KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

INFORMACIJOS IR INFORMACINIŲ TECHNOLOGIJŲ SAUGOS STUDIJŲ PROGRAMA

GRIGAS PETRAITIS

SLAPTO KANALO ORGANIZAVIMO NAUDOJANT DISKINIUS KAUPIKLIUS METODAS

Magistro baigiamasis darbas

Darbo vadovas
dr. N. Morkevičius

KAUNAS, 2013

KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

INFORMACIJOS IR INFORMACINIŲ TECHNOLOGIJŲ SAUGOS STUDIJŲ PROGRAMA

GRIGAS PETRAITIS

SLAPTO KANALO ORGANIZAVIMO NAUDOJANT DISKINIUS KAUPIKLIUS METODAS

Magistro baigiamasis darbas

Darbo vadovas
dr. N. Morkevičius

Recenzentas
doc. dr. T. Adomkus

KAUNAS, 2013

AUTORIŲ GARANTINIS RAŠTAS
DĖL PATEIKIAMO KŪRINIO

2013 - 05 - 22 d.
Kaunas

Autorius, Grigas Petraitis **,**
(vardas, pavardė)

patvirtina, kad Kauno technologijos universitetui pateiktas baigiamasis magistro darbas (toliau vadinama – Kūrinys) Slapto kanalo organizavimo naudojant diskinius kaupiklius metodas
(kūrinio pavadinimas)
pagal Lietuvos Respublikos autorių ir gretutinių teisių įstatymą yra originalus ir užtikrina, kad

1) jį sukūrė ir parašė Kūrinyje įvardyti autoriai;
2) Kūrinys nėra ir nebus įteiktas kitoms institucijoms (universitetams) (tiek lietuvių, tiek užsienio kalba);
3) Kūrinyje nėra teiginių, neatitinkančių tikrovės, ar medžiagos, kuri galėtų pažeisti kito fizinio ar juridinio asmens intelektinės nuosavybės teises, leidėjų bei finansuotojų reikalavimus ir sąlygas;
4) visi Kūrinyje naudojami šaltiniai yra cituojami (su nuoroda į pirminį šaltinį ir autorių);
5) neprieštarauja dėl Kūrinio platinimo visomis oficialiomis sklaidos priemonėmis.
6) atlygins Kauno technologijos universitetui ir tretiesiems asmenims žalą ir nuostolius, atsiradusius dėl pažeidimų, susijusių su aukščiau išvardintų Autorių garantijų nesilaikymu;
7) Autoriai už šiame rašte pateiktos informacijos teisingumą atsako Lietuvos Respublikos įstatymų nustatyta tvarka.

Autorius

_____ _____
(vardas, pavardė) (parašas)

SANTRAUKA

Šiame darbe analizuojama slaptos informacijos slėpimo diskiniuose kompiuterių kaupikliuose problema. Visų pirma išanalizuotas tyrimo objektas, t.y. klasterinės failų sistemos, jų fragmentacija, informacijos slėpimo būdai. Apžvelgti šiuo metu egzistuojantys informacijos slėpimo diskiniuose kaupikliuose metodai. Darbe pasiūlytas naujas slapto kanalo metodas, skirtas slėpti duomenis klasterinėse failų sistemose. Siūlomas metodas naudoja keletą dengiančių failų, o informaciją perduoda keisdamas tų failų klasterių tarpusavio išsidėstymą failų sistemoje. Sukurta eksperimentinė programa, paremta pasiūlytu metodu. Naudojantis šia programa buvo atlikti metodo tyrimai, kurie patvirtino, kad metodas pasižymi dvigubo pagrįsto išsigynimo savybe.

SUMMARY

This work is based on the problem of information hiding in computer disk drives. Firstly, the object of study, i.e. clustered file systems, their fragmentation, ways to hide information, has been analyzed. Moreover, currently existing methods to hide information in computer disks has been reviewed. Then, a new covert channel method, designed to hide data in clustered file systems, has been proposed. The method uses multiple cover files and hides data by changing relative positions of clusters of those files. Finally, an experimental application that exploits the proposed method has been created. With the help of the application the method has been examined and it is proved that it has the property of two-fold plausible deniability.

TURINYS

LENTELIŲ SĄRAŠAS

PAVEIKSLŲ SĄRAŠAS

TERMINŲ IR SANTRUMPŲ ŽODYNAS

1. ANSI (angl. American Nation Standards Institute) – nepelno organizacija, kuri atsakinga už produktų, paslaugų, procesų, sistemų bei personalo standartų priėmimą bei vystymą JAV.
2. API (angl. application programming interface) - tai sąsaja, kurią suteikia kompiuterinė sistema, biblioteka ar programa tam, kad programuotojas per kitą programą galėtų pasiekti jos funkcionalumą ar apsikeistų su ja duomenimis.
3. CF (angl. collision flag) – aptiktos sunkios kolizijos požymis.
4. CR (angl. collision resolution) – sunkios kolizijos išsprendimo bitas.
5. EOF (angl. end-of-file) – failo pabaiga.
6. FAT (angl. file allocation table) – 1977 m. Microsoft sukurta failų sistemų šeima, pasižyminti paprastumu bei santykinai našiu veikimu. Šiai šeimai priklauso failų sistemos FAT12, FAT16, FAT32 ir kt. 1996 m. Tuo pačiu principu paremta ir 1996 m. Microsoft pasiūlyta exFAT failų sistema.
7. FFS (angl. fast file system) – 1983 m. sukurta failų sistema, naudojama (dabar jau tik kaip pagrindas) daugelyje Unix šeimos operacinių sistemų.
8. Failo fragmentas „atgal“ – failo fragmentas, kurio pirmojo klasterio numeris failų sistemoje yra mažesnis už prieš jį einančio fragmento pirmojo klasterio numerį.
9. Failo fragmentas „pirmyn“ – failo fragmentas, kurio pirmojo klasterio numeris failų sistemoje yra didesnis už prieš jį einančio fragmento pirmojo klasterio numerį.
10. FS (angl. file system) – failų sistema.
11. GPL licenzija – plačiausiai naudojama nemokamos programinės įrangos licenzija, kuri leidžia keisti, platinti programinę įrangą tik su sąlyga, kad ji taip pat bus su GPL licenzija.
12. IDE (angl. integrated development environment) – programinė įranga, skirta programuotojams. Paprastai į ja įeina kodo redaktorius, kompiliatorius, automatizavimo įrankiai bei derintuvė.
13. JFS (angl. journaled file system) – 64 bitų IBM sukurta failų sistema, skirta Unix šeimos operacinėms sistemoms.
14. LFN (angl. long file names) – galimybė FAT failų sistemose saugoti failus, kurių pavadinimai sudaryti iš iki 256 simbolių.
15. MIT licenzija – nemokamos programinės įrangos licenzija, leidžianti keisti bei platinti produktą su sąlyga, kad visos kopijos bus platinamos taip pat su šia licenzija.
16. OS (angl. operating system) – operacinė sistema.
17. PA – panaudojimo atvejis.
18. Slapto kanalo metodo talpa – reikšmė, nurodanti, kiek slepiamų bitų galima paslėpti vienu dengiančio failo klasteriu.
19. SQL (angl. structured query language) – struktūrizuota užklausų kalba.
20. UFS (angl. Unix file system) – 1994 m. sukurta failų sistema, kuri perėmė daugelį FFS savybių.

ĮVADAS

Duomenų šifravimas naudojant standartinius kriptografijos metodus gali būti efektyvus būdas paversti pradinę informaciją į neaiškių simbolių seką, kurią iššifruoti gali tik tie asmenys, kuriems ji skirta, tačiau yra atvejų, kai kur kas naudingiau paslėpti patį informacijos egzistavimo faktą.

Šią problemą gerai apibūdina G. J. Simmons savo straipsnyje „The prisoners problem and the subliminal channel“ [21]: du nusikaltimo bendrininkai yra areštuojami ir uždaromi į skirtingas kameras. Vienintelis būdas jiems bendrauti – keistis žinutėmis, kurias gali matyti prižiūrėtojas. Taigi slapta informacija apsikeisti bendrininkai gali tik paslėpdami ją nekaltai atrodančiose žinutėse, kurias perskaitęs prižiūrėtojas nieko neįtars. Akivaizdu, kad pamatęs užšifruotą informaciją prižiūrėtojas atitinkamomis priemonėmis galės priversti atskleisti slaptą raktą ir tuo pačiu – perduodamą informaciją.

Informacijos ir technologijų amžiuje informacijos apsaugojimas tampa vis svarbesnis: turime rūpintis kortelių PIN kodų, banko sąskaitų bei interneto prieigos slaptažodžių saugumu, kitos privačios informacijos saugojimu.

Tuo tarpu informacijai saugoti kompiuteriai naudoja diskinius kaupiklius bei klasterines failų sistemas. Tai sukuria terpę paslėpti privačią informaciją, tačiau iškyla problema – kaip tą geriausia padaryti. Taigi šiame darbe ir sprendžiama informacijos paslėpimo diskiniuose kaupikliuose taip, kad nebūtų įmanoma aptikti jos egzistavimo, problema.

Norint išspręsti šią problemą, darbe buvo siekiama sukurti efektyvų metodą slaptam duomenų perdavimui, naudojant klasterines failų sistemas. Šiam tikslui pasiekti buvo užsibrėžti šie uždaviniai: susipažinti su duomenų saugojimu klasterinėse failų sistemose standartais, praktiškai ištirti failų sistemų užpildymo specifiką realiose OS, išanalizuoti duomenų slėpimo kituose duomenyse bei slapto duomenų perdavimo metodus, pasiūlyti slapto duomenų perdavimo, panaudojant klasterines failų sistemas, metodą bei, praktiškai jį realizavus, atlikti jo kiekybinę bei kokybinę analizę

Taigi darbe analizuojamos įvairių failų sistemų struktūros, duomenų saugojimo bei jų paslėpimo kituose duomenyse metodai, slapto duomenų perdavimo problema ir jos sprendimo būdai, diskinių kaupiklių failų sistemų naudojimo ir užpildymo specifika.

Išanalizavus tyrimo sritį, pasiūlytas naujas efektyvus metodas slaptam duomenų perdavimui, naudojant klasterines failų sistemas. Pasiūlyto metodo savybės ištirtos eksperimentiškai, įrodoma, kad metodas pasižymi dvigubo pagrįsto išsigynimo savybe.

Galiausiai, panaudojant sukurtą eksperimentinę metodo realizaciją, metodo charakeristikos išanalizuotos tiek kiekybiniu, tiek kokybiniu atžvilgiu: ištirtas paslėptų duomenų atsparumas darbui su dengiančiais failais, failų sistemos defragmentacijai, atliktas laikinių charakteristikų tyrimas.

1. ANALIZĖ

Norint išspręsti slaptos informacijos perdavimo problemą, galima panaudoti duomenų slėpimo metodus, tokius kaip steganografija arba slaptas kanalas [17]. Steganografiniai metodai paslepia duomenis „įprastoje“ dengiančioje terpėje taip, kad nesukeltų galimo stebėtojo įtarimo [20], pvz. slaptą tekstinį pranešimą galima paslėpti skaitmeniniame vaizde. Slapti kanalai leidžia paslėpti duomenis terpėje, kuri nėra skirta duomenų perdavimui ar saugojimui. Tikslui pasiekti jie dengiančią informaciją perduoda tam tikru būdu išdėstytą laike, laikmenoje ar kitoje terpėje taip, kad gavėjas tik pagal jos išdėstymą gali išgauti slaptą pranešimą [21, 14]. Dažniausiai duomenų slėpimo metodai skirti naudoti informacijos perdavimo protokoluose [12, 7, 3], tačiau egzistuoja daugelis darbų [19, 9, 1, 15, 16, 2], kurie aprašo efektyvius duomenų slėpimo metodus kietuosiuose kompiuterių diskuose.

1.1. Analizės tikslas

Analizės tikslas buvo suprasti kaip veikia klasterinės failų sistemos, kurias naudoja diskiniai kaupikliai. Taip pat svarbu suprasti slapto duomenų perdavimo metodų problemas ir išanalizuoti jau egzistuojančius sprendimus.

Analizės uždaviniai:

1. Klasterinių failų sistemų analizė. Palyginti kuo tarpusavyje skiriasi šiuolaikinės failų sistemos.
2. Išanalizuoti skirtingus duomenų slėpimo metodus bei jų pritaikymą klasterinėse failų sistemose.
3. Išsiaiškinti siūlomų informacijos slėpimo metodų specifiką, juos palyginti.
4. Atlikti failų sistemų analizės įrankių atranką.

1.2. Tyrimo objekto analizė

1.2.1. Klasterinės failų sistemos

Failų sistemų paskirtis yra labai paprasta: kompiuteriams reikia metodo, kuris leistų ilgai saugoti informaciją ir ją atgauti. Jos suteikia mechanizmą naudotojams saugoti duomenis aplankų bei failų hierarchijoje. Failų sistemą sudaro sisteminiai bei naudotojo duomenys, kurie yra saugomi taip, kad kompiuteris žinotų kur jų ieškoti. Dažniausiai šis saugojimo būdas yra nepriklausomas nuo konkretaus kompiuterio.

Tam, kad būtų galima paprasčiau palyginti failų sistemas tarpusavyje, jų saugomi duomenys yra skirstomi į penkias kategorijas: failų sistemos, turinio, metaduomenų, failų vardų, taikymo. Bet kokie duomenys failų sistemai priklauso vienai iš šių kategorijų [4].

Failų sistemos kategorijai priklauso pagrindinė jos informacija, tokia kaip kur ieškoti tam tikrų duomenų struktūrų ir kokio jos yra dydžio. Šios kategorijos duomenys – lyg failų sistemos žemėlapis.

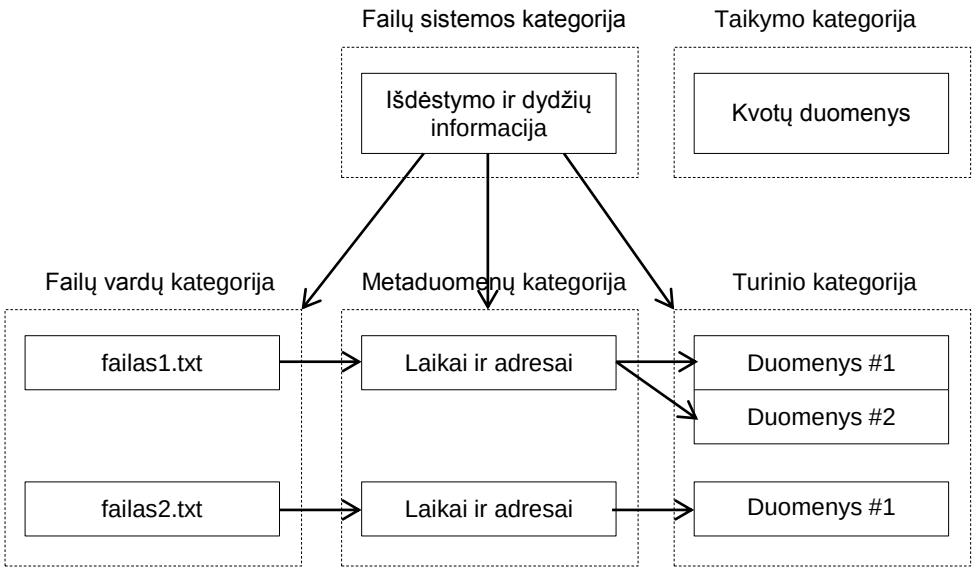
Į turinio kategoriją įeina naudotojo saugomų failų duomenys, todėl šios kategorijos duomenys užima didžiąją dalį sistemos bei yra suskirstyti į tam tikro dydžio blokus. Kiekvienas blokas turi savo adresą bei standartinį dydį.

Metaduomenų kategorijai priklauso duomenys, kurie apibūdina failus – ta yra duomenys apie duomenis. Jai priklauso informacija, nusakanti kur tam tikras failas yra patalpintas, kokio jis dydžio, sukūrimo, keitimo, nuskaitymo datos ir laikai, prieigos kontrolės informacija. Verta pastebėti, kad šiai kategorijai nepriklauso failų turinys ir failų vardai taip pat gali nepriklausyti.

Failų vardų, kitaip vadinamos sąsajos žmogui, kategorija saugo duomenis, kurie priskiria vardus kiekvienam failui. Daugumoje failų sistemų šie duomenys yra tam tikrame kataloge, kuriame yra failų vardai bei metaduomenų adresas.

Programų failų kategorijai priklausantys duomenys suteikia papildomas funkcijas, kurios nėra naudojamos failų nuskaitymui ar įrašymui ir jų gali išvis nebūti failų sistemoje. Tai gali būti duomenys apie naudotojų kvotas ar failų sistemos žurnalai.

Ryšys tap šių penkių kategorijų pavaizduotas žemiau esančiame 1 pav.



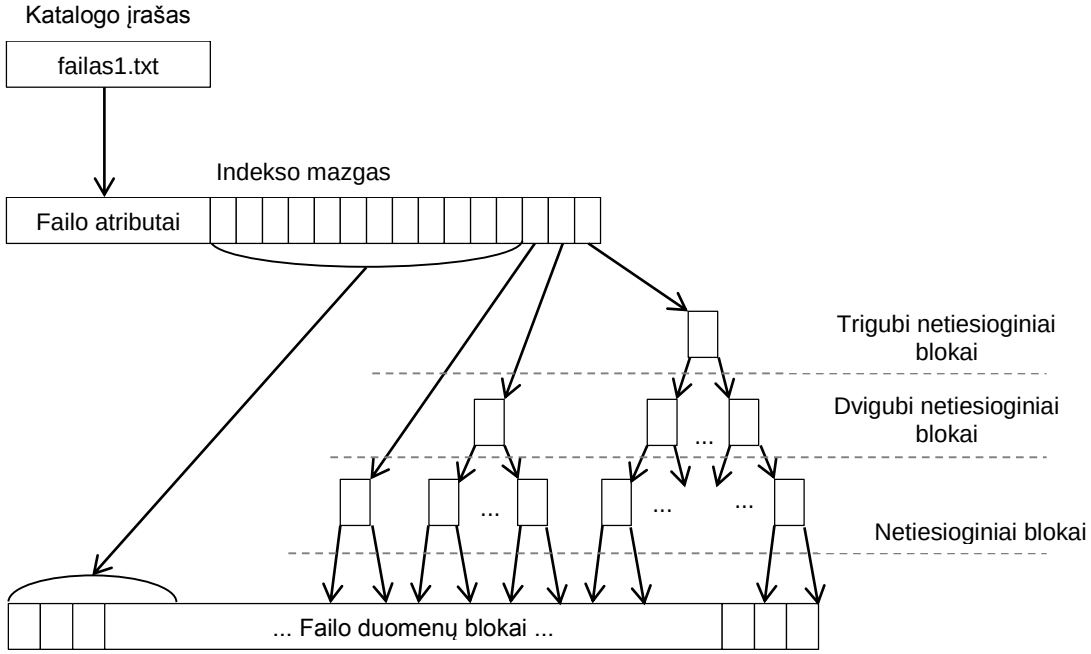
1 pav. Ryšys tarp duomenų kategorijų failų sistemoje

1.2.1.1. FFS (Fast File System)

FFS – UNIX šeimos failų sistemų ExtX pradininkė, 1983 m. buvo sukurta veikti greitai ir patikimai [22].

Viena iš pagrindinių jos struktūrų – superblokas yra patalpintas diskinio kaupiklio pradžioje fiksuotoje pozicijoje. Jis nusako sistemos būseną bei saugo tokią informaciją kaip bloko dydis, blokų skaičius, nuorodos į kitas sistemos struktūras bei kitą informaciją.

Kiekvienas failas šiose failų sistemoje yra aprašomas struktūra, vadinama *inode* – indekso mazgu. Ši struktūra saugo informaciją apie failo atributus bei nuorodas į blokus, kuriuose saugomi duomenys. Saugomi atributai – failo savininkas ir jo grupė, prieigos kontrolės duomenys, failo dydis, sukūrimo bei modifikavimo laikai. Kadangi šios struktūros dydis yra fiksuotas (128 baitai), nuorodų į duomenų blokus skaičius yra ribotas. Tam, kad būtų įmanoma saugoti didelius failus, naudojamas netiesioginis adresavimas (žr. 2 pav.)



2 pav. Pavyzdinis FFS blokų išdėstymas

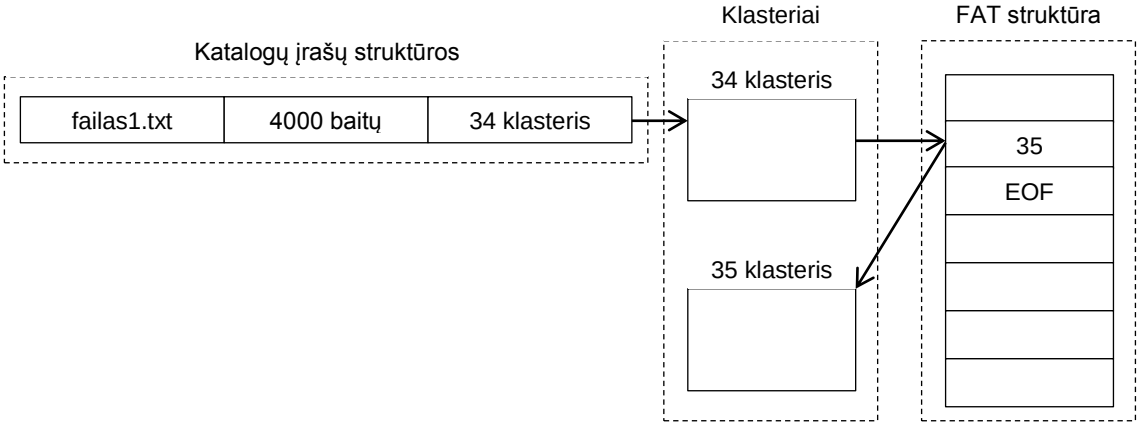
Aukščiau pateiktoje schemoje matome, kad indekso mazgas turi 15 nuorodų į disko blokus. Pirmi dvylika talpina nuorodas į pirmus 12 failo duomenų blokų. Kiti 3 skirti nuorodoms į netiesioginius blokus. Pirmasis rodo į pirmo lygio netiesioginės adresacijos bloką, antrasis į antro

lygio, rodantį į kitus pirmo lygio adresacijos blokus ir trečiasis – trečio. Tokio tipo adresacija leidžia greičiau pasiekti nedidelius failus, o naudojant numatytąjį 8 kB bloko dydį galima pasiekti 64 TB failo dydį. Netiesioginiai blokai yra užimami tik tada, kai failo dydis to reikalauja.

Pradinė šios failų sistemos versija atlikdavo visas rašymo ir skaitymo operacijas atskirai su kiekvienu bloku, t.y. jei failas užima 64 kB ir bloko dydis yra 8 kB, būtų atliekamos 8 skirtingos operacijos. Tam kad išvengti našumo problemų, blokai buvo apjungti į klasterius, paprastai 64 kB dydžio. Tai leido dirbti iškart su 8 blokais, t.y. net jei mums reikia tik vieno bloko, nuskaitomas yra visas klasteris tam atvejui, jei būtų atliekamas nuoseklusis skaitymas.

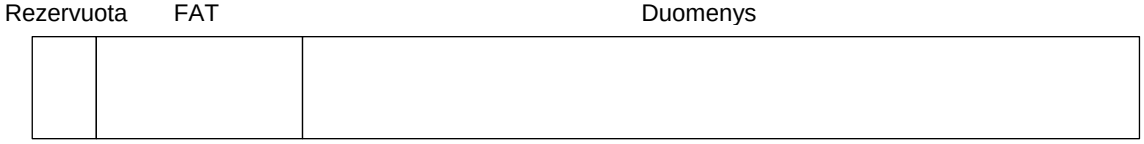
1.2.1.2. FAT (angl. File Allocation Table)

Viena iš pagrindinių priežasčių kodėl šį failų sistema vadinama paprasta – ji turi mažai duomenų struktūrų. Taip pat jos duomenys negali būti skirstomi į anksčiau pateiktas penkias kategorijas, nes joje failų vardų kategorija sujungta su metaduomenimis, o programų kategorijos nėra. Šios failų sistemos pagrindą sudaro dvi struktūros – katalogų įrašų bei failų paskirstymo lentelės (FAT). Grafiškai ryšys tarp šių struktūrų pavaizduotas 3 pav.



3 pav. Ryšys tarp katalogų įrašų, klasterių bei FAT struktūros

Kaip matome iš aukščiau pateiktos schemos, kiekvienas failas turi įrašą katalogų įrašų struktūroje, kurioje be failo vardo dar saugomas jo dydis, pradžios adresas bei kiti metaduomenys. Failų bei katalogų duomenys yra saugomi duomenų blokuose, vadinamuose klasteriais. Jei failas ar katalogas užima daugiau nei vieną klasterį, kiti klasteriai yra randami naudojant struktūrą, vadinamą FAT. Ji yra naudojama kito failui priklausančio klasterio radimui bei klasterių vietų užimtumo nustatymui. Fizinis FAT failų sistemos išdėstymas pateikiamas žemiau.



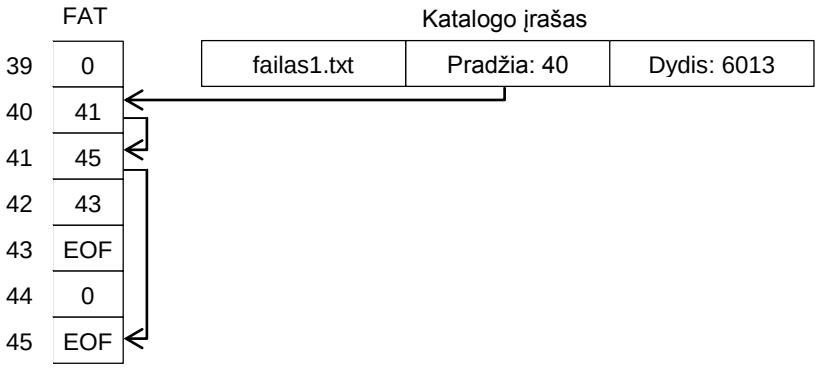
4 pav. Fizinis FAT failų sistemos išdėstymas

1.2.1.2.1. Klasterių paskirstymas

FAT failų sistemoje naudojami duomenų blokai vadinami klasteriais. Tai yra grupės iš eilės einančių sektorių, o sektorių skaičius turi būti 2 laipsnis, t.y. 1, 2, 4, 8, 32 ar 64. Maksimalus klasterio dydis šioje failų sistemoje yra 32 kB. Visi klasteriai turi savo adresus, kurie pradedami numeruoti nuo 2 ir prasideda duomenų srityje.

Kaip jau buvo minėta aukščiau, klasterių panaudojimo būsena nustatoma pagal FAT struktūrą – ji turi po vieną įrašą kiekvienam klasteriui. Įrašų skaičius priklauso nuo naudojamos FAT failų sistemos versijos: FAT12 naudoja 12-ą bitų klasterių adresams, FAT16 – 16, o FAT32 – 32 (tačiau tik 28 bitai yra naudojami adresams). Jei lentelėje ties adresu yra 0, vadinasi klasteris yra laisvas, jei 0x0ffffff7 (FAT32) – klasteris pažeistas, o visos kitos reikšmės reiškia, kad jis užimtas.

Klasterių paskirstymo algoritmą nurodo operacinė sistema, tačiau dauguma sistemų naudoja „pirmas laisvas“ algoritmą, kai imamas pirmas pasitaikęs laisvas klasteris po jau užimtojo. Taip daroma todėl, kad diskiniai kaupikliai daug sparčiau dirba su klasteriais, kai jie yra šalia vienas kito. Tačiau dėl aktyvaus darbo su failų sistema dažnai nepavyksta rezervuoti visų klasterių iš eilės - susidaro fragmentacija (plačiau apie ją 1.2.2 skyriuje). Tokiu atveju visus failų klasterius rasti pavyksta naudojant FAT struktūrą panašiai kaip dinaminį sąrašą – žr. 5 pav.



5 pav. Failo užimtų klasterių grandinė

Palygindami šią failų sistemą su prieš tai apžvelgtaja FFS galime teigti, kad FFS failų sistema yra sudėtingesnė, tačiau tai leidžia pasiekti didesnį našumą. Visgi turint omenyje, kad FAT failų sistema yra labiau paplitusi dėl savo suderinamumo su skirtingomis sistemomis, joje sukeliama didesnė failų fragmentacija (kas mūsų atveju yra naudinga) ir ji yra paprastesnė.

1.2.2. Failų sistemų fragmentacija

Tam, kad failai galėtų būti skaitomi ir rašomi diskiniame kaupiklyje reali fizinė failo vieta jame nėra labai aktuali. Tačiau norint šiuos veiksmus atlikti efektyviai, fizinė failo blokų vieta yra labai svarbi. Norint pasiekti didžiausią nuskaitymo greitį, reikia, kad šie blokai diske fizine prasme būtų kuo arčiau vienas kito, o jų išsidėstymo tvarka atitiktų loginę sujungimo tvarką failų sistemoje. Jei failo blokai bus išsimėtę po diską, jo galvutė turės šokinėti nuo vieno bloko prie kito vietoje to, kad nuskaitytų juos visus vienu ypu. Kadangi tai užima daug laiko, failų sistemos stengiasi fragmentacijos vengti.

Kai naudojamas „pirmo laisvo“ bloko radimo (FAT failų sistemų šeimos pagrindinis) algoritmas, diskui esant tuščiam visi failų blokai eina nuosekliai vienas po kito. Tačiau sistemą naudojant ilgiau, trinant bei įrašant naujus failus, laisvi blokai išsimėto po visą diską ir tuomemt naujo failo įrašymo metu jis yra fragmentuojamas.

Naujesnės, tokios kaip NTFS ar ExtX šeimos, failų sistemos naudoja efektyvesnius algoritmus failų fragmentacijai vengti, tačiau ir jose egzistuoja tam tikrų atvejų, kai sukelta fragmentacija gali būti netgi didesnė už FAT sistemos [6].

„The effects of filesystem fragmentation“ [6] autoriai atliko tyrimą, kurio metu nustatė aktyviai naudojamos failų sistemos fragmentacijos lygį. Bandymai buvo atliekami su 138 GB failų sistema, paliekant mažiausiai 5% laisvos vietos, naudojant failus nuo 500 MB iki 5 GB, atliekant 2500 naujų failų įrašymų, panaudojant iki 4 atskirų gijų. Rezultatai parodė, kad didžiausia fragmentacija atsiranda laisvos vietos diske sumažėjus iki 5%. Taip pat pastebėta, kad skirtingos failų sistemos, kai įrašymas vykdomas nuosekliai, nepatiria didelių fragmentacijos problemų, tačiau daugiagijėje aplinkoje sistemos, kurios į tai nekreipia dėmesio, patiria didelę fragmentaciją bei našumo sumažėjimą.

„Proceedings of the international conference on Measurement and modeling of computer systems“ dalyviai Douceur J. R. ir Bolosky W. J savo pranešime [8] pristatė tyrimo rezultatus, kurie atskleidė, kad paprastai failų sistemos būna tik pusiau užpildytos, todėl galima manyti, kad ir failų fragmentacija tokiose sistemose nėra didelė.

Savo straipsnyje „Carving contiguous and fragmented files with fast object validation“ [11] S. L. Garfinkel pateikia savo diskų fragmentacijos tyrimo, kuriame jis siekė išsiaiškinti failų fragmentaciją panaudotuose diskiniuose kaupikliuose, rezultatus. Jie parodė, kad 6% visų failų buvo

fragmentuoti, tačiau fragmentacijos pasiskirstymas kaupikliuose nebuvo vienodas – net apie pusė diskų neturėjo nė vieno fragmentuoto failo. Taip pat jis pateikia tris pagrindines priežastis, dėl kurių atsiranda failų fragmentacija:

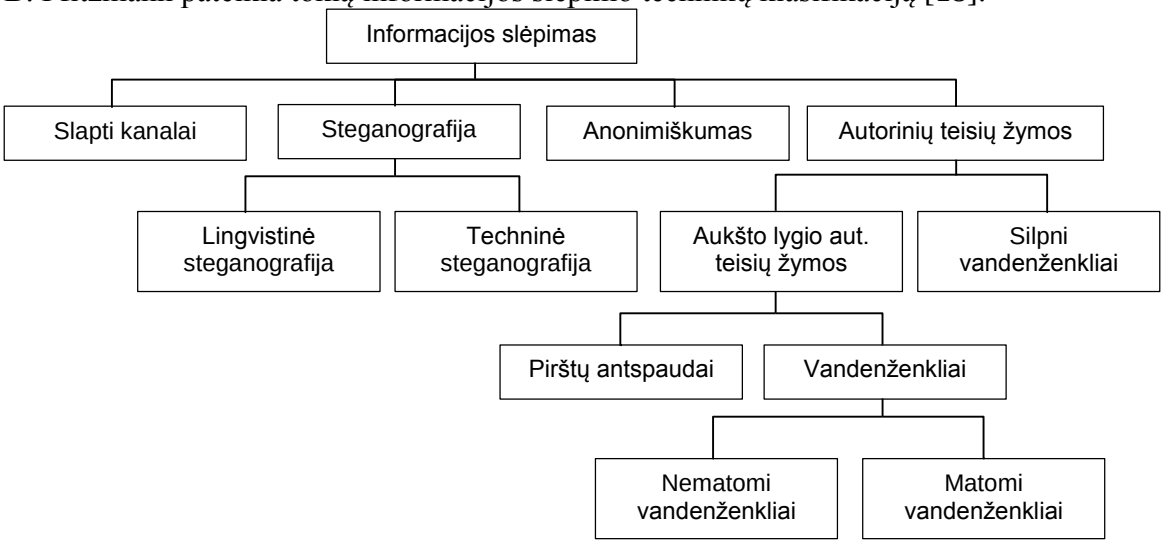
1. Naudojamas ilgai ir daug naudotas diskinis kaupiklis, kuriame yra mažai laisvos vietos.
2. Papildant failą gali būti rezervuota nepakankamai vietos, todėl reikia visą failą perkelti kitur arba papildomus duomenis rašyti į kitą vietą (dažniausiai naudojamas pastarasis variantas).
3. Kai kurios failų sistemos (pvz., UFS) didelius failus fragmentuoja specialiai.

Taip pat tame pačiame straipsnyje autorius išskyrė labiausiai fragmentuotus failus pagal jų tipą. Nustatyta, kad labiausiai fragmentuoti failai yra .tmp (66%), .pst (58%), .pnf (41%), .log (31%), .mdb (27%), .avi (20%).

1.2.3. Informacijos paslėpimo būdai

Nors šifravimas ir yra populiariausias būdas apsaugoti duomenis, tačiau jis turi ir savo minusą – užšifruoti duomenys sukelia įtarimą, kas tam tikrose situacijose gali reikšti ir jų priverstinį atskleidimą [17]. Tokiais atvejais geriau paslėpti patį informacijos egzistavimo faktą.

B. Pfitzmann pateikia tokią informacijos slėpimo technikų klasifikaciją [18]:



6 pav. Informacijos slėpimo technikos

Informacijos slėpime aktualūs terminai: slapti duomenys – tai pranešimas, kurį norime slaptai nusiųsti, slepiantys (dengiantys) duomenys – duomenys, kurie veikia kaip priedanga slaptam pranešimui paslėpti.

Eckstein K. ir Jahnke M. savo straipsnyje [9] pateikė 3 abstrakcijos lygmenis, kuriuose yra galimas duomenų paslėpimas:

1. Laikmenų valdymo lygmuo. Laikmena gali būti padalinta į keletą dalių, vadinamų skirsniais (angl. partition). Šis padalinimas yra įrašomas skirsnių lentelėse ar panašiose laikmenų valdymo struktūrose.
    1. Tuščių laikmenos vietų panaudojimas. Tai „standartinis“ būdas paslėpti informaciją laikmenų valdymo lygmenyje. Informacijos slėpimui naudojama vieta, kuri, remiantis skirsnių lentelių duomenimis, yra nenaudojama. Tarp skirsnių taip pat yra tam tikri tarpai, kuriuose galima patalpinti slaptą informaciją.
    2. Failų sistemos „užrašymas“ ant netuščio katalogo. Labai paprastas būdas paslėpti informaciją, esančią kažkuriame kataloge. Panaudojant operacinės sistemos funkcijas, aplanką galima panaudoti kaip nuorodą į tam tikrą failų sistemą. Egzistuojanti ar naujai sukurta failų sistema yra prijungiama (angl. mount) prie šio katalogo taip, kad jis rodo į tos failų sistemos pradžios direktoriją. Taip priėjimas prie aplanke buvusių duomenų yra užblokuojamas ir ten buvę duomenys paslėpiami.

2. Failų sistemos lygmuo. Jame galima pritaikyti daug įvairių informacijos paslėpimo metodų, kurie taip pat priklauso ir nuo failų sistemos struktūros, kurioje norima paslėpti duomenis (daugiau apie struktūras 1.2.1 skyriuje).
    1. Failų sistemos sisteminių duomenų struktūra. Sisteminės failų sistemos duomenų struktūros – tai skirsnių užkrovimo blokai, superblokai ir kitos globalios suktūros, kurios apibūdina failų sistemos pagrindines savybes. Šios struktūros ne visada panaudoja visą joms skirtą loginį bloką, todėl tam tikra dalis šio loginio bloko gali būti panaudota slėpti duomenims.
    2. Turinio kategorija: atliekama vieta blokuose. Kadangi failų turinys yra skirstomas į fiksuoto dydžio blokus (klasterius), o failų dydžiai paprastai nėra bloko dydžio kartotiniai, paskutiniame bloke dažniausiai lieka kažkiek laisvos vietos. Pvz., jei failas užima 10 kB, o bloko dydis yra 4 kB, tuomet bus užimti 3 blokai, iš kurių paskutiniame 2 kB bus laisvi. Tokiu atveju šioje bloko dalyje galima paslėpti duomenis, o šį failą būtų galima pavadinti slepiančiuoju failu.
    3. Metaduomenų kategorija: rezervuotų indekso mazgų (angl. inode) panaudojimas. Duomenims slėpti galima panaudoti indekso mazgus, kurių failų sistema dėl tam tikrų priežasčių nenaudoja. Pavyzdžiui, FFS failų sistemoje nenaudojami pirmi 2 indekso mazgai, o kai kurios failų sistemos tokių mazgų turi ir daugiau (JFS jų turi 13).
    4. Metaduomenų kategorija: išplėstieji failų atributai. Naujesnės failų sistemos turi papildomos vietos, skirtos aprašyti papildomiems failo atributams. Šią vietą galima panaudoti duomenims slėpti, jeigu žinoma, kad OS, kurioje tos failų sistemos naudojamos, nepalaiko tų atributų. Tokiu atveju, papildoma vieta yra tiesiog ignoruojama operacinės sistemos ir ją galima panaudoti savo reikmėms. Vietos kiekis priklauso nuo konkrečios failų sistemos.
    5. Failų vardų kategorija: „specialūs“ vardai. Pasitelkus tam tikrus failų vardus, tokius, kaip „..“ arba saugant juos failų pripildytuose sisteminiuose kataloguose, galima padaryti failus sunkiai pastebimus neįgudusiam analitikui.
    6. Failų vardų kategorija: atidarytų failų ištrynimas. Kai kurios operacinės sistemos leidžia ištrinti naudojimui atidarytą failą. Tokiu atveju, jis yra panaikinamas iš egzistuojančių failų sąrašo, tačiau jo metaduomenys ir turinys lieka iki kol jį naudojanti programa baigia darbą. Tai leidžia paslėpti failą nuo paieškos mechanizmų, tačiau neapsaugo nuo žemo lygio disko analizės.
3. Programų lygmuo.
    1. Paslėptos kilpinės (angl. loopback) failų sistemos. Kilpinės sistemos Unix šeimos operacinėse sistemose yra plačiai naudojamos ir gali būti atpažįstamos pagal specialų požymį. Tačiau šį požymį galima suklastoti kuriant kilpinę failų sistemą nurodant papildomą postūmį (angl. offset) taip ją prijungiant prie kito failo. Tokiu atveju failas yra sugadinamas, tačiau nustatyti, kad jame yra kilpinė failų sistema – sunku. Vėliau šios failų sistemos duomenis galima nesunkiai nuskaityti, jei žinomas panaudotas postūmis.
    2. Nenaudojama vieta failuose. Dauguma failų formatų turi tam tikrą nenaudojamą (ar retai naudojamą) plotą (pvz., komentaro laukas jpeg tipo failuose). Šią vietą galima panaudoti duomenims slėpti, tačiau jos dydis priklauso naudojamo failo formato ir jos nėra daug. Didelio kiekio dengiančių failų sukūrimas gali būti lengvai pastebėtas, tad reikėtų naudoti jau egzistuojančius failus.
    3. Steganografija. Ją informacijai paslėpti galima panaudoti labai įvariai ir priklauso nuo slepiančio failo tipo ir dydžio. Galimi variantai – muzikos, video, failai, nuotraukos ir pan. Plačiau apie steganografiją 1.2.4 skyriuje.

1.2.4. Steganografija

N. Provos ir P. Honeyman [20] steganografiją apibrėžia kaip komunikacijos slėpimo meną ir mokslą ir teigia, kad steganografinė sistema įterpia slaptą turinį į niekuo neišsiskiriančią dengiančią

terpę tam, kad nesukeltų stebėtojo įtarimo. Praeityje tam buvo naudojamas nematomas rašalas, o šiais laikais – kompiuteriai bei tinklų technologijos.

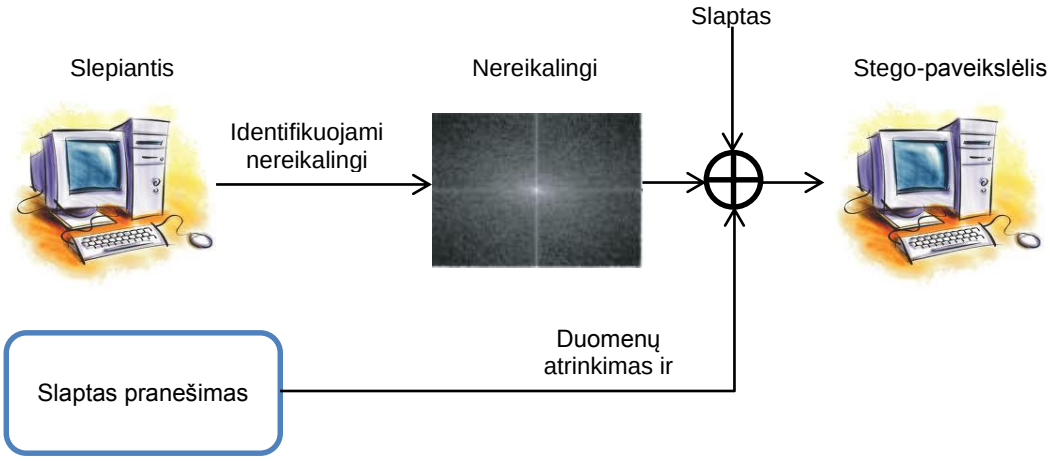
Informacijos slėpimo procesas prasideda nereikalingų bitų (tokių, kuriuos galima keisti nepažeidžiant terpės vientisumo) suradimu slepiančioje terpėje. Tuomet šie bitai yra pakeičiami slapto pranešimo bitais.

Steganografijos tikslas yra likti visiškai nepastebėtai, tačiau dėl atliekamų (kad ir kokių nepastebimų) pakeitimų slepiančioje terpėje, jos sukeltą „triukšmą“ aptikti galima. Tokio triukšmo aptikimas ir analizė vadinama statistine steganalize.

Informaciją slepiančias sistemas galima vertinti 3 aspektais: talpa (kiek duomenų galima paslėpti), saugumu (kaip sunkiai aptinkami paslėpti duomenys), atsparumu (kiek pakeitimų reikia atlikti slepiančioje terpėje, kad slaptas pranešimas būtų sugadintas). Vandenženklių sistemose svarbiausias yra pastarasis aspektas (norint nuimti vandens ženklą turi kartu susinaikinti ir terpė), tuo tarpu steganografinės sistemos orientuojasi į talpą bei saugumą, o atsparumui dėmesio visiškai neskiriama arba skiriama labai mažai.

Steganografijos saugumas, skirtingai nei šifravimo, pagrįstas algoritmo slaptumu. Paprastai, atskleidus algoritmą, nustatyti slaptus duomenis yra labai paprasta. Todėl modernūs steganografiniai algoritmai į slėpimo algoritmą įterpia dar ir slaptą raktą.

7 pav. matome, kaip į paveikslėlį įterpiamas slaptas pranešimas panaudojant slaptą raktą. Pirmiausia paveikslėlyje identifikuojami mažiausios reikšmės bitai, kurie gali būti panaudoti duomenų slėpimui. Tuomet, panaudojant tam tikrą steganografinį metodą, identifikuoti bitai apjungiami su slaptu raktu bei slepiamais duomenimis, gaunamas šiek tiek pakeistas „nereikalingų bitų“ paveikslėlis. Galiausiai, jis sujungiamas su slepiančiu paveikslėliu ir gaunamas nuo originalo beveik nesiskiriantis paveikslėlis su paslėptais duomenimis.



7 pav. Pranešimo paslėpimas, panaudojant moderniąją steganografiją

Įgyvendinta daug steganografinių metodų, skirtų naudoti diskiniuose kaupikliuose, paslepiant slaptus duomenis nenaudojamose failų sistemos vietose. Tarp jų: StegFS, sukurta Pang H., Tan K., Zhou X. [16], taip pat tokio pačio pavadinimo StegFS, pasiūlyta McDonald A.D. ir Kuhn M. G. [15].

1.2.5. Slaptas kanalas

Slaptas kanalas padeda paslėpti informaciją tokioje terpėje, kuri nėra skirta perduoti ar saugoti duomenis [14].

Nors steganografijos tikslas yra lygiai toks pat, kaip ir slapto kanalo, tačiau yra vienas esminis skirtumas. Kaip jau minėta prieš tai buvusiame skyriuje, steganografinė sistema paslepia slaptą pranešimą nekaltai atrodančioje terpėje. Tuo tarpu slapti kanalai informaciją perduoda panaudodami esybes, kurios nėra skirtos perduoti informacijai. Savo tikslą slapti kanalai pasiekia dengiančią informaciją perduodami išdėstytą laike, laikmenoje ar kitoje terpėje taip, kad gavėjas tik pagal jos išdėstymą gali išgauti slaptą pranešimą.

Kaip matome, slaptų kanalų stiprybė slypi sugebėjime perduoti duomenis per terpę, kuri tam nėra skirta. Tinkamai sukurta sistema paprastai suteikia labai nedaug galimybių ją panaudoti slaptai

komunikacijai, nes komunikacijos subjektai negali jos kontroliuoti. Taigi slaptų kanalų talpa smarkiai nusileidžia tai, kurią galima pasiekti steganografiniais metodais.

Taigi, lyginant su steganografiniais metodais, slapto kanalo sprendimai gali paslėpti mažiau informacijos, tačiau dėl kitokios duomenų slėpimo specifikos tų duomenų aptikti praktiškai neįmanoma.

1.2.6. Dvigubo pagrįsto išsigynimo savybė

H. Khan et al. savo darbe [14] įveda du saugos metodų, skirtų duomenų slėpimui, efektyvumo įvertinimo lygius – tai pagrįstas išsigynimas ir dvigubas pagrįstas išsigynimas. Šios sąvokos apibūdina kaip efektyviai metodas gali paslėpti duomenis. Pagrįstas išsigynimas yra tokia saugos metodo savybė, kuri vienai iš šalių leidžia pagrįstai teigti kitoms šalims (pvz., teisėjui arba tyrėjui), kad paslėpti duomenys neegzistuoja [15]. Dažnai šia sąvoka apibūdinami ir metodai, leidžiantys šaliai atskleisti tik dalį paslėptų ir/ar užšifruotų duomenų ir sėkmingai paneigti bet kokių kitų duomenų egzistavimą. Šiuo atveju pats paslėptų duomenų egzistavimo faktas nėra efektyviai paslepiamas. Pavyzdžiui, didžioji dalis steganografinių metodų slepia duomenis nenaudojamose kietojo disko vietose [15, 16, 2, 13, 5]. Tyrėjas, atliekantis disko analizę, gali lengvai rasti nenaudojamus disko sektorius, užpildytus tam tikrais duomenimis, bet nežinodamas rakto, jis negali įrodyti, jog tai paslėpti svarbūs duomenys, o ne atsitiktiniai duomenys įrašyti anksčiau dėl intensyvaus disko naudojimo. Visi steganografiniai metodai turi tik pagrįsto išsigynimo savybę.

Dvigubo pagrįsto išsigynimo savybę turi tokie duomenų slėpimo metodai, kurie užtikrina, kad šalis gali du kartus pagrįstai paneigti paslėptų duomenų egzistavimą. Pirmuoju bandymu galima neigti, kad išvis egzistuoja bet kokie paslėpti duomenys. Nepavykus pirmajam bandymui, šie metodai palieka galimybę dar kartą paneigti duomenų (ar svarbesnės jų dalies) egzistavimą taip pat kaip viengubo pagrįsto išsigynimo atveju. Šią, dvigubo pagrįsto išsigynimo, savybę turi tik slapto kanalo metodai, kurie slėpdami informaciją į terpę neįrašo jokių papildomų duomenų. Du slapto kanalo metodai, tinkami naudoti kietuosiuose diskuose ir turintys dvigubo pagrįsto išsigynimo savybę, aprašyti [14] bei apžvelgti 1.3 skyriuje.

1.3. Esamų sprendimų analizė

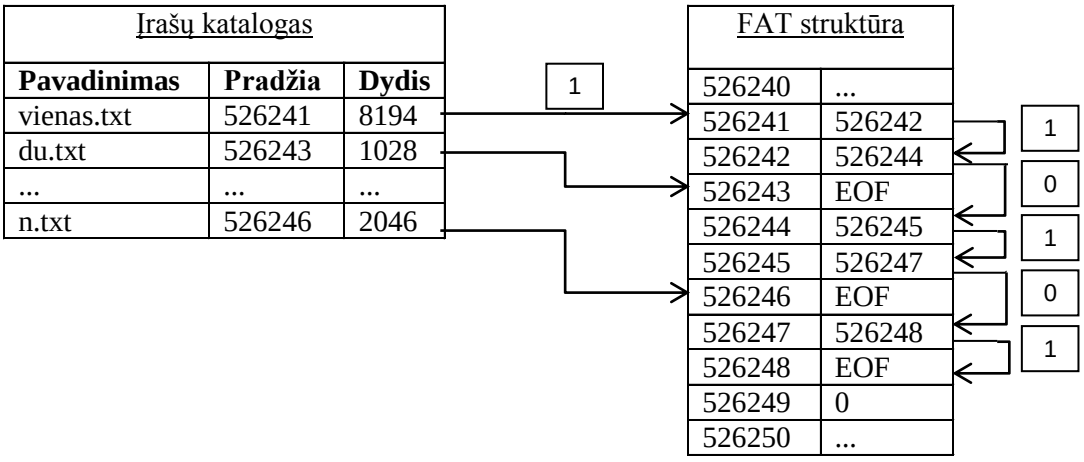
Slapto kanalo organizavimas panaudojant diskinius kaupiklius iki šiol nebuvo labai plačiai naudojamas. Tačiau 2011 m. straipsnyje „Designing a cluster-based covert channel to evade disk investigation and forensics“ [14] yra pateikiami du sprendimai – vienas paprastas ir neefektyvus, kitas – sudėtingas ir sąlyginai efektyvus. Abu metodai naudoja klasterių išdėstymą FAT failų sistemoje kaip būdą perduoti slaptai informacijai, todėl nesaugo nė vieno jos bito. Būtent šiuo atžvilgiu šiame straipsnyje pateikti metodai skiriasi nuo Pang H., Tan K., Zhou X. [16] ar McDonald A.D. ir Kuhn M. G. [15] pasiūlytų steganografinių metodų.

Abu pasiūlyti sprendimai yra saugesni už steganografinius atitikmenis, nes patys nesaugo duomenų, o tik pakeičia dengiančių duomenų išdėstymą failų sistemoje. Toks metodas turi šalutinį poveikį – fragmentaciją, tačiau, kaip rašyta 1.2.2 skyriuje, tai nėra retas reiškinys naudotuose diskiniuose kaupikliuose.

1.3.1. Paprastas sprendimas

Šis metodas naudoja FAT struktūrą bei klasterių sujungimą dinaminio sąrašo būdu. Dvejetainės žinutės paslėpimui naudojami tarpai tarp gretimų failo klasterių: jei norime paslėpti tokį patį bitą, kaip buvo praėjęs pranešimo bitas, klasterį rašome į gretimą vietą praėjusiam failo klasteriui. Kitu atveju, paliekame vieną laisvą tarpą ir rašome klasterį už jo. Tokia realizacija reikalauja pakeisti pirmojo failo klasterio vietą, nurodytą katalogo įraše, ir visą jo FAT struktūrą.

Norėdami paslėpti pranešimą 110101 FAT failų sistemoje aukščiau aprašytu būdu, turėsime panaudoti dengiantį failą, kuris užima bent 6 klasterius:



8 pav. Paprasto slapto kanalo metodo įgyvendinimas

Pirmiausia yra paimamas pirmas norimas rašyti bitas, kuris šiuo atveju yra „1“. Tada dengiančio failo pirmą klasterį rašome į pirmą laisvą nelyginę vietą (tokiu būdų išsaugodami slaptą „1“). Jei pirmas bitas būtų buvęs „0“, pirmą klasterį būtume rašę į lyginę klasterio vietą. Antras bitas taip pat lygus „1“, todėl kitą dengiančio failo klasterį rašome į vietą, esančią iškart po praėjusio klasterio (taip išsaugodami antrą slaptą bitą „1“). Kadangi trečias bitas ne toks pat, kaip antrasis, trečią slepiančio failo klasterį rašome į tolimesnę vietą, taip išsaugodami trečią slaptą bitą. Tokiu būdu išsaugoma visa slapta žinutė.

Šis metodas nėra idealus ir turės problemų, kai norėsime rašyti į iškart po einamo klasterio esančią vietą, o ją bus užėmęs kitas failas. Tokiu atveju metodą galima pakeisti taip, kad kitą klasterį rašytume į nelyginę vietą, kai norime paslėpti „1“ ir lyginę, kai „0“.

Sprendimo saugumą galima padidinti naudojant daugiau klasterių nei reikia. Tada bitus slepiantys dengiančiojo failo klasteriai nei pradės, nei užbaigs failo. Tokiu atveju paslėpti duomenys bus tik kažkurioje slepiančio failo dalyje ir jų ribų nebus įmanoma nustatyti be specialaus požymio, kurį žinotų tik slapta informacija besikeičiantys subjektai.

1.3.2. Sudėtingas sprendimas

Naudojant šį metodą pasiekiamas didesnis talpinamų slaptų duomenų kiekis nei prieš tai aprašytajame.

FAT32 failų sistema, turinti n klasterių turi FAT struktūroje įrašus sunumeruotus 1, 2, ..., n. Kiekvienam tokiam įrašui šioje struktūroje yra skirti 32 bitai, iš kurių 4 yra rezervuoti, o 28 naudojami nurodyti klasteriui. Tai suteikia leistiną adresų 0x0 – 0x10000000 intervalą, kuris gali būti panaudotas slaptos informacijos įterpimui į klasterių grandinę. Slaptai informacijai saugoti šis metodas naudoja atstumą tarp fragmentuotų klasterių.

Tarkime, norime paslėpti 8 baitus duomenų: 0x0123456789ABCDEF. Padaliname juos į 3 dalis po 3 baitus ir pridedame 0 prie pirmojo baito. Gauname 0x000123, 0x456789 ir 0xABCDEF. Šiems trims baitų rinkiniams paslėpti reikės mažiausiai 4 klaserių FAT failų sistemoje. Taigi jei klasterio dydis yra 2 kB, reikės bent 8 kB failo, norint paslėpti 8 baitus. Tarkime pirmas klasteris įrašytas adresu 0x4BFA60. Tam, kad paslėptume pirmą 3 baitų rinkinį prie šio adreso pridedame 0x000123, gauname 0x4BFB83, kur rašome kitus 2 kB slepiančiojo failo. Lygiai taip pat paslepiame ir kitus du trijų baitų rinkinius: 0x4BFB83 + 0x456789 = 0x91630C, 0x91630C + 0xABCDEF = 0x13D30FB.

Toks metodas leidžia paslėpti 3 baitus slaptų duomenų viename dengiančio failo klasteryje, tačiau yra didelė kolizijų tikimybė. Kolizija gali atsitikti tada, kai bandome rašyti klasterį į vietą, kuri jau yra užimta kito failo klasterio. Tokios kolizijos skirstomos yra 2 kategorijas – lengvas ir sunkias. Lengva kolizija nutinka, kai bandome rašyti į vietą, kuri jau yra priskirta kitam failui ir jis nėra dengiantis failas. Sunki kolizija atsitinka, kai bandome rašyti į vietą, kuri jau yra priskirta kitam failui ir jis yra dengiantis failas.

Tam, kad nustatytume kolizijos tipą, turime patikrinti ar klasteris, į kurį norime rašyti, priklauso egzistuojančiam dengiančiam failui. Toks patikrinimas reikalauja:

1. Nustatyti failą, kuriam priklauso klasteris;

2. Patikrinti ar tai dengiantis failas, naudojant turimas atpažinimo priemones.

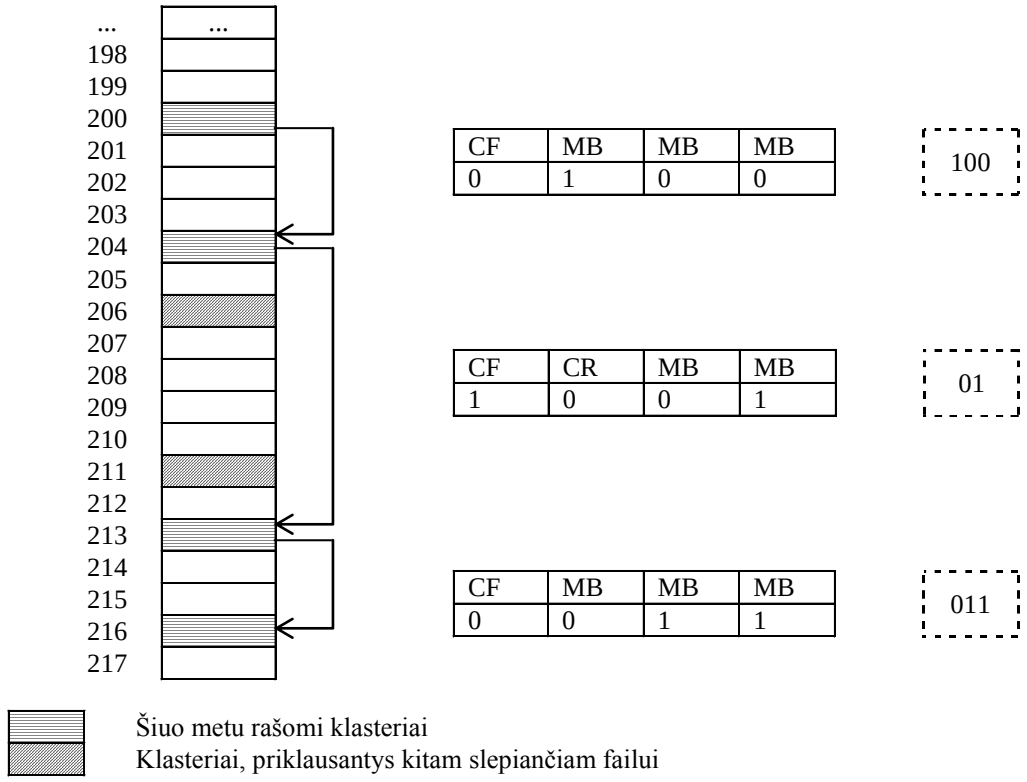
Lengvos kolizijos atveju užtenka surasti bet kokią laisvą klasterio vietą ir ten perkelti besimaišantį klasterį, taip atsilaisvinant vietą savo reikmėms. Sunkios kolizijos atveju klasteris negali būti perkeliamas, nes tuomet sugadinsime slaptus duomenis, kuriuos saugo dengiantis failas. Taigi būtinas metodas kolizijų problemai spręsti.

Metodo autoriai sunkių kolizijų problemą sprendžia įterpdami papildomą bitą į kiekvieną slepiamų ℓ bitų rinkinį, tam kad būtų galima atpažinti sunkią koliziją ir pavadino tą bitą kolizijos požymiu (CF). Šis bitas yra įjungiamas, jei aptinkama sunki kolizija, ir tokiu atveju kiti δ (kur $\delta \leq {}^{\ell}/_{2}$) bitai yra naudojami kaip kolizijos atstatymo (CR) bitai, kurių reikšmė rodo į laisvą klasterio vietą. Pavyzdžiui, jei naudojami 4 CR bitai, turėsime 16 galimų vietų sunkios kolizijos išsprendimui. Pirmos $2^{\delta} - 1$ CR bitų kombinacijų yra naudojamos laisvos klasterio vietos radimui. Mažai tikėtinu atveju, kai visos šios vietos yra užimtos, paskutinė CR kombinacija yra naudojama kaip klasterio praleidimo požymis, kuris reiškia, kad šis bitų rinkinys neperduoda slapto pranešimo bitų, nes nepavyko išspręsti sunkios kolizijos.

Taigi naudojant aukščiau aprašytą kolizijų išsprendimo techniką ℓ bitų rinkinys, paslėptas klasterio vietoje k gali turėti tokias 3 semantikas:

1. CF bitas, $\ell - 1$ paslėptų bitų; jei nebuvo sunkios kolizijos.
2. CF bitas, δ CR bitų, $\ell - 1 - \delta$ paslėptų bitų; jei pavyko išspręsti sunkią koliziją.
3. CF bitas, $\ell - 1$ CR bitų; jei sunkios kolizijos išspręsti nepavyko naudojant δ CR bitų.

9 pav. pateikiamas pavyzdys, kuriame paslepiama dvejetainė žinutė 10001011 su sunkiomis kolizijomis. Žinutė yra padalinama į 3 bitų rinkinius ir į jų pradžią pridedamas CF bitas. Tarkime pirmasis dengiančio failo klasteris įrašytas į 200-ąją vietą. Pirmąjį bitų rinkinį paslepiame $(200)_{10}$ + $(100)_2$ = $(204)_{10}$ vietoje, kuri yra laisva. Tam, kad paslėptume antrąjį rinkinį, turėtume rašyti į 206 vietą, tačiau ji jau užimta kito slepiančio failo. Šiai sunkiai kolizijai išspręsti nustatomas CF bitas ir kitas bitas naudojamas kaip CR bitas. Kadangi CR bitas užėmė vieną vietą, šiuo klasteriu galima saugoti jau tik 2 slapto pranešimo bitus $(01)_2$. Kadangi vieta $(204)_{10}$ + $(1001)_2$ = $(213)_{10}$ yra laisva, klasterio duomenys surašomi ten ir kolizija yra išspręsta. Jei 213 vieta būtų taip pat užimta, kitas klasteris būtų įrašomas į poziciją su $(11xx)_2$ postūmiu, reiškiančią, kad sunkios kolizijos išspręsti nepavyko ir tas klasteris slaptų bitų neneša. Kadangi praeitame rinkinyje sutalpinti buvo tik 2 bitai, išmestąjį bitą pridedame prie paskutinio rinkinio. Taigi kitas klasteris rašomas į $(213)_{10}$ + $(0011)_2$ = $(216)_{10}$ poziciją, kuri yra laisva.



9 pav. Pranešimo paslėpimas, naudojant sudėtingesnį metodą

Verta pastebėti, kad vietos rezervavimas CR bitams turi gana didelį neigiamą poveikį algoritmo sugebėjimui slėpti didesnį kiekį informacijos, taigi jie turėtų būti naudojami tik tada, kai jų tikrai reikia. Norint nustatyti reikiamą CR bitų skaičių galima prieš rašant nuskaityti klasterių vietas ir atlikti slėpimo operacijos simuliaciją. Tokia išankstinė simuliacija leistų netgi išvengti CF bito, jei matytume, kad sunkių kolizijų nebus.

Straipsnio autoriai apskaičiavo, kad be sunkių kolizijų šis metodas leidžia paslėpti 3000 baitų 2 MB slepiančiame faile arba 2643 baitus tame pačiame faile, jei 50% užimtų klasterių priklauso kitiems slepiantiems failams. Talpos atžvilgiu šis metodas daug geresnis už pirmąjį, tačiau smarkiai nusileidžia steganografiniams analogams.

1.4. Siekiamas sprendimas

Šio darbo siekiamas sprendimas yra sukurti metodą, kurio pagalba būtų galima efektyviai organizuoti slaptą kanalą duomenų perdavimui panaudojant klasterinę failų sistemą FAT32. Sukurtas metodas turėtų pasižymėti dvigubo pagrįsto išsigynimo savybe.

1.4.1. Įrankių analizė

Tam, kad galėtume realizuoti sukurtą metodą bei jį tinkamai ištirti, reikės programinės bibliotekos darbui su diskiniais kaupikliais, programavimo aplinkos (IDE), failų sistemos analizės įrankių.

Kadangi norint įgyvendinti slaptą kanalą diskiniame kaupiklyje reikės keisti atskirų failų klasterių išdėstymą, būtina programinė įranga, galinti dirbti tiesiogiai su diskiniu kaupikliu. Surasti du atviro kodo sprendimai, kuriuos pamodifikavus galima įgyvendinti slaptą kanalą: ThinFAT32 [24] bei „FAT16/32 File System Library“ [10]. Abi bibliotekos sukurtos įterptinėms (angl. embedded) sistemoms, todėl yra paprastos.

ThinFAT32 privalumai: maža, lengva įgyvendinti, parašyta ANSI C kalba, palaiko LFN, beveik nereikalauja papildomų C bibliotekų, naudoja mažai operatyviosios atminties, paprasta naudoti. Biblioteka platinama su MIT licenzija, kas reiškia, kad jos kodas gali būti laisvai keičiamas ir platinamas su sąlyga, kad tai bus atliekama taip pat su ta pačia licenzija. Tačiau nors jos programinis kodas ir yra gana aiškus, bet beveik nedokumentuotas, todėl gali būti sudėtinga jį keisti.

„FAT16/32 File System Library“ privalumai: standartinės įvesties ir išvesties funkcijos, FAT16 bei FAT32 palaikymas, LFN palaikymas, formatavimo funkcija, katalogų naršymo funkcija,

galimybė naudoti podėlį. Platinama su GPL licenzija, kas reiškia, kad šia biblioteką taip pat galima keisti bei platinti su sąlyga, kad platinimas vyks su GPL licenzija. Programinio kodo yra gerokai daugiau nei ThinFAT32 bibliotekoje dėl platesnio funkcionalumo, tačiau jis geriau dokumentuotas, kas reiškia lengvesnį jo modifikavimą.

Failų sistemai tirti skirtas „The Sleuth Kit“ [23] leidžia analizuoti daug skirtingų failų sistemų (NTFS, FAT, UFS1, UFS2, EXT2, EXT3), veikia tiek Windows, tiek Unix sistemose ir analizuoja visą diskinių kaupiklių turinį, o ne tik tai, ką mato operacinė sistema. Taip pat šis įrankis turi ir grafinę sąsają, kuri palengvina darbą juo.

Kadangi programos realizacija rašoma C ir C++ kalbomis, naudojama Microsoft Visual Studio 2010 programavimo aplinka, kuri leidžia ne tik patogiai rašyti kodą, bet taip pat ir jį derinti.

1.5. Išvados

Išanalizavę tyrimo objektą bei jau esamus problemos sprendimus, galime padaryti šias išvadas:

1. Išanalizuotos dvi skirtingos failų sistemos FFS bei FAT ir nustatyta, kad eksperimentui atlikti FAT failų sistema yra tinkamesnė dėl savo plataus pritaikymo diskiniuose kaupikliuose net ir šiandien, o taip pat paprastesnių duomenų struktūrų ir valdymo, paprastai didesnės fragmentacijos.
2. Išsiaiškinti skirtingi būdai slėpti informacijai, labiau atkreipiant dėmesį į galimybę tai padaryti diskiniuose kaupikliuose. Nustatyta, kad tam geriausiai tinka steganografiniai arba slapto kanalo sprendimai, tačiau tik slapto kanalo metodai gali pasiūlyti dvigubo pagrįsto išsigynimo savybę, t.y. visišką slaptumą.
3. Palyginti du slapto kanalo metodai, leidžiantys paslėpti duomenis prisidengiant kita informacija diskiniuose kaupikliuose, išnaudojant failų klasterių išdėstymą. Nustatyta, kad jie turi šiuos trūkumus:
    1. neįmanoma pilnai užpildyti disko;
    2. sunkiai sprendžiama kolizijų problema;
    3. mažėjanti talpa, kai didėja failų sistemos užpildymas.
4. Nustatyta, kad, norint pasiekti aukštą slaptumo lygį, duomenis slėpti reikia ilgai naudotose sistemose, kurios yra kuo labiau užpildytos ir kuo labiau fragmentuotos. Dengiančiu failu geriausiai tinka būti failai, kurie paprastai būna smarkiai fragmentuoti.
5. Išanalizuoti įrankiai, kuriais naudojantis galima realizuoti slapto kanalo sistemą ir ją ištirti kiekybiškai bei kokybiškai.

2. SIŪLOMO SLAPTO KANALO ORGANIZAVIMO DISKINIUOSE KAUPIKLIUOSE METODO APRAŠYMAS

Metodo esmė yra slaptam duomenų perdavimui panaudoti keleto failų klasterių tarpusavio išsidėstymą failų sistemoje. Tokiu atveju į kitus failų sistemos duomenis galima nekreipti dėmesio ir analizuoti tik dengiančių failų klasterius. Galima sudaryti didėjimo tvarka surikiuotą masyvą su dviejų dengiančių failų klasterių numeriais ir stebėti, kaip vieno failo klasteriai „pinasi“ su klasteriais, priklausančiais kitam failui. Jei kažkurioje masyvo vietoje po pirmojo failo klasterio eina kito failo klasteris, tai atitinkamoje paslėptos žinutės vietoje yra bitas „1“. Jei kažkurioje masyvo vietoje yra du iš eilės to paties failo klasteriai, tai reiškia, jog paslėptoje žinutėje yra „0“. Tokiu būdu galima paslėpti visą slaptą žinutę neįrašant nei vieno bito į diską.

Norint padidinti slapto kanalo talpą, galima panaudoti daugiau dengiančiųjų failų. Tada kiekvienam failui reikia skirti numerius ir kiekvienam dengiančių failų klasterių masyvo elementui apskaičiuoti skirtumą tarp einamojo ir prieš tai buvusio failo numerių. Šis skirtumas ir bus atitinkamų paslėptos žinutės bitų reikšmė. Naudojant tokią taktiką galima paslėpti daugiau nei vieną slaptą bitą viename dengiančio failo klasteryje. Tokiu atveju dengiančių failų vardai bei jų pateikimo eilės tvarka tampa slaptojo rakto dalimi, nes juos būtina žinoti, norint nuskaityti paslėptus duomenis.

2.1. Duomenų slėpimo algoritmas

Duomenų paslėpimo algoritmą galima pavaizduoti blokine diagrama (žr. 10 pav.). Detalesnis jo aprašymas pateiktas žemiau.

2.1.1. Pradinė būsena ir duomenys

- Slepiama žinutė $M = [b_0, b_1, \ldots, b_{n-1}]$, kur n yra žinutę sudarančių bitų skaičius.
- Dengiantys failai $F_1, F_2, \ldots, F_p$, kur p yra dengiančių failų skaičius, $p = 2^m$, $m \in N$. Natūralusis skaičius m laikomas slapta informacija ir sudaro dalį slapto rakto. Failų F_i vardai yra simbolių eilutės t_i, $i = 1,2,\ldots,p$. Jų eilės tvarka taip pat yra dalis slapto rakto.
- c_{ij} yra failų sistemos klasteriai, kuriuose patalpintas failas F_i po slėpimo procedūros, $j = 0,1,\ldots,L_i$. Tokiu būdu kiekvienas dengiantis failas F_i gali būti atvaizduotas kaip rikiuotas klasterių numerių masyvas $F_i = [c_{i0}, c_{i1}, \ldots, c_{iL_i}]$. Čia L_i yra failą F_i sudarančių klasterių skaičius. Blogiausiu atveju (kai visa dengiančių failų informacija privalo būti išsaugota į diską) dengiančius failus sudarančių klasterių skaičius turėtų būti $L_i \geq k$, $k = n/m$.
- Masyvas, atspindintis tuščių failų sistemos klasterių numerius prieš slėpimo procedūrą, $D = [c_1, c_2, \ldots, c_{L_D}]$, $c_1 < c_2 < \ldots < c_{L_D}$, čia c_i, $i = 1,2,\ldots,L_D$, yra tuščių failų sistemos klasterių numeriai. Blogiausiu atveju L_D turėtų tenkinti šią lygybę:

$$L_D \geq \sum_{i=1}^{p} L_i \tag{1}$$

2.1.2. Inicializacija

Žinutė M padalinama į blokus po m bitų, $M = [B_1, B_2, \ldots, B_k]$. Jei paskutinis blokas nepilnas, jis užpildomas nulinėmis reikšmėmis. Kiekvienas bitų blokas B_i interpretuojamas kaip natūralusis skaičius: $0 \leq B_i \leq p-1$, $B_i \in N$, $i = 0,1,\ldots,k$. Inicializacijos vektoriaus B_0 reikšmė yra parenkama laisvai ir yra papildoma slapto rakto dalis.

2.1.3. Slėpimas

Duomenų paslėpimo algoritmo pseudokodas:

begin

$j_i := 0$, $i = 0,1,\ldots,p-1$;

for i=1 **to** k **do**

$N_i := B_{i-1} + B_i \pmod p$, $0 \le N_i \le p-1$; (2)

$c_i := c_{N_i j_{N_i}}$ (įrašyti vieną dengiančio failo F_{N_i} klasterį į FS poziciją c_i);

$j_{N_i} = j_{N_i} + 1$;

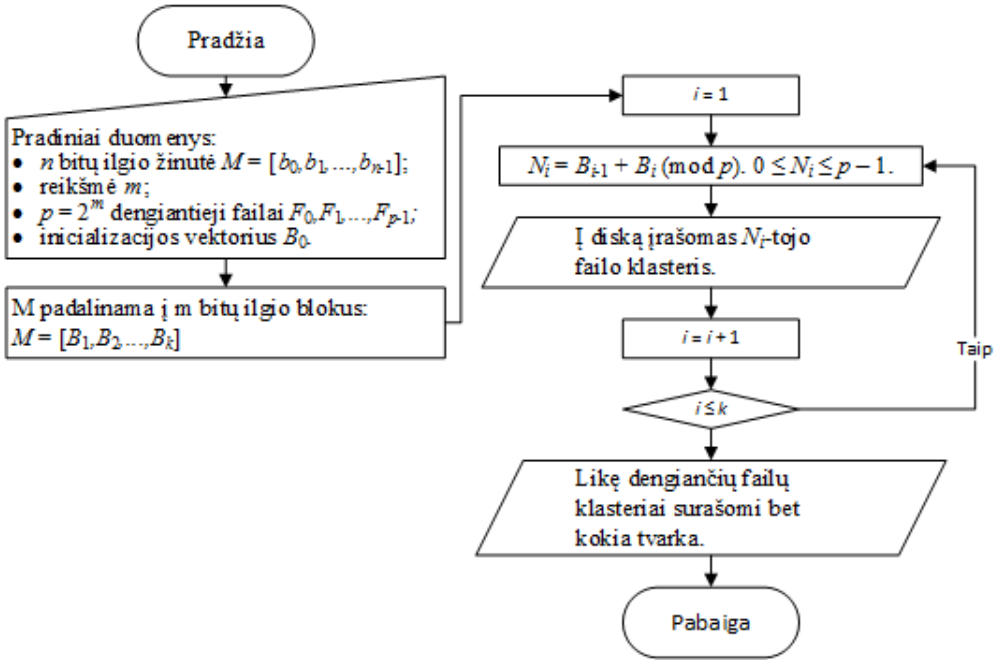
end

surašyti likusius dengiančių failų $F_1, F_2, \ldots, F_p$ klasterius į failų sistemą (jei reikia);

end

Atlikus slėpimo operaciją failų sistemoje, gaunamas pakeistas dengiančių failų klasterių išdėstymas D'.

Informacija, reikalinga nuskaityti paslėptai informacijai (slaptas raktas): reikšmė m, inicializacijos vektorius B_0, dengiančių failų vardai t_i tiksliai tokia tvarka, kokia buvo pateikti slėpimo algoritmui.



10 pav. Duomenų paslėpimo algoritmo blokinė diagrama

2.2. Duomenų nuskaitymo algoritmas

Duomenų nuskaitymo algoritmą galima pavaizduoti blokine diagrama (žr. 11 pav.). Detalesnis jo aprašymas pateiktas žemiau.

2.2.1. Pradinė būsena ir duomenys

- Slapto rakto informacija: reikšmė m, $m \in N$, inicializacijos vektorius B_0, dengiančių failų $F_1, F_2, \ldots, F_p$, $p = 2^m$, vardai t_i tiksliai tokia tvarka, kokia buvo pateikti slėpimo algoritmui ir žinutės ilgis n.
- Rikiuotas dengiančių failų F_i, $i = 1,2,\ldots,p$, klasterių failų sistemoje masyvas $D' = [c_1, c_2, \ldots, c_{L_D}]$, $c_1 < c_2 < \ldots < c_{L_D}$. c_{ij} yra numeriai klasterių, priklausančių failams F_i, $j = 0,1,\ldots,L_i$, čia L_i yra bendras atitinkamą failą sudarančių klasterių skaičius.

Taigi $F_i = [c_{i0}, c_{i1}, \ldots, c_{iL_i}]$, $L_D = \sum_{i=1}^{p} L_i$, $k = n/m$. Kiekvienas klasteris c_l masyve D' priklauso vienam ir tik vienam dengiančiam failui ($c_l = c_{xy}$, $l = 1,2,\ldots,L_D$).

2.2.2. Inicializacija

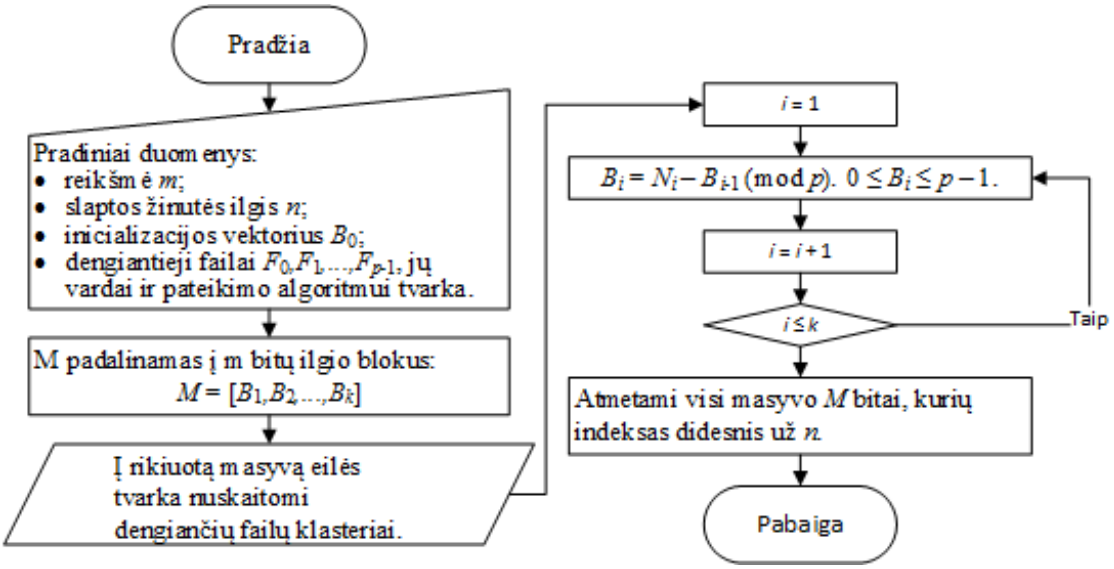
Masyvas, į kurį bus nuskaitoma paslėpta žinutė, suskaidomas į k blokų po m bitų, $M = [B_1, B_2, \ldots, B_k]$. Kiekvienas blokas B_i laikomas natūraliuojumi skaičiumi.

2.2.3. Nuskaitymas

Duomenų nuskaitymo algoritmo pseudokodas:

begin
  $j_i := 0$, $i = 0,1,\ldots,p-1$;
  for i=1 **to** k **do**
    nuskaityti i-tąjį klasterį c_i;
    rasti dengiančio failo indeksą N_i tokį, kad būtų tenkinama lygybė $c_i = c_{N_i j_{N_i}}$;
    $j_{N_i} = j_{N_i} + 1$;
    $B_i := N_i - B_{i-1} \pmod p$, $0 \le B_i \le p-1$; (3)
  end
  atmesti žinutės M bitus, kurie viršija ilgį n;
end

Po nuskaitymo procedūros gauname nuskaitytą slaptą žinutę M.



11 pav. Duomenų nuskaitymo algoritmo blokinė diagrama

2.3. Pavyzdžiai

Pateikiami du pavyzdžiai, padedantys lengviau suprasti metodo veikimą. Abiejuose pavyzdžiuose bus slepiamas tas pats pranešimas $M = [1,0,0,0,0,1,1,1,0]$, panaudojant skirtingus skaičius dengiančių failų. Abiem atvejais naudojamas inicializacijos vektorius $B_0 = 0$.

2.3.1. Slėpimas naudojant 2 dengiančius failus

Norėdami naudoti du dengiančius failus, parenkame atitinkamas reikšmes: $m = 1$, $p = 2^m = 2$. Imame dengiančius failus „A.txt“ ir „B.txt“, jų indeksai masyve bus atitinkamai 0 ir 1. Kadangi

$m=1$, t. y. slėpsime po vieną bitą vienu dengiančių failų klasteriu, tai žinutės M blokas B_i yra lygus vienam žinutės bitui.

Pagal (2) formulę paskaičiuojame, kurio failo klasterį rašysime pirmą: $N_1 := B_0 + B_1 \pmod p = 0+1 \pmod 2 = 1$. Į pirmą laisvą disko klasterį įrašome failo su pirmu indeksu („B.txt“) pirmąjį klasterį ir algoritmą vykdome toliau. Analogiškai $N_2 = 1+0 \pmod 2 = 1$, todėl į antrąjį nuo disko pradžios laisvą klasterį rašysime pirmojo failo antrąjį klasterį. Nesunku apskaičiuoti, kad $N_3 = 0$, $N_4 = 0$ ir t. t. Paslėpus visus bitų blokus, pabaigiame rašyti dengiančius failus bet kokia tvarka (arba likusių jų dalių visai neberašome į diską).

12 pav. pateiktoje schemoje matoma kaip atrodo FAT struktūra, paslėpus duotąją žinutę.

Įrašų katalogas		
Pavad.	*Pradžia*	*Dydis*
...	...	...
B.txt	256241	?
A.txt	526246	?
...	...	...

FAT struktūra		
Klasteris	*Kitas*	*Failas*
...		...
526241	526243	B.txt
526242	526244	failas1.txt
526243	526249	B.txt
526244	526245	failas1.txt
526245	526250	failas1.txt
526246	526247	A.txt
526247	526248	A.txt
526248	526251	A.txt
526249	526254	B.txt
526250	526252	failas1.txt
526251	526253	A.txt
526252	EOF	failas1.txt
526253	EOF	A.txt
526254	?	B.txt
...		...

12 pav. FAT struktūra po duomenų paslėpimo operacijos

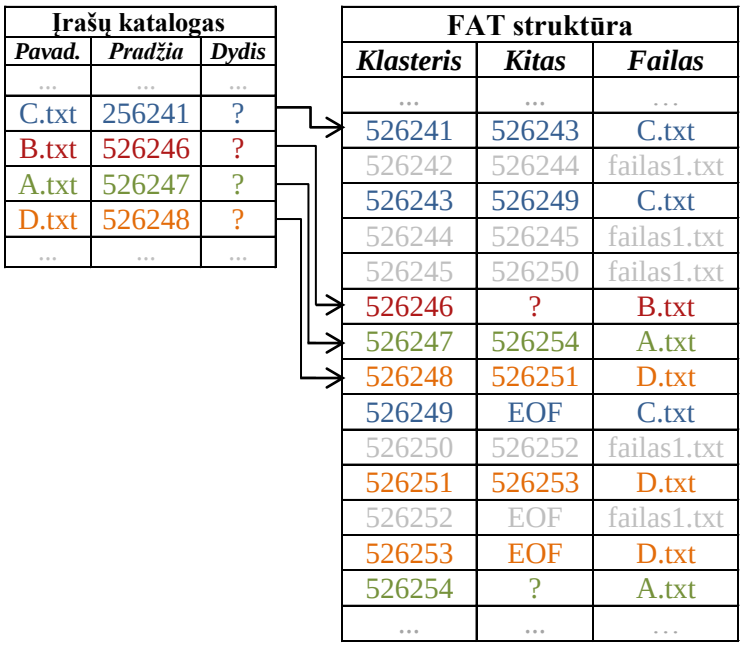
Norėdami nuskaityti slaptą pranešimą, turime žinoti paslėpimui naudotus parametrus. Kadangi žinome, kurių failų klasteriai slepia duomenis, išrenkame juos į atskirą masyvą ir žiūrime, kuriam failui priklauso pirmasis masyvo elementas. Tuomet pagal (3) formulę paskaičiuojame pirmojo bitų bloko reikšmę $B_1 = N_1 - B_0 \pmod p = 1-0 \pmod 2 = 1$. Kitų blokų reikšmės paskaičiuojamos analogiškai: $B_2 = N_2 - B_1 \pmod p = 1-1 \pmod 2 = 0$, $B_3 = 0-0 \pmod 2 = 0$, $B_4 = 0$, $B_5 = 0$, $B_6 = 1$, $B_7 = 1$, $B_8 = 1$, $B_9 = 0$. Bitų blokus surašius į masyvą, gauname nuskaitytą pranešimą $M = [1,0,0,0,0,1,1,1,0]$.

2.3.2. Slėpimas naudojant 4 dengiančius failus

Norėdami naudoti keturis dengiančius failus, parenkame atitinkamas reikšmes: $m=2$, $p = 2^m = 4$. Imame dengiančius failus „A.txt“, „B.txt“, „C.txt“ ir „D.txt“, jų indeksai masyve bus atitinkamai 0, 1, 2, 3. Kadangi $m=2$, t. y. slėpsime po du bitus vienu dengiančių failų klasteriu, tai žinutės M blokas B_i yra lygus dviem žinutės bitams. Padalinę žinutę į blokus ir laikydami juos natūraliaisiais skaičiais, gauname $M = [2,0,1,3,0]$.

Pagal (2) formulę paskaičiuojame, kurio failo klasterį rašysime pirmą: $N_1 := B_0 + B_1 \pmod p = 0+2 \pmod 4 = 2$. Į pirmą laisvą disko klasterį įrašome failo su antruoju indeksu („C.txt“) pirmąjį klasterį ir algoritmą vykdome toliau. Analogiškai $N_2 = 2+0 \pmod 4 = 2$, todėl į antrąjį nuo disko pradžios laisvą klasterį rašysime antrojo failo antrąjį klasterį. Nesunku apskaičiuoti, kad $N_3 = 1$, $N_4 = 0$ ir $N_5 = 3$. Paslėpus visus bitų blokus, pabaigiame rašyti dengiančius failus bet kokia tvarka (arba likusių jų dalių visai neberašome į diską).

13 pav. pateiktoje schemoje matoma kaip atrodo FAT struktūra, paslėpus duotąją žinutę.



Įrašų katalogas		
Pavad.	*Pradžia*	*Dydis*
...	...	...
C.txt	256241	?
B.txt	526246	?
A.txt	526247	?
D.txt	526248	?
...	...	...

FAT struktūra		
Klasteris	*Kitas*	*Failas*
...	...	...
526241	526243	C.txt
526242	526244	failas1.txt
526243	526249	C.txt
526244	526245	failas1.txt
526245	526250	failas1.txt
526246	?	B.txt
526247	526254	A.txt
526248	526251	D.txt
526249	EOF	C.txt
526250	526252	failas1.txt
526251	526253	D.txt
526252	EOF	failas1.txt
526253	EOF	D.txt
526254	?	A.txt
...	...	...

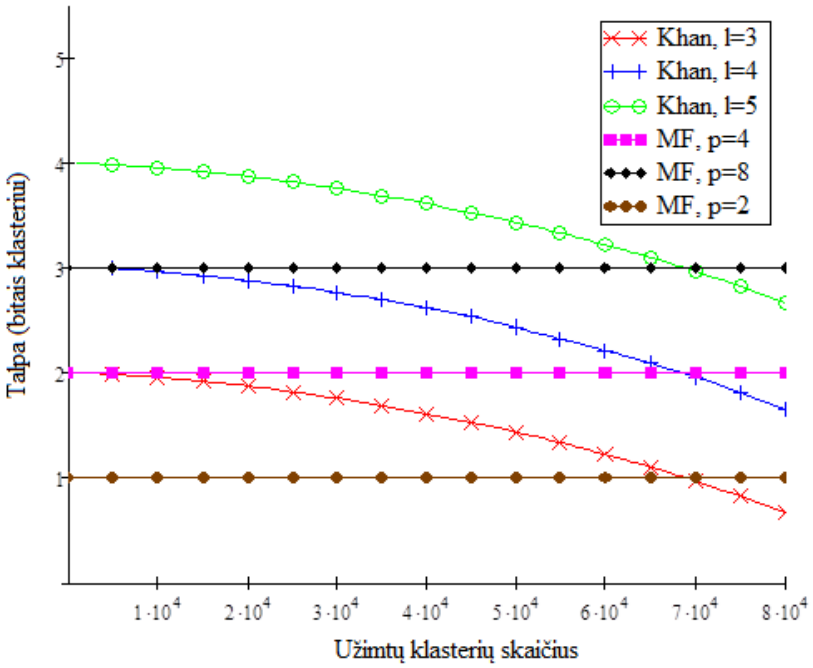
13 pav. FAT struktūra po duomenų paslėpimo operacijos

Norėdami nuskaityti slaptą pranešimą, turime žinoti paslėpimui naudotus parametrus. Kadangi žinome, kurių failų klasteriai slepia duomenis, išrenkame juos į atskirą masyvą ir žiūrime, kuriam failui priklauso pirmasis masyvo elementas. Tuomet pagal (3) formulę paskaičiuojame pirmojo bitų bloko reikšmę $B_1 = N_1 - B_0 \pmod p = 2 - 0 \pmod 4 = 2$. Kitų blokų reikšmės paskaičiuojamos analogiškai: $B_2 = N_2 - B_1 \pmod p = 2 - 2 \pmod 4 = 0$, $B_3 = 1 - 0 \pmod 4 = 1$, $B_4 = 3$, $B_5 = 0$. Bitų blokus surašome į masyvą, $M = [2,0,1,3,0]$, o juos pavertę į bitus ir numetę nereikalingą bitą, gauname nuskaitytą pranešimą $M = [1,0,0,0,0,1,1,1,0]$.

2.4. Metodo privalumai

Žemiau pateikiami siūlomo metodo privalumai, lyginant su 1.3 skyriuje apžvelgtais jau esamais metodais:

1. Galimas pilnas disko užpildymas. Apžvelgti metodai slaptai informacijai perduoti naudoja atitinkamo dydžio tarpus tarp klasterių. Dėl šios priežasties neįmanoma pilnai užpildyti disko dengiančiais duomenimis. Mūsų siūlomas metodas šio trūkumo neturi.
2. Didelė metodo talpa, naudojant daugiau dengiančių failų. [11] straipsnyje aprašyto tyrimo metu nustatyta, kad dažniausiai pasitaikantys tarpai tarp klasterių yra 8, 16 ir 32 blokų dydžio. Tuo tarpu [14] straipsnyje teigiama, kad, norint nesukelti disko tyrėjo įtarimo, reikėtų naudoti 6 – 8 klasterių tarpus. Įvertinus šias sąlygas, galima palyginti pastarajame straipsnyje aprašyto antrojo (efektyvesnio) metodo talpą su siūlomo metodo talpa. Pirmosios trys 14 pav. kreivės (pažymėtos „Khan“) atspindi [14] straipsnyje aprašyto metodo talpą. Parametras *l* reiškia bitus klasteriui. Tuo atveju, kai norime, kad tarpai tarp klasterių neviršytų 32 klasterių, turime tris tinkamas *l* reikšmes (3, 4 ir 5). Kitus parametrus metodui imame tokius, kokius pateikia autoriai [14] 6 pav.. Kitos trys kreivės (pažymėtos „MF“) atspindi siūlomo metodo talpą, naudojant atitinkamai 4, 8 ir 2 dengiančius failus.



14 pav. Duomenų paslėpimo metodų talpos palyginimas

3. Papildoma duomenų sauga. Norint atgaminti paslėptus duomenis, būtina ne tik žinoti kurie failai juos slepia, bet ir tų failų eilės tvarką, kuri buvo nurodyta informaciją paslepiant. Tai suteikia papildomą saugumą paslėptiems duomenims.
4. Paprastumas. Kadangi naudojant siūlomą metodą dengiančių failų klasterių rašymo tvarka į diską visiškai nepriklauso nuo diske jau egzistuojančios informacijos, nereikia spręsti kolizijų problemos, kas labai supaprastina metodą.

2.5. Metodo silpnosios pusės

Žemiau pateikiami siūlomo metodo trūkumai, lyginant su 1.3 skyriuje apžvelgtais jau esamais metodais:

1. Būtini bent 2 dengiantys failai. Tai gali metodo realizaciją padaryti sudėtingesne, nes programa turės vienu metu atitinkama tvarka rašyti kelių failų klasterius. Tačiau, neskaitant truputį didesnių atminties reikalavimų programai, tai nėra sunkiai įgyvendinama.
2. Sukeliamas dengiančių failų fragmentų susipynimas. Failų sistemose atskirų failų fragmentai tarpusavyje paprastai neturėtų susipinti, todėl toks fragmentų išsidėstymas disko analizės metu gali sukelti įtarimą. Tačiau praktiškai tokie atvejai pasitaiko (žr. 3.4 ir 3.5 skyrius).

3. METODO TALPOS ĮVERTINIMAS IR EKSPERIMENTINIS SAUGOS SAVYBIŲ TYRIMAS

3.1. Siūlomo metodo talpa

Siūlomas metodas pasižymi pastovia $C = \log_2 p$ talpa. Čia *C* yra slapto kanalo talpa matuojant bitais, paslepiamais vienu dengiančio failo klasteriu, *p* – dengiančių failų skaičius. Ši išraiška teisinga tik tuo atveju, kai nebūtina į diską pilnai įrašyti kiekvieno dengiančiojo failo. Vaizdesnė algoritmo talpos priklausomybė nuo naudojamų dengiančių failų skaičiaus pavaizduota 1 lentelėje.

1 lentelė. Metodo talpos priklausomybė nuo naudojamų dengiančių failų skaičiaus

Naudojamų dengiančių failų skaičius *f*	**Slaptų bitų skaičius viename dengiančio failo klasteryje *C***
2	1
4	2
8	3
16	4
32	5
64	6
128	7
256	8

3.2. Siūlomo metodo saugumo įvertinimas

Siūlomas slapto kanalo metodas yra atsparus žemo lygio disko analizei. Diską analizuojantis tyrėjas neras jokių papildomų duomenų „nestandartinėse“ disko vietose, visos failų sistemos struktūros bus korektiškos ir visiškai atitiks FAT specifikaciją. Vienintelis šalutinis metodo naudojimo efektas yra padidėjusi dengiančių failų fragmentacija. Tačiau fragmentacija aptinkama ir natūraliomis sąlygomis, kai failų sistema yra naudojama ilgą laiką ir joje vykdomas aktyvus darbas su failais. Taigi, norint įvertinti pasiūlyto metodo saugumą, reikėtų atsakyti į klausimą: su kokiais parametrais turėtų būti naudojamas pasiūlytasis metodas, kad sukeliama fragmentacija būtų tokia pati, kaip ir kylanti įprasto failų sistemose naudojimo atveju? Į šį klausimą atsakoma atliekant eksperimentinius tyrimus tolimesniuose skyriuose.

3.3. Failų sistemų analizės eiga

Eksperimento metu failų sistemos buvo analizuojamos šiais etapais:

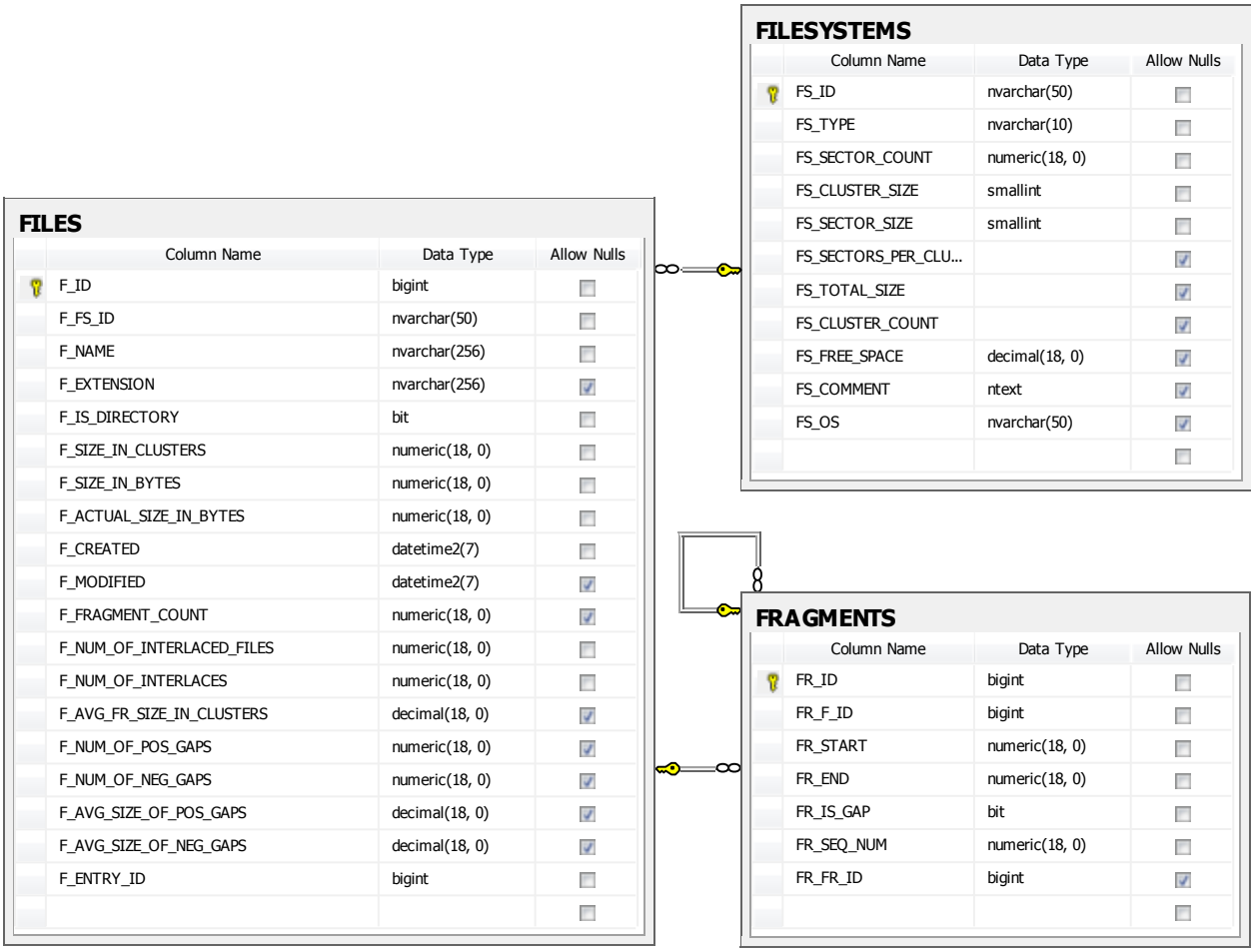
1. Sukuriamas failų sistemos disko vaizdas (naudojant Unix programą „dd“).
2. Panaudojant shell scenarijų bei programų paketo „The Sleuth Kit“ įrankius „fsstat“, „fls“ bei „istat“ buvo atliekama tarpinė disko vaizdo analizė, kurios rezultatas – 3 failai:
   - FS_vardas.fsstat – informacija apie failų sistemą.
   - FS_vardas.fls – visų failų sistemoje esančių failų sąrašas.
   - FS_vardas.istat – kiekvieno failų sistemoje esančio failo klasterių išsidėstymas.
3. Naudojant sukurtą .NET programą tarpinės analizės failai buvo nuskaitomi ir duomenys suimportuojami į suprojektuotą duomenų bazę (žr. 15 pav.).
4. Panaudojant SQL užklausas duomenų bazėje, išrenkami dominantys rezultatai.

3.3.1. Failų sistemų analizės duomenų bazė

Duomenų bazę, į kurią buvo importuojami nuskaityti duomenys, sudaro 3 lentelės: „FILESYSTEMS“, „FILES“ bei „FRAGMENTS“. Detaliau duomenų bazės schema aprašyta 3.3.1.1 bei 3.3.1.2 skyriuose.

3.3.1.1. Duomenų bazės schema

15 pav. esančioje duomenų bazės schemoje pavaizduotas ryšys tarp ją sudarančių lentelių.



15 pav. Failų sistemų analizės rezultatų duomenų bazės schema

3.3.1.2. Lentelių aprašymai

Žemiau pateikti kiekvienos lentelės aprašymai. Juose paaiškinta, kam reikalinga konkreti lentelė, kas joje saugoma.

Lentelėje „FILESYSTEMS“ saugoma informacija apie failų sistemas. Jos struktūros aprašymas pateiktas 2 lentelėje.

2 lentelė. „FILESYSTEMS“ lentelės aprašymas

	Pavadinimas	**Tipas**	**Privalomas?**	**Paaiškinimas**
PK	FS_ID	nvarchar(50)	+	Įrašo identifikatorius
	FS_TYPE	nvarchar(10)	+	Failų sistemos tipas (FAT32, NTFS, Ext3 ir t.t.)
	FS_SECTOR_COUNT	numeric(18,0)	+	Sektorių skaičius
	FS_CLUSTER_SIZE	smallint	+	Klasterio dydis baitais
	FS_SECTOR_SIZE	smallint	+	Sektoriaus dydis baitais
	FS_SECTORS_PER_CLUSTER	computed		Kiek sektorių sudaro vieną klasterį
	FS_TOTAL_SIZE	computed		Failų sistemos dydis baitais
	FS_CLUSTER_COUNT	computed		Klasterių skaičius
	FS_FREE_SPACE	decimal(18,0)		Laisva vieta procentais
	FS_COMMENT	ntext		Komentaras apie failų sistemą
	FS_OS	nvarchar(50)		OS, kurioje buvo naudojama failų sistema

Lentelėje „FILES“ saugoma informacija apie failus, esančius failų sistemose. Jos struktūros aprašymas pateiktas 3 lentelėje.

3 lentelė. „FILES“ lentelės aprašymas

	Pavadinimas	**Tipas**	**Privalomas?**	**Paaiškinimas**
PK	F_ID	bigint	+	Įrašo identifikatorius
	F_FS_ID	nvarchar(50)	+	Nuoroda į „FILESYSTEMS“ lentelę – failų sistema, kuriai priklauso failas
	F_NAME	nvarchar(256)	+	Failo vardas
	F_EXTENSION	nvarchar(256)		Failo plėtinys
	F_IS_DIRECTORY	bit	+	Failas ar aplankas (0 – failas, 1 – aplankas)
	F_SIZE_IN_CLUSTERS	numeric(18,0)	+	Užimta vieta klasteriais
	F_SIZE_IN_BYTES	numeric(18,0)	+	Užimta vieta baitais
	F_ACTUAL_SIZE_IN_BYTES	numeric(18,0)	+	Realus failo dydis baitais
	F_CREATED	datetime2(7)	+	Failo sukūrimo data
	F_MODIFIED	datetime2(7)		Failo keitimo data
	F_FRAGMENT_COUNT	numeric(18,0)		Failą sudarančių fragmentų skaičius
	F_NUM_OF_INTERLACED_FILES	numeric(18,0)	+	Skaičius failų, su kuriais šis failas yra susipynęs
	F_NUM_OF_INTERLACES	numeric(18,0)	+	Susipynimo atvejų skaičius
	F_AVG_FR_SIZE_IN_CLUSTERS	decimal(18,0)		Vidutinis failą sudarančių fragmentų dydis klasteriais
	F_NUM_OF_POS_GAPS	numeric(18,0)		Tarpų tarp fragmentų „pirmyn“ skaičius
	F_NUM_OF_NEG_GAPS	numeric(18,0)		Tarpų tarp fragmentų „atgal“ skaičius
	F_AVG_SIZE_OF_POS_GAPS	decimal(18,0)		Vidutinis tarpų tarp fragmentų „pirmyn“ dydis
	F_AVG_SIZE_OF_NEG_GAPS	decimal(18,0)		Vidutinis tarpų tarp fragmentų „atgal“ dydis
	F_ENTRY_ID	bigint	+	Failo įrašo failų lentelėje identifikatorius

Lentelėje „FRAGMENTS“ saugoma informacija apie failų fragmentus. Jos aprašymas struktūros pateiktas 4 lentelėje.

4 lentelė. „FRAGMENTS“ lentelės aprašymas

	Pavadinimas	**Tipas**	**Privalomas?**	**Paaiškinimas**
PK	FR_ID	bigint	+	Įrašo identifikatorius
	FR_F_ID	bigint	+	Nuoroda į „FILES“ lentelę – failas, kuriam priklauso šis fragmentas
	FR_START	numeric(18,0)	+	Pirmas fragmento sektorius
	FR_END	numeric(18,0)	+	Paskutinis fragmento sektorius
	FR_IS_GAP	bit	+	Fragmentas ar tarpas (0 – fragmentas, 1 – tarpas)
	FR_SEQ_NUM	numeric(18,0)	+	Fragm. eilės nr. failo fragmentų ir tarpų eilutėje
	FR_FR_ID	bigint		Nuoroda į „FRAGMENTS“ lentelę – kito fragmento identifikatorius

3.4. Naudotų diskinių kaupiklių analizė

Naudotų diskinių kaupiklių analizės tikslas yra ištirti realiose sąlygose naudotas diskinių kaupiklių failų sistemas ir nustatyti, ar jose aptinkami mūsų siūlomo metodo sukeliamai fragmentacijai analogiški atvejai. Taip pat rasti atitinkamus metodo konfigūracijos parametrus, su kuriais galima pasiekti mus tenkinantį metodo saugumą bei talpą.

Buvo išanalizuota 19 failų sistemų, kuriose rasta ir ištirta daugiau nei $0{,}9 \cdot 10^6$ failų ir daugiau nei $1{,}1 \cdot 10^6$ fragmentų.

3.4.1. Rezultatai

3.4.1.1. Failų fragmentavimosi statistika

Išanalizuotose failų sistemose vidutiniškai 5,49% failų buvo fragmentuoti. Fragmentavimosi statistika matoma žemiau esančiose lentelėse bei grafikuose.

5 lentelė. Fragmentavimosi priklausomybė nuo failų sistemos tipo

Failų sistemos tipas	Fragmentuotų failų, %
FAT32	6,327279305194
NTFS	4,694333341825

Lentelėje matome, kad FAT32 tipo failų sistemose fragmentuoti failai yra aptinkami dažniau, negu naujesnėje NTFS.

6 lentelė. Fragmentavimosi priklausomybė nuo operacinės sistemos, kurioje FS buvo naudota, tipo

OS tipas	Fragmentuotų failų, %
Windows 98	8,886615128088
Windows XP	6,698864881448
Windows Vista	2,083285856445

Aukščiau pateikta lentelė atspindi failų fragmentavimosi priklausomybę, nuo naudojamos OS. Matome, kad senesnės operacinės sistemos šiuo atžvilgiu tvarkosi prasčiau.

7 lentelė. Labiausiai fragmentuoti failų tipai

Failo plėtinys	Vidutinis failų skaičius failų sistemoje	Vidutinis failo dydis, B	Vidutinis fragmentų skaičius faile
zip	14,57	430428102,93	11,25
pak	6,30	131956893,28	10,39
ilg	1,03	596339,79	8,55
sqlite	9,22	18673624,27	8,47
lo_	1,08	65728,93	6,90
msp	21,97	15210570,72	5,29
msi	13,59	8422140,52	5,12
cmp	1,27	2414663,23	4,89
log	113,49	755514,30	4,55
cab	31,00	12625330,84	4,35
000	8,32	711679,20	4,15

Aukščiau esančioje lentelėje matome, kad labiausiai fragmentuojasi archyvo, žurnalo, duomenų bazės tipų failai. Taip matome, kad tokių tipų failai aptinkami failų sistemose gana dažnai.

Turint galvoje, kad mūsų siūlomas metodas sukelia žymią fragmentaciją, o jo saugumas yra pagrįstas slaptumu turėtume jį taikyti tokiose aplinkose, kur jo naudojimas sukeltų mažiausiai įtarimo. Pagal atliktą diskinių kaupiklių analizę matome, kad tam labiausiai tiktų FAT32 tipo failų sistemos, naudojamos senesnėse Windows versijose. Kaip dengiančius duomenis geriausia naudoti archyvo, žurnalinius ar duomenų bazių failus, nes jų failų sistemose yra pakankamai daug ir jie yra labiausiai linkę fragmentuotis.

3.4.1.2. Failų susipynimo statistika

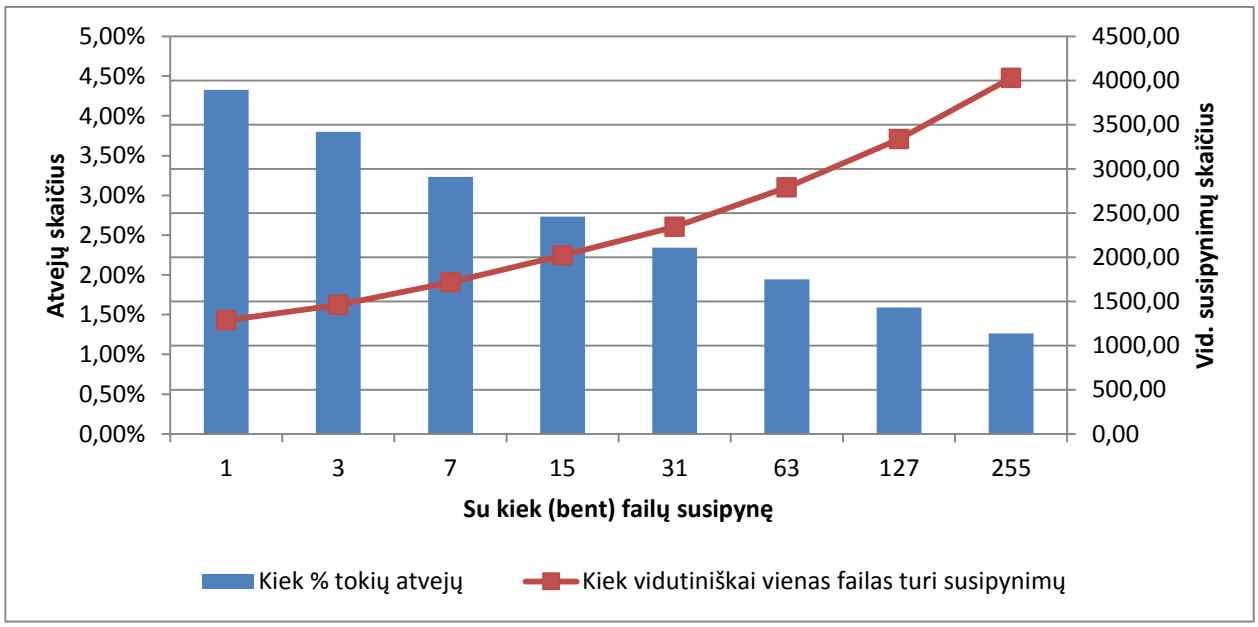
Norint suprasti šio eksperimento rezultatus, reikia formaliai apibrėžti susipynusių failų sąvoką: du failai F_1 ir F_2 laikomi tarpusavyje susipynusiais, jeigu egzistuoja bent viena situacija, kai tarp failo F_1 dviejų gretimų klasterių yra bent vienas failo F_2 klasteris ir atvirkščiai (tarp dviejų gretimų failo F_2 klasterių yra bent vienas failo F_1 klasteris). Jei ši situacija diske aptinkama keletą kartų, sakoma, kad failai susipynę atitinkamą skaičių kartų. Suprantama, failas gali būti susipynęs ne tik su vienu, bet su keliais kitais failais. Gretimais failo klasteriais laikomi iš eilės einantys to failo klasteriai (pagal failo turinį) ir turintys didėjančius klasterių numerius (dengiantiems failams ši sąlyga bus tenkinama visiems klasteriams, kitiems failams – ne visada).

Ištyrus turimas failų sistemas nustatyta, kad 4,33% visų failų yra susipynę su bent vienu kitu failu, 3,80% su bent trimis, 3,23% su bent 7, o 2,73% su bent 15 failų. Net 1,26% failų yra susipynę su 255 ar daugiau kitų failų:

8 lentelė. Susipynimų skaičiaus priklausomybė nuo susipynusių failų skaičiaus

Su kiek failų susipynę (bent)	Kiek bitų galima paslėpti vienu klasteriu	Kiek % tokių atvejų	Vid. vieno failo susipynimų skaičius	Vid. failo dydis, klast.
1	1	4,33%	1284,37	206,88
3	2	3,80%	1460,85	223,18
7	3	3,23%	1715,37	242,11
15	4	2,73%	2021,27	256,90
31	5	2,34%	2343,29	256,16
63	6	1,94%	2791,35	269,69
127	7	1,59%	3336,12	285,75
255	8	1,26%	4029,56	261,13

Grafinė priklausomybė tarp susipynusių failų skaičiaus bei susipynimo atvejų pavaizduota 16 pav.



16 pav. Susipynimų skaičiaus priklausomybė nuo susipynusių failų skaičiaus

Iš aukščiau pateiktų rezultatų matome, kad didinant dengiančių failų skaičių, kiekvienas failas susipina su vis didesniu skaičiumi kitų failų. Tai reiškia, kad didėja ne tik vienu dengiančio failo klasteriu perduodamų duomenų kiekis, bet ir bendras klasterių skaičius.

Išsamesnė sukategorizuota fragmentų susipynimo statistika matoma žemiau esančiose lentelėse. Atlikti palyginimai tarp skirtingų failų sistemų, operacinių sistemų, failų tipų, failų dydžių.

9 lentelė. Fragmentų persipynimo priklausomybė nuo failų sistemos tipo

FS tipas	Vid. failo dydis, klast.	Su kiek vid. failų susipynę	Vid. susipynimo atvejų skaičius	Vid. vieno failo susipynimo atvejų sk.
NTFS	31,45	31,68	115,20	3,64
FAT32	42,38	1,53	6,41	4,19

Aukščiau esančioje lentelėje matome, kad nepaisant to, jog NTFS tipo failų sistemose esantys failai yra gerokai mažesni, jie yra persipynę su kur kas daugiau failų, kas, savaime suprantama, sąlygoja ir atitinkamai didesnį susipynimo atvejų skaičių. Visgi vidutinis vieno failo susipynimo atvejų skaičius (gaunamas padalinant vid. susipynimo atvejų skaičių iš vid. susipynusių failų skaičiaus) didesnis FAT32 failų sistemose. Tai parodo, kad NTFS failų sistemose failų klasteriai yra labiau išsimėtę po failų sistemą, taip sukeldami susipynimą su dideliu skaičiumi failų.

10 lentelė. Fragmentų persipynimo priklausomybė nuo OS, kurioje FS buvo naudota, tipo

OS tipas	Vid. failo dydis, klast.	Su kiek vid. failų susipynę	Vid. susipynimo atvejų skaičius	Vid. vieno failo susipynimo atvejų sk.
Windows XP	39,25	19,79	70,21	3,55
Windows Vista	26,20	15,66	61,61	3,93
Windows 98	35,12	2,01	9,05	4,50
	188,91	0,01	0,03	3,00

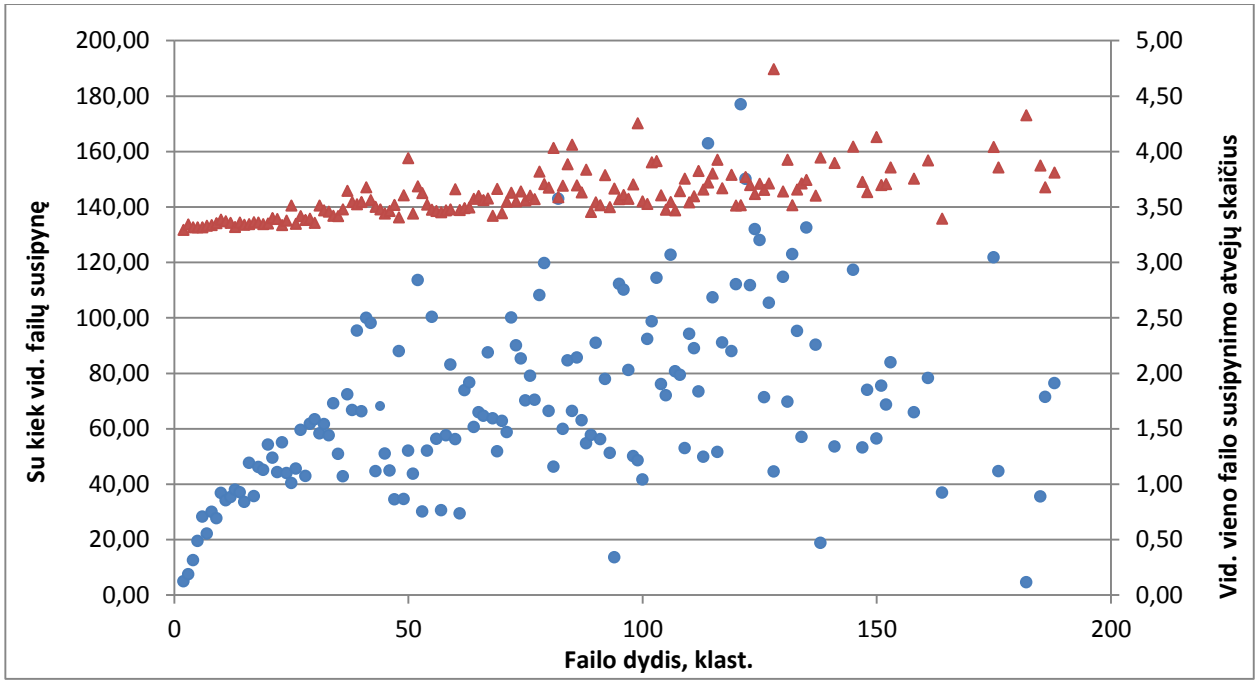
Pažvelgę į 10 lentelę matome, kad failų sistemose, naudotose Windows XP ir Windows Vista operacinėse sistemose, failai yra susipynę su daugiau failų, o vieno failo susipynimo atvejų skaičius yra didžiausias Windows 98 OS naudotose FS.

Žemiau esančioje lentelėje matome failų tipus, kurių vidutinis vieno failo susipynimo atvejų skaičius yra didžiausias. Paprastai tai yra arba dideli archyviniai failai (pak, zip, cab, msi), arba mažesni, bet dažnai papildomi failai (log, sqlite, doc). Šie failai ne tik susipynę su daug kitų failų, bet turi ir daug susipynimo atvejų su kiekvienu iš jų, kas padaro šių tipų failus puikiai tinkamais naudoti kaip dengiančius duomenis mūsų siūlomame metode.

11 lentelė. Fragmentų susipynimo priklausomybė nuo failo tipo

Failo plėtinys	Vid. failo dydis, klast.	Su kiek vid. failų susipynę	Vid. susipynimo atvejų skaičius	Vid. vieno failo susipynimo atvejų sk.
pak	391,77	6,01	49,12	8,17
lib	91,30	81,54	407,96	5,00
zip	2152,61	126,90	589,36	4,64
cab	884,87	80,78	367,79	4,55
bin	226,85	17,81	79,81	4,48
log	40,51	102,39	458,12	4,47
sqlite	673,59	156,68	699,91	4,47
json	22,51	131,35	583,79	4,44
adm	144,08	7,71	33,73	4,37
drv	14,31	5,49	23,13	4,21
doc	66,32	4,09	17,13	4,19
mfl	65,13	35,22	146,86	4,17
msi	892,87	326,31	1349,40	4,14

Žemiau pateiktame grafike matome, kaip nuo failo dydžio priklauso jo susipynimo atvejų su vienu failu skaičius bei su kiek kitų failų jis yra susipynęs. Matome, kad vidutinis vieno failo susipynimo atvejų skaičius beveik nepriklauso nuo failo dydžio. Vidutinis skaičius failų, su kuriais buvo susipynęs imamas failas, didėja augant failo dydžiui, kol pasiekiama maždaug 50 klasterių failo dydžio riba. Verta pastebėti, kad buvo imami tik failai, kurių failų sistemose buvo vidutiniškai bent po 4 ir kurie buvo susipynę bent su 3 kitais failais.



17 pav. Vid. supintų failų bei susipynimo atvejų skaičiaus priklausomybė nuo failo dydžio

Atlikus failų fragmentų persipynimo failų sistemose analizę buvo nustatyta, kad praktikoje yra nemažai atvejų, kai failų fragmentai tarpusavyje susipina. Tokie susipynimai vyksta ne tik tarp keleto failų, tačiau ir tarp daugiau nei 255, o mūsų metodu panaudojant tokį skaičių failų galima sutalpinti netgi 8 bitus duomenų viename dengiančio failo klasteryje. Susipynusių failų skaičius didesnis NTFS tipo failų sistemose, kurios buvo naudotos Windows XP operacinėse sistemose. Taip pat sužinota, kad dažniau susipynę būna didesni (bent 50 klasterių) ir dažniau atnaujinami ar papildomi failai.

3.5. Situacijų modeliavimas

Kita eksperimentinių tyrimų dalis buvo skirta nustatyti, ar įprastai dirbant su failų sistemomis (naudojant tik standartines OS funkcijas ir tiesiogiai nekeičiant žemo lygio disko ir failų sistemų struktūrų) susidaro situacijos, kai du ar daugiau failų tarpusavyje susipina daugelį kartų. Tam tikslui buvo sumodeliuotos trys situacijos Windows 7, Windows XP ir Ubuntu 11.04 operacinėse sistemose bei FAT32 ir NTFS failų sistemose.

3.5.1. Sumodeliuotos situacijos

Projekto metu buvo modeliuojamos šios situacijos:

1. Du failai buvo vienu metu kopijuojami į tą pačią failų sistemą OS priemonėmis. Šiam bandymui įgyvendinti buvo parašyti scenarijai, kurie vienu metu paleisdavo OS suteikiamas programas (Ubuntu atveju „cp“, Windows – „copy“) dviejų 100 MB dydžio failų kopijavimui.
2. Du failai vienu metu siunčiami į tą pačią failų sistemą, naudojant BitTorrent protokolo programą. Buvo surasti du nuorodas į apie 100 MB failus turintys „.torrent“ failai ir, panaudojant atitinkamas programas, siunčiami į tą pačią vietą.
3. Duomenys buvo lygiagrečiai rašomi į du failus, esančius toje pačioje failų sistemoje, papildymo būdu. Šio eksperimento įgyvendinimui sukurta programa Java kalba. Programa naudojo atskiras gijas kiekvieno failo kopijavimui, o kiekvienos gijos veikimas atitiko šį scenarijų: atidaromas failas papildymui, įrašoma 1 KB duomenų, failas uždaromas. Scenarijus vykdomas, kol failo dydis pasiekia 100 MB.

Situacijos pakartotos naudojant šias operacines sistemas:

1. Windows 7
2. Windows XP
3. Ubuntu 11.04

Situacijos pakartotos naudojant šiuos failų sistemų tipus:

1. FAT32
2. NTFS

Bandymų metu 100 MB failai buvo rašomi į 2 GB USB atmintuką. Kiekvienu atveju po atlikto duomenų įrašymo failų sistema buvo analizuojama 3.3 skyriuje aprašyta metodika.

3.5.2. Rezultatai

3.5.2.1. Lygiagretus dviejų failų kopijavimas

Atlikto bandymo rezultatai pateikti žemiau esančioje lentelėje:

12 lentelė. Lygiagretaus dviejų failų kopijavimo simuliacijos rezultatai

	Windows 7		Windows XP		Ubuntu 11.04	
	FAT32	NTFS	FAT32	NTFS	FAT32	NTFS
Failų skaičius	2	2	2	2	2	2
Vid. fragmentų skaičius faile	1	1,5	1	1	9,5	5
Su kiek vidutiniškai kitų failų susipynęs vienas failas	0	0	0	0	1	1
Vid. vieno failo susipynimų skaičius	0	0	0	0	17	8

Matome, kad su šia užduotimi Windows šeimos operacinės sistemos susitvarkė kur kas geriau – jose beveik neįvyko fragmentacija, tad ir failai nesusipynė. Ubuntu OS tiek FAT32, tiek NTFS tipo failų sistemose kopijuojamus failus sufragmentavo, o fragmentuoti failai tarpusavyje susipynė. Tai reiškia, kad Ubuntu operacinėje sistemoje vienu metu kopijuojant keletą failų į tą pačią vietą greičiausiai sukurs tarpusavyje susipynusius fragmentuotus failus.

3.5.2.2. Lygiagretus dviejų failų siuntimas naudojant „BitTorrent“ tinklą

Atlikto tyrimo rezultatai patikti 13-oje lentelėje:

13 lentelė. Lygiagretaus dviejų failų siuntimo, naudojant „BitTorrent“ tinklą, simuliacijos rezultatai

	Windows 7		Windows XP		Ubuntu 11.04	
	FAT32	NTFS	FAT32	NTFS	FAT32	NTFS
Failų skaičius	2	2	2	2	2	2
Vid. fragmentų skaičius faile	1	1	1	1	1	1
Su kiek vidutiniškai kitų failų susipynęs vienas failas	0	0	0	0	0	0
Vid. vieno failo persipynimų skaičius	0	0	0	0	0	0

„BitTorrent“ tinklu siunčiami failai būna suskaidyti į daug mažų dalių, todėl buvo nuspręsta patikrinti, ar lygiagrečiai tuo pačiu metu juos siunčiant nebus tos dalys tarpusavyje supintos. Iš bandymo rezultatų matome, kad visais atvejais failai buvo parsiųsti tvarkingai, nesufragmentuoti.

3.5.2.3. Lygiagretus kelių failų rašymas papildymo būdu

Eksperimentui Java kalba buvo sukurta programa, kuri, naudodama nurodytą skaičių gijų (angl. threads), papildymo būdu rašė po 100 MB duomenų į kiekvieną failą (viena gija – vienas failas) dalimis po 1 kB. Rašymo procesas:

Kartoti, kol bus įrašyta 100 MB:

1. Failas atidaromas papildymui (jei neegzistuoja – sukuriamas naujas).
2. Įrašoma 1 kB atsitiktinių duomenų.
3. Failas uždaromas.

Eksperimento rezultatai pateikti žemiau esančioje lentelėje:

14 lentelė. Lygiagretaus kelių failų rašymo papildymo būdu simuliacijos rezultatai

	Windows 7				Windows XP		Ubuntu 11.04	
	FAT32		NTFS		FAT32	NTFS	FAT32	NTFS
Failų skaičius	2	4	2	4	2	2	2	2
Vid. fragmentų skaičius faile	25579,5	11409,75	19869	25520,75	16457,5	24675	24774	7
Su kiek vidutiniškai kitų failų susipynęs vienas failas	1	3	1	3	1	1	1	1
Vid. vieno failo persipynimų skaičius	51157	44228,5	29684	148634	14812	49341	49546	12

Matome, kad visais atvejais, išskyrus NTFS failų sistemą Ubuntu operacinėje sistemoje, toks failų rašymas sukėlė labai didelę fragmentaciją. Visose OS FAT32 failų sistemose lygiagretus failų rašymas papildymo būdu sukelia labai aukštą failų fragmentaciją ir tuo pačiu – fragmentų susipynimą tarpusavyje. Atlikus eksperimentą su 4, o ne 2 failais, kiekvieno failo vidutinis fragmentų bei susipynimų skaičius truputį sumažėja, tačiau vis dar išlieka labai aukštas.

Išskirtinis Ubuntu 11.04 OS ir NTFS failų sistemos atvejis tik įrodo, kad failų fragmentacija smarkiai priklauso nuo operacinės sistemos naudojamų tvarkyklių tam tikrai failų sistemai. T.y. failų fragmentacija lemia ne failų ar operacinė sistema, o jų pora.

3.6. Išvados

1. Ištirta 19 realiai naudotų failų sistemų ir nustatyta, kad siūlomo metodo sukeliama fragmentacija ir failų fragmentų persipynimas pasitaiko ir realiose sistemose. Taip pat pateikti pasiūlymai, kaip padidinti metodo saugą, atsižvelgiant į OS ir FS tipus, naudojamų dengiančių failų tipus bei dydį.
2. Atliktas trijų situacijų modeliavimas, bandant sukelti failų fragmentų persipynimą. Nustatyta, kad failų fragmentai gali tarpusavyje susipinti Ubuntu 11.04 OS lygiagrečiai kopijuojant keletą failų bei lygiagrečiai papildymo būdu rašant duomenis į kelis failus vienu metu (išskyrus Ubuntu 11.04 operacinės sistemos su NTFS failų sistema atvejį).
3. Remiantis pirmaja ir antraja išvadomis galima teigti, kad siūlomas slapto kanalo metodas pasižymi dvigubo pagrįsto išsigynimo savybe.

4. EKSPERIMENTINIS METODO STABILUMO IR LAIKINIŲ CHARAKTERISTIKŲ TYRIMAS

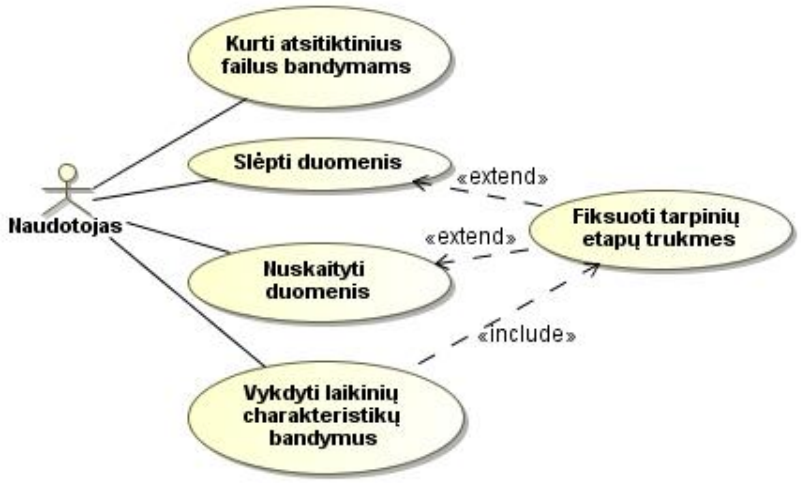
Tyrimo metu atlikta siūlomo metodo kokybinė ir kiekybinė analizė. Kokybinio tyrimo metu nustatytas metodo atsparumas dengiančių failų modifikavimui, failų sistemos defragmentacijai. Kiekybinės analizės tikslas buvo išmatuoti metodo laikines charakteristikas, nustatyti galimus optimizavimo būdus.

Analizei atlikti buvo sukurta eksperimentinė programa, naudojanti pasiūlytą slapto kanalo organizavimo klasterinėse failų sistemose metodą. 4.1 skyriuje pateikiamas jos aprašymas.

4.1. Eksperimentinės programos aprašymas

4.1.1. Panaudojimo atvejai

Žemiau (18 pav.) pateikiama diagrama, atspindinti kuriamos eksperimentinės programos funkcijas.



18 pav. Eksperimeninės programos panaudojimo atvejų diagrama

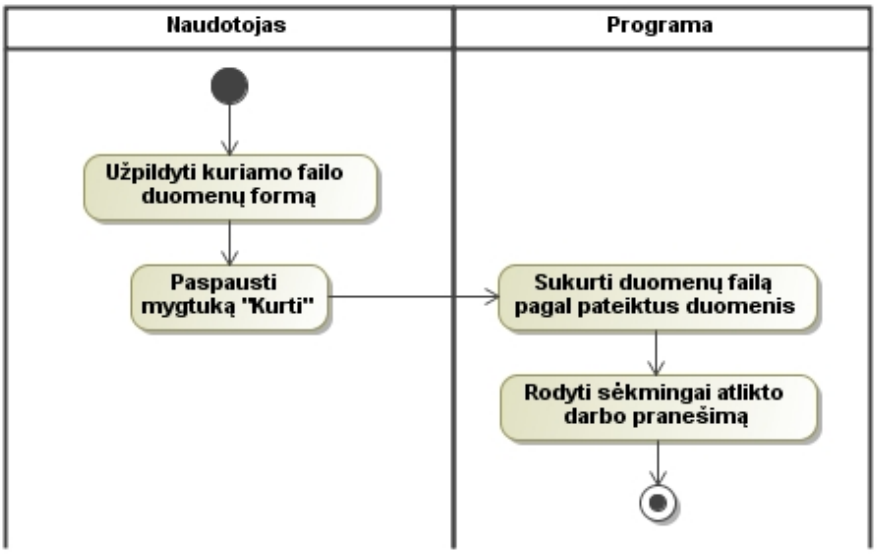
4.1.2. Panaudojimo atvejų specifikacija

4.1.2.1. Panaudojimo atvejo „Kurti atsitiktinius failus bandymams“ specifikacija

Panaudojimo atvejis aprašytas lentele (15 lentelė) bei veiklos diagrama (19 pav.).

15 lentelė. Panaudojimo atvejo „Kurti atsitiktinius failus bandymams“ specifikacija

PA „Kurti atsitiktinius failus bandymams“		
Tikslas: sukurti atsitiktinį failą bandymams.		
Aprašymas: užpildęs duomenų formą aktorius inicijuoja failo kūrimo procedūrą mygtuko paspaudimu. Programa pagal pateiktus parametrus sukuria failą ir baigusi darbą pateikia sėkmės pranešimą.		
Prieš sąlyga:		1. Failų sistemoje pakanka vietos nurodyto dydžio failui.
Sužadinimo sąlyga:		Aktorius atsidaro atsitiktinio failo kūrimo kortelę programoje.
Aktorius:		Naudotojas
Susiję PA	**Išplečiantys PA**	
	Apimami PA	
	Specializuoti PA	
Pagrindinis įvykių srautas		**Programos reakcija**
1. Užpildyti kuriamo failo duomenų formą.		
2. Paspausti mygtuką „Kurti“.		2.1. Sukurti duomenų failą pagal pateiktus duomenis. 2.2. Rodyti sėkmingai atlikto darbo pranešimą.
Po sąlygos:		Sukurtas failas bandymams.



19 pav. Panaudojimo atvejo „Kurti atsitiktinius failus bandymams“ veiklos diagrama

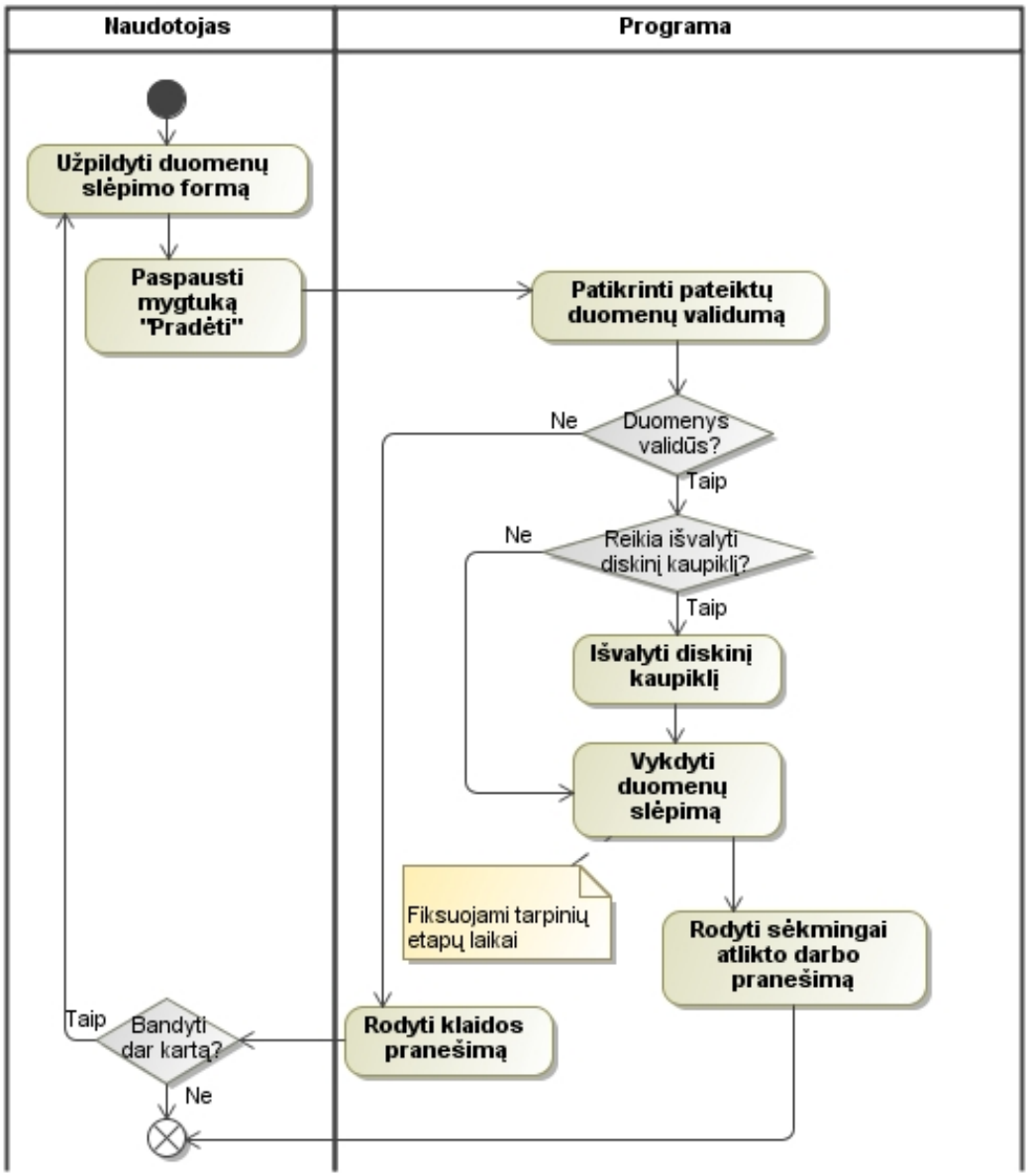
Kuriamo failo duomenų formos aprašymas pateiktas skyriuje 4.1.3.1 Kortelė „Atsitiktinių duomenų failų kūrimas“.

4.1.2.2. Panaudojimo atvejo „Slėpti duomenis“ specifikacija

Panaudojimo atvejis aprašytas lentele (16 lentelė) ir veiklos diagrama (20 pav.).

16 lentelė. Panaudojimo atvejo „Slėpti duomenis“ specifikacija

PA „Slėpti duomenis“		
Tikslas: panaudojant siūlomą metodą, paslėpti pateiktus duomenis klasterinėje failų sistemoje, panaudojant nurodytus dengiančius failus.		
Aprašymas: užpildęs duomenų formą aktorius inicijuoja slėpimo procedūrą mygtuko paspaudimu. Programa validuoja pateiktus duomenis ir, neteisingų duomenų atveju, parodo klaidos pranešimą ir pasiūlo bandyti vėl. Kitu atveju, pagal pateiktus duomenis atliekamas duomenų paslėpimas ir, baigus darbą, parodomas sėkmės pranešimas.		
Prieš sąlyga:		1. Turimas failas su slepiamais duomenimis. 2. Turimi dengiantys failai.
Sužadinimo sąlyga:		Aktorius atsidaro failo slėpimo kortelę programoje.
Aktorius:		Naudotojas
Susiję PA	**Išplečiantys PA**	1. Fiksuoti tarpinių etapų trukmes.
	Apimami PA	
	Specializuoti PA	
Pagrindinis įvykių srautas		**Sistemos reakcija**
1. Užpildyti duomenų slėpimo formą.		
2. Paspausti mygtuką „Pradėti“.		2.1. Patikrinti pateiktų duomenų validumą. 2.2. Jei reikia, išvalyti diskinį kaupiklį. 2.3. Vykdyti duomenų slėpimą. 2.4. Rodyti sėkmingai atlikto darbo pranešimą.
Po sąlygos:		Failų sistemoje, panaudojant nurodytus dengiančius failus, paslėptas nurodytas dokumentas.
Alternatyvūs scenarijai		
1.a. Pateikti duomenys yra nevalidūs.		1.1.a. Rodyti klaidos pranešimą. 1.2.a. Pasiūlyti bandyti dar kartą.



20 pav. Panaudojimo atvejo „Slėpti duomenis“ veiklos diagrama

Duomenų slėpimo formos aprašymas pateiktas skyriuje 4.1.3.2 Kortelė „Duomenų slėpimas“.

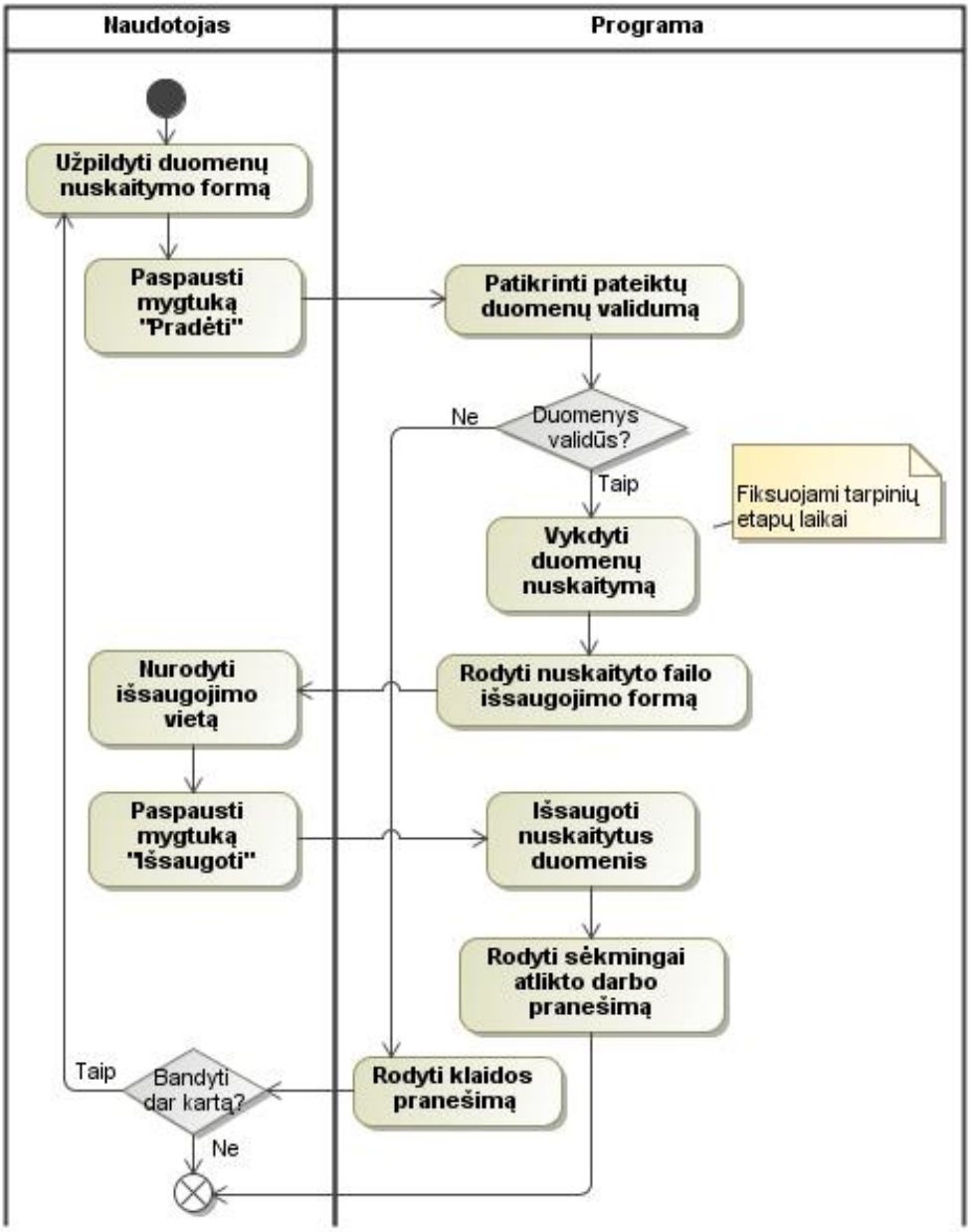
4.1.2.3. Panaudojimo atvejo „Nuskaityti duomenis“ specifikacija

Panaudojimo atvejis aprašytas lentele (17 lentelė) ir veiklos diagrama (21 pav.).

17 lentelė. Panaudojimo atvejo „Nuskaityti duomenis“ specifikacija

PA „Nuskaityti duomenis“	
Tikslas: nuskaityti duomenis, paslėptus panaudojant siūlomą metodą.	
Aprašymas: užpildęs duomenų formą aktorius inicijuoja atstatymo procedūrą mygtuko paspaudimu. Programa validuoja pateiktus duomenis ir, neteisingų duomenų atveju, parodo klaidos pranešimą ir pasiūlo bandyti vėl. Kitu atveju, pagal pateiktus duomenis atliekamas duomenų nuskaitymas ir, baigus darbą, aktoriui pateikiamas duomenų išsaugojimo į failą dialogas. Aktorius nurodo, kur išsaugoti failą, programa jį išsaugo ir parodo sėkmės pranešimą.	
Prieš sąlyga:	1. Nurodytoje failų sistemoje, panaudojant siūlomą metodą, paslėpti duomenys. 2. Aktoriui žinomi visi slaptą raktą sudarantys duomenys.
Sužadinimo sąlyga:	Aktorius atsidaro paslėptų duomenų nuskaitymo kortelę programoje.
Aktorius:	Naudotojas

Susiję PA	**Išplečiantys PA**	1. Fiksuoti tarpinių etapų trukmes.
	Apimami PA	
	Specializuoti PA	
Pagrindinis įvykių srautas		**Sistemos reakcija**
1. Užpildyti duomenų nuskaitymo formą.		
2. Paspausti mygtuką „Pradėti“.		2.1. Patikrinti pateiktų duomenų validumą. 2.2. Vykdyti duomenų nuskaitymą. 2.3. Rodyti nuskaityto failo išsaugojimo formą.
3. Nurodyti išsaugojimo vietą.		
4. Paspausti mygtuką „Išsaugoti“.		4.1. Išsaugoti nuskaitytus duomenis. 4.2. Rodyti sėkmingai atlikto darbo pranešimą.
Po sąlygos:		Nurodytame faile išsaugoti prieš tai failų sistemoje paslėpti duomenys.
Alternatyvūs scenarijai		
1.a. Pateikti duomenys yra nevalidūs.		1.1.a. Rodyti klaidos pranešimą. 1.2.a. Pasiūlyti bandyti dar kartą.



21 pav. Panaudojimo atvejo „Nuskaityti duomenis“ veiklos diagrama

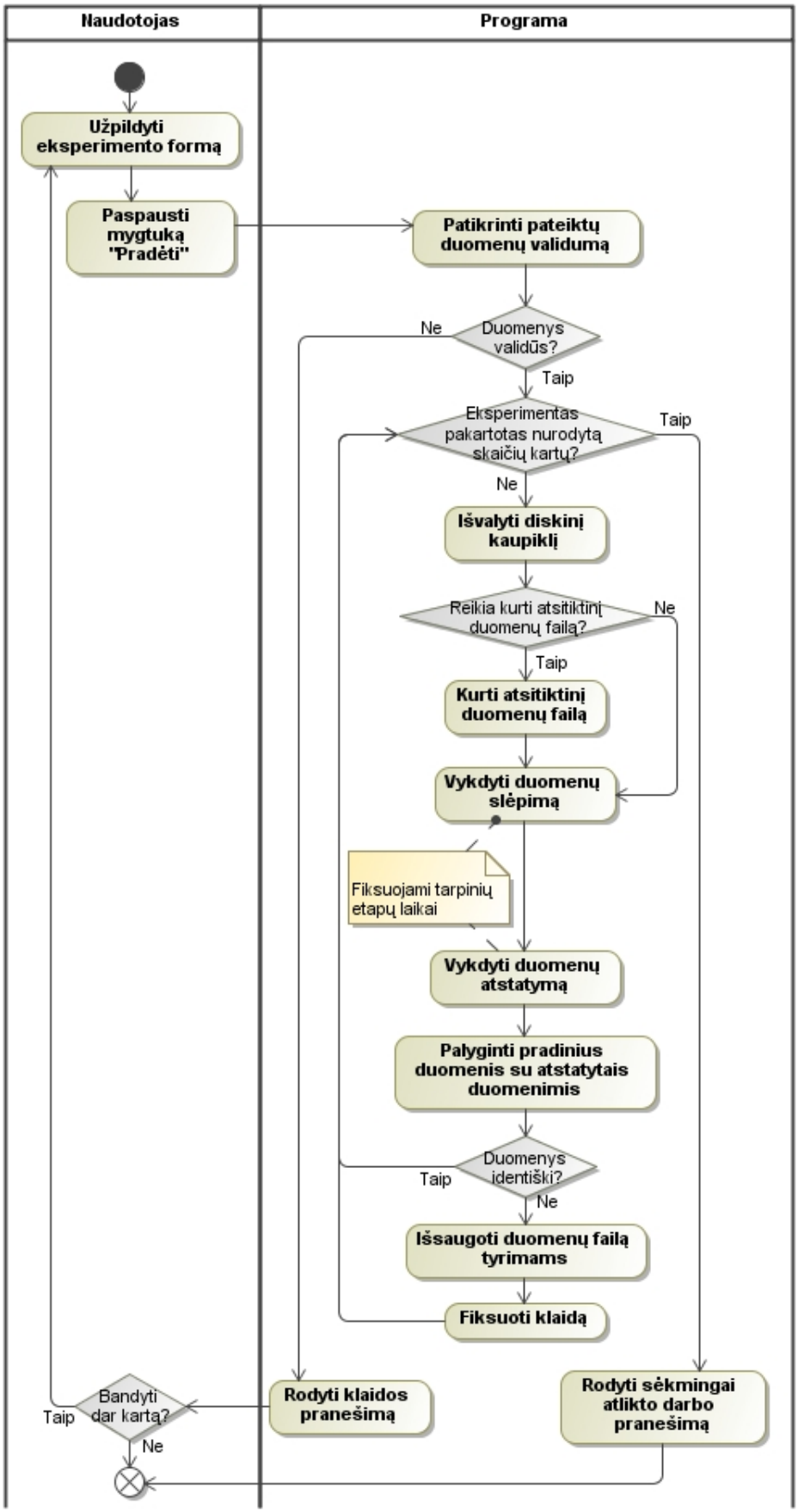
Duomenų nuskaitymo formos aprašymas pateiktas skyriuje 4.1.3.3 Kortelė „Duomenų nuskaitymas“.

4.1.2.4. Panaudojimo atvejo „Vykdyti laikinių charakteristikų bandymus“ specifikacija

Panaudojimo atvejis aprašytas lentele (18 lentelė) ir veiklos diagrama (22 pav.).

18 lentelė. Panaudojimo atvejo „Vykdyti laikinių charakteristikų bandymus“ specifikacija

PA „Vykdyti laikinių charakteristikų bandymus“		
Tikslas: daug kartų kartoti duomenų slėpimą bei jų nuskaitymą tuo pačiu metu matuojant tarpinių etapų trukmes bei patikrinant duomenų ir rezultatų identiškumą.		
Aprašymas: užpildęs eksperimento duomenų formą aktorius inicijuoja eksperimento procedūrą mygtuko paspaudimu. Programa validuoja pateiktus duomenis ir, neteisingų duomenų atveju, parodo klaidos pranešimą ir pasiūlo bandyti vėl. Kitu atveju, pagal pateiktus duomenis daug kartų atliekama: išvalomas diskinis kaupiklis, jei reikia, sukuriamas atsitiktinis duomenų failas, vykdomas duomenų paslėpimas, vykdomas duomenų nuskaitymas, duomenys bei gauti rezultatai palyginami tarpusavyje. Jei rezultatai nesutampa su duomenimis, duomenų failas išsaugomas tolimesniems tyrimams, o žurnalo dokumente užfiksuojama klaida. Baigusi bandymus programa parodo sėkmės pranešimą.		
Prieš sąlyga:		1. Aktorius turi paruošęs FAT32 failų sistemą, kuri yra pakankamai talpi ir kurią galima daug kartų išvalyti.
Sužadinimo sąlyga:		Aktorius atsidaro laikinių charakteristikų eksperimento kortelę programoje.
Aktorius:		Naudotojas
Susiję PA	**Išplečiantys PA**	
	Apimami PA	1. Fiksuoti tarpinių etapų trukmes.
	Specializuoti PA	
Pagrindinis įvykių srautas		**Sistemos reakcija**
1. Užpildyti eksperimento formą.		
2. Paspausti mygtuką „Pradėti“.		2.1. Patikrinti pateiktų duomenų validumą. 2.2. Nurodytą skaičių kartų kartoti: 2.2.1. Išvalyti diskinį kaupiklį. 2.2.2. Jei reikia, sukurti atsitiktinių duomenų failą. 2.2.3. Vykdyti duomenų paslėpimą. 2.2.4. Vykdyti duomenų nuskaitymą. 2.2.5. Palyginti pradinius duomenis su nuskaitytais duomenimis. 2.2.6. Jei duomenys ir rezultatai nesutampa, išsaugoti duomenų failą ir žurnalo dokumente užfiksuoti klaidą. 2.3. Rodyti sėkmingai atlikto darbo pranešimą.
Po sąlygos:		Nurodytame faile išsaugoti prieš tai failų sistemoje paslėpti duomenys.
Alternatyvūs scenarijai		
1.a. Pateikti duomenys yra nevalidūs.		1.1.a. Rodyti klaidos pranešimą. 1.2.a. Pasiūlyti bandyti dar kartą.



22 pav. Panaudojimo atvejo „Vykdyti laikinių charakteristikų bandymus“ veiklos diagrama

Kuriamo failo duomenų formos aprašymas pateiktas skyriuje 4.1.3.4 Kortelė „Laikinių charakteristikų eksperimentas“.

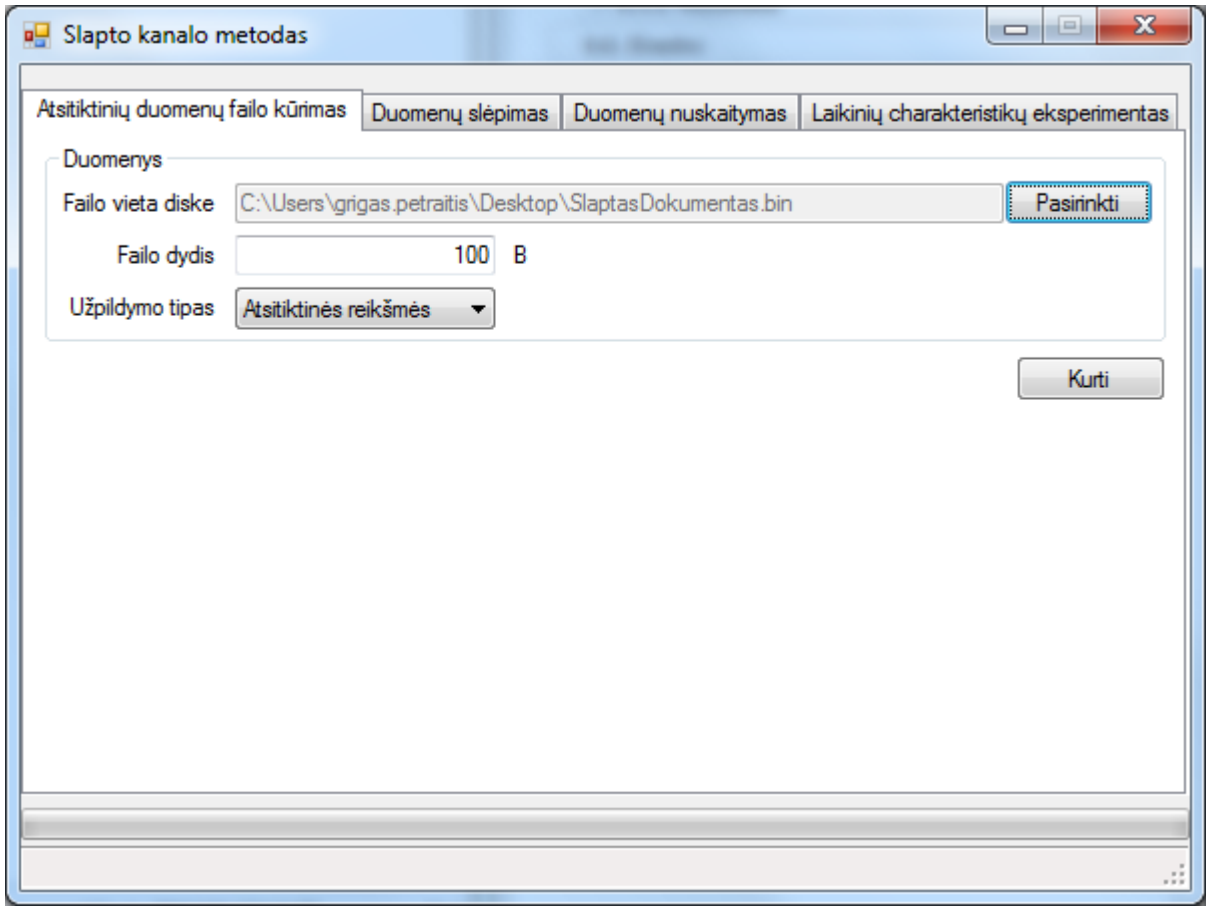
4.1.3. Varotojo grafinės sąsajos prototipas

Vartotojo grafinę sąsają sudaro 4 kortelės:

1. Atsitikitinių duomenų failo kūrimas.
2. Duomenų slėpimas.
3. Duomenų nuskaitymas.
4. Laikinių charakteristikų eksperimentas.

Taip pat po kortelėmis yra progreso bei būsenos juostos.

4.1.3.1. Kortelė „Atsitiktinių duomenų failų kūrimas“

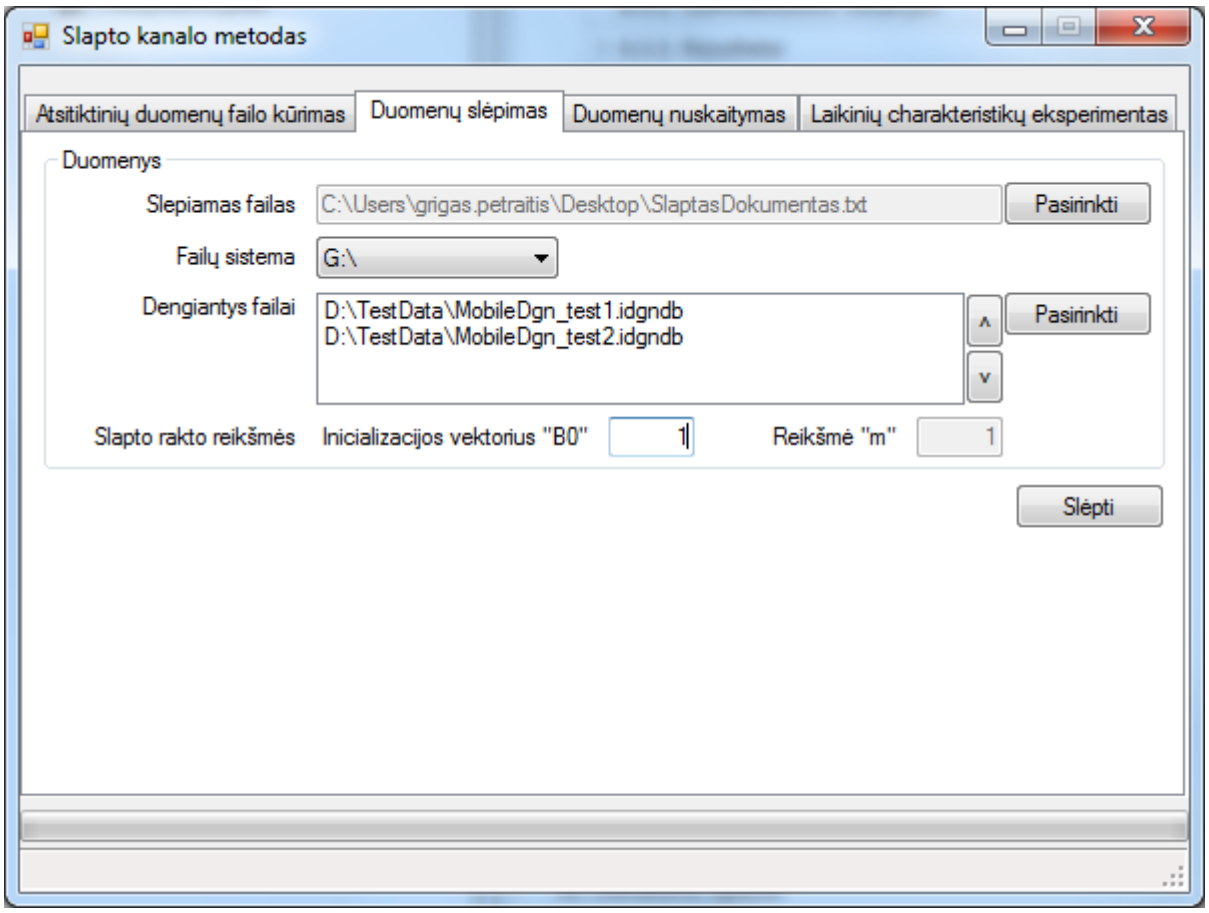


23 pav. Kortelės „Atsitiktinių duomenų failų kūrimas“ grafinės sąsajos prototipas

Kuriamo atsitiktinio failo duomenų formą sudaro:

- Failo vieta diske. Pateikiamas kelias iki kuriamo failo. Failas nurodomas paspaudus mygtuką „Pasirinkti“.
- Failo dydis. Kuriamo failo dydis baitais.
- Failo užpildymo tipas. Kokiais duomenimis bus užpildomas generuojamas failas. Galimi 2 variantai:
    - nulinės reikšmės – tuščias failas, užpildytas nulinėmis reikšmėmis;
    - atsitiktinės reikšmės – failas užpildomas atsitiktinių skaičių generavimo metodu gautomis reikšmėmis.

4.1.3.2. Kortelė „Duomenų slėpimas“

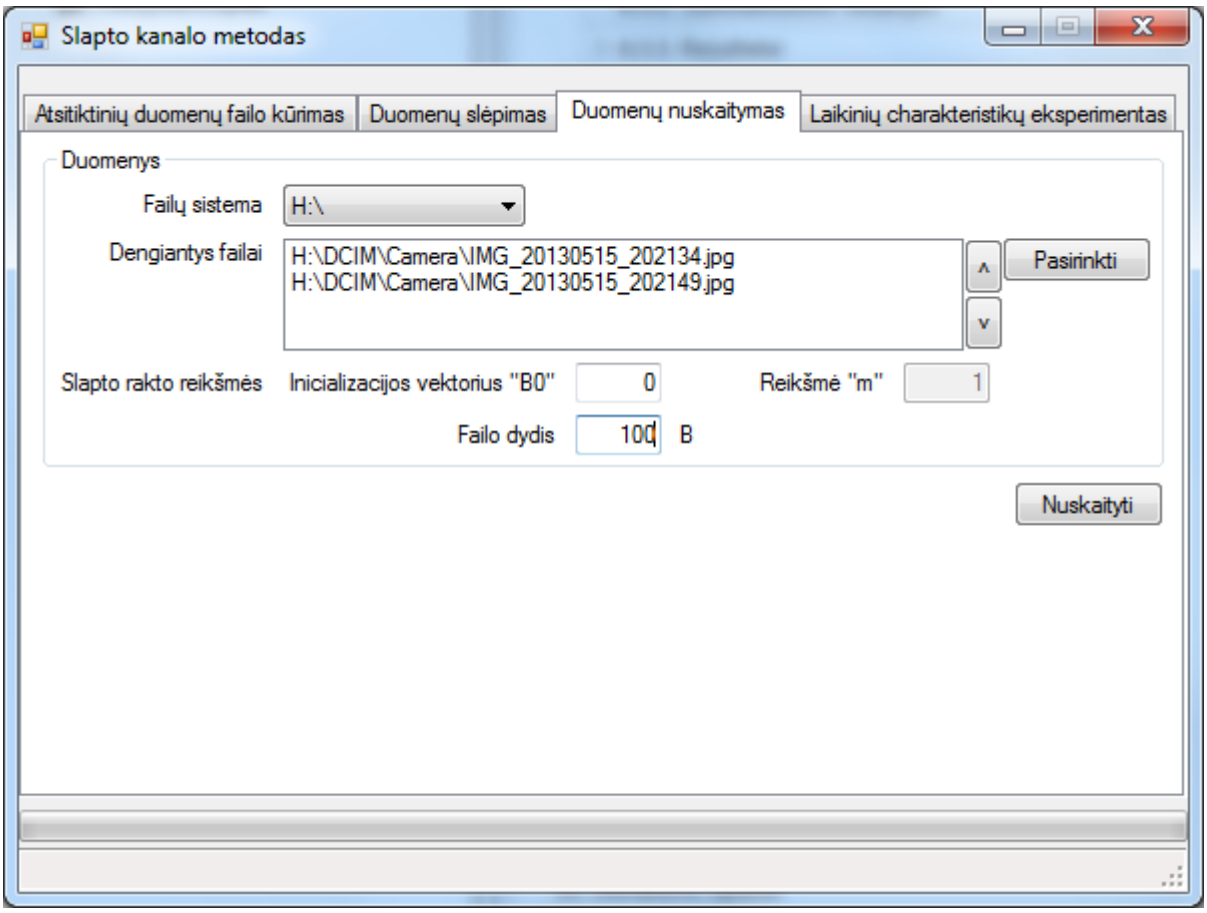


24 pav. Kortelės „Duomenų slėpimas“ grafinės sąsajos prototipas

Duomenų slėpimo formą sudaro:

- Slepiamas failas. Pateikiamas kelias iki slepiamo failo. Failas pasirenkamas paspaudus mygtuką „Pasirinkti“.
- Failų sistema. Failų sistema, kurioje bus slepiamas failas. Pateikiamas visų išorinėse (ne sisteminėse) laikmenose esančių FAT32 failų sistemų sąrašas.
- Dengiantys failai. Pateikiami keliai iki visų pasirinktų dengiančių failų. Failai pasirenkami mygtuko „Pasirinkti“ pagalba“. Sąrašo dešinėje esančiais mygtukais galima keisti failų pateikimo algoritmui tvarką.
- Inicializacijos vektorius B_0. Slapto rakto dalis, natūralusis skaičius.
- Reikšmė m. Neredaguojamas. Pagal pasirinktų failų skaičių automatiškai paskaičiuojama slapto rakto dalis.

4.1.3.3. Kortelė „Duomenų nuskaitymas“

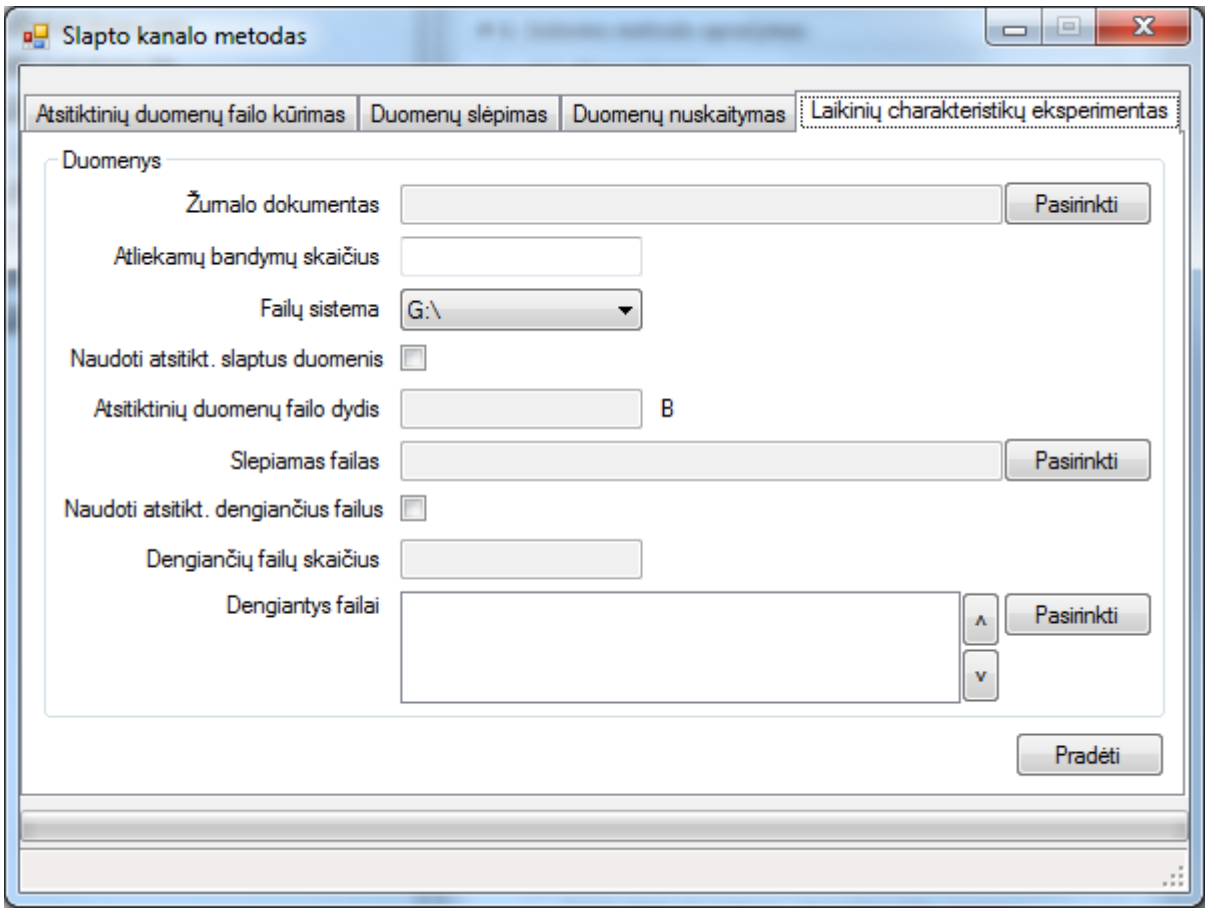


25 pav. Kortelės „Duomenų nuskaitymas“ grafinės sąsajos prototipas

Paslėptų duomenų nuskaitymo formą sudaro:

- Failų sistema. Failų sistema, kurioje bus slepiamas failas. Pateikiamas visų išorinėse (ne sisteminėse) laikmenose esančių failų sistemų sąrašas.
- Dengiantys failai. Pateikiami keliai iki visų pasirinktų dengiančių failų. Failai pasirenkami mygtuko „Pasirinkti“ pagalba“. Sąrašo dešinėje esančiais mygtukais galima keisti failų pateikimo algoritmui tvarką.
- Inicializacijos vektorius B_0. Slapto rakto dalis, natūralusis skaičius.
- Reikšmė m. Neredaguojamas. Pagal pasirinktų failų skaičių automatiškai paskaičiuojama slapto rakto dalis.
- Failo dydis. Nurodomas nuskaitymų duomenų dydis (baitais).

4.1.3.4. Kortelė „Laikinių charakteristikų eksperimentas“



26 pav. Kortelės „Laikinių charakteristikų eksperimentas“ grafinės sąsajos prototipas

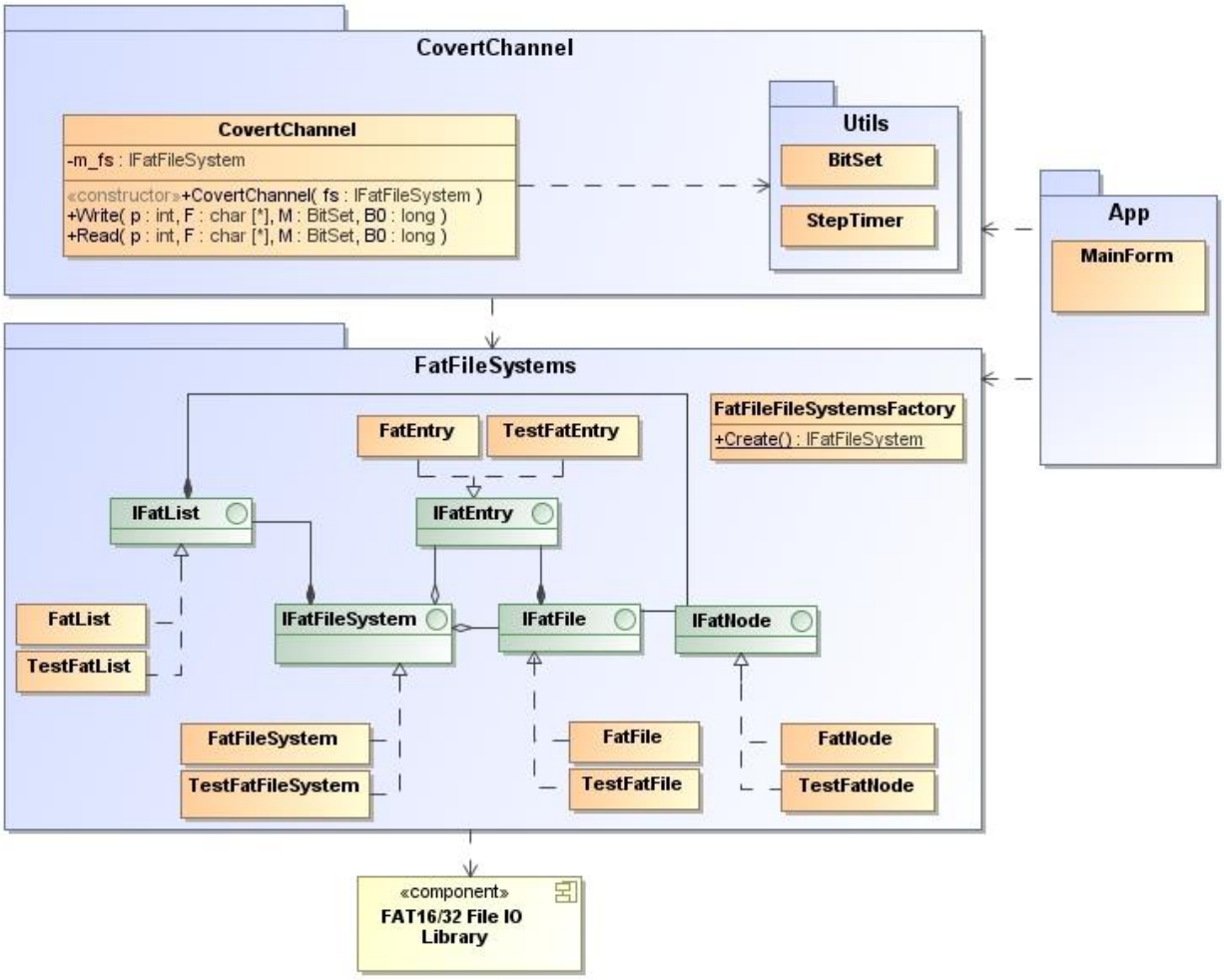
Laikinių charakteristikų eksperimento formą sudaro:

- Žurnalo dokumentas. Kelias iki žurnalo dokumento, kuriame išsaugomi tarpinių etapų laikai, klaidų pranešimai. Failas pasirenkamas paspaudus mygtuką „Pasirinkti“.
- Atliekamų bandymų skaičius. Natūralusis skaičius. Kiek kartų kartoti kiekvieną bandymą.
- Failų sistema. Failų sistema, kurioje bus slepiamas failas. Pateikiamas visų išorinėse (ne sisteminėse) laikmenose esančių failų sistemų sąrašas.
- Naudoti atsitikt. slaptus duomenis“. Požymis, nusakantis, ar bus naudojamas vartotojo pateiktas slepiamų duomenų failas, ar bus automatiškai generuojamas atsitiktinių duomenų failas.
- Atsitiktinių duomenų failo dydis. Aktyvus, kai pažymėtas „Naudoti atsitikt. slaptus duomenis“. Kokio dydžio (baitais) failus su atsitiktiniais duomenimis generuoti.
- Slepiamas failas. Aktyvus, kai nepažymėtas „Naudoti atsitiktinius slaptus duomenis“. Pateikiamas kelias iki slepiamo failo. Failas pasirenkamas paspaudus mygtuką „Pasirinkti“.
- Naudoti atsitikt. dengiančius failus. Požymis, nusakantis, ar bus naudojami vartotojo pateikti dengiantys failai, ar jie bus automatiškai generuojami.
- Dengiančių failų skaičius. Aktyvus, kai pažymėtas „Naudoti atsitikt. dengiančius failus“. Natūralusis skaičius, nusakantis, kiek dengiančių failų turėtų būti naudojama slėpimo procese.
- Dengiantys failai. Aktyvus, kai nepažymėtas „Naudoti atsitikt. dengiančius failus“. Pateikiami keliai iki visų pasirinktų dengiančių failų. Failai pasirenkami mygtuko „Pasirinkti“ pagalba“. Sąrašo dešinėje esančiais mygtukais galima keisti failų pateikimo algoritmui tvarką.

4.1.4. Architektūra

Programą sudaro 4 sluoksniai (žr. 27 pav.):

1. FAT16/32 File IO Library. Naudojama išorinė biblioteka, skirta žemo lygio darbui su FAT16 ir FAT32 failų sistemomis.
2. FileSystems. Darbui su FAT failų sistemomis skirtas sluoksnis, kurio API suteikia objektinę sąsają darbui su FAT sistemomis. Priklausomai nuo ją naudojančios programos, biblioteka gali dirbti su realia failų sistema arba supaprastinta fiktyvia failų sistemos imitacija.
3. CovertChannel. Siūlomą slapto kanalo metodą realizuojanti biblioteka, kuri savo funkcijas realizuoja panaudodama FileSystems API.
4. App. Vartotojo sąsajos sluoksnis, kuris vartotojui pateikia grafinę sąsają, inicijuoja darbą su failų sistema bei kviečia CovertChannel API funkcijas.



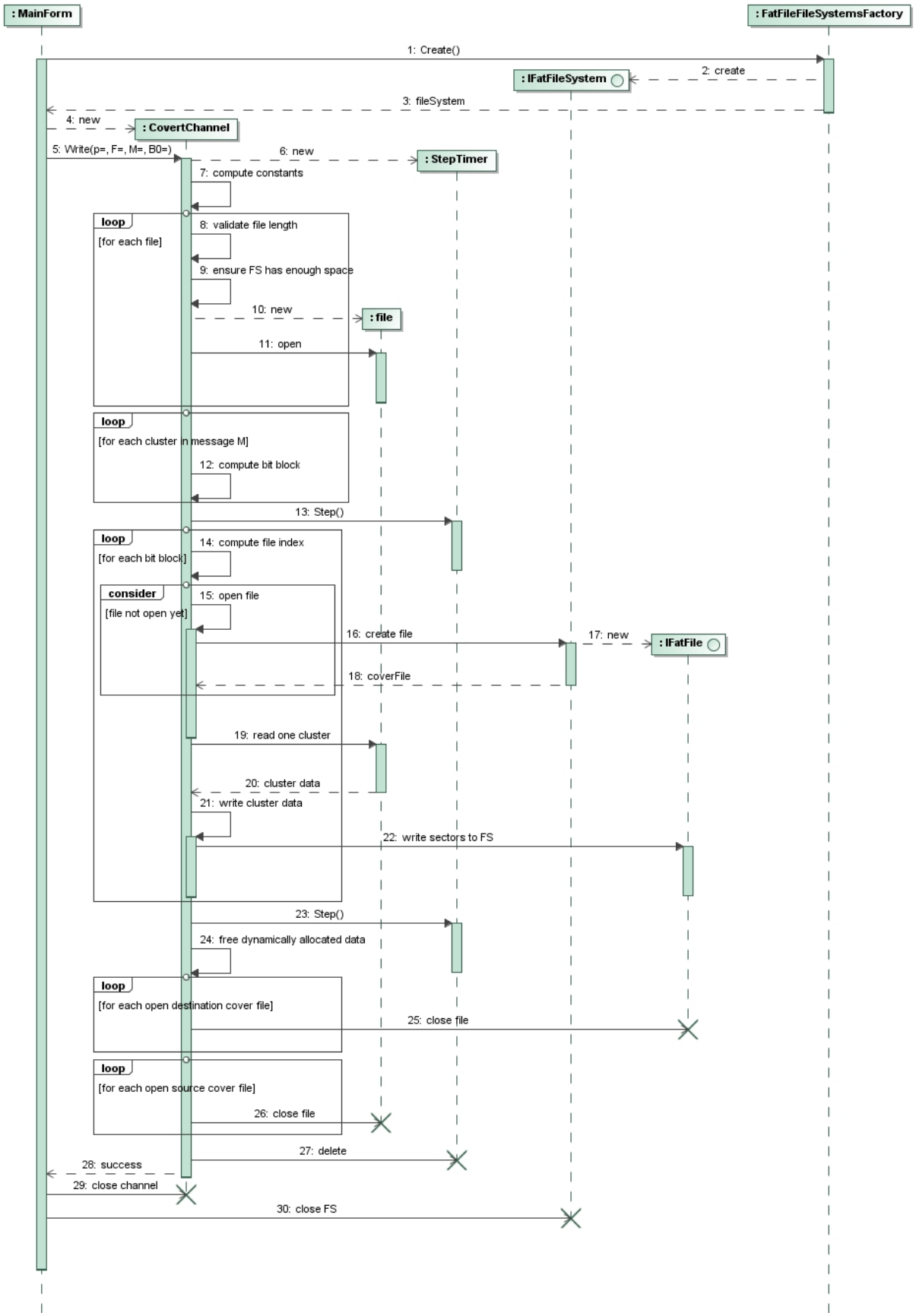
27 pav. Eksperimentinės metodą realizuojančios programos klasių diagrama

4.1.5. Algoritmų aprašymai

Žemiau pateikti 2.1 bei 2.2 skyriuose aprašytų algoritmų sekų diagramos, grafiškai parodant, kaip šie algoritmai realizuoti eksperimentinėje programoje.

4.1.5.1. Duomenų paslėpimas

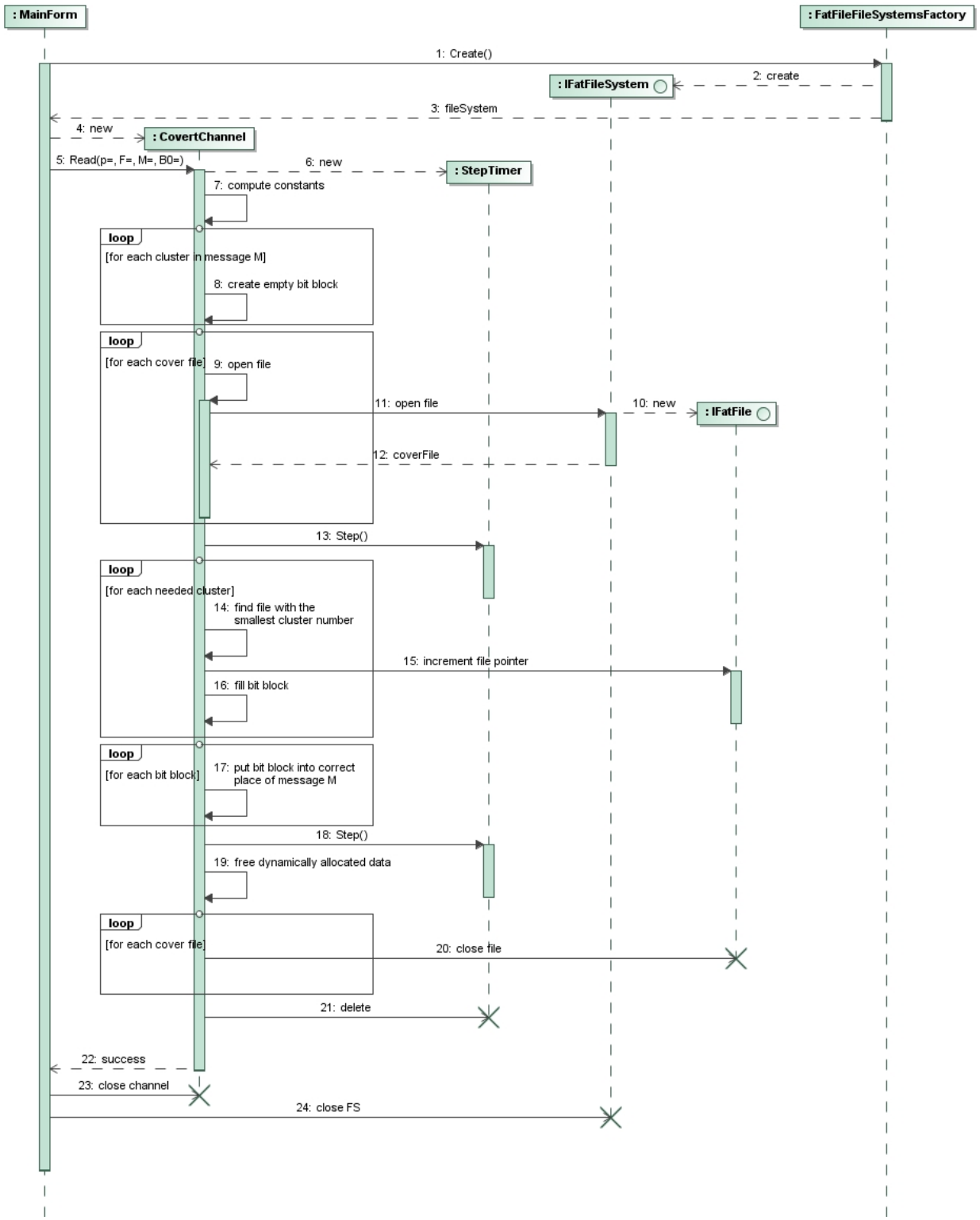
28 pav. pateikta duomenų paslėpimo sekų diagrama atspindi, kaip pasiūlytas paslėpimo algoritmas realizuotas eksperimentinėje programoje.



28 pav. Duomenų slėpimo algoritmo sekų diagrama

4.1.5.2. Duomenų nuskaitymas

29 pav. pateikta duomenų nuskaitymo sekų diagrama atspindi, kaip pasiūlytas paslėptų duomenų nuskaitymo algoritmas realizuotas eksperimentinėje programoje.



29 pav. Duomenų nuskaitymo algoritmo sekų diagrama

4.2. Tyrimų planas

Tyrimams testinėje aplikoje buvo sukurta 1 GB dydžio FAT32 failų sistema su 512 B dydžio klasteriais. Panaudojant žemiau aprašytus planus, atlikti metodo bandymai, panaudojant eksperimentinę programą.

4.2.1. Atsparumo dengiančių failų modifikavimui tyrimo planas

Atsparumas dengiančių failų modifikavimui buvo matuojamas atliekant šiuos žingsnius:

1. Panaudojant eksperimentinę programą sukuriami 2 failai:
    1. užpildytas nulinėmis reikšmėmis;

2. užpildytas atsitiktiniais duomenimis.
2. Paskaičiuojama kiekvieno failo MD5 maišos reikšmė.
3. Panaudojant tekstinius dengiančius failus kiekvienas iš šių failų paslepiamas.
4. Dengiantiems failams atliekama viena iš šių operacijų:
    1. papildoma į pabaigą;
    2. papildoma į pradžią;
    3. pakeičiama pradžia;
    4. ištrinama dalis teksto pradžioje;
    5. ištrinama dalis teksto pabaigoje.
5. Panaudojant eksperimentinę programą failas nuskaitomas ir išsaugomas.
6. Paskaičiuojama nuskaityto failo MD5 maišos reikšmė.
7. Gauta reikšmė palyginama su paskaičiuota #2 žingsnyje.
8. Tyrimas atliekamas su kiekvienu failu, sukurtu #1 žingsnyje, ir kiekviena operacija, pateikta #4 žingsnyje.

4.2.2. Atsparumo failų sistemos defragmentacijai tyrimo planas

Atsparumas failų sistemos defragmentacijai buvo matuojamas atliekant šiuos žingsnius:

1. Panaudojant eksperimentinę programą sukuriami 2 failai:
    1. užpildytas nulinėmis reikšmėmis;
    2. užpildytas atsitiktiniais duomenimis.
2. Paskaičiuojama kiekvieno failo MD5 maišos reikšmė.
3. Kiekvienas iš šių failų paslepiamas.
4. Panaudojant defragmentacijos programą, failų sistema sudefragmentuojama.
5. Failas nuskaitomas ir išsaugomas.
6. Paskaičiuojama nuskaityto failo MD5 maišos reikšmė.
7. Gauta reikšmė palyginama su paskaičiuota #2 žingsnyje.
8. Tyrimas atliekamas su kiekvienu failu, sukurtu #1 žingsnyje.

4.2.3. Metodo laikinių charakteristikų tyrimas

Metodo laikinės charakteristikos ištirtos panaudojant eksperimentinės programos „Laikinių charakteristikų eksperimentas“ kortelėje esančiu funkcionalumu. Panaudojant skirtingo dydžio slepiamų duomenų failus (10 B, 100 B, 1 kB, 100 kB, 1 MB) ištirti šie atvejai:

1. Slepiamas tuščiomis reikšmėmis užpildytas failas, panaudojant [2, 4, 8] atsitiktinai sugeneruotus dengiančius failus.
2. Slepiamas atsitiktinėmis reikšmėmis užpildytas failas, panaudojant [2, 4, 8] atsitiktinai sugeneruotus dengiančius failus.
3. Slepiamas tuščiomis reikšmėmis užpildytas failas, panaudojant 2 realius dengiančius failus.
4. Slepiamas atsitiktinėmis reikšmėmis užpildytas failas, panaudojant 2 realius dengiančius failus.

Kiekvienas bandymas atliekamas po 10 kartų (nurodant „Atliekamų bandymų skaičius“ reikšmę).

4.3. Atsparumo dengiančių failų modifikavimui tyrimas

Atlikto tyrimo metodologija pateikta 4.2.1 skyriuje. Žemiau esančiose 19 ir 21 lentelėse pateikta pradinė failų sistemos būsena prieš analizę. Jose nurodytas slepiamo failo vardas, jo užpildymo tipas (nulinės arba atsitiktinės reikšmės), dydis, jam paslėpti panaudoti dengiantys failai. Taip pat pateikta slepiamo failo MD5 maišos reikšmė bei dengiančių failų klasterių išsidėstymo schema failų sistemoje. Ši schema sugeneruojama, naudojant „The Sleuth Kit“ [23] įrankį bei papildomą „savo“ programą, kuri skirtingomis spalvomis parodo, kaip failų sistemoje išsidėstę skirtingų failų klasteriai.

Kaip matome 19-oje lentelėje, nulinėmis reikšmėmis užpildytas failas (t.y. slepiami vien nuliniai bitai) gali būti paslėptas paprastai surašant 2 failų klasterius ir nesukeliant jokios fragmentacijos. Pirmiausia eina pirmojo dengiančio failo klasteriai, po to – antrojo. Šiuo atveju

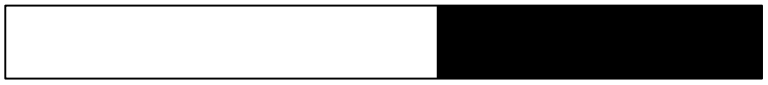
inicializacijos vektorius buvo pateiktas toks, kad pirmas klasteris priklauso antram dengiančiam failui, tuomet surašomi visi pirmojo failo klasteriai ir galiausiai – visi antrojo failo klasteriai.

19 lentelė. Pradiniai duomenys paslėptų nulinių reikšmių atsparumo modifikavimui tyrimui

Failo vardas	Testas_1.bin
Užpildymo tipas	Nulinės reikšmės
Failo dydis	128 B
Dengiantys failai	Dengiantis_1.txt Dengiantis_2.txt
Maišos reikšmė	f09f35a5637839458e462e6350ecbce4
Klasterių išsidėstymas FS	

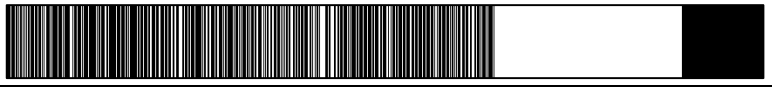
20 lentelėje pateikti atsparumo modifikavimui, kai slepiamas nulinėmis reikšmėmis užpildytas failas, tyrimo rezultatai. Iš jų matome, kad duomenys sugadinami tik tais atvejais, kai dengiančių failų dydis sumažėja – ištrinant pradžią arba pabaigą. Atliekant dengiančių failų papildymą arba keičiant duomenis, papildomi duomenys rašomi į naujus klasterius, kurie neįtakoja jau esančių klasterių išsidėstymo. Dėl šios priežasties, atliekant šias operacijas, paslėpti duomenys nedingsta.

20 lentelė. Paslėptų nulinių reikšmių atsparumo modifikavimui tyrimo rezultatai

Operacija	**Nuskaityto failo maišos reikšmė, klasterių išsidėstymas failų sistemoje**	**Duomenys sugadinti?**
Papildymas į pabaigą	f09f35a5637839458e462e6350ecbce4 	Ne
Papildymas į pradžią	f09f35a5637839458e462e6350ecbce4 	Ne
Pradžios pakeitimas	f09f35a5637839458e462e6350ecbce4 	Ne
Pradžios ištrynimas	88d29edd1bfd661cb402b80c8b3f9cc0 	Taip
Pabaigos ištrynimas	9b2db65d310c63287265aed4c2479704 	Taip

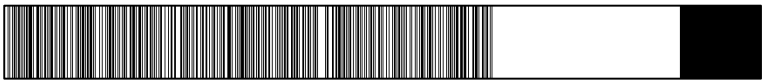
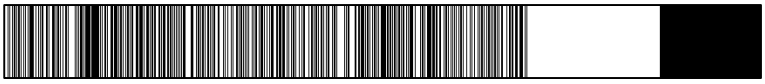
Žemiau esanti lentelė atspindi, kad atsitiktinėmis reikšmėmis užpildyto failo paslėpimas sukelia žymią dengiančių failų fragmentaciją. Taip pat matome, kad metodas, pabaigęs slėpti duomenis, likusius dengiančių failų klasterius surašo įprasta tvarka (šiuo atveju pirma visus pirmojo failo klasterius, tuomet – antrojo).

21 lentelė. Pradiniai duomenys paslėptų atsitiktinių reikšmių atsparumo modifikavimui tyrimui

Failo vardas	Testas_2.bin
Užpildymo tipas	Atsitiktinės reikšmės
Failo dydis	128 B
Dengiantys failai	Dengiantis_1.txt Dengiantis_2.txt
Maišos reikšmė	3c731817afa9d619f0b22833520b6c7a
Klasterių išsidėstymas FS	

22 lentelėje pateikti atsparumo modifikavimui, kai slepiamas atsitiktinėmis reikšmėmis užpildytas failas, tyrimo rezultatai. Šiuo atveju nė viena iš atliktų operacijų nesugadino paslėptų duomenų. Papildant failą, kaip ir nulinių reikšmių paslėpimo atveju, sukuriami papildomi klasteriai, kurie prijungiami prie failo paskutinio klasterio ir paslėptų duomenų neįtakoja. Tačiau šiuo atveju ir ištrinant dalį tekstinių dengiančių failų informacijos, paslėpti duomenys nebuvo pažeisti. Tai galima paaiškinti tuo, kad dengiantys failai buvo didesni nei iš tikro reikėjo (dydžiai tenkino lygybę $C = \log_2 f$, žr. 3.1 skyrių). Šis papildomas failų dydis sukūrė tam tikrą apsaugą nuo paslėptų duomenų vientisumo pažeidimo mažėjant dengiančių failų dydžiui. Kaip matome iš lentelėje pateiktų dengiančių failų klasterių išsidėstymo schemų, ištrintas informacijos kiekis pašalino dalį failams skirtų paskutinių klasterių, tačiau ištrintas kiekis nebuvo pakankamas, kad pažeistų paslėptus duomenis.

22 lentelė. Paslėptų atsitiktinių reikšmių atsparumo modifikavimui tyrimo rezultatai

Operacija	**Nuskaityto failo maišos reikšmė, klasterių išsidėstymas failų sistemoje**	**Duomenys sugadinti?**
Papildymas į pabaigą	3c731817afa9d619f0b22833520b6c7a 	Ne
Papildymas į pradžią	3c731817afa9d619f0b22833520b6c7a 	Ne
Pradžios pakeitimas	3c731817afa9d619f0b22833520b6c7a 	Ne
Pradžios ištrynimas	3c731817afa9d619f0b22833520b6c7a 	Ne
Pabaigos ištrynimas	3c731817afa9d619f0b22833520b6c7a 	Ne

Atlikę atsparumo modifikavimui tyrimą matome, kad paslėpti duomenys nėra pažeidžiami atliekant dengiančių failų pakeitimus, kurie nesumažina failų dydžio. Taip pat, jei naudojami didesni dengiantys failai, negu yra būtina paslėpti duomenims, sukuriama papildoma paslėptų duomenų apsauga nuo dengiančių failų modifikavimo pašalinant informaciją ir taip sumažinant jų klasterių skaičių.

Verta atkreipti dėmesį į tai, kad nesvarbu kurią tekstinio failo dalį betrintume, fizinė jo vieta visada traukiasi nuo galo. T.y. ištrynę failo pradžią, prarasime paskutinius jo klasterius. Taigi, jei naudojami dengiantys failai turi perteklinių klasterių, kuriais nėra slepiami duomenys, gaunamas atsparumas dalies failo turinio ištrynimui (bet kurioje failo vietoje).

4.4. Atsparumo disko defragmentavimui tyrimas

Atlikto tyrimo metodologija pateikta 4.2.2 skyriuje. Buvo slepiami dviejų rūšių (užpildyti nulinėmis bei atsitiktinėmis reikšmėmis) failai, failų sistema sudefragmentuojama, panaudojant Windows 7 Defragmentation Tool programą ir bandoma nuskaityti paslėptus duomenis.

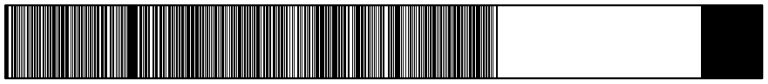
23 lentelėje pateikti tyrimo, panaudojant nulines reikšmes, rezultatai. Iš klasterių išsidėstymo failų sistemoje prieš defragmentaciją schemos matome, kad pirmiausia buvo įrašytas vienas antrojo dengiančio failo klasteris, tuomet visi pirmojo dengiančio failo klasteriai ir galiausiai – visi likę antrojo failo klasteriai. Defragmentacijos metu pirmasis antrojo dengiančio failo klasteris buvo perkeltas prie visų likusių, o tai pažeidė paslėptų duomenų vientisumą.

23 lentelė. Paslėptų nulinių reikšmių atsparumo disko defragmentavimui tyrimo rezultatai

Failo vardas	Testas_1.bin
Užpildymo tipas	Nulinės reikšmės
Failo dydis	128 B
Dengiantys failai	Dengiantis_1.txt Dengiantis_2.txt
Maišos reikšmė prieš defragmentaciją	f09f35a5637839458e462e6350ecbce4
Klasterių išsidėstymas FS prieš defragmentaciją	
Maišos reikšmė po defragmentacijos	eb7a491f09f61a338a2b9fc2dfdf0040
Klasterių išsidėstymas FS po defragmentacijos	
Duomenys sugadinti?	Taip

24 lentelėje pateikti tyrimo, panaudojant atsitiktines reikšmes, rezultatai. Iš klasterių išsidėstymo failų sistemoje prieš defragmentaciją schemos matome, kad dengiančių failų klasteriai tarpusavyje yra smarkiai persipynę. Po defragmentacijos skirtingų failų klasteriai buvo sugruputi, o tai pažeidė paslėptų duomenų vientisumą.

24 lentelė. Paslėptų atsitiktinių reikšmių atsparumo disko defragmentavimui tyrimo rezultatai

Failo vardas	Testas_2.bin
Užpildymo tipas	Atsitiktinės reikšmės
Failo dydis	128 B
Dengiantys failai	Dengiantis_1.txt Dengiantis_2.txt
Maišos reikšmė prieš defragmentaciją	90e71766080505d00f3635fb2cbc320f
Klasterių išsidėstymas FS prieš defragmentaciją	
Maišos reikšmė po defragmentacijos	eb7a491f09f61a338a2b9fc2dfdf0040
Klasterių išsidėstymas FS po defragmentacijos	
Duomenys sugadinti?	Taip

Atsparumo disko defragmentavimui tyrimas atskleidė, kad siūlomas metodas yra neatsparus disko defragmentavimui. Defragmentavimo operacija radikaliai pakeičia failų sistemoje esančių failų klasterių išdėstymą, kuris naudojamas paslėpti duomenims. Po klasterių išdėstymo pakeitimo, paslėpti duomenys sunaikinami.

4.5. Laikinių metodo charakteristikų tyrimas

Laikinių charakteristikų tyrimo metodologija aprašyta 4.2.3 skyriuje. Panaudojant eksperimentinę programą analizuoti dviejų dydžių (mažas, 10 B dydžio, ir sąlyginai didelis, 1 kB dydžio) ir dviejų užpildymo tipų (nulinėmis bei atsitiktinėmis reikšmėmis) failų paslėpimo laikai sekundėmis.

Tiriant laikus, duomenų paslėpimas buvo suskirstytas į tris etapus (žr. 2.1 sk.):

1. Inicializacija. Apskaičiuojamos konstantos, atliekamos reikalingos duomenų modifikacijos, atliekami patikrinimai, ar užtenka failų sistemoje vietos ir dengiančių failų dydžių duomenims paslėpti.
2. Duomenų slėpimas. Panaudojant siūlomą metodą, dengiančių failų klasteriai rašomi į failų sistemą, kol visiškai paslepiami duomenys.

3. Likusių klasterių surašymas. Surašomi visi likę dengiančių failų klasteriai, kurie nebuvo panaudoti slepiant duomenis.

Duomenų nuskaitymas taip pat suskirstytas į 3 etapus (žr. 2.2 sk.):

1. Inicializacija. Apskaičiuojamos konstantos, rezervuojama atmintis rezultatams.
2. Duomenų nuskaitymas. Pagal dengiančių failų klasterių tarpusavio išsidėstymą nuskaitomi paslėpti duomenys ir patalpinami į rezervuotą atminties vietą.
3. Duomenų apdorojimas. Nuskaityti duomenys paverčiami iš blokų masyvo į naudojamą duomenų struktūrą.

Kiekvienas bandymas atliktas po 10 kartų ir apskaičiuotos vidutinės etapų laikų vertės. Toliau aptariant laikus bus kalbama tik apie vidutines vertes.

Žvelgdami į žemiau pateiktas 25, 26 bei 27 lenteles matome, kad didžiąją dalį laiko užima paslėpimo inicializacija, kurios metu pasiruošiama slėpti duomenis. Taip pat matome, kad duomenų paslėpimo etapas trumpėja daugėjant dengiančių failų skaičiui (paslepiama daugiau bitų panadojant vieną dengiančio failo klasterį), tačiau tuo pačiu daugėja perteklinės dengiančių failų dydžių vertės, dėl kurių ilgėja likusių klasterių surašymo etapas. Kadangi bendras visų dengiančių failų klasterių skaičius yra vienodas, bendras (rašymo slepiant ir likusių klasterių surašymo) laikas yra panašus.

Duomenų nuskaitymas ypač greitas, vienintelė užfiksuota laiko vertė yra inicializacijos etape, kai paruošiamos duomenų struktūros.

25 lentelė. 10 B dydžio nulinėmis reikšmėmis užpildyto failo paslėpimo, panaudojant 2 deng. failus, laikai sekundėmis

Failo vardas	Testas_1.bin										
Užpildymo tipas	Nulinės reikšmės										
Failo dydis	10 B										
Dengiačių failų skaičius	2										
Bandymo nr. / Žingsnis	1	2	3	4	5	6	7	8	9	10	Vid.
Rašymas. Inicializacija	4,180	3,823	3,620	3,729	3,619	3,728	3,603	3,822	3,619	3,744	3,749
Rašymas. Duomenų slėpimas	0,266	0,249	0,312	0,281	0,312	0,296	0,343	0,312	0,327	0,234	0,293
Rašymas. Likusių klasterių surašymas	0,078	0,063	0,140	0,062	0,063	0,063	0,140	0,063	0,047	0,047	0,077
Skaitymas. Inicializacija	0,015	0,000	0,016	0,016	0,016	0,032	0,016	0,000	0,016	0,000	0,013
Skaitymas. Duomenų nuskaitymas	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
Skaitymas. Duomenų apdorojimas	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000

26 lentelė. 10 B dydžio nulinėmis reikšmėmis užpildyto failo paslėpimo, panaudojant 4 deng. failus, laikai sekundėmis

Failo vardas	Testas_1.bin										
Užpildymo tipas	Nulinės reikšmės										
Failo dydis	10 B										
Dengiačių failų skaičius	4										
Bandymo nr. / Žingsnis	1	2	3	4	5	6	7	8	9	10	Vid.
Rašymas. Inicializacija	4,134	3,915	3,931	3,947	4,165	4,088	3,993	4,103	4,181	4,415	4,087
Rašymas. Duomenų slėpimas	0,234	0,203	0,203	0,234	0,234	0,172	0,156	0,156	0,141	0,140	0,187
Rašymas. Likusių klasterių surašymas	0,156	0,109	0,093	0,140	0,140	0,062	0,125	0,078	0,156	0,093	0,115
Skaitymas. Inicializacija	0,015	0,000	0,016	0,015	0,015	0,016	0,015	0,016	0,015	0,015	0,014
Skaitymas. Duomenų nuskaitymas	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000

Skaitymas. Duomenų apdorojimas	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000

27 lentelė. 10 B dydžio nulinėmis reikšmėmis užpildyto failo paslėpimo, panaudojant 8 deng. failus, laikai sekundėmis

Failo vardas	Testas_1.bin										
Užpildymo tipas	Nulinės reikšmės										
Failo dydis	10 B										
Dengiačių failų skaičius	8										
Bandymo nr. / **Žingsnis**	1	2	3	4	5	6	7	8	9	10	Vid.
Rašymas. Inicializacija	3,947	3,713	3,838	3,916	4,118	3,838	3,869	4,087	3,931	3,963	3,922
Rašymas. Duomenų slėpimas	0,109	0,140	0,171	0,203	0,171	0,140	0,187	0,125	0,203	0,187	0,164
Rašymas. Likusių klasterių surašymas	0,156	0,281	0,218	0,390	0,234	0,234	0,188	0,156	0,187	0,203	0,225
Skaitymas. Inicializacija	0,016	0,016	0,015	0,016	0,015	0,031	0,016	0,016	0,016	0,016	0,017
Skaitymas. Duomenų nuskaitymas	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
Skaitymas. Duomenų apdorojimas	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000

Žemiau pateiktose 28, 29 bei 30 lentelėse matome laikinių charakteristikų tyrimo, slepiant 1 kB nulinėmis reikšmėmis užpildytą failą, rezultatus. Paslėpimo etapo inicializacija ir šiuo atveju buvo pastovi ir truko 3 – 4 sek. Slepiamų duomenų kiekis lėmė, kad didžiąją dalį laiko užima kiti du rašymo etapai, kai surašomi dengiančių failų klasteriai. Vėlgi kartojasi tendencija, kad daugėjant dengiančių failų, pats duomenų paslėpimas trumpėja ir ilgėja likusių klasterių surašymo etapas.

Ir šiuo atveju nuskaitymas buvo žymiai spartesnis už paslėpimą. Inicializacija truko 0,006 – 0,014 sek., o nuskaitymas 0,014 – 0,027 sek.

28 lentelė. 1 kB dydžio nulinėmis reikšmėmis užpildyto failo paslėpimo, panaudojant 2 deng. failus, laikai sekundėmis

Failo vardas	Testas_2.bin										
Užpildymo tipas	Nulinės reikšmės										
Failo dydis	1 kB										
Dengiačių failų skaičius	2										
Bandymo nr. / **Žingsnis**	1	2	3	4	5	6	7	8	9	10	Vid.
Rašymas. Inicializacija	4,368	3,869	3,775	3,915	3,681	3,963	3,697	3,900	3,885	6,717	4,177
Rašymas. Duomenų slėpimas	91,35	89,10	94,80	95,19	95,23	97,27	89,76	103,0	88,10	159,3	100,310
Rašymas. Likusių klasterių surašymas	202,4	201,0	198,1	204,7	190,8	191,2	196,8	197,3	303,8	302,0	218,810
Skaitymas. Inicializacija	0,015	0,000	0,016	0,000	0,016	0,000	0,000	0,000	0,008	0,007	0,006
Skaitymas. Duomenų nuskaitymas	0,032	0,015	0,047	0,015	0,031	0,031	0,015	0,031	0,028	0,024	0,027
Skaitymas. Duomenų apdorojimas	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000

29 lentelė. 1 kB dydžio nulinėmis reikšmėmis užpildyto failo paslėpimo, panaudojant 4 deng. failus, laikai sekundėmis

Failo vardas	Testas_2.bin
Užpildymo tipas	Nulinės reikšmės
Failo dydis	1 kB

Dengiačių failų skaičius	4										
Bandymo nr. / Žingsnis	1	2	3	4	5	6	7	8	9	10	Vid.
Rašymas. Inicializacija	3,366	3,153	2,801	2,873	2,857	2,867	3,781	2,760	2,864	2,674	3,000
Rašymas. Duomenų slėpimas	24,35	24,79	26,74	24,72	24,83	25,02	31,33	26,45	23,38	29,18	26,079
Rašymas. Likusių klasterių surašymas	179,4	177,1	176,4	183,1	189,4	182,4	182,1	189,6	176,0	180,7	181,620
Skaitymas. Inicializacija	0,010	0,010	0,011	0,011	0,005	0,009	0,010	0,010	0,011	0,008	0,010
Skaitymas. Duomenų nuskaitymas	0,015	0,017	0,017	0,020	0,013	0,010	0,014	0,015	0,016	0,013	0,015
Skaitymas. Duomenų apdorojimas	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000

30 lentelė. 1 kB dydžio nulinėmis reikšmėmis užpildyto failo paslėpimo, panaudojant 8 deng. failus, laikai sekundėmis

Failo vardas	Testas_2.bin										
Užpildymo tipas	Nulinės reikšmės										
Failo dydis	1 kB										
Dengiačių failų skaičius	8										
Bandymo nr. / Žingsnis	1	2	3	4	5	6	7	8	9	10	Vid.
Rašymas. Inicializacija	3,031	5,981	2,843	2,749	2,601	2,937	3,060	2,878	2,927	3,010	3,202
Rašymas. Duomenų slėpimas	15,85	16,47	18,98	17,12	17,43	17,32	18,27	17,86	17,76	17,97	17,503
Rašymas. Likusių klasterių surašymas	352,9	370,7	349,8	327,5	328,2	333,5	331,4	330,7	335,9	338,2	339,880
Skaitymas. Inicializacija	0,020	0,018	0,010	0,011	0,020	0,015	0,010	0,011	0,010	0,019	0,014
Skaitymas. Duomenų nuskaitymas	0,029	0,013	0,010	0,011	0,016	0,011	0,011	0,013	0,011	0,014	0,014
Skaitymas. Duomenų apdorojimas	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000

10 B atsitiktinai sugeneruotų duomenų paslėpimo laikinių charakteristikų matavimo rezultatai pateikti 31, 32 bei 33 lentelėse. Palyginę rezultatus su rezultatais, gautais slepiant nulinėmis reikšmėmis užpildytą failą, matome, kad jie žymiai nesiskiria. Taip yra todėl, kad realizuota programa duomenis rašo ne maksimaliais galimais blokais, o po vieną klasterį. Optimizavus programą rašyti po maks. galimą duomenų bloką, nulinėmis reikšmėmis užpildyti failai būtų surašomi žymiai sparčiau.

31 lentelė. 10 B dydžio atsitiktinėmis reikšmėmis užpildyto failo paslėpimo, panaudojant 2 deng. failus, laikai sekundėmis

Failo vardas	Testas_3.bin										
Užpildymo tipas	Atsitiktinės reikšmės										
Failo dydis	10 B										
Dengiačių failų skaičius	2										
Bandymo nr. / Žingsnis	1	2	3	4	5	6	7	8	9	10	Vid.
Rašymas. Inicializacija	3,828	3,828	4,300	4,676	4,309	6,365	5,723	4,659	4,613	5,136	4,744
Rašymas. Duomenų slėpimas	0,801	0,427	0,405	0,258	0,462	0,747	0,550	0,349	0,573	0,324	0,490
Rašymas. Likusių klasterių surašymas	0,100	0,043	0,178	0,101	0,062	2,277	0,107	0,121	0,082	0,066	0,314

Skaitymas. Inicializacija	0,017	0,005	0,008	0,024	0,007	0,105	0,010	0,019	0,023	0,011	0,023
Skaitymas. Duomenų nuskaitymas	0,000	0,000	0,001	0,000	0,001	0,025	0,011	0,000	0,000	0,000	0,004
Skaitymas. Duomenų apdorojimas	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000

32 lentelė. 10 B dydžio atsitiktinėmis reikšmėmis užpildyto failo paslėpimo, panaudojant 4 deng. failus, laikai sekundėmis

Failo vardas	Testas_3.bin										
Užpildymo tipas	Atsitiktinės reikšmės										
Failo dydis	10 B										
Dengiačių failų skaičius	4										
Bandymo nr. / **Žingsnis**	1	2	3	4	5	6	7	8	9	10	Vid.
Rašymas. Inicializacija	3,268	2,977	2,995	2,756	2,783	2,741	2,775	2,795	2,921	2,763	2,877
Rašymas. Duomenų slėpimas	0,121	0,211	0,413	0,153	0,139	0,147	0,124	0,089	0,149	0,162	0,171
Rašymas. Likusių klasterių surašymas	0,059	0,205	0,093	0,090	0,105	0,110	0,141	0,106	0,062	0,115	0,109
Skaitymas. Inicializacija	0,006	0,009	0,009	0,010	0,010	0,010	0,012	0,010	0,006	0,005	0,009
Skaitymas. Duomenų nuskaitymas	0,000	0,000	0,001	0,016	0,001	0,000	0,000	0,000	0,000	0,000	0,002
Skaitymas. Duomenų apdorojimas	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000

33 lentelė. 10 B dydžio atsitiktinėmis reikšmėmis užpildyto failo paslėpimo, panaudojant 8 deng. failus, laikai sekundėmis

Failo vardas	Testas_3.bin										
Užpildymo tipas	Atsitiktinės reikšmės										
Failo dydis	10 B										
Dengiačių failų skaičius	8										
Bandymo nr. / **Žingsnis**	1	2	3	4	5	6	7	8	9	10	Vid.
Rašymas. Inicializacija	3,095	3,121	3,027	2,903	2,970	2,792	2,610	2,947	3,275	3,220	2,996
Rašymas. Duomenų slėpimas	0,109	0,162	0,188	0,177	0,147	0,128	0,141	0,144	0,194	0,189	0,158
Rašymas. Likusių klasterių surašymas	0,505	0,165	0,174	0,172	0,127	0,229	0,160	0,149	0,140	0,219	0,204
Skaitymas. Inicializacija	0,017	0,018	0,017	0,018	0,018	0,019	0,017	0,015	0,019	0,017	0,018
Skaitymas. Duomenų nuskaitymas	0,000	0,001	0,000	0,001	0,000	0,000	0,000	0,000	0,000	0,000	0,000
Skaitymas. Duomenų apdorojimas	0,000	0,001	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000

Žemiau esančiose 34, 35 ir 36 lentelėse pateikti 1 kB atsitiktinai sugeneruotų duomenų paslėpimo ir nuskaitymo laikų tyrimo rezultatai. Ir šiuo atveju jie panašūs į rezultatus, slepiant nulinėmis reikšmėmis užpildytą failą.

34 lentelė. 1 kB dydžio atsitiktinėmis reikšmėmis užpildyto failo paslėpimo, panaudojant 2 deng. failus, laikai sekundėmis

Failo vardas	Testas_4.bin
Užpildymo tipas	Atsitiktinės reikšmės
Failo dydis	1 kB
Dengiačių failų skaičius	2

Bandymo nr. / Žingsnis	1	2	3	4	5	6	7	8	9	10	Vid.
Rašymas. Inicializacija	4,126	3,447	2,772	3,417	2,341	3,278	3,463	3,272	3,384	3,456	3,296
Rašymas. Duomenų slėpimas	94,51	87,15	86,16	143,5	87,43	87,43	91,56	97,38	97,38	96,59	96,909
Rašymas. Likusių klasterių surašymas	179,9	198,6	181,8	213,6	239,5	239,5	187,8	188,2	188,2	181,2	199,830
Skaitymas. Inicializacija	0,05	0,004	0,003	0,030	0,012	0,012	0,007	0,007	0,007	0,003	0,014
Skaitymas. Duomenų nuskaitymas	0,025	0,020	0,034	0,021	0,047	0,047	0,029	0,032	0,032	0,020	0,031
Skaitymas. Duomenų apdorojimas	0,000	0,001	0,000	0,001	0,001	0,001	0,001	0,000	0,000	0,000	0,001

35 lentelė. 1 kB dydžio atsitiktinėmis reikšmėmis užpildyto failo paslėpimo, panaudojant 4 deng. failus, laikai sekundėmis

Failo vardas	Testas_4.bin										
Užpildymo tipas	Atsitiktinės reikšmės										
Failo dydis	1 kB										
Dengiačių failų skaičius	4										
Bandymo nr. / Žingsnis	1	2	3	4	5	6	7	8	9	10	Vid.
Rašymas. Inicializacija	3,095	2,736	3,011	3,107	3,032	2,876	2,887	2,836	5,096	2,980	3,166
Rašymas. Duomenų slėpimas	25,60	25,45	29,01	29,50	28,70	28,65	28,83	29,42	51,07	30,38	30,661
Rašymas. Likusių klasterių surašymas	174,6	177,3	200,5	198,3	182,1	183,3	185,6	197,7	253,8	183,7	193,690
Skaitymas. Inicializacija	0,011	0,012	0,011	0,006	0,022	0,012	0,010	0,051	0,006	0,011	0,015
Skaitymas. Duomenų nuskaitymas	0,017	0,036	0,018	0,017	0,021	0,019	0,013	0,013	0,030	0,014	0,020
Skaitymas. Duomenų apdorojimas	0,000	0,001	0,001	0,001	0,000	0,000	0,001	0,000	0,001	0,001	0,001

36 lentelė. 1 kB dydžio atsitiktinėmis reikšmėmis užpildyto failo paslėpimo, panaudojant 8 deng. failus, laikai sekundėmis

Failo vardas	Testas_4.bin										
Užpildymo tipas	Atsitiktinės reikšmės										
Failo dydis	1 kB										
Dengiačių failų skaičius	8										
Bandymo nr. / Žingsnis	1	2	3	4	5	6	7	8	9	10	Vid.
Rašymas. Inicializacija	3,352	2,733	2,864	2,732	2,679	2,758	2,703	2,333	2,894	3,464	2,851
Rašymas. Duomenų slėpimas	17,81	15,05	16,56	16,64	17,62	21,89	17,13	13,52	16,09	17,67	16,998
Rašymas. Likusių klasterių surašymas	327,7	337,9	334,8	338,6	328,3	337,5	322,4	382,3	344,4	339,8	339,370
Skaitymas. Inicializacija	0,010	0,128	0,016	0,015	0,017	0,012	0,011	0,010	0,016	0,015	0,025
Skaitymas. Duomenų nuskaitymas	0,012	0,018	0,017	0,010	0,010	0,012	0,011	0,012	0,016	0,032	0,015
Skaitymas. Duomenų apdorojimas	0,000	0,000	0,001	0,001	0,001	0,000	0,000	0,001	0,000	0,000	0,000

Atlikus laikinių charakteristikų eksperimentą, galima padaryti šias išvadas:

1. Paslėpimo bei nuskaitymo funkcijos turi pastovų laiko tarpą trunkantį inicializacijos etapą, kurio metu duomenys paruošiami paslėpimui, atliekami failų sistemos bei dengiančių failų patikrinimai.
2. Duomenų paslėpimo etapas trumpėja didėjant dengiančių failų skaičiui (reikia rašyti mažiau dengiančių failų klasterių, norint paslėpti tą patį kiekį informacijos).
3. Naudojant taktiką, kai surašomi visi dengiančių failų klasteriai, nenumetant nereikalingų, ilgėja paslėpimo etapas „likusių klasterių surašymas“. Taip atsitinka todėl, kad kiekvienas dengiantis failas privalo turėti pakankamai klasterių visiems slaptiems duomenims paslėpti. Didėjant dengiančių failų skaičiui, mažėja reikalingų dengiančių klasterių skaičius, tačiau daugėja dengiančių failų, kurių klasteriai galiausiai vistiek surašomi į FS šiame etape.
4. Paslėptų duomenų nuskaitymas yra labai greitas.
5. Eksperimentinę programą galima optimizuoti rašant duomenis maksimaliais galimais blokais, o ne visais atvejais po vieną klasterį. Tokia optimizacija stipriai padidintų duomenų rašymo greitį, ypač kai rašoma po kelis to pačio failo klasterius iš eilės.

4.6. Išvados

1. Sukurti ir praktiškai realizuoti veikiantys duomenų paslėpimo ir nuskaitymo algoritmai, naudojantys pasiūlytą metodą. Atliktas realizuoto metodo testavimas įvairiose situacijose ir įsitikinta, kad jis veikia.
2. Realizuota eksperimentinė programa, naudojanti siūlomą slapto kanalo organizavimo diskiniuose kaupikliuose metodą.
3. Atlikta eksperimentinė metodo analizė ir nustatyta, kad:
    1. Sukurtas metodas yra visiškai atsparus dengiančių failų papildymui bei keitimui, kai keičiant nemažėja failo dydis.
    2. Naudojant taktiką, kai nebereikalingi dengiančių failų klasteriai nenumetami, o surašomi į failų sistemą, gauname papildomą apsaugą nuo dengiančių failų keitimo, kai jų dydis mažėja. Tokiu atveju galime dengiančio failo dydį sumažinti tiek, kad liktų slėpimui panaudoti klasteriai. Tokių klasterių skaičius (įprastu atveju) mažėja, didinant dengiančių failų skaičių.
    3. Išmatuotos eksperimentinės programos laikinės charakteristikos. Nustatyta, kurie metodo etapai užima daugiausiai laiko ir numatyti būdai optimizuoti eksperimentinę programą.

5. IŠVADOS

1. Išanalizuota tyrimo sritis. Analizės metu nustatyta, kad FAT failų sistema yra lengviausiai pritaikoma duomenų slėpimui, panaudojant slapto kanalo metodus, o taip pat turi šiems metodams parankias savybes – aukštą failų fragmentaciją, populiarumą nešiojamose laikmenose. Taip pat apžvelgti egzistuojantys duomenų paslėpimo klasterinėse failų sistemose metodai ir nustatyti pagrindiniai jų trūkumai: negebėjimas pilnai užpildyti disko, neatsparumas darbui su failų sistema, sudėtingumas, dvigubo pagrįsto išsigynimo savybės trūkumas.
2. Pasiūlytas slapto kanalo metodas, pasižymintis šiomis savybėmis: galimybė pilnai užpildyti diską (duomenų slėpimui nereikalingi tarpai tarp dengiančių failų klasterių), didelė talpa (lyginant su kitais metodais, kai naudojami rekomenduojami parametrai), kurią galima didinti parenkant daugiau dengiančių failų, paprastumas (lengvai realizuojamas, nereikia spręsti kolizijų problemos), lyginant su analogiškai metodais, didesnė duomenų sauga dėl sudėtingo slapto rakto (svarbūs ne tik paslėptų duomenų kiekis ir dengiančių failų vardai, bet ir jų eilės tvarka, inicializacijos vektorius).
3. Atlikta eksperimentinė realiai naudotų failų sistemų analizė. Jos metu nustatyta, kad metodo sukeliama fragmentacija bei failų klasterių susipynimai yra realiai aptinkami ir „natūraliomis“ sąlygomis. Taip pat atlikta trijų skirtingų situacijų simuliacija, norint nustatyti, kada „natūraliomis“ sąlygomis kyla failų klasterių susipynimai. Panaudojus eksperimentinę programą surasta, kad klasterių susipynimas kyla, kai programos papildymo būdu rašo į failų sistemoje vienas šalia kito esančius failus.
4. Eksperimentinis tyrimas parodė, kad, norint metodo panaudojimą išlaikyti nepastebimą, reikėtų pasirinkti FAT32 failų sistemą, naudotą ilgiau naudotoje operacinėje sistemoje, nes šiuo atveju failai yra labiau linkę fragmentuotis bei susipinti klasteriais tarpusavyje. Taip pat, reikėtų pasirinkti tinkamus dengiančius failus. Tiek naudotų diskų analizė, tiek situacijų simuliacija porodė, kad geriausiai tam tinkami dažnai papildomi failai (žurnalo, duomenų bazės, archyvo ir pan.). Laikantis šių rekomendacijų metodas nesukelia anomalijų failų sistemoje ir išlieka nepastebimas disko analizės metu, t.y. pasižymi dvigubo pagrįsto išsigynimo savybe.
5. Eksperimentinės programos pagalba nustatyti būdai pagerinti metodo našumą. Laikinių charakteristikų tyrimas parodė, kad metodo spartą galima padidinti rašant po kelis klasterius vienu metu bei nerašant perteklinių dengiančių failų klasterių.
6. Nustatyta, kad nesvarbu kurį tekstinio failo dalis trinama, fizinė jo vieta visada traukiasi nuo galo, t.y. ištrynus failo pradžią, prarandami paskutiniai jo klasteriai. Taigi, jei naudojami dengiantys failai turi perteklinių klasterių, kuriais nėra slepiami duomenys, gaunamas atsparumas dalies failo turinio ištrynimui (bet kurioje failo vietoje).
7. Nustatyta, kad metodas yra atsparus dengiančių failų papildymui, keitimui, iš dalies – turinio trynimui. Tačiau visiškai neatsparus failų sistemos defragmentacijai. Nepasaisant to, kad defragmentavimo metu pagrindinė metodo esmė – duomenų slaptumo išsaugojimas – nėra pažeidžiama, verta panagrinėti galimybę svarbius dengiančių failų klasterius pažymėti, kaip „nejudinamus“ tam, kad defragmentavimo įrankis jų neperkėlinėtų ir nesunaikintų paslėptų duomenų.

6. LITERATŪRA

1. Anderson R. J. Information Hiding Terminology - Results of an Informal Plenary Meeting and Additional Proposals. // Information Hiding, Springer, 1996, P. 347–350.
2. Anderson R. J., Needham R. M., Shamir A. The Steganographic File System. // Proceedings of the Second International Workshop on Information Hiding, London, UK: Springer-Verlag, 1998, P. 73–82.
3. Bellare M., Boldyreva A. The Security of Chaffing and Winnowing. // Proceedings of the 6th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology, London, UK, UK: Springer-Verlag, 2000, P. 517–530.
4. Carrier B., File system forensic analysis. - Crawfordsville: R. R. Donnelley, 2005. – 382 p. ISBN 0-32-126817-2.
5. Czeskis A., Hilaire D. J. S., Koscher K., Gribble S. D., Kohno T., Schneier B. Defeating Encrypted and Deniable File Systems: TrueCrypt v5.1a and the Case of the Tattling OS and Applications. // HotSec, USENIX Association, 2008, P. 1–7.
6. D. de Nijs, Biesheuvel A., Denissen A., Lambert N. The effects of filesystem fragmentation // „Proceedings of the Linux Symposium“, konferencija (2006; Otava, Kanada). Proceedings of the Linux Symposium. Volume One. – Otava, 2006.
7. Dakhane D. M., Patil S., Patil M. Detection and elimination of covert communication in Transport and Internet layer - A Survey. // IJCA Proceedings on International Conference on Recent Trends in Information Technology and Computer Science (ICRTITCS-2011), New York, USA: Foundation of Computer Science, 2012, P. 36–41.
8. Douceur J. R., Bolosky W. J. A Large-scale study of file-system contents // „Proceedings of the international conference on Measurement and modeling of computer systems“, konferencija (1999; Atlanta, JAV). Microsoft Research. –Redmond, 1999.
9. Eckstein K., Jahnke M., Data Hiding in Journaling File Systems // Eingereicht beim Digital Forensic Research Workshop. – 2005, p. 1-8.
10. FAT16/32 File System Library [interaktyvus]. 2010. Žiūrėta 2012 m. sausio 15 d. Prieiga per internetą: <http://ultra-embedded.com/?fat_filelib>.
11. Garfinkel S. L. Carving contiguous and fragmented files with fast object validation // Digital Investigation. – ISSN 17422876. – 2007, nr. 4, p. 2-12.
12. Hanaoka G., Hanaoka Y., Hagiwara M., Watanabe H., Imai H. Unconditionally secure chaffing-and-winnowing: a relationship between encryption and authentication. // Proceedings of the 16th international conference on Applied Algebra, Algebraic Algorithms and Error-Correcting Codes, Berlin, Heidelberg: Springer-Verlag, 2006, P. 154–162.
13. Huebner E., Bem D., Wee C. K. Data hiding in the NTFS file system. // Digital Investigation, 2006 Dec, Vol. 3, No. 4, P. 211–226.
14. Khan H., Javed M., Khayam S. A., Mirza F. Designing a cluster-based covert channel to evade disk investigation and forensics // Computers Security. – ISSN 01674048. – 2011, nr. 30, p. 35-49.
15. McDonald A. D., Kuhn M. G. StegFS: a steganographic file system for Linux // Lecture notes in computer science. – ISBN 978-3-540-67182. – Heidelberg, 2000, nr. 1768, p. 463-477.
16. Pang H., Tan K. L., Zhou X. StegFS: a steganographic file system // „Data Engineering, 2003“. – Singapore, 2003. – ISBN 0-7803-7665-X. –p 657-667.
17. Petitcolas F. A. P., Anderson R. J., Kuhn M. G. Information hiding – a survey // Proceedings of the IEEE. – ISSN 0018-9219. – 1999, nr. 87, p. 1062-1078.
18. Pfitzmann B. Information hiding terminology // „Results of an informal plenary meeting and additional proposals“. – Anderson, 1996. - ISBN 3-540-61996-8. – p. 347-350.
19. Piper S., Davis M., Manes G., Shenoi S. Detecting Hidden Data in Ext2/Ext3 File Systems. // IFIP Int. Conf. Digital Forensics, Springer, 2005, P. 245–256.

20. Provos N., Honeyman P. Hide and seek: an introduction to steganography // Security & Privacy, IEEE. – ISSN 1540-7993. – 2003, nr. 1, p. 32-44.
21. Simmons G. J. The prisoners problem and the subliminal channel // Advances in Cryptology Proceedings of Crypto 83. – 1984, nr. 83, p. 51-67.
22. Smith K. A. Workload-specific file system benchmarks: daktaro disertacija, kompiuterių mokslas, filosofija // Harvard University. – Cambridge, 2001. – 159 p. – Aut. darbas.
23. The Sleuth Kit: Description [interaktyvus]. 2012. Žiūrėta 2012 m. sausio 15 d. Prieiga per internetą: <http://www.sleuthkit.org/sleuthkit/desc.php>.
24. thinfat32 – An implementation of the FAT32 filesystem specification for embedded systems [interaktyvus]. 2011. Žiūrėta 2012 m. sausio 15 d. Prieiga per internetą: <http://code.google.com/p/thinfat32/>.

7. PRIEDAS NR. 1. PUBLIKACIJOS MAGISTRO DARBO TEMA

Buvo publikuoti ar pateikti publikacijai šie straipsniai:

1. Petraitis G., Morkevičius N., Disko analizei atsparus duomenų slėpimo metodas. // Informacinės technologijos, Kaunas. – ISSN 2029-249X. – 2013, p. 102-107.
2. Morkevičius N., Petraitis G., Venčkauskas A., Covert Channel for Cluster-based File Systems Using Multiple Cover Files. 2013 m. sausio mėn. pateiktas žurnalo "Information technology and control" redakcijai. Gautas vieno recenzento teigiamas įvertinimas.

INFORMACINĖS TECHNOLOGIJOS • IVUS 2013 • ISSN 2029-249X • eISSN 2029-4824

Disko analizei atsparus duomenų slėpimo metodas

Grigas Petraitis, Nerijus Morkevičius
Kompiuterių katedra
Kauno technologijos universitetas
Kaunas, Lietuva
grigas.petraitis@stud.ktu.lt

Santrauka. ***Šiame darbe analizuojama slaptos informacijos slėpimo diskiniuose kompiuterių kaupikliuose problema. Straipsnyje pasiūlytas ir išanalizuotas naujas metodas, skirtas informacijos slėpimui klasterinėse failų sistemose. Metodas naudoja kelis dengiančiuosius failus ir informaciją saugo keisdamas šių failų klasterių tarpusavio padėtį. Eksperimento rezultatai patvirtina, jog pasiūlytas metodas turi dvigubo pagrįsto išsigynimo savybę.***

Reikšminiai žodžiai: slaptas kanalas, steganorgrafija, duomenų slėpimas, FAT, failų sistema, klasteris.

I. Įvadas

Duomenų saugumas bei skaitmeninės informacijos saugus perdavimas yra labai aktuali tema. Dažniausiai slapta informacija yra užšifruojama panaudojant kriptografinius algoritmus, kurie yra pastoviai tobulinami ir papildomi naujais metodais [1, 2]. Visgi kai kuriais atvejais duomenų šifravimas panaudojant kriptografinius metodus neužtikrina pakankamos jų saugos. Tradiciškai užšifruotų duomenų egzistavimo faktas nėra slepiamas arba yra nesunkiai aptinkamas, todėl jų savininkas gali būti priverstas atskleisti šifravimo raktą [3, 4]. Vienas iš tokios situacijos pavyzdžių yra B. W. Lampson darbe [5] aprašyta „kalinių problema“. Norint išspręsti šią problemą, galima panaudoti duomenų slėpimo metodus, tokius kaip steganografija ar slaptas kanalas [3]. Steganografiniai metodai paslepia duomenis „įprastoje“ dengiančioje terpėje taip, kad nesukeltų galimo stebėtojo įtarimo [4], pvz., slaptą tekstinį pranešimą galima paslėpti skaitmeniniame vaizde. Slapti kanalai leidžia paslėpti duomenis terpėje, kuri nėra skirta duomenų perdavimui ar saugojimui. Tikslui pasiekti jie dengiančią informaciją perduoda tam tikru būdu išdėstytą laike, laikmenoje ar kitoje terpėje taip, kad gavėjas tik pagal jos išdėstymą gali išgauti slaptą pranešimą [6, 7]. Dažniausiai duomenų slėpimo metodai skirti naudoti informacijos perdavimo protokoluose [8, 9, 10], tačiau egzistuoja daugelis darbų [11, 12, 13, 14, 15, 16], kurie aprašo efektyvius duomenų slėpimo metodus kietuosiuose kompiuterių diskuose.

H. Khan et al. savo darbe [7] įveda du saugos metodų, skirtų duomenų slėpimui, efektyvumo įvertinimo lygius – tai pagrįstas išsigynimas ir dvigubas pagrįstas išsigynimas. Šios sąvokos apibūdina kaip efektyviai metodas gali paslėpti duomenis. Pagrįstas išsigynimas yra tokia saugos metodo savybė, kuri vienai iš šalių leidžia pagrįstai teigti kitoms šalims (pvz., teisėjui arba tyrėjui), kad paslėpti duomenys neegzistuoja [14]. Dažnai šia sąvoka apibūdinami ir metodai, leidžiantys šaliai atskleisti tik dalį paslėptų ir/ar užšifruotų duomenų ir sėkmingai paneigti bet kokių kitų duomenų egzistavimą. Šiuo atveju pats paslėptų duomenų egzistavimo faktas nėra efektyviai paslepiamas. Pavyzdžiui, didžioji dalis steganografinių metodų slepia duomenis nenaudojamose kietojo disko vietose [14, 15, 16, 17, 18]. Tyrėjas, atliekantis disko analizę, gali lengvai rasti nenaudojamus disko sektorius, užpildytus tam tikrais duomenimis, bet nežinodamas rakto, jis negali įrodyti, jog tai paslėpti svarbūs duomenys, o ne atsitiktiniai duomenys įrašyti anksčiau dėl intensyvaus disko naudojimo. Visi steganografiniai metodai turi tik pagrįsto išsigynimo savybę.

Dvigubo pagrįsto išsigynimo savybę turi tokie duomenų slėpimo metodai, kurie užtikrina, kad šalis gali du kartus pagrįstai paneigti paslėptų duomenų egzistavimą. Pirmuoju bandymu galima neigti, kad išvis egzistuoja bet kokie paslėpti duomenys. Nepavykus pirmajam bandymui, šie metodai palieka galimybę dar kartą paneigti duomenų (ar svarbesnės jų dalies) egzistavimą taip pat kaip viengubo pagrįsto išsigynimo atveju. Šią, dvigubo pagrįsto išsigynimo, savybę turi tik slapto kanalo metodai, kurie slėpdami informaciją į terpę neįrašo jokių papildomų duomenų. Du slapto kanalo metodai, tinkami naudoti kietuosiuose diskuose ir turintys dvigubo pagrįsto išsigynimo savybę, aprašyti [7].

Šiame straipsnyje aprašytas naujas, efektyvus ir lengvai įgyvendinamas metodas jautrios informacijos paslėpimui diskiniuose kaupikliuose, naudojančiuose klasterinę FAT [19] failų sistemą. Pasiūlytas metodas buvo išanalizuotas saugos požiūriu. Straipsnyje pateikti atliktų eksperimentinių tyrimų rezultatai. Eksperimentų rezultatų analizė leidžia daryti išvadą, jog siūlomas metodas pasižymi dvigubo pagrįsto išsigynimo savybe.

II. Egzistuojantys duomenų paslėpimo klasterinėse failų sistemose metodai

Keletas steganografinių metodų, slepiančių duomenis neužimtose failų sistemos struktūrose, aprašyti [11, 12, 17] darbuose. Anderson et al. [16] darbe pasiūlė steganografinę failų sistemą, naudojančią du skirtingus metodus informacijos paslėpimui, užpildant dengiančius failus atsitiktiniais duomenimis. Kiti autoriai [14, 15] dar labiau patobulino šiuos metodus. Visose metodo modifikacijose į įvairias failų sistemos duomenų struktūras rašoma informacija, kuri gali būti lengvai aptikta žemame lygyje analizuojant diską, todėl visi šie metodai pasižymi tik viengubo pagrįsto išsigynimo savybe.

H. Khan et al. pasiūlė [7] du slapto kanalo metodus, pasižyminčius dvigubo pagrįsto išsigynimo savybe ir skirtus paslėpti duomenis FAT failų sistemoje. Šie metodai į diską

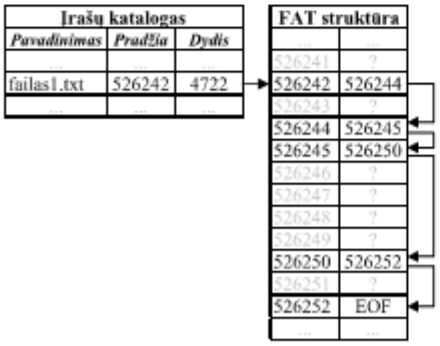
neįrašo jokių papildomų duomenų, o informacijos saugojimui naudoja FAT failų sistemos klasterių, priklausančių dengiančiajam failui, išdėstymą (tarpus tarp gretimų failo klasterių). Kadangi tam tikrai reikšmei paslėpti būtinas tam tikro dydžio tarpas, kyla kolizijų problema, kurios sprendimai yra gana sudėtingi. Be to, kolizijų kiekis stipriai išauga, kai failų sistema yra užpildyta duomenimis. Dėl to neįmanoma iki galo užpildyti diskų. Abejų metodų talpa (matuojama bitais, kuriuos galima paslėpti vienu dengiančiojo failo klasteriu) labai krenta, kai failų sistema yra artipilnė. Iš anksto įvertinti, kiek naudingų duomenų bus galima paslėpti duotajame diske, neįmanoma, nes tai taip pat priklauso net ir nuo pačių duomenų.

III. FAT FAILŲ SISTEMŲ STRUKTŪRA

FAT šeimos failų sistemos yra palaikomos daugumoje OS, dėl paprastumo ir suderinamumo jos yra naudojamos daugumoje šiuolaikinių prietaisų (televizorių, vaizdo grotuvų ir t. t.). FAT yra labai populiari nedidelės talpos nešiojamuose atmintinėse, dėl to kompanija Microsoft neseniai sukūrė naują FAT išplėtimą (exFAT), skirtą naudoti didesnės talpos diskuose.

FAT failų sistemas [19] sudaro tik 2 iš 5 kitose failų sistemose aptikamų duomenų struktūrų – įrašų katalogo struktūra bei failų paskirstymo lentelės (FAT). Grafiškai ryšys tarp šių struktūrų pavaizduotas 1 pav. Schemoje matome, kad kiekvienas failas turi įrašą katalogo struktūroje, kurioje, be failo vardo, dar saugomas jo dydis, pirmojo klasterio adresas bei kiti metaduomenys. Failų bei katalogų duomenys yra saugomi duomenų blokuose, vadinamuose klasteriais. Jei failas ar katalogas užima daugiau nei vieną klasterį, kiti klasteriai yra randami naudojant struktūrą, vadinamą FAT. Ji yra naudojama kito, failui priklausančio, klasterio radimui bei klasterių vietų užimtumo nustatymui.

Klasterių paskirstymo failams algoritmas priklauso nuo operacinės sistemos, tačiau dauguma sistemų naudoja „pirmas laisvas“ algoritmą, kai imamas pirmas pasitaikęs laisvas klasteris po jau užimtojo. Taip daroma todėl, kad diskiniai kaupikliai daug sparčiau dirba su klasteriais, kai jie yra šalia vienas kito. Tačiau dėl aktyvaus darbo su failų sistema dažnai nepavyksta rezervuoti visų klasterių iš eilės – susidaro failų sistemos fragmentacija. Tokiu atveju visus failo klasterius rasti pavyksta naudojant FAT struktūrą, panašiai kaip dinaminį sąrašą (1 pav.).



1 pav. Ryšys tarp įrašų katalogo ir FAT struktūros

IV. SIŪLOMAS METODAS

Pagrindinė siūlomo duomenų slėpimo FAT failų sistemoje metodo idėja - panaudoti daugiau nei vieną dengiantį failą ir slaptą žinutę perduoti panaudojant dengiančių failų klasterių tarpusavio išsidėstymą. Tokiu atveju į kitus failų sistemos duomenis galima nekreipti dėmesio ir analizuoti tik dengiančių failų klasterius. Galima sudaryti didėjimo tvarka surikiuotą masyvą su dviejų dengiančių failų klasterių numeriais ir stebėti, kaip vieno failo klasteriai „pinasi“ su klasteriais, priklausančiais kitam failui. Jei kažkurioje masyvo vietoje po pirmojo failo klasterio eina kito failo klasteris, tai atitinkamoje paslėptos žinutės vietoje yra bitas „1“. Jei kažkurioje masyvo vietoje yra du iš eilės to paties failo klasteriai, tai reiškia, jog paslėptoje žinutėje yra „0“. Tokiu būdu galima paslėpti visą slaptą žinutę neįrašant nei vieno bito į diską.

Norint padidinti slapto kanalo talpą, galima panaudoti daugiau dengiančiųjų failų. Tada kiekvienam failui reikia skirti numerius ir kiekvienam dengiančių failų klasterių masyvo elementui apskaičiuoti skirtumą tarp einamojo ir prieš tai buvusio failo numerių. Šis skirtumas ir bus atitinkamų paslėptos žinutės bitų reikšmė. Naudojant tokią taktiką galima paslėpti daugiau nei vieną slaptą bitą viename dengiančio failo klasteryje. Tokiu atveju dengiančių failų vardai bei jų pateikimo eilės tvarka tampa slaptojo rakto dalimi, nes juos būtina žinoti, norint nuskaityti paslėptus duomenis.

V. DUOMENŲ PASLĖPIMO ALGORITMAS

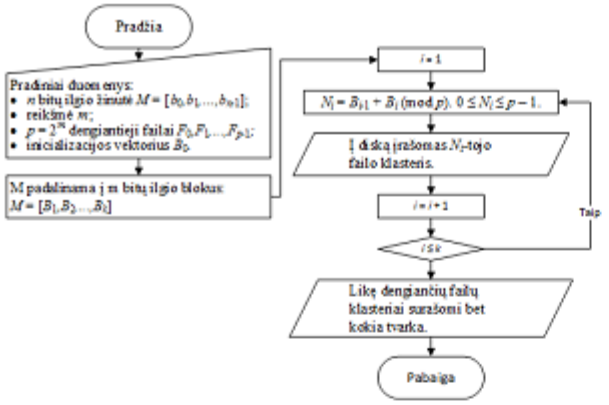
Tarkime, norime paslėpti n bitų ilgio žinutę $M=[b_0,b_1,\ldots,b_{n-1}]$. Pasirenkame $p=2^m$ dengiančiųjų failų $F_0,F_1,\ldots,F_{p-1}$. Kiekvienam iš šių failų priskiriame sveiką skaičių (failo eilės numerį). Įrašydami vieną reikiamo failo klasterį į diską galėsime paslėpti m žinutės bitų, todėl žinutę reikia suskaidyti į m bitų ilgio blokus $M=[B_1,B_2,\ldots,B_k]$ ir juos apdoroti atskirai, pradedant nuo pirmojo. Jei žinutės ilgis nesutampa su sveiku blokų skaičiumi, tai žinutę reikia papildyti nuliniais bitais iki užpildyto bloko. Kiekvienas bitų blokas B_i, $i=0,1,\ldots,k$ interpretuojamas kaip natūralusis skaičius. Prieš pradedant slėpimą, reikia laisvai pasirinkti nulinio bloko reikšmę B_0. Jei slepiamos kelios žinutės iš eilės, tai šia reikšme galima pasirinkti paskutinio prieš tai apdorotos žinutės bloko reikšmę.

Iš eilės peržiūrimi visi pradinės žinutės blokai B_i $i=1,2,\ldots,k$, ir dengiančiojo failo eilės numeris N_i apskaičiuojamas pagal formulę:

$$N_i := B_{i-1}+B_i \pmod p,\ 0\le N_i\le p-1. \qquad (1)$$

Pradedant nuo pirmojo laisvo disko klasterio, į diską įrašomas eilinis dar neįrašytas N_i-tojo failo klasteris. Procedūra kartojama tol, kol paslepiama visa žinutė.

Duomenų paslėpimo algoritmo blokinė diagrama pateikta 2 pav.



2 pav. Duomenų paslėpimo algoritmo blokinė diagrama

Norint atkurti pradinę žinutę, paslėptą diske, būtina žinoti tokią informaciją:

- reikšmę m;
- slaptos žinutės ilgį bitais n;
- inicializacijos vektorių B_0;
- visų dengiančiųjų failų $F_0, F_1, \ldots, F_{p-1}$, vardus ir jų pateikimo algoritmui tvarką.

Visa ši atkūrimui būtina informacija sudaro siūlomo slapto kanalo metodo slaptąjį raktą, be kurio neįmanoma atkurti pradinės žinutės net ir turint diską, kuriame ji įrašyta.

Duomenų nuskaitymo algoritmas yra analogiškas. Tarkime, kad žinoma visa informacija, sudaranti algoritmo slaptąjį raktą. Tuščias bitų masyvas M suskaidomas į k blokų po m bitų, $M = [B_1, B_2, \ldots, B_k]$. Kiekvienas blokas B_i interpretuojamas kaip natūralusis skaičius.

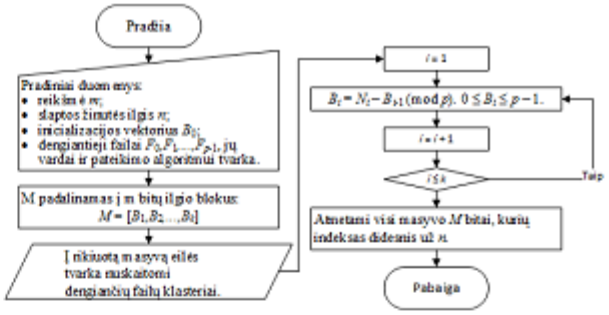
Disko, kuriame paslėpta žinutė, failų sistema analizuojama pradedant nuo mažiausio klasterio numerio. Tiriami tik tie klasteriai, kurie priklauso vienam (bet kuriam) iš dengiančiųjų failų. Iš tokių klasterių numerių galima sudaryti surikiuotą masyvą.

Kiekvienam atkuriamos žinutės blokui B_i, $i = 1,2\ldots,k$ pagal surikiuoto klasterių masyvo i-tąjį elementą, panaudojant FAT failų sistemos struktūras, randamas atitinkamo dengiančiojo failo numeris N_i. Bitų bloko B_i reikšmė apskaičiuojama pagal formulę:

$$B_i := N_i - B_{i-1} \pmod p,\ 0 \le B_i \le p-1. \qquad (2)$$

Procedūra kartojama tol, kol apdorojami visi blokai. Rezultato masyvo M bitai, kurių indeksas didesnis už n, atmetami.

Duomenų nuskaitymo blokinė diagrama pateikta 3 pav.



3 pav. Duomenų nuskaitymo algoritmo blokinė diagrama

Reikėtų atkreipti dėmesį į dengiančiųjų failų ilgiams taikomus apribojimus. Visų pirma, maksimalią visų dengiančių failų dydžio sumos reikšmę apibrėžia likusi laisva failų sistemoje vieta. Antra, galimos dvi strategijos, naudojamos rašyti dengiančius failus – pabaigus rašyti slaptam pranešimui paslėpti reikalingus klasterius, likusius dengiančiųjų failų klasterius tiesiog „išmesti“, t. y. neberašyti į diską. Antra strategija – įrašyti visus likusius dengiančiųjų failų klasterius.

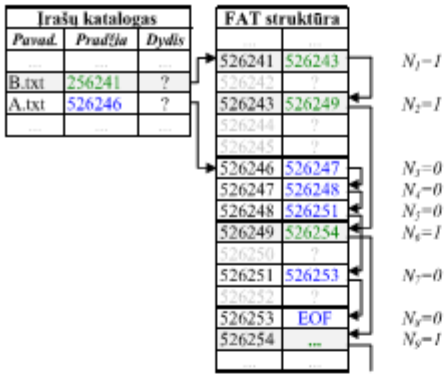
Dengiančiųjų failų pakankamo ilgio problema kyla todėl, kad siūlomas metodas, priklausomai nuo slepiamų duomenų, gali nevienodai „panaudoti“ skirtingų failų klasterius. Egzistuoja išskirtinės situacijos, kai visas pranešimas gali būti paslepiamas naudojant tik vieno failo klasterius, nors dengiančiųjų failų reikia daug daugiau. Dėl tokio galimo failų panaudojimo asimetriškumo, jei nenorime naudoti pirmosios strategijos, būtina užtikrinti, kad kiekvienas failas būtų pakankamo ilgio paslėpti visą pranešimą arba prieš jį slepiant apskaičiuoti kiekvieno failo reikiamą dydį atliekant žinutės paslėpimo simuliaciją, t.y. suskaičiuoti, kiek kiekvieno failo klasterių reikės, nerašant jų į failų sistemą.

VI. Pavyzdys

Tarkime, norime paslėpti dvejetainę žinutę $M = [1,0,0,0,0,1,1,1,0]$. Parenkame duomenis slaptam raktui: $B_0 = 0$, $m = 1$, $p = 2^m = 2$. Tai reiškia, kad mums reikės dviejų dengiančių failų – imame „A.txt“ ir „B.txt“, jų indeksai masyve bus atitinkamai 0 ir 1. Kadangi $m = 1$, t. y. slėpsime po vieną bitą vienu dengiančių failų klasteriu, tai žinutės M blokas B_i yra lygus vienam žinutės bitui.

Pagal (1) formulę paskaičiuojame, kurio failo klasterį rašysime pirmą: $N_1 := B_0 + B_1 \pmod p = 0 + 1 \pmod 2 = 1$. Į pirmą laisvą disko klasterį įrašome failo su pirmu indeksu („B.txt“) pirmąjį klasterį ir algoritmą vykdome toliau. Analogiškai $N_2 = 1 + 0 \pmod 2 = 1$, todėl į antrąjį nuo disko pradžios laisvą klasterį rašysime pirmojo failo antrąjį klasterį. Nesunku apskaičiuoti, kad $N_3 = 0$, $N_4 = 0$ ir t. t. Paslėpus visus bitų blokus, pabaigiame rašyti dengiančius failus bet kokia tvarka (arba likusių jų dalių visai neberašome į diską).

4 pav. pateiktoje schemoje matoma kaip atrodo FAT struktūra, paslėpus duotąją žinutę.



4 pav. FAT struktūra po duomenų paslėpimo operacijos

Norėdami nuskaityti slaptą pranešimą, turime žinoti paslėpimui naudotus parametrus. Kadangi žinome, kurių failų klasteriai slepia duomenis, išrenkame juos į atskirą masyvą ir žiūrime, kuriam failui priklauso pirmasis masyvo elementas. Tuomet pagal (2) formulę paskaičiuojame pirmojo bitų bloko reikšmę $B_1 = N_1 - B_0 \pmod p = 1 - 0 \pmod 2 = 1$. Kitų blokų reikšmės paskaičiuojamos analogiškai: $B_2 = N_2 - B_1 \pmod p =$ $= 1 - 1 \pmod 2 = 0$, $B_3 = 0 - 0 \pmod 2 = 0$, $B_4 = 0$, $B_5 = 0$, $B_6 = 1$, $B_7 = 1$, $B_8 = 1$, $B_9 = 0$. Bitų blokus surašius į masyvą, gauname nuskaitytą pranešimą $M = [1,0,0,0,0,1,1,1,0]$

VII. Metodo talpos įvertinimas

Siūlomas metodas pasižymi pastovia $C = \log_2 p$ talpa. Čia C yra slapto kanalo talpa matuojant bitais, paslepiamais vienu dengiančio failo klasteriu, p – dengiančių failų skaičius. Ši išraiška teisinga tik tuo atveju, kai nebūtina į diską pilnai įrašyti kiekvieno dengiančiojo failo. Vaizdesnė algoritmo talpos priklausomybė nuo naudojamų dengiančių failų skaičiaus pavaizduota 1 lentelėje.

1 lentelė. Talpos priklausomybė nuo dengiančių failų skaičiaus

Naudojamų dengiančių failų skaičius p	Slaptų bitų skaičius viename dengiančio failo klasteryje C
2	1
4	2
8	3
16	4
32	5
64	6
128	7
256	8

VIII. Kokybinių charakteristikų įvertinimas

Metodo kokybinių charakteristikų analizės metu įvertintas paslėptų duomenų atsparumas pokyčiams failų sistemoje ir dengiančiuose failuose. Taip pat, remiantis įvairiais kriterijais, įvertintas metodo saugumas.

A. Paslėptų duomenų stabilumas

Steganografiniai metodai, kurie remiasi slaptos informacijos saugojimu nenaudojamose failų sistemos vietose, yra neatsparūs paslėptų duomenų sugadinimui, kai su failų sistema dirba apie slaptus duomenis nieko nežinanti OS ar kitos programos. Pavyzdžiui, naujai sukūrus failą, jo duomenys gali perrašyti prieš tai „laisvose“ disko vietose paslėptą informaciją.

Mūsų siūlomas metodas nesaugo jokios papildomos informacijos failų sistemoje, todėl yra atsparus įprastam darbui su failų sistema (failų kūrimui, ne dengiančių failų keitimui, ištrynimui). Visgi kai kurie žemo lygio darbui su disku skirti įrankiai, tokie kaip disko defragmentavimo programos, keičia jau esančių failų klasterių išsidėstymą ir gali sunaikinti paslėptą informaciją. Dengiančių failų papildymas ar kitoks keitimas gali sukelti klasterių išdėstymo pakeitimą, o tuo pačiu ir slaptų duomenų vientisumo pažeidimą. Galiausiai vieno iš dengiančių failų pervardinimas, perkėlimas, nukopijavimas ar ištrynimas garantuos paslėptų duomenų sunaikinimą.

B. Saugumo įvertinimas

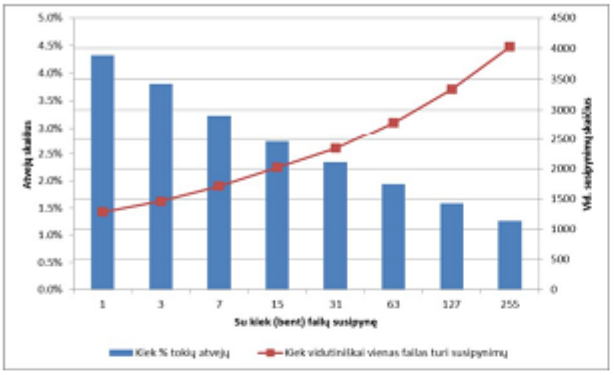
Siūlomas slapto kanalo metodas yra atsparus žemo lygio disko analizei. Diską analizuojantis tyrėjas neras jokių papildomų duomenų „nestandartinėse“ disko vietose, visos failų sistemos struktūros bus korektiškos ir visiškai atitiks FAT specifikaciją. Vienintelis šalutinis metodo naudojimo efektas yra padidėjusi dengiančių failų fragmentacija. Tačiau fragmentacija aptinkama ir natūraliomis sąlygomis, kai failų sistema yra naudojama ilgą laiką ir joje vykdomas aktyvus darbas su failais. Taigi, norint įvertinti pasiūlyto metodo saugumą, reikėtų atsakyti į klausimą: su kokiais parametrais turėtų būti naudojamas pasiūlytasis metodas, kad sukeliama fragmentacija būtų tokia pati, kaip ir kylanti įprasto failų sistemose naudojimo atveju?

IX. Metodo saugumo eksperimentinis tyrimas

Eksperimentinio tyrimo metu buvo išanalizuoti universiteto laboratorijose naudojami kompiuterių diskai norint nustatyti, kaip failai fragmentuojasi kasdienio naudojimo sąlygomis. Pradinių duomenų apie failų sistemą nuskaitymui naudotas „The Sleuth Kit“ [20, 21]. Vėliau šie duomenys buvo įkelti į duomenų bazę, kurioje SQL užklausų pagalba buvo siekiama nustatyti, kiek tiriamose failų sistemose buvo fragmentuotų ir susipynusių failų ir kokiais atvejais jie susipina. Norint suprasti eksperimentų rezultatus, reikia formaliai apibrėžti susipynusių failų sąvoką: du failai F_1 ir F_2 laikomi tarpusavyje susipynusiais, jeigu egzistuoja bent viena situacija, kai tarp failo F_1 dviejų gretimų klasterių yra bent vienas failo F_2 klasteris ir atvirkščiai (tarp dviejų gretimų failo F_2 klasterių yra bent vienas failo F_1 klasteris). Jei ši situacija diske aptinkama keletą kartų, sakoma, kad failai susipynę atitinkamą skaičių kartų. Suprantama, failas gali būti susipynęs ne tik su vienu, bet su keliais kitais failais. Gretimais failo klasteriais laikomi iš eilės einantys to failo klasteriai (pagal failo turinį) ir turintys didėjančius klasterių numerius (dengiantiems failams ši sąlyga bus tenkinama visiems klasteriams, kitiems failams – ne visada).

Atlikta diskų analizė parodė, kad 4,33% visų failų yra susipynę bent su vienu kitu failu (šie failai turėjo vidutiniškai po 1284 susipynimų), 3,80% su bent trim kitais failais (vid. po 1461 susipynimų), 3,23% su bent septyniais kitais failais (vid.

po 1715 susipynimų), o 1,26% analizuotų failų buvo susipynę su 255 ir daugiau kitų failų, ir kiekvienas failas turėjo vid. po 4030 susipynimų (5 pav.). Šie rezultatai parodo, kad panaudojant didesnį dengiančių failų skaičių ne tik padidiname metodo talpą (1 lentelė), bet taip pat gauname didesnį susipynimų skaičių, kas leidžia prisidengiant „natūraliomis sąlygomis“ paslėpti daugiau duomenų.



5 pav. Susipynimo atvejų ir susipynimų priklausomybė nuo failų skaičiaus

Kitas eksperimentinių tyrimų tikslas buvo nustatyti, kokiose failų sistemose, operacinėse sistemose bei kokio tipo ir dydžio failai yra labiausiai linkę susipinti.

Atlikta analizė atskleidė, kad, lyginant su FAT32 tipo failų sistemomis, susipynusių failų daugiau turi NTFS tipo FS (2 lentelė). Taip pat 3 lentelėje matome, kad daugiausia susipynusių failų bei susipynimo atvejų aptikta failų sistemose, naudotose Windows XP bei Windows Vista operacinėse sistemose.

2 lentelė. FAILŲ SUSIPYNIMO PRIKLAUSOMYBĖ NUO FS TIPO

FS tipas	Vid. failo dydis, klast.	Su kiek vid. kitų failų susipynęs vienas failas	Vid. susipynimo atvejų skaičius
NTFS	31,45	31,68	115,20
FAT32	42,38	1,53	6,41

3 lentelė. FAILŲ SUSIPYNIMO PRIKLAUSOMYBĖ NUO OS

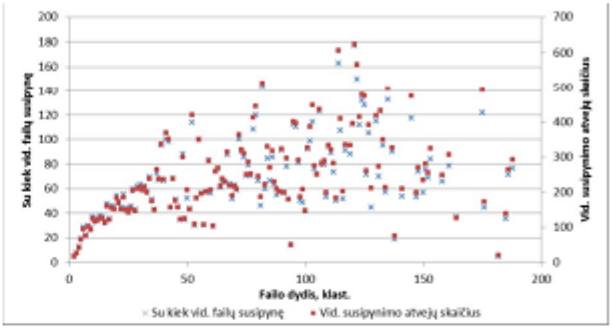
OS	Vid. failo dydis, klast.	Su kiek vid. kitų failų susipynęs vienas failas	Vid. susipynimo atvejų skaičius
Windows XP	39,25	19,79	70,21
Windows Vista	26,20	15,66	61,61
Windows 98	35,12	2,01	9,05

4 lentelėje pateikti 10 daugiausiai susipynimo atvejų turinčių failų tipų. Matome, kad „natūraliomis sąlygomis“ labiausiai susipina archyvo bei įrašų (duomenų bazės, žurnalo) tipo failai. Taigi norint saugiai paslėpti duomenis geriausia naudoti tokių tipų dengiančiuosius failus.

4 lentelė. FAILŲ SUSIPYNIMO PRIKLAUSOMYBĖ NUO FAILO TIPO

Failo tipas	Vid. failo dydis, klast.	Su kiek vid. kitų failų susipynęs vienas failas	Vid. susipynimo atvejų skaičius
.msi	892,87	326,31	1349,40
.zip	2152,61	126,90	589,36
.jar	204,50	133,79	500,25
.msp	2319,98	114,41	458,41
.log	40,51	102,39	458,12
.msf	39,50	106,04	396,75
.cab	884,87	80,78	367,79
.otp	46,48	100,27	326,33
.db	67,38	75,85	274,27
.frx	7,37	61,34	202,82

6 pav. pateiktame grafike matome kaip keičiasi vidutiniai supynusių failų bei susipynimo atvejų skaičiai didėjant failą sudarančių klasterių skaičiui.



6 pav. Susipynusių failų bei susipynimo atvejų skaičiaus priklausomybė nuo failo dydžio

Kita eksperimentinių tyrimų dalis buvo skirta nustatyti, ar įprastai dirbant su failų sistemomis (naudojant tik standartines OS funkcijas ir tiesiogiai nekeičiant žemo lygio disko ir failų sistemų struktūrų) susidaro situacijos, kai du ar daugiau failų tarpusavyje susipina daugelį kartų. Tam tikslui buvo sumodeliuotos trys situacijos Windows 7, Windows XP ir Ubuntu 11.04 operacinėse sistemose bei FAT32 ir NTFS failų sistemose:

1. Du failai buvo vienu metu kopijuojami į tą pačią failų sistemą OS priemonėmis. Šiam bandymui įgyvendinti buvo parašyti scenarijai, kurie vienu metu paleisdavo OS suteikiamas programas (Ubuntu atveju „cp“, Windows – „copy“) dviejų 100 MB dydžio failų kopijavimui.
2. Du failai vienu metu siunčiami į tą pačią failų sistemą, naudojant BitTorrent protokolo programą. Buvo surasti du nuorodas į apie 100 MB failus turintys „.torrent“ failai ir, panaudojant atitinkamas programas, siunčiami į tą pačią vietą.
3. Duomenys buvo lygiagrečiai rašomi į du failus, esančius toje pačioje failų sistemoje, papildymo būdu. Šio eksperimento įgyvendinimui sukurta programa Java kalba. Programa naudojo atskiras gijas kiekvieno failo kopijavimui, o kiekvienos gijos veikimas atitiko šį scenarijų: atidaromas failas papildymui, įrašoma 1 KB duomenų, failas uždaromas. Scenarijus vykdomas, kol failo dydis pasiekia 100 MB.

Kiekvienu atveju po atlikto duomenų įrašymo failų sistema buvo analizuojama saugumo įvertinimo skyriuje (žr. VIII skyriaus VIII.B poskyrį) aprašyta metodika.

Pirmos dvi situacijos nesukėlė jokios išskirtinės fragmentacijos ar failų susipynimo. Iš 5 lentelėje pateiktų rezultatų matoma, jog beveik visais atvejais rašant į failus vienu metu papildymo būdu sukeliama labai stipri fragmentacija ir failų tarpusavio susipynimas. Lentelė taip pat parodo, kad failų klasterių išsidėstymas stipriai priklauso nuo operacinės sistemos bei jos tvarkyklių. Labiau optimizuotos sistemos sugeba išvengti stiprios fragmentacijos (Ubuntu 11.04 OS, NTFS failų sistemos atvejis).

5 lentelė. RAŠYMO PAPILDYMO BŪDU SUKELTA FRAGMENTACIJA

	Windows 7		Windows XP		Ubuntu 11.04	
	FAT32	*NTFS*	*FAT32*	*NTFS*	*FAT32*	*NTFS*
Vid. fragmentų skaičius faile	25580	19869	16458	24675	24774	7
Vid. vieno failo susipynimų skaičius	51157	29684	14812	49341	49546	12

Reikia pabrėžti, kad rašant į failus buvo naudotos tik OS suteikiamos aukšto lygio funkcijos. Sukurta rašymo programa prie failų sistemos struktūrų priėjimo neturėjo. Tai parodo, kad įprastos programos gali sukelti labai stiprų failų tarpusavio susipynimą, jei rašoma į juos papildymo būdu. Dažnai tokiu būdu rašoma į žurnalo tipo failus ir tą patvirtina 4 lentelėje matomas 5 vietoje pagal susipynimo atvejus esantis žurnalo failo tipas.

X. IŠVADOS

Straipsnyje buvo pristatytas naujas metodas saugiam duomenų paslėpimui klasterinėje FAT failų sistemoje. Siūlomas metodas naudoja keletą dengiančių failų ir yra pagrįstas tų failų klasterių tarpusavio išsidėstymu failų sistemoje. Metodas yra paprastai įgyvendinamas, nes darbo metu nesukelia kolizijų problemos [7]. Siūlomo metodo talpumas nepriklauso nuo failų sistemos dydžio ar likusios laisvos vietos joje. Metodas užtikrina pastovų paslėptų bitų, tenkančių vienam dengiančiųjų failų klasteriui, skaičių ir leidžia visiškai užpildyti failų sistemą. Dengiantys failai, jų skaičius bei jų eilės tvarka sudaro slaptąjį raktą, kuris yra reikalingas paslėptiems duomenims nuskaityti. Dengiančių failų skaičiaus parinkimas leidžia keisti metodo talpos charakteristiką (kiek slaptos informacijos paslepiama vienu dengiančio failo klasteriu), kuri yra panaši, o kai kuriais atvejais netgi geresnė negu kitų panašių metodų. Siūlomo metodo naudojimas sukelia failų fragmentaciją bei ypatingą jos tipą – failų susipynimą. Eksperimentuose nustatyta, kad toks klasterių susipynimas yra dažnai sutinkamas senose aktyviai naudotose failų sistemose, be to, labai didelis kelių failų tarpusavio susipynimas gali būti sukeltas programų, kurios dažnai rašo į failus papildymo būdu. Eksperimentiniai tyrimai patvirtina, jog pasiūlytas metodas pasižymi dvigubo pagrįsto išsigynimo savybe.

LITERATŪRA

[1] Toldinas J., Stuikys V., Damasevicius R., Ziberkas G., Banionis M. Energy Efficiency Comparison with Cipher Strength of AES and Rijndael Cryptographic Algorithms in Mobile Devices. // Electronics and Electrical Engineering, 2011, Vol. 108, No. 2, P. 11–14.

[2] Luksys K., Sakalauskas E., Venckauskas A. Implementation Analysis of Matrix Power Cipher in Embedded Systems. // Electronics and Electrical Engineering, 2012, Vol. 118, No. 2, P. 95–98.

[3] Petitcolas F. A. P., Anderson R. J., Kuhn M. G. Information hiding-a survey. // Proceedings of the IEEE: Special Issue on Protection of Multimedia Content, 1999, Vol. 87, No. 7, P. 1062–1078.

[4] Provos N., Honeyman P. Hide and Seek: An Introduction to Steganography. // IEEE Security and Privacy, 2003 May, Vol. 1, No. 3, P. 32–44.

[5] Lampson B. W. A note on the confinement problem. // Communications of the ACM, 1973 Oct, Vol. 16, No. 10, P. 613–615.

[6] Simmons G. J. The Prisoners' Problem and the Subliminal Channel. // Advances in Cryptology, Proceedings of Crypto 83, New York: Plenum Press, 1984, P. 51–67.

[7] Khan H., Javed M., Khayam S. A., Mirza F. Designing a cluster-based covert channel to evade disk investigation and forensics. // Computers & Security, 2011, Vol. 30, No. 1, P. 35–49.

[8] Hanaoka G., Hanaoka Y., Hagiwara M., Watanabe H., Imai H. Unconditionally secure chaffing-and-winnowing: a relationship between encryption and authentication. // Proceedings of the 16th international conference on Applied Algebra, Algebraic Algorithms and Error-Correcting Codes, Berlin, Heidelberg: Springer-Verlag, 2006, P. 154–162.

[9] Dakhane D. M., Patil S., Patil M. Detection and elimination of covert communication in Transport and Internet layer - A Survey. // IJCA Proceedings on International Conference on Recent Trends in Information Technology and Computer Science (ICRTITCS-2011), New York, USA: Foundation of Computer Science, 2012, P. 36–41.

[10] Bellare M., Boldyreva A. The Security of Chaffing and Winnowing. // Proceedings of the 6th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology, London, UK, UK: Springer-Verlag, 2000, P. 517–530.

[11] Piper S., Davis M., Manes G., Shenoi S. Detecting Hidden Data in Ext2/Ext3 File Systems. // IFIP Int. Conf. Digital Forensics, Springer, 2005, P. 245–256.

[12] Eckstein K., Jahnke M. Data Hiding in Journaling File Systems. // Refereed Proceedings of the 5th Annual Digital Forensic Research Workshop, New Orleans, Louisiana, USA, 2005, P. 1–8.

[13] Anderson R. J. Information Hiding Terminology - Results of an Informal Plenary Meeting and Additional Proposals. // Information Hiding, Springer, 1996, P. 347–350.

[14] McDonald A. D., Kuhn M. G. StegFS: A Steganographic File System for Linux. // Information Hiding, Springer, 1999, P. 462–477.

[15] Pang H., Tan K. L., Zhou X. StegFS: A steganographic filesystem. // Proceedings of the 19th International Conference on Data Engineering, IEEE, 2003, P. 657–668.

[16] Anderson R. J., Needham R. M., Shamir A. The Steganographic File System. // Proceedings of the Second International Workshop on Information Hiding, London, UK: Springer-Verlag, 1998, P. 73–82.

[17] Huebner E., Bem D., Wee C. K. Data hiding in the NTFS file system. // Digital Investigation, 2006 Dec, Vol. 3, No. 4, P. 211–226.

[18] Czeskis A., Hilaire D. J. S., Koscher K., Gribble S. D., Kohno T., Schneier B. Defeating Encrypted and Deniable File Systems: TrueCrypt v5.1a and the Case of the Tattling OS and Applications. // HotSec, USENIX Association, 2008, P. 1–7.

[19] Carrier B. File System Forensic Analysis. Addison-Wesley Professional, 2005, 382 p.

[20] Garfinkel S. L. Automating Disk Forensic Processing with SleuthKit, XML and Python. SADFE, IEEE Computer Society, 2009, P. 73–84.

[21] Forte D. V. Open Source Forensics: The PTK: An alternative advanced interface for Sleuth Kit. // Network Security, 2008 Apr, Vol. 2008, No. 4, P. 10–13.

Nerijus Morkevicius, Grigas Petraitis, Algimantas Venčkauskas.

Covert Channel for Cluster-based File Systems Using Multiple Cover Files.

Abstract. Problems of sensitive information hiding in disk drives using cluster-based file systems are analyzed in this study. A new covert channel method for information hiding in disk drives is proposed and discussed. The method uses multiple cover files and is based on relative allocation of clusters of cover files in relation to one another. The experimental results presented in this paper show that the proposed method is easy to implement, provides good (for the covert channel) storage capacity and has the property of two-fold plausible deniability. The proposed covert channel method can be used for the storage of small and very sensitive information (such as passwords or encryption keys) on removable disk drives.

Nerijus Morkevicius, Grigas Petraitis, Algimantas Venčkauskas.

Slaptas kanalas klasterinėms failų sistemoms naudojant kelis dengiančiuosius failus

Darbe analizuojama slaptos informacijos slėpimo diskiniuose kaupikliuose problema. Straipsnyje pasiūlytas ir išanalizuotas naujas metodas skirtas informacijos slėpimui klasterinėse failų sistemose. Metodas naudoja kelis dengiančiuosius failus ir informaciją saugo keisdamas šių failų klasterių tarpusavio padėtį. Eksperimento rezultatai patvirtina, jog pasiūlytas metodas turi dvigubo pagrįsto išsigynimo savybę.