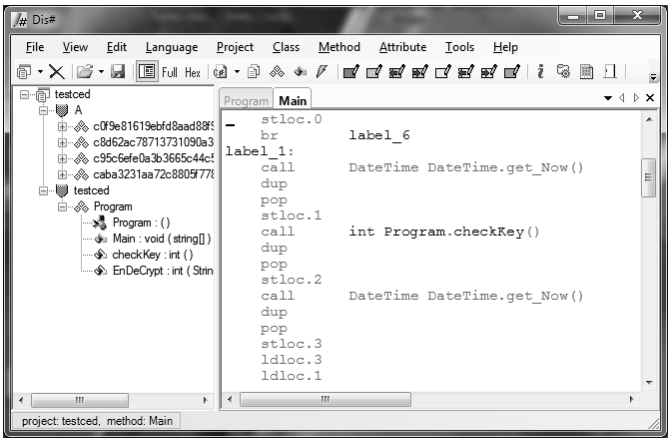
Pirmos dalies antram tyrimo etapui buvo naudojamas pirmame etape tirtas komercinis DLP-D apsaugos raktas ir eksperimentinė programa. Šiame etape panaudotas programos kodo maskavimo metodas. Maskavimo metodas paverčia programos kodą į sunkiau suprantamą sintaksę nei originalo kodo sintaksė. Todėl šio tyrimo etapo tikslas yra patikrinti, kiek kodo maskavimas pailgina kreipinių į apsaugos raktą suradimo laiką. Maskavimą panaudojome Java, C++ ir C# programų apsaugai.

Maskavimas naudingas programos kodo logikos apsaugojimui, tačiau įsilaužėliui, kuris nori apeiti apsaugos raktą, svarbu surasti kreipinius į apsaugos raktą ir juos apeiti. Maskavimas pakeičia vartotojo suprogramuotas kodo dalis, tačiau nepakeičia kreipinių per API bibliotekas pavadinimų. Todėl kreipinių pavadinimai randami taip pat kaip nemaskuotoje programoje. Pirmiausia C# programos kodą užmaskavome pasinaudoję Visual Studio 2008 priedu Dotfuscator. Po to dekompiliavome programą su .NET Reflector, po dekompiliavimo programos kreipiniai į apsaugos raktą liko nepakitę ir atrasti per kelias minutes. Kreipinių radimas pavaizduotas 30 pav. Vėliau buvo išmėgintas Crypto Obfuscator For .NET maskavimo programa, kuri be maskavimo gali taikyti prieš apgrąžos inžineriją nukreiptus veiksmus, tačiau buvo naudojamas tik maskavimas. Dekompiliavimui panaudoti .NET Reflector ir Dis# įrankiai. .NET Reflector negalėjo teisingai dekompiliuoti užmaskuotos eksperimentinės programos, buvo gauti klaidų pranešimai. Dis# dekompiliavo programą be klaidų, o po paieškos programoje buvo aiškiai rasti kreipiniai į apsaugos raktą. Rastas kreipinys pavaizduotas 31 pav.



30 pav. Kreipinio radimas po programos dekompiliavo su .NET Reflector

C++ programa buvo užmaskuota su *C++ Obfuscator* bandomaja versija. C++ programoje buvo nenaudojami dekompiliatoriai, o pirmiausia atlikta kreipinių į apsaugos raktą paieška, kuri buvo sėkminga.



31 pav. Kreipinio radimas po programos dekompiliavo su Dis#

Java programa buvo maskuota su ProGuard maskavimo įrankiu. Šis įrankis naudoja įvairius maskavimo metodus, tačiau kreipinių pavadinimų per API bibliotekas į apsaugos raktą pakeisti negali. Todėl po dekompiliavimo atlikus kreipinių paiešką, buvo rasti visi kreipiniai į apsaugos raktą. Kreipiniai buvo apeiti ir kodas sukompiliuota į pilnai veikiančią programą be apsaugos rakto.

Eksperimentas parodė, kad maskavimo metodas neužtikrina kelis kartus ilgesnės apsaugos nei neužmaskuota programa, nes kreipiniai į apsaugos raktą per API bibliotekas lieka nepakeisti.

3.2.3. Komercinio apsaugos rakto ir eksperimentinės programos su suspaudimo ir nederinimo apsaugomis tyrimas

Pirmos dalies trečiam tyrimo etapui buvo naudojamas pirmame ir antrame etapuose tirtas komercinis DLP-D apsaugos raktas ir eksperimentinė programa. Šiame etape panaudoti programos suspaudimo (angl. *packers*) ir nederinimo (angl. *antidebugging*) metodai. Šioms užduotims panaudota bandomoji Software Passport versija. Software Passport paremtas Armadillo suspaudimo branduoliu. Suspaudimas ir nederinimas panaudotas Java, .NET ir C++ programų apsaugai.

Pirmiausia atliktas nederinimo apsaugos apėjimas, nes kitaip su OllyDbg įrankiu derinimo atlikti nebuvo galima, buvo gaunami klaidos pranešimai arba programa nebeveikdavo. Tačiau panaudojus OllyDbg priedą (angl. *plug-in*) *Olly Advanced* ir pasirinkus kelis nustatymus buvo galima programą derinti. Suspaudimo programa paslepia originalios programos pradžią ir kreipinius į API bibliotekas. Taip pat originali programa išpakuojama vykdimo metu, todėl sekdami programos eigą žingsnis po žingsnio arba ieškodami konkrečių kreipinių buvo galima užtrukti kelias valandas aiškinantis programos veikimą ir ieškant originalios programos eigos arba buvo galima susidurti programos vykdimui skirto laiko apribojimais. Todėl buvo panaudotas kitas metodas. Internete rasta *Armadillo* suspaudėjui priešinga programa, kuri išskleidžia suspaustą programą. Tam buvo panaudota *ArmStripper*

v0.1 beta 6. Pastebėta, kad suspausta programa skiriasi nuo išskleistos programos, tačiau jų funkcionalumas yra vienodas. Taigi, išskleistoje programoje atlikta kreipinių į apsaugos raktą per API bibliotekas paieška, kaip ir ankstesniuose tyrimo etapuose. Aptikti kreipiniai buvo apeiti panaudojus JMP arba NOP komandas.

Taigi, suspaudimo ir nederinimo apsaugoms apeiti reikėjo daugiau laiko nei apeiti kodo maskavimą, tačiau paros ir daugiau laiko trunkančios apsaugos neužtikrino.

3.2.4. Komercinio apsaugos rakto ir eksperimentinės programos tyrimo rezultatai

Atlikus komercinio apsaugos rakto ir eksperimentinės programos be apsaugos ir su apsauga pirmos dalies tyrimą gauti rezultatai parodė, kad nė vienas apsaugos metodas neužtikrina ilgesnio kaip paros apsaugos laiko. Apsaugos vien su apsaugos raktu įveikimas užtruko nuo kelių minučių iki 2 val. Programos maskavimo apsauga įveikiama nuo keliasdešimties minučių iki 3 val. Programos suspaudimo ir nederinimo apsaugą palyginus su pirmais dviem apsaugos metodais truko ilgiausiai įveikti, nuo keliasdešimties minučių iki ~4 val. Gauti rezultatai pateikti 10 lentelėje.

Lentelė Nr. 10. Eksperimentinės programos su komercinio apsaugos rakto apsauga įveikimo laikai

Programavimo kalba parašyta programa	Programos apsaugos su apsaugos raktu įveikimo laikas	Programos apsaugos su apsaugos raktu ir programos maskavimu įveikimo laikas	Programos apsaugos su apsaugos raktu, programos suspaudimu ir nederinimo funkcija įveikimo laikas
C++	~2min. – ~2val.	~30min . ~3val.	~50min. ~4val.
Java	~2min. – ~2val.	~30min . ~3val.	~50min. ~4val.
.NET C#	~2min. – ~2val.	~30min . ~3val.	~50min. ~4val.

Gautų apsaugos įveikimo laiko rezultatų dispersija svyruoja tarp 1val. ir 1,5val., tai priklauso nuo žmogiškojo faktoriaus, tačiau dėsnis aiškus: kuo programos apsaugai naudojama daugiau apsaugos metodų tuo ilgiau užtrunki juos studijuodamas ir apeidamas.

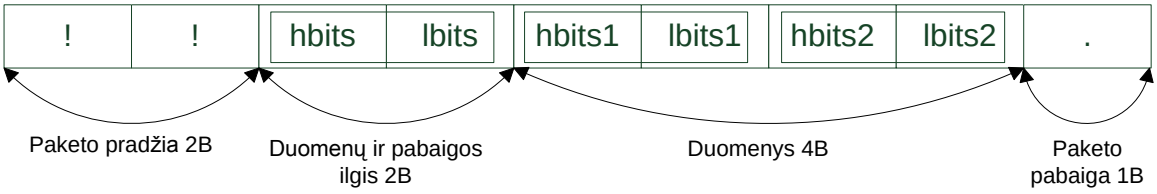
3.3. II tyrimo dalis. Eksperimentinės programos apsaugos su apsaugos raktu, realizuotu siūlomo modelio pagrindu, tyrimas

Antroje tyrimo dalyje pirmame etape panaudota eksperimentinė C++ kalba suprogramuota lango tipo programa, kuri susideda iš vieno mygtuko ir tekstinio lauko. Programos dydis 56 kB. Eksperimentinė programa sukompiliuota panaudojus *Microsoft Visual Studio 2010 Ultimate*. Programos projektas susideda iš 8 failų: 3 antraštiniai (angl.

header) failai, 2 išteklių (angl. *resource*) failai ir 3 šaltinio (angl. *source*) failai. Failų turinys pateiktas priede Nr. 3. Programoje paspaudus mygtuką į USB modulį siunčiamos vertės, kurios naudojamos kaip funkcijos argumentai. Apskaičiuotas rezultatas siunčiamas atgal eksperimentinei programai, kuri gautą rezultatą pavaizduoja tekstiniame lauke.

Antroje tyrimo dalyje antrame etape panaudota eksperimentinė C++ kalba suprogramuota komandinės eilutės tipo programa. Programos dydis 52 kB. eksperimentinė programa sukompiliuota panaudojus *Microsoft Visual Studio 2010 Ultimate*. Programos projektas susideda iš 4 failų: 2 antraštiniai (angl. *header*) ir 2 šaltinio (angl. *source*) failai. Failų turinys pateiktas priede Nr. 4. Programa, po paleidimo, tikrina ar prijungtas eksperimentinis *Tmote sky* apsaugos raktas, jei raktas prijungtas ir jame vykdoma reikiama funkcija, tai apskaičiavęs vertę siunčia programai, kuri gautą rezultatą pavaizduoja komandinėje eilutėje.

Su apsaugos raktu programa komunikuoja siųsdama paketą, kurio struktūra pavaizduota 32 pav. Paketo dydis 9B, iš kurių 4B skirti duomenims. Eksperimentui paketas paruoštas taip, kad būtų siunčiami du sveiki skaičiai, kurių kiekvieno dydis nedaugiau 2B.



32 pav. Siunčiamo paketo struktūra

Paketo pradžia yra du unikalūs simboliai, kurie apsaugos raktui parodo, kad siunčiami tikri duomenys, o ne „šlamštas“. Duomenų ir pabaigos ilgis nurodo kiek baitų sudaro duomenys ir paketo pabaigos simbolis.

Eksperimentinis apsaugos raktas paruoštas panaudojus *Tmote sky* programuojamą USB modulį, kuriame panaudota *Contiki* operacinė sistema. Į *Tmote sky* modulį įdiegta 326kB dydžio C kalba suprogramuotą programą, kuri vykdo funkciją. Apsaugos raktas suprogramuotas panaudojus *VMware* virtualios aplinkos *Linux* ruošinį, kuriame įdiegtas *Instant Contiki* paketas leidžia suprogramuoti *Tmote Sky* USB modulius. Atskirtai funkcijai, kuri įrašyta *Tmote Sky* modulyje, panaudota *Deffie Hellman* uždavinio funkciją:

$$\alpha^{a*b} \mathrm{mod}(p), \tag{1}$$

kuri yra vienkryptė funkcija. Vertės α ir p yra siunčiamos iš pagrindinės programos į apsaugos raktą, kuriame įrašytos a ir b vertės. Gautas rezultatas siunčiamas atgal pagrindinei programai, kuri rezultatą atvaizduoja tekstiniame lauke.

3.3.1. Eksperimentinio *Tmote sky* apsaugos rakto ir langų tipo programos be papildomos apsaugos tyrimas

Tyrimo eiga pradėta nuo eksperimentinės programos vertimo į asemblerio kodą su *OllyDbg* ir *IDA Pro* apgrąžos inžinerijos įrankiais. Abiem atvejais, kai įrankiai nenaudoja programinių papildinių (angl. *plugins*), kurie skirti prieš nederinimą, *OllyDbg* kodą išverčia, o *IDA Pro* meta klaidos pranešimą (angl. *can not set the target processor type "cli"*), kuris reiškia nesuderinamumą su *Microsoft.NET* platforma, ir išsijungia. Vadinasi sukompiliuota eksperimentinė programa turi nederinimo savybių, nors papildomų nederinimo įrankių nenaudota.

OllyDbg leidžia pažingsniui eiti per visą eksperimentinės programos kodą, tačiau langų tipo programa nukreipia per *Windows* operacinės sistemos sistemines bylas, kuriose pagal nutylėjimą nustatyti programos lango dydžio parametrai, sąveika su pele ir pan., tačiau iki pagrindinės programos neprieita. Taip apsunkinamas kodo sekimas, nes yra grėsmė pažeisti operacinės sistemos bylas ir apribojami derinimo įrankio veiksmai.

OllyDbg parodo kreipinių dėklą (angl. *Call Stack*) pagal kurį galima rasti pagrindinius programos kreipinius, tačiau eksperimentinėje programoje kreipinių masyvas neinformatyvus, randami tik operacinės sistemos bylų tarpusavio ryšiai. Eksperimentinėje programoje atlikus teksto ir sekų paiešką randami vien operacinės sistemos bylų sisteminiai pavadinimai, kurie įtakos eksperimentiniui apsaugos raktui neturi.

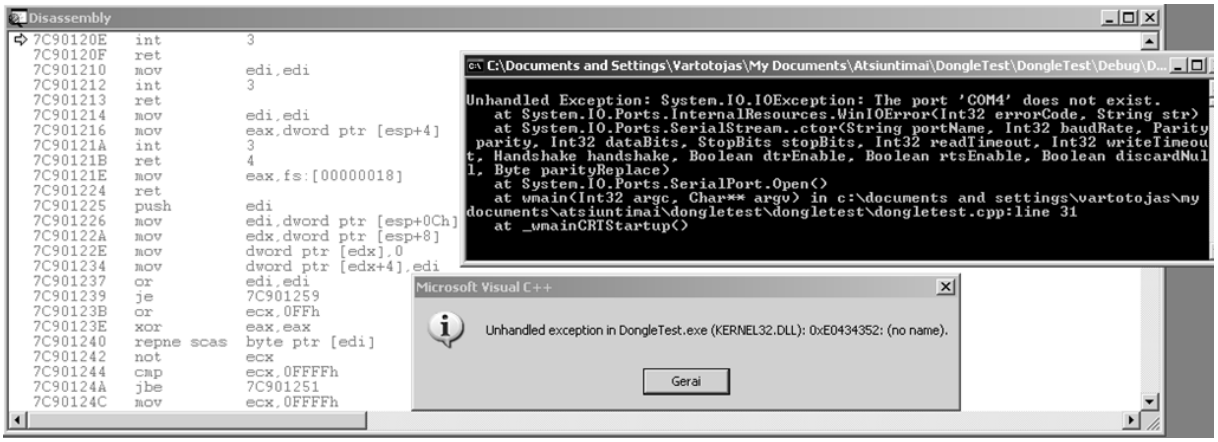
Praėjus apytiksliai septynioms valandoms ir panaudojus skirtingus įsilaužimų kombinacijų bandymus, kreipinys į eksperimentinį apsaugos raktą rastas kai *OllyDbg* įrankis nutraukdavo programos veikimą, tačiau kreipinio apėjimas programos neatrišdavo nuo apsaugos rakto. Paleidus modifikuotą programą gaunamas klaidos pranešimas. Palyginus su pirma tyrimo dalimi ilgą kreipinio radimo laiką lėmė *Microsoft Visual Studio* automatiškai sugeneruota programos struktūra, kuri nukreipdavo klaidinančia kodo seka.

Papildomi programos maskavimo, suspaudimo ir nederinimo įrankiai nebuvo naudojami, nes tai padidintų kreipinio į apsaugos raktą aptikimą daugiau nei 7 valandomis, tačiau iš esmės nepagerintų pačios apsaugos.

3.3.2. Eksperimentinio Tmote sky apsaugos rakto ir komandinės eilutės tipo programos be papildomos apsaugos tyrimas

Tyrimo eiga pradėta nuo eksperimentinės programos vertimo į asemblerio kodą su *OllyDbg* apgrąžos inžinerijos įrankiu. Atlikti tie patys kodo analizės bandymai ir kreipinių paieška kaip langų tipo programos atveju. Pastebėta, kad komandinės eilutės programos kodo

fragmentai ir kodo vykdymas sutampa su langų tipo programos dalimis, todėl kreipinys į *Tmote sky* apsaugos raktą surasti nedaugiau kaip per 20 min. Tačiau bandymai apeiti arba pakeisti kreipinius ir vykdyti programą be apsaugos rakto buvo nesėkmingi, nes gaunama klaida susijusi su nežinomais (angl. *unknown*) programos adresais. Jei apsaugos raktas neprijungtas, tai paleidus programą gaunamas pranešimas apie nenumatytą klaidą ir leidžiama pasirinkti derinti programą ar ne. Pasirinkus derinimą įsijungia *Microsoft Visual Studio* asemblerio langą, kuris pavaizduotas 33 pav.



33 pav. Kreipinys į apsaugos raktą

Derinimo įrankis parodo kokiu adresu vykdomas kreipinys į apsaugos raktą.

3.4. I ir II tyrimo dalių rezultatų palyginimas

I ir II tyrimo dalių rezultatai pateikti 11 lentelėje.

Lentelė Nr. 11. Eksprimentinių programų apsaugos įveikimo rezultatų palyginimas

		Programavimo kalba parašyta programa	Programos apsaugos su *dongle* moduliu įveikimo laikas	Programos apsaugos su *dongle* moduliu ir programos maskavimu įveikimo laikas	Programos apsaugos su *dongle* moduliu, programos suspaudimu ir nederinimo funkcija įveikimo laikas
I tyrimo dalis		C++	~2min. – ~2val.	~30min . ~3val.	~50min. ~4val.
		Java	~2min. – ~2val.	~30min . ~3val.	~50min. ~4val.
		.NET C#	~2min. – ~2val.	~30min . ~3val.	~50min. ~4val.
II tyrimo dalis	I etapas	C++	Apsaugos įveikti nepavyko (kreipinys į apsaugos raktą rastas per ~7 val.)		
	II etapas	C++	Apsaugos įveikti nepavyko (kreipinys į apsaugos raktą rastas per ~20 min.)		

Rezultatai parodo, kad eksperimentinės langų ir komandinės eilutės tipo programos su eksperimentiniu *Tmote sky* apsaugos raktu yra žymiai geriau apsaugotos nei eksperimentinės programos su komerciniu DLP-D apsaugos raktu. Tai priklauso nuo to, kad langų ir komandinės eilutės tipo programos dalis yra vykdoma apsaugos rakte, o prie apsaugos rakto prieiga nebuvo gauta. Paskirstytų skaičiavimų funkcijos nepavyko „nulaužti“, nes buvo rastas kreipinys į apsaugos raktą, bet ne į funkciją. Be to, vienkryptės Diffie Hellman funkcijos rezultatų radimas matematiškai yra sudėtingas uždavinys.

3.5. Išvados

1. Komercinio apsaugos rakto, be programos kodo dalies vykdymo, eksperimentinis įvertinimas parodė, kad tokio tipo apsaugą galima įveikti per 50 min. – 4 val.
2. Darbe sukurto apsaugos rakto, su programos kodo dalies vykdymu, eksperimentinis įvertinimas parodė, kad tokio tipo apsaugos įveikti paprastais apgrąžos inžinerijos metodais, apeinant kreipinį į apsaugos raktą, neįmanoma. Nors po 10-20 min. randamas kreipinys į apsaugos raktą, tačiau bandymas apeiti ar pakeisti kreipinį baigiasi klaidos pranešimu apie nežinomus adresus, todėl paskirstytais skaičiavimais pagrįstas aparatinis apsaugos metodas yra atsparus apgrąžos inžinerijos atakoms, lyginant su tradiciniais apsaugos metodais.
3. Paskirstytam skaičiavimui buvo pasirinkta *Diffie Hellman* vienkryptė funkcija, kurios du argumentai buvo įrašyti į pagrindinę programą, o funkcijos du laipsnio rodikliai įrašyti apsaugos rakte, todėl aptikus du argumentus pagrindinėje programoje, funkcijos apėjimas taptų sudėtingu uždaviniu.

IŠVADOS

1. Atlikus taikomųjų programų apsaugos nuo nelegalaus naudojimo aparatiniu metodu analizę išsiaiškinta, kad pažeidžiamiausia apsaugos vieta yra programos ir apsaugos rakto komunikavimas.
2. Atlikus programų apsaugos aparatiniais ir programiniais metodais analizę išsiaiškinta, kad dalies programos slėpimas ir vykdymas apsaugos rakte geriau apsaugo nuo apgrąžos inžinerijos, nuotolinės kodo įterpimo ir „klonavimo“ atakų, nei maskavimas, filigranavimas, kopijas aptinkančios sistemos ir kodo suspaudimas.
3. Sukurtas išmaniojo aparatinio apsaugos rakto modelis, kuris pagrįstas paskirstytų skaičiavimų principais, atliktas modelio imitacinis modeliavimas ir nustatyta, kad pritaikius paskirstytų skaičiavimų principą modelis užtikrina apsaugą nuo apgrąžos inžinerijos atakos.
4. Komercinio apsaugos rakto, be programos kodo dalies vykdymo, eksperimentinis įvertinimas parodė, kad tokio tipo apsaugą galima įveikti per 50 min. – 4 val.
5. Darbe sukurto apsaugos rakto, su programos kodo dalies vykdymu, eksperimentinis įvertinimas parodė, kad tokio tipo apsaugos įveikti paprastais apgrąžos inžinerijos metodais, apeinant kreipinį į apsaugos raktą, neįmanoma. Nors po 10-20 min. randamas kreipinys į apsaugos raktą, tačiau bandymas apeiti ar pakeisti kreipinį baigiasi klaidos pranešimu apie nežinomus adresus, todėl paskirstytais skaičiavimais pagrįstas aparatinis apsaugos metodas yra atsparus apgrąžos inžinerijos atakoms, lyginant su tradiciniais apsaugos metodais.

LITERATŪRA

1. *SEVENTH annual BSA-IDC GloBal SofTware 09 PIRACY STUDY*, Business Software Alliance, 2010 May.
2. Software piracy rate (most recent) by country. NationMaster.com. 2007. [žiūrėta 2010-12-10]. Prieiga per internetą <http://www.nationmaster.com/graph/cri_sof_pir_rat-crime-software-piracy-rate>.
3. Kazanvičius E., Venčkauskas A., Liutkevičius A., Vrubliauskas A. Informacijos saugos vadyba. Kaunas. Kauno technologijos universitetas. 2008. p. 8, 26.
4. LANVA [interaktyvus]. [žiūrėta 2010-12-10]. Prieiga per internetą <http://www.lanva.org>.
5. Piazzalunga U., Salvaneschi P., Balducci F., Jacomuzzi P., Moroncelli C. Security Strenght Measurement for Dongle-Protected Software. EEE Security & Privacy. 2007 November/December.
6. Atallah M. J., Bryant E. D., Stytz M. R. A Survey of Anti-Tamper Technologies. The Journal of Defense Software Engineering. 2004 November.
7. Djekic P., Loebbecke C. Preventing application software piracy: An Empirical investigation of technical copy protections. Journal of Strategic Information Systems. 2007.
8. Anderson R. J. Security Engineering: A Guide to Building Dependable Distributed Systems. Wiley. 2008. p 285.
9. Copy Protection Software Products for Microsoft Windows 95, 98, Me, XP, NT, 2000, 2003, Vista [interaktyvus]. Zapper Software. [žiūrėta 2010-10-29]. Prieiga per internetą <http://www.zappersoftware.com/home.html>.
10. Matrix Software Protection System [interaktyvus]. TechnoData Interware. [žiūrėta 2010-10-28]. Prieiga per internetą <http://www.matrixlock.de/english/index.htm>.
11. MICROCOSM [interaktyvus]. Software based and hardware based protection. [žiūrėta 2010-10-28]. Prieiga per internetą <http://www.microcosm.co.uk>.
12. SG-Lock [interaktyvus]. SG-Lock Copy Protection & Crypto System. [žiūrėta 2010-10-28]. Prieiga per internetą <http://www.sg-lock.com/us/index.php>.
13. Computing dictionary [interaktyvus]. Dictionary.com. [žiūrėta 2010-10-29]. Prieiga per internetą <http://dictionary.reference.com/browse/dongle>.
14. Memon J. M., Khan A., Baig A., Shah A. A Study of Software Protection Techniques. Innovations and Advanced Techniques in Computer and Information Sciences and Engineering. 2007, 249-253, DOI: 10.1007/978-1-4020-6268-1_45.
15. Codework [interaktyvus]. [žiūrėta 2010-10-28] <http://codework.com/wibu>.

16. USB dongle paveikslas. Rechargeguru.com. [žiūrėta 2010-10-29]. Prieiga per internetą <http://rechargeguru.com/do/images/USB-Dongle.jpg>.
17. Codework [interaktyvus]. [žiūrėta 2010-10-29]. Prieiga per internetą <http://www.codework.com/wibu/hardware.html>.
18. Alkonost Software [interaktyvu]. Alconost ContraCopy. [žiūrėta 2010-10-28]. Prieiga per internetą <http://www.contracopy.com>.
19. Jun W. J. Smart Card Technology Capabilities. 2003, July 8. Giesecke & Devrient. Prieiga per internetą < http://csrc.nist.gov/publications/ >.
20. TechnoData Interware [interaktyvu]. Matrix High quality software protection user manual. p. 28, 37, 64. Prieiga per internetą < http://www.matrixlock.de/english/index.htm >.
21. SG-Lock [interaktyvus]. Copy Protection System Developer Manual. 2007 October. Prieiga per internetą <http://www.sg-lock.com>.
22. TopBits [interaktyvus]. Two factor authentication. [žiūrėta 2010-10-29]. Prieiga per internetą <http://www.tech-faq.com/two-factor-authentication.shtml>.
23. Jansh [interaktyvus]. [žiūrėta 2010-12-09]. Prieiga per internetą <http://jansh.com/web/index.php?option=com_content&view=article&id=10&Itemid=9> .
24. SafeNet, Alladin. HASP SRM HL hardware-Based Protection Keys. 2010. Prieiga per internetą <ftp://ftp.ealaddin.com/pub/marketing/HASP/HASP_SRM/Datasheets/DS_HASP_HL.pdf>.
25. Rockey.com.my, ROCKEY6 Smart, [žiūrėta 2011-01-16]. Prieiga per internetą <http://www.rockey.com.my/prod-dongle-rockey6.php>.
26. KEYLOK, A Code Porting Capability Within Smart Card Driverless Dongle, [žiūrėta 2011-01-16]. Prieiga per internetą <http://www.keylok.com/product/code_vault.aspx>.
27. Jozwiak I. J., Libert A., Marczak K. A Hardware-Based Software Protection Systems Analysis of Security Dongles with Memory. 2007. International Multi-Conference on Computing in the Global Information Technology. IEEE Explore.
28. Barak B. On The (Im)possibility of Software Obfuscation. 2001. Department of Computer Science. Princeton University. New Jersey.
29. Chen H.Y., Hou T.W. Changing data type method of data obfuscation on java software. Int. Computer Symphosium. 2004 Dec. 15-17. Taipei, Taiwan.

30. Myles G., Collberg C. Software Watermarking via Opaque Predicates: Implementation, Analysis and Attacks. Electronic Commerce Research Volume 6, Number 2, 155-171, DOI: 10.1007/s10660-006-6955-z.
31. MeiHong L., JiQiang L. USB Key-Based Approach for Software Protection. International Conference on Industrial Mechatronics and Automation. 2009.
32. SecuTech Solution Inc [interaktyvus]. How to protect your software Profesional Advise from SecuTech. 2006 February. Prieiga per internetą <http://www.esecutech.com>.
33. DESkey [interaktyvus]. DESkey hardware. [žiūrėta 2010-10-28]. Prieiga per internetą <http://www.deskey.co.uk>.
34. Borges M., Ribeiro A. N., Campos J. C. A Push Infrastructure for Mobile Application Deployment in Mobile Environments.. Escola de Engenharia. Universidade do Minho, Portugal. 2006 June.
35. Eliasson M. A Study of Terminal Server Technology. 2009:170-ISSN:1402-1773-ISRN:LTU-CUPP—09/170—SE.
36. Rizzoli, A.E. et al. ,2009. Updated version of final design and of the architecture of SEAMLESS-IF Report No.47, SEAMLESS integrated project, EU 6th Framework Programme, contract no. 010036-2, www.SEAMLESS-IP.org, 31 pp, ISBN no. 978-90-8585-590-3.
37. Hutchinson C., Ward J., Castilon K. Navigating the Next-Generation Application Architecture. IEEE Computer Society. 2009 March/April.
38. Blom S., Book Matthias, Gruhn V., Hrushchak R., Köhler A. Write Once, Run Anywhere – A Survey of Mobile Runtime Environments. Applied Telematics/e-Business Group, Dept. Of Computer Science, University of Leipzig Klostergasse 3,04109 Leipzig, Germany.
39. White S. R., Comerford L. ABYSS: An Architecture for Software Protection. IEEE TRANSACTION ON SOFTWARE ENGINEERING , VOL 16. 1990 6 June.
40. Mana A., Lopez J., Ortega J.J., Pimentel E., Troya J.M. A framework for secure execution of software. Springer-Verlag. 2004.
41. Venčkauskas A., Toldinas J. Kompiuterių ir Operacinių sistemų sauga. Mokomoji knyga. Kauno Technologijos universitetas. Kaunas. 2008. psl. 158, 160-161.
42. Perry M., Oskov N. Introduction to Reverse Engineering Software. [žiūrėta 2010-10-29]. Prieiga per internetą <http://www.acm.uiuc.edu/sigmil/RevEng>.
43. SafeNet [interaktyvus]. MD 21017. [žiūrėta 2010-10-28]. Prieiga per internetą <http://www.safenet-inc.com>.

44. Jozwiak I. J., Marczak K. A Hardware-Based Software Protection Systems – Analysis of Security Dongles with Time Meters. 2nd International Conference on Dependability of Computer Systems. IEEE Explore. 2007.
45. Bedrune J.B., Filiol É., Raynal F. Cryptography:all-out attacks or how to attack cryptography without intensive cryptanalysis. Journal in Computer Virology Volume 6, Number 3, 207-237, DOI: 10.1007/s11416-008-0117-x.
46. Hoglung G., McGraw G. Exploiting Software: How to Break Code. Addison-Wesley Professional, 2004 Feb 17.
47. Software Informer [interaktyvus]. PIC Simulator IDE. [žiūrėta 2011-01-06]. Prieiga per internetą <http://pic-simulator-ide.software.informer.com>.
48. Giffin J.T., Christodorescu M., Kruger L. Strengthening Software Self-Checksumming via Self-Modifying Code. 21st Annual Computer Security Applications Conference, IEEE. 2005.
49. SecurityFocus [interaktyvus]. [žiūrėta 2010-10-29]. Prieiga per internetą <http://www.securityfocus.com>.
50. Mana A., Pimentel E. An Efficient Software Protection Scheme. Advances in Cryptology — CRYPTO' 89 Proceedings Lecture Notes in Computer Science, 1990, Volume 435/1990, 610-611, DOI: 10.1007/0-387-34805-0_55
51. Microsoft [interaktyvus]. Debugging Tools for Windows 32-bit Version. [žiūrėta 2011-01-06]. Prieiga per internetą <http://www.microsoft.com/whdc/devtools/debugging/installx86.mspx>.
52. OllyDbg [interaktyvus]. 32-bit assembler level analysing debugger for Microsoft® Windows. [žiūrėta 2011-01-06]. Prieiga per internetą <http://www.ollydbg.de/>.
53. Cifuentes C., Fitzgerald A. The legal status of reverse engineering of computer software. Annals of Software Engineering 9. 2000. p. 337-351.
54. Free Download Manager [interaktyvus]. Antidebug software. [žiūrėta 2011-01-06]. Prieiga per internetą <http://www.freedownloadmanager.org/downloads/antidebug_software/>.
55. Klein T. Process Dump Analyses Forensical acquisition and analyses of volatile data. University of Applied Science, Augsburg, 2006-07-22.
56. Microsoft [interaktyvus]. User Mode Process Dumper Version 8.1. [žiūrėta 2011-01-05]. Prieiga per internetą <http://www.microsoft.com/downloads/details.aspx?FamilyID=E089CA41-6A87-40C8-BF69-28AC08570B7E&displaylang=en>.

57. Giacobbi G. What is Netcat? [žiūrėta 2011-01-05]. 2006. Prieiga per internetą <http://netcat.sourceforge.net/>.
58. Klein T. Memory Parser. [žiūrėta 2011-01-05]. Prieiga per internetą <http://www.trapkit.de/research/forensic/mmp/index.html>.
59. Roy C. K., Cordy J. R. A Survey on Software Clone Detection Research. School of Computing. Queen's University at Kingston. Ontario. Canada. 2007 September 26. p. 8, 15-17, 62, 64-65.
60. Maartmann C. M., Thorkildsen S. E., Årnes A. The persistence of memory: Forensic identification and extraction of cryptographic keys. Elsevier. 2009.
61. Verizon Business RISK Team. 2009 Data Breach Investigations Supplemental Report Anatomy of a Data Breach. 2009. Prieiga per internetą <www.verizonbusiness.com>.
62. DongleLabs. Emulator for any Protection type. [žiūrėta 2011-01-06]. Prieiga per internetą. <http://www.unpacking.net/>.
63. asciitable.com [interaktyvus]. ASCII Table and Description. [žiūrėta 2011-01-01]. Prieiga per internetą <http://www.asciitable.com/>.
64. Sakalauskas E. Kriptografinės sistemos. Mokomoji knyga. Kauno technologijos universitetas. Kaunas. 2008.
65. Dunkels A. Tutorial: Running Contiki 2.1 on the Tmote Sky. [žiūrėta 2011-05-22]. Prieiga per internetą <http://www.sics.se/contiki/tutorials/running-contiki-on-the-tmote-sky.html>.
66. ContikiWiki [interaktyvus]. Develop your first application. [žiūrėta 2011-05-22]. Prieiga per internetą <http://www.sics.se/contiki/wiki/index.php/Develop_your_first_applicatio >.

Research on hardware-based software protection methods and distributed computing model for security dongle implementation

SUMMARY

Software protection is important problem of nowaday. In 2009 the piracy rate reaches 43% of all globe softwares usage, especialy high piracy rate is in Eastern Europe, where 64% of softwares are illegal. This brings hardware-based protection to be one of the prime defense against illegal software usage. The analysis of hardware-based software protection showed that the weakest part of hardware-based protection is communication with software, so dongle method, which computes part of the software inside dongle, could withstand most of reverse engineer attack methods. To prove this Matlab model of distributed dongle-based protection scheme was created and its experimental evaluation showed, that suggested software protection model is resistant against deassembling, debbuging and software cloning attacks. Equally, experimental distributed computing protection dongle prototype was created and tested with two experimental programs against deassembling, debbuging, decompilation and software cloning attacks. This shows that attackers can find calls to the dongle, but can not jump or nop it, such it was done in experimental software protected with traditional commercial dongle.

PRIEDAI

1. Priedas Nr. 1. Programos dalis, vykdoma eksperimentiniame *Tmote sky* apsaugos rakte

```
2.  #include "contiki.h"
3.  #include "uart1.h"
4.  #include <stdio.h>
5.
6.  #define READY 0
7.  #define WAITING_SECOND_START_SYMBOL 1
8.  #define WAITING_FIRST_LENGTH_SYMBOL 2
9.  #define WAITING_SECOND_LENGTH_SYMBOL 3
10. #define RECEIVING 4
11.
12. void processByte(unsigned char c);
13. void parsePacket();
14. int uart_rx_callback(unsigned char c);
15.
16. typedef unsigned short UINT16;
17. typedef unsigned char UINT8;
18.
19. unsigned char buff[20];
20.
21. int state = READY;
22. int buff_cnt = 0;
23. UINT16 frame_size;
24. UINT8 frame_size_hbits;
25. UINT8 frame_size_lbits;
26.
27. PROCESS(com_listener_process, "COM Listener process");
28. AUTOSTART_PROCESSES(&com_listener_process);
29.
30. int uart_rx_callback(unsigned char c) {
31.         //printf("Got symbol:%c\n", c);
32.         //uart1_writeb(c);
33.         processByte(c);
34.         return 0;
35. }
36.
37. void parsePacket() {
38.         int numberOfDataFields = (frame_size - 1) / 2;
39.         UINT16 number[numberOfDataFields];
40.         int index = 0;
41.         UINT8 hbits;
42.         UINT8 lbits;
43.         UINT16 res;
44.         UINT16 alpha = 1;
45.         UINT16 p = 1;
46.         //printf("Number of received data fields: %d \n", numberOfDataFields);
47.         //printf("All packet %s, restart receiving\n", buff);
48.         int i = 0;
49.         for (i = 0; i < frame_size - 1; i = i + 2) {
50.                 hbits = (UINT8) buff[i];
51.                 //printf("First symbol: %d \n", hbits);
52.                 lbits = (UINT8) buff[i + 1];
53.                 //printf("Second symbol: %d \n", lbits);
54.                 number[index++] = hbits << 8 | lbits; // convert two symbols into one short number
55.                 //printf("Field: %d \n", number[index - 1]);
56.         }
57.         //!!! SVARBU !!! ( PAVYZDYS: res=Param_1 * Param_2 ..... - Param_N)
58.         for (i = 0; i < index - 1; i++) {
59.                 // paimam po viena gauta parametra ir dauginam
60.                 alpha = alpha * number[i];
61.         }
62.         // pridedam paskutini parametra
63.         p = p * number[i];
64.         //Diffie Hellman
65.         res = (alpha ^ (7 * 13)) % p;
66.         // pakonvertuojam gauta 16bit skaiciu i 2 byte (8bit) tipo skaicius
67.         lbits = res & 0x00FF;
68.         hbits = (res >> 8) & 0x00FF;
69.
70.         // grazinam rezultatus (pradzioje vyresnieji bitai, paskui zemesnieji)
```

```
71.             printf("%c%c\n", hbits, lbits);
72.
73.     }
74.
75.     PROCESS_THREAD(com_listener_process, ev, data)
76.     {
77.             PROCESS_BEGIN();
78.             uart1_set_input(uart_rx_callback);
79.
80.             for(;;) {
81.                     PROCESS_YIELD();
82.             }
83.
84.             PROCESS_END();
85.     }
86.
87.     void processByte(unsigned char c) {
88.
89.             switch (state) {
90.
91.             case READY:
92.                     if (c == '!') {
93.                             state = WAITING_SECOND_START_SYMBOL;
94.                             //printf("Got symbol %c and waiting for second\n", c);
95.                     }
96.                     break;
97.
98.             case WAITING_SECOND_START_SYMBOL:
99.                     if (c == '!') {
100.                            state = WAITING_FIRST_LENGTH_SYMBOL;
101.                            //printf("Got symbol %c and waiting for first length symbol\n", c);
102.                    } else {
103.                            state = READY;
104.                            //printf("Wrong second beginning symbol %c, restart receiving\n", c);
105.                    }
106.                    break;
107.
108.            case WAITING_FIRST_LENGTH_SYMBOL:
109.                    frame_size_hbits = (UINT8) c; // save high bits of frame length
110.                    state = WAITING_SECOND_LENGTH_SYMBOL;
111.                    //printf("Got symbol %d and waiting for second length symbol\n",
112.                                    //frame_size_hbits);
113.                    break;
114.
115.            case WAITING_SECOND_LENGTH_SYMBOL:
116.                    frame_size_lbits = (UINT8) c;
117.                    //printf("Got symbol %d\n", frame_size_lbits);
118.                    frame_size = frame_size_hbits << 8 | frame_size_lbits; // save packet frame length
119.                    if (frame_size > 800) // wrong frame length
120.                    {
121.                            //                                      printf(
122.                            //                                                      "Error,  wrong frame size (%d)
    received, droping frame...\n",
123.                            //                                                      frame_size);
124.                            frame_size = 0;
125.                            state = READY;
126.                            buff_cnt = 0;
127.                    } else {
128.                            state = RECEIVING;
129.                            memset(buff, '\0', 20);
130.                            buff_cnt = 0;
131.                            //printf("Got packet length %d\n", frame_size);
132.                    }
133.                    break;
134.
135.            case RECEIVING:
136.                    buff[buff_cnt++] = c;
137.                    //printf("Got symbol %c\n", c);
138.                    if (buff_cnt == frame_size) {
139.                            if (buff[buff_cnt - 1] == '.') {
140.                                    //printf("Got all packet %s, restart receiving\n", buff);
141.                                    parsePacket(); // parse packet and take function arguments
142.                                    frame_size = 0;
143.                                    state = READY;
144.                                    buff_cnt = 0;
145.                            } else {
146.                                    //printf("Error,  wrong frame end symbol, droping frame...\n");
147.                                    frame_size = 0;
148.                                    state = READY;
```

```
149.                                                buff_cnt = 0;
150.                                        }
151.                                }
152.                                break;
153.
154.                        default:
155.                                frame_size = 0;
156.                                state = READY;
157.                                buff_cnt = 0;
158.                        }
159.
160. }
```

2. Priedas Nr. 2. Eksperimentinė programa, bandyta su komerciniu apsaugos raktu

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Runtime.InteropServices;

namespace testced
{
   class Program
   {
      [DllImport(@"ced.dll")]
      public static extern int checkKey();


      static void Main(string[] args)
      {
         for (int i = 0; i < 10; i++)
         {
            DateTime startTime = DateTime.Now;

            int status = checkKey();

            DateTime stopTime = DateTime.Now;

            TimeSpan duration = stopTime - startTime;

            if (status == 0) // 0 : passed, -1 : failed
               Console.WriteLine("checkKey()=PASSED, execution time = " + duration);
            else
               Console.WriteLine("checkKey()=FAILED, execution time = " + duration);
         }

         Console.WriteLine("END.");
      }
   }
}
```

3. Priedas Nr. 3. Eksperimentinė langų tipo programa. Form1.h byla

```
4.    #pragma once
5.
6.    #using <System.dll>
7.
8.    using namespace System;
9.    using namespace System::IO::Ports;
10.   using namespace System::Threading;
11.
12.   namespace TestForms {
13.
14.           using namespace System;
15.           using namespace System::ComponentModel;
16.           using namespace System::Collections;
17.           using namespace System::Windows::Forms;
18.           using namespace System::Data;
19.           using namespace System::Drawing;
20.
21.           /// <summary>
22.           /// Summary for Form1
23.           ///
```

```
24.         /// WARNING: If you change the name of this class, you will need to change the
25.         ///          'Resource File Name' property for the managed resource compiler tool
26.         ///          associated with all .resx files this class depends on.  Otherwise,
27.         ///          the designers will not be able to interact properly with localized
28.         ///          resources associated with this form.
29.         /// </summary>
30.         public ref class Form1 : public System::Windows::Forms::Form
31.         {
32.         public:
33.                     Form1(void)
34.                     {
35.                                 InitializeComponent();
36.                                 //
37.                                 //TODO: Add the constructor code here
38.                                 //
39.                     }
40.
41.         protected:
42.                     /// <summary>
43.                     /// Clean up any resources being used.
44.                     /// </summary>
45.                     ~Form1()
46.                     {
47.                                 if (components)
48.                                 {
49.                                             delete components;
50.                                 }
51.                     }
52.         private: System::Windows::Forms::Button^  button1;
53.                                  static bool _continue;
54.                                  static SerialPort^ _serialPort;
55.         private: System::Windows::Forms::TextBox^  textBox1;
56.         protected:
57.
58.         private:
59.                     /// <summary>
60.                     /// Required designer variable.
61.                     /// </summary>
62.                     System::ComponentModel::Container ^components;
63.
64.  #pragma region Windows Form Designer generated code
65.                     /// <summary>
66.                     /// Required method for Designer support - do not modify
67.                     /// the contents of this method with the code editor.
68.                     /// </summary>
69.                     void InitializeComponent(void)
70.                     {
71.                                 this->button1 = (gcnew System::Windows::Forms::Button());
72.                                 this->textBox1 = (gcnew System::Windows::Forms::TextBox());
73.                                 this->SuspendLayout();
74.                                 //
75.                                 // button1
76.                                 //
77.                                 this->button1->Location = System::Drawing::Point(20, 25);
78.                                 this->button1->Name = L"button1";
79.                                 this->button1->Size = System::Drawing::Size(132, 36);
80.                                 this->button1->TabIndex = 0;
81.                                 this->button1->Text = L"button1";
82.                                 this->button1->UseVisualStyleBackColor = true;
83.                                 this->button1->Click += gcnew System::EventHandler(this, &Form1::button1_Click);
84.                                 //
85.                                 // textBox1
86.                                 //
87.                                 this->textBox1->Location = System::Drawing::Point(12, 91);
88.                                 this->textBox1->Multiline = true;
89.                                 this->textBox1->Name = L"textBox1";
90.                                 this->textBox1->Size = System::Drawing::Size(260, 159);
91.                                 this->textBox1->TabIndex = 1;
92.                                 //
93.                                 // Form1
94.                                 //
95.                                 this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
96.                                 this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
97.                                 this->ClientSize = System::Drawing::Size(284, 262);
98.                                 this->Controls->Add(this->textBox1);
99.                                 this->Controls->Add(this->button1);
100.                                this->Name = L"Form1";
101.                                this->Text = L"Form1";
102.                                this->ResumeLayout(false);
```

```
103.                                 this->PerformLayout();
104.
105.                         }
106. #pragma endregion
107.             private: System::Void button1_Click(System::Object^  sender, System::EventArgs^  e) {
108.
109.                                                 this->textBox1->Text = "";
110.
111.                                                 // Create a new SerialPort object with default settings.
112.                                                 _serialPort = gcnew SerialPort();
113.
114.                                                 // Sset the appropriate properties.
115.                                                 _serialPort->PortName = "COM4";
116.                                                 _serialPort->BaudRate = 115200;
117.                                                 _serialPort->Parity =  Parity::None;
118.                                                 _serialPort->DataBits = 8;
119.                                                 _serialPort->StopBits = StopBits::One;
120.                                                 _serialPort->Handshake = Handshake::None;
121.
122.                                                 // Set the read/write timeouts
123.                                                 _serialPort->ReadTimeout = 500;
124.                                                 _serialPort->WriteTimeout = 500;
125. //-------------------------SVARBU!-------------------------------------------------------------------------------------------
126.                                                 _serialPort->Open();
127.                                                 _continue = true;
128.
129.                                                 String^ message = "";
130.                                                 array<unsigned char,1>^ data = gcnew array<unsigned char,1>(11);
131.                                                 data[0] = '!';
132.                                                 data[1] = '!';
133.                                                 unsigned short number = 5;
134.                                                 unsigned short number1 = 11;
135.                                                 unsigned short number2 = 13;
136.                                                 unsigned char lbits = number & 0x00FF;
137.                                                 unsigned char hbits = (number >> 8) & 0x00FF;
138.                                                 unsigned char lbits1 = number1 & 0x00FF;
139.                                                 unsigned char hbits1 = (number1 >> 8) & 0x00FF;
140.                                                 unsigned char lbits2 = number2 & 0x00FF;
141.                                                 unsigned char hbits2 = (number2 >> 8) & 0x00FF;
142.                                                 data[2] = hbits; // size
143.                                                 data[3] = lbits; // size
144.                                                 data[4] = hbits1;
145.                                                 data[5] = lbits1;
146.                                                 data[6] = hbits2;
147.                                                 data[7] = lbits2;
148.                                                 data[8] = '.';
149.                                                 _serialPort->Write(data, 0, 1);
150.                                                 _serialPort->Write(data, 1, 1);
151.                                                 _serialPort->Write(data, 2, 1);
152.                                                 _serialPort->Write(data, 3, 1);
153.                                                 _serialPort->Write(data, 4, 1);
154.                                                 _serialPort->Write(data, 5, 1);
155.                                                 _serialPort->Write(data, 6, 1);
156.                                                 _serialPort->Write(data, 7, 1);
157.                                                 _serialPort->Write(data, 8, 1);
158.                                                 message = _serialPort->ReadLine();
159.
160.                                                // padarom is dvieju gautu baitu viena skaiciu
161.                                                 hbits = message[0];
162.                                                 lbits = message[1];
163.                                                 number = hbits << 8 | lbits;
164.
165.                                                 this->textBox1->Text += number + "\r\n";
166.
167.                                                 _serialPort->Close();
168.                                 }
169.             };
170. }
```

4. Priedas Nr. 4. Eksperimentinė komandinės eilutės tipo programa. DongleTest.cpp byla

```
5.    // DongleTest.cpp : Defines the entry point for the console application.
6.    //
7.
8.    #include "stdafx.h"
9.
```

```
#using <System.dll>

using namespace System;
using namespace System::IO::Ports;
using namespace System::Threading;


int _tmain(int argc, _TCHAR* argv[])
{

	// Create a new SerialPort object with default settings.
	SerialPort^ _serialPort = gcnew SerialPort();

	// Sset the appropriate properties.
	_serialPort->PortName = "COM4";
	_serialPort->BaudRate = 115200;
	_serialPort->Parity =  Parity::None;
	_serialPort->DataBits = 8;
	_serialPort->StopBits = StopBits::One;
	_serialPort->Handshake = Handshake::None;

	// Set the read/write timeouts
	_serialPort->ReadTimeout = 500;
	_serialPort->WriteTimeout = 500;
//------------------------------------------------------------SVARBU!-------------------------------------
	_serialPort->Open();

	String^ message = "";
	array<unsigned char,1>^ data = gcnew array<unsigned char,1>(11);
	data[0] = '!';
	data[1] = '!';
	//unsigned short number = 7;
	unsigned short number = 5;
	unsigned short number1 = 11;
	unsigned short number2 = 13;
	unsigned char lbits = number & 0x00FF;
	unsigned char hbits = (number >> 8) & 0x00FF;
	unsigned char lbits1 = number1 & 0x00FF;
	unsigned char hbits1 = (number1 >> 8) & 0x00FF;
	unsigned char lbits2 = number2 & 0x00FF;
	unsigned char hbits2 = (number2 >> 8) & 0x00FF;
	data[2] = hbits; // size
	data[3] = lbits; // size
	data[4] = hbits1;
	data[5] = lbits1;
	data[6] = hbits2;
	data[7] = lbits2;
	//data[8] = hbits;
	//data[9] = lbits;
	//data[10] = '.';
	data[8] = '.';
	_serialPort->Write(data, 0, 1);
	_serialPort->Write(data, 1, 1);
	_serialPort->Write(data, 2, 1);
	_serialPort->Write(data, 3, 1);
	_serialPort->Write(data, 4, 1);
	_serialPort->Write(data, 5, 1);
	_serialPort->Write(data, 6, 1);
	_serialPort->Write(data, 7, 1);
	_serialPort->Write(data, 8, 1);
	//_serialPort->Write(data, 9, 1);
	//_serialPort->Write(data, 10, 1);
	message = _serialPort->ReadLine();

	// padarom is dvieju gautu baitu viena skaiciu
	hbits = message[0];
	lbits = message[1];
	number = hbits << 8 | lbits;

	printf("%d\r\n", number);

	_serialPort->Close();

	return 0;
```