Latvijas Universitāte
Datorikas Fakultāte

Informācijas paslēpšana digitālajos attēlos

kvalifikācijas darbs

Autors : **Andrejs Žmakins**
Apl. nr. : az08189
Darba vadītājs : Kārlis Freivalds

Rīga 2010

ANOTĀCIJA

Šī darba mērķis bija izstrādāt un dokumentēt moduli, kas ir paredzēts informācijas slēpšanai digitālajos attēlos. Bija novērtēti daži informācijas slēpšanas algoritmi un realizēti divi no tiem. Modulis ir izstrādāts programmēšanas valodā *Java*. Izstrāde noritēja prakses ietvaros; prakse izgāja Latvijas Universitātes Matemātikas un Informātikas institūtā.

Atslēgvārdi: informācijas slēpšana attēlos, steganogrāfija, *Java*.

Abstract

The goal of this qualification work is a development of a module capable to conceal information within digital images. These were some concealment algorithms evaluated. Two of them are implemented in this work. The module is developed in Java programming language. The development was performed in the Institute of Mathematics and Informatics of University of Latvia during the period of practice.

Keywords: steganography, digital watermark, digital watermarking.

Аннотация

Целью данной работы является разработка и документирование модуля, предназначенного для сокрытия информации в цифровых изображениях. Были оценены несколько алгоритмов сокрытия информации и реализованы два из них. Модуль разработан на языке программирования *Java*. Разработка велась в рамках практики, проходившей в Институте математики и информатики Латвийского университета.

Ключевые слова: сокрытие информации в изображениях, стеганография, *Java*.

Saturs

Ievads....9
Moduļa Steganography prasību specifikācija....10
1. Ievads....10
1.1. Nolūks....10
1.2. Darbības sfēra....10
1.3. Definīcijas....10
1.4. Steganogrāfijas pamatjedzieni....10
1.5. Lietošanas scenāriji....11
1.5.1. Autortiesību aizsardzība....11
1.5.2. Aizsargātas informācijas noplūdes avota noteikšana....11
1.5.3. Cenzūras pārvarēšana....11
1.6. Saistītie dokumenti....11
2. Vispārējs apraksts....12
3. Funkcionālas prasības....13
3.1. Ziņas slēpšana un izņemšana....13
3.1.1. Liela izmēra ziņas slēpšana....13
3.1.2. Izturīgā ziņas slēpšana....14
3.1.3. Ziņas izņemšana no konteinera....15
3.2. Prasības drošībai....16
3.2.1. Ziņas šifrēšana....16
4. Prasības lietotāja saskarnei....17
5. Nefunkcionālās prasības....18
5.1. Prasības izpildes videi....18
5.2. Prasības ātrdarbībai....18
6. Mijējdarbība ar «AtteluApstrade»....19
Moduļa Steganography projektējuma apraksts....20
1. Ievads....20
1.1. Nolūks....20
1.2. Darbības sfēra....20
1.3. Definīcijas un saīsinājumi....20
1.4. Saistība ar citiem dokumentiem....20
2. Sistēmas interfeisa apraksts....21
3. Pamatprogrammas «AttēluApstrāde» izsaucamās funkcijas....22
4. Dekompozīcijas apraksts....23
4.1. Pakotņu diagramma....23
4.2. Klašu diagrammas....24
4.2.1. Pakotne steganography....24
4.2.2. Pakotne ECC....24
4.2.3. Pakotne gui....24
4.3. Klašu apraksts....25
4.3.1. Pakotne zhmakin....25
4.3.2. Pakotne staganography....25
4.3.3. Pakotne ECC....25
4.3.4. Pakotne gui....25
5. Detalizēts komponenšu apraksts....26
5.1. Algoritmu izvēle....26
5.1.1. Informācijas slēpšanas algoritmu apskate....26
5.1.2. Šifrēšanas metodes izvēle....26
5.2. Algoritmu projektējums....27
5.2.1. Slēpšana jaunākajos bitos....27

5.2.2. Slēpšana ar vienkāršoto Zhao un Koch metodi....28
5.2.3. Rīda-Solomona kļūdu korekciju kodi....29
5.2.4. Ziņas struktūra un tas atkodēšanas algoritms....29
6. Lietotāja saskarnes dizains....30
6.1. Ziņas slēpšanas dialogs....30
6.2. Ziņas izņemšanas dialogs....31
Moduļa Steganography testēšanas plāns....32
1. Ievads....32
2. Saistītie dokumenti....32
3. Testējamie vienumi....32
4. Testējamās raksturiezīmes....32
5. Netestējamās raksturiezīmes....32
6. Testēšanas pieēja....32
7. Testējamā vienuma novērtēšanas kritēriji....33
8. Atlikšanas kritēriji un atsākšanas prasības....33
9. Testēšanas nodevumi....33
10. Testēšanas uzdevumi....33
11. Vides vajadzības....33
Moduļa Steganography testa piemēru specifikācija....34
1. Klase zhmakin.LinearProgressEstimation....34
1.1. Testpiemēru kopa....34
1.1.1. Testpiemērs 1....34
1.1.2. Testpiemērs 2....34
1.1.3. Testpiemērs 3....34
2. Klase zhmakin.YCbCr....35
2.1. Testpiemēru kopa 1....35
2.1.1. Testpiemērs 1....35
2.1.2. Testpiemērs 2....35
3. Klase zhmakin.steganography.BitsOfImageBySignificance....36
3.1. Testpiemēru kopa 1....36
3.1.1. Testpiemērs 1....36
3.1.2. Testpiemērs 2....36
3.2. Testpiemēru kopa 2....36
3.2.1. Testpiemērs 1....36
3.2.2. Testpiemērs 2....36
4. Klase zhmakin.steganography.BitString....38
4.1.1. Testpiemērs 1....38
4.1.2. Testpiemērs 2....38
4.1.3. Testpiemērs 3....38
4.1.4. Testpiemērs 4....38
5. Klase zhmakin.steganography.ConformityTable....39
5.1. Testpiemēru kopa 1....39
5.1.1. Testpiemērs 1 – nenegatīva permutācija....39
5.1.2. Testpiemērs 2 – pozitīva permutācija....39
5.1.3. Testpiemērs 3 – permutācija no veseliem skaitļiem....39
6. Klase zhmakin.steganography.Message....40
6.1. Testpiemēru kopa 1....40
6.1.1. Testpiemērs 1....40
6.1.2. Testpiemērs 2....40
7. Klase zhmakin.steganography.ZhaoAndKoch....41
7.1. Testpiemēru kopa 1....41
7.1.1. Testpiemērs 1....41

7.1.2. Testpiemērs 2....41
7.1.3. Testpiemērs 3....41
7.1.4. Testpiemērs 4....41
7.2. Testpiemēru kopa 2....42
7.2.1. Testpiemērs 1 – rakstīšanās ātrums....42
7.2.2. Testpiemērs 2 – lasīšanās ātrums....42
Moduļa Steganography testēšanas žurnāls....43
1. Ievads....43
2. Saistītie dokumenti....43
3. Testēšanas gaitā atklātās problēmas un risinājumi....44
3.1. Problēma 1 (PRB01)....44
3.2. Problēma 2 (PRB02)....44
Projekta organizācija....45
Kvalitātes nodrošināšana....46
Konfigurāciju pārvaldība....47
Darbietilpības novērtējums....48
Rezultāti un secinājumi....49
Izmantotā literatūra....50
Pielikums 1 – Steganogrāfijas algoritmu salīdzināšana....51
Pielikums 2 – JavaDoc....52
zhmakin.steganography
Class BitsOfImageBySignificance....52
1.1. NUM_OF_LAYERS....53
1.2. BitsOfImageBySignificance....53
1.3. BitsOfImageBySignificance....54
1.4. initConformityTables....54
1.5. getSize....54
1.6. getLayerSize....54
1.7. getIndexInLayer....55
1.8. getLayer....55
1.9. getComponent....55
1.10. getIndexInImage....56
1.11. get....56
1.12. set....57
1.13. get....57
1.14. set....57
1.15. encode_trivially....58
1.16. decode_trivially....58
2. zhmakin.steganography
Class BitString....59
2.1. BitString....60
2.2. BitString....60
2.3. BitString....61
2.4. encodeString....61
2.5. decodeString....61
2.6. loadFile....61
2.7. loadFile....62
2.8. saveToFile....62
2.9. set....62
2.10. set....63
2.11. get....63
2.12. size....63

2.13. setSize....64
2.14. equals....64
2.15. attach....64
2.16. toString....64
3. zhmakin.steganography
Class ConformityTable....65
3.1. ConformityTable....65
3.2. ConformityTable....65
3.3. get....66
Pielikums 3 – Programmas kods....67
Dokumentārā lapa....115

Ievads

Mūsdienās strauji attīstās autortiesību aizsardzības tehnoloģijas. Bieži vien šīs tehnoloģijas prasa atzīmēt aizsargāta materiāla kopiju tā, lai kopijas turētājam nebūtu ne jausmas, ka kopijā ir paslēpta turētāju unikāli identificējoša informācija. Vispārīgā gadījumā informācijas slēpšanu kādā citā objektā sauca par steganogrāfiju. Šajā darbā ir aprakstīts autora izstrādāts programmas «AttēluApstrāde» modulis, ko izmanto steganogrāfijskos paņēmienos un kas darbojas ar digitālajiem attēliem. Starp moduļa funkcijām ir iespēja paslēpt patvaļīgo tekstu vai datnes saturu, kā arī regulēt, kādas izredzes ir paslēptai informācijai pārdzīvot attēla izmaiņas.

MODUĻA *STEGANOGRAPHY* PRASĪBU SPECIFIKĀCIJA

1. IEVADS

1.1. Nolūks

Šī dokumenta nolūks ir precīzi specificēt programmas «AttēluApstrāde» moduli *Steganography*, lai identificēt visas prasības modulim un veidot pamatu programmatūras projektējumam.

Dokuments ir domāts gan pasūtītājam, gan izstrādātājam.

1.2. Darbības sfēra

LU MII izstrādātas programmas «AttēluApstrāde» modulis *Steganography* ir paredzēts informācijas slēpšanai rastra attēlos, kā arī paslēptas informācijas izņemšanai. Par slēpšanu ir domāts tāds attēla pārveidojums, kā tajā tiks integrēta ziņa attēla izskatu pamanāmi nemainot.

1.3. Definīcijas

Saspiešana ar zudumiem (*lossy compression*)	— tāds informācijas saspiešanas veids, kas atmet daļu no informācijas, lai samazinātu informācijas izmēru cik vien iespējams.
JPEG	— populārs attēlu uzglabāšanas formāts, kas lieto saspiešanu ar zudumiem.

1.4. Steganogrāfijas pamatjedzieni

Slēpjamo informāciju sauksim par ziņu (*message*), bet attēlu, kurā slēpsim ziņu – par konteineri (*container*). Ja konteinerā nav nekas nav paslēpts, tad saka ka konteiners ir tukšs. To pašu teiksim, ja pie ziņas nevaram tikt un to eksistenci nevaram pierādīt.

Ziņu uztversim kā bitu masīvu, līdz ar to ziņas garumu parasti izteiksim bitos.

Par slēpšanu sauksim tādu konteinera pārveidojumu, kā cilvēka uztverē tas vizuāli nemainās vai izmaiņas ir mazsvarīgas; bet pārveidojuma rezultātā pie dotas paroles (*password*) ziņa tiks iekodēta konteinerā.

No konteinera ziņu var izņemt ārā zinot paroli.

Ziņas spēju pārdzīvot konteinera izmaiņas pie attēlu apstrādes procedūrām, pārveidojumu formātā ar saspiešanu ar zudumiem (*lossy compression*) sauksim par izturību (*robustness*). Ja ziņa spēj pārdzīvot relatīvi stiprākas izmaiņas, tad saka ka tai izturības pakāpe ir augstāka. (Acīmredzami, pie lielākas izturības pakāpes ziņas slēpšana stiprāk ietekmēs konteineri).

Iesaistīto pusi, kas tīšām vai netīšam mēģina iznīcināt ziņu konteinera, vai [tīšām] noteikt ziņas klātbūtni konteinerā sauksim par uzbrucēju (*attacker*). Ziņas iznīcināšanas mēģinājumu sauksim par uzbrukumu (*attack*).

Cita izturības definīcija: Izturība ir ziņas spēja pārdzīvot tādus uzbrukumus, kas stipri nemaina konteinera izskatu.

Moduļa funkcijas šajos terminos precīzas definēsim sekojoši:

1. Modulis S*teganography* nodrošina iespēju paslēpt konteinerī ziņu, norādot paroli un

izturības pakāpi.

2. No konteinera izņemt tajā paslēpto ziņu, ja ir zināma attiecīgā parole.

1.5. Lietošanas scenāriji

1.5.1. Autortiesību aizsardzība

Reportierim paveicās nofotografēt kādu «karsto» notikumu. Viņam gribās to nopublicēt internetā, lai foto būtu visiem redzams, tajā skaitā lai piedāvātu to ziņu aģentūrām nopublicēt par maksu. Bet jebkurš var lejuplādēt foto un teikt, ka notikumu nofotografēja viņš. Kvalitātes samazināšana, necaurspīdīgs un puscaurspīdīgs teksts uz bildes var būt nepietiekams – fotogrāfiju no malas var apgriezt, puscaurspīdīgo ūdenszīmi var restaurēt. Bet ja autors atzīmēs savu fotogrāfiju ar neredzamo ūdenszīmi, potenciālam autortiesību pārkāpējam par to nebūs ne jausmas. Bet ja tomēr viņš par to uzzinās, vienīgs veids no tā garantēti izvairīties būs stipra fotogrāfijas izkropļošana. Tad īstajām autoram būs labs pieradījums tiesas prāvā.

1.5.2. Aizsargātas informācijas noplūdes avota noteikšana

Kāda kompānija publicē datorspēli. Spēles diska kopijas nosūtītas vairākām disku rakstīšanas rūpnīcām. Netālu no oficiālā pārdošanās sākuma datorspēle ir noplūdusi failu apmaiņas tīklos, kas var nelabvēlīgi ietekmēt pārdošanu. Lai noteikt no kādas rūpnīcas notika noplūde, var katrai rūpnīcai iesūtīt diska kopiju ar spēles resursiem (tas bieži ir attēli), kas ir atzīmētas ar unikālajiem ūdenszīmēm. Labums šeit ir tāds, ka noplūdes avotu var noteikt, ja parādījās ne paša spēle, bet pat neautorizētas ekrānkopijas, kas liecina par noplūdi.

1.5.3. Cenzūras pārvarēšana

Kādā totalitārā valstī disidentu grupa cīnās par cilvēktiesību ievērošanu. Šī valsts ir stipra interneta cenzūra. Disidentu grupai jākontaktē ar līdzjutējiem citās (brīvākajās) valstīs. Īpaši ātri jāpārraida ziņas, ka valdība uzsāka represijas pret oponentiem. Var iepriekš norunāt, ka šīs nelaimes gadījumā kāds mazsvarīgs (lai nepievērstu lielu uzmanību) organizācijas biedrs nopublicēs sludinājumu par noteiktas lietas pārdošanu kādā starptautiskajā sludinājumu sarakstā. Fotogrāfijās, kas nāk kopā ar sludinājumu var ievietot ziņu ar svarīgo informāciju. Organizācijas līdzjutējiem paliek tikai novērot vai neparādījās ziņas par norunātas preces pārdošanu. Tas fakts, kā viens no organizācijas biedriem kaut ko pārdod visticamāk paliks nepamanīts ar valsts drošības dienestiem pietiekami ilgu laiku, lai pārraidīt ziņu. Sludinājumu saraksta vietā bildes pārraidei var izmantot *peer-to-peer* tīklus.

1.6. Saistītie dokumenti

Šī prasību specifikācija ir izstrādāta saskaņā ar Latvijas Valsts Standarta 68:1996 «Programmatūras prasību specifikācijas ceļvedis».

Programma «AttēluApstrāde» ir aprakstītā kvalifikācijas darbā «Teksta zīmju attēlu uzlabošana», Jevgeņijs Jonass, 2009.

2. Vispārējs apraksts

1. Programmai jānodrošina divas paslēpšanas metodes:
    1. pirmajai metodei jābūt pēc iespējas ātrai un tai arī jānodrošina liels ziņas apjoms; izturība pret uzbrukumiem šai metodei nav svarīga, vienīgais kas jāiztur ir trokšņa pielīkšana vai kādas daļas aizkrāsojums;
    2. otrajai metodei jānodrošina regulējamo paslēptas informācijas izturības līmeni (ziņās apjoms nav vitāli svarīgs, kā arī ātrdarbība); izturību jānodrošina pret sekojošām izmaiņām:
        1. pārveidojumu starp formātiem bez zudumiem un saglabāšanu tajā pašā formātā;
        2. konteineris ir pārveidots formātā, kas izmanto saspiešanu ar zudumiem (piemērām, *JPEG*);
        3. trokšņa pielikšana;
        4. izmēra mainīšana: ja konteiners ir pārveidots par lielāko un pēc tam atgriezts pie sākotnēja izmēra;
        5. kādas konteinera daļas zaudējums: ja konteiners ir apgriezts no malas un/vai konteinera iekšienē kaut kas ir aizkrāsots (piemērām, bildei virsū «uzzīmēts» teksts);
        6. pārveidojumu par melnbalto;
        7. ja konteinerā ir paslēpta vēl viena ziņa (ar citu paroli).
2. Nedrīkst būt iespējai nolasīt ziņu nezinot pareizo paroli. Un pat nevar bez paroles konstatēt ziņas esamību.

3. Funkcionālas prasības

3.1. Ziņas slēpšana un izņemšana

3.1.1. Liela izmēra ziņas slēpšana

Identifikators	Embed.1.Capacious
Nosaukums	Liela izmēra ziņas slēpšana
Ievads	
Nodrošina iespēju paslēpt attēlā liela izmēra ziņu, kā arī izdarīt to ātri. Par lielu ziņas izmēru ir domāts tāds, kurš ir lielāks par pikseļu skaitu konteinerī. Izturība pret izmaiņām nav tik svarīga. Ziņai jāpārdzīvo tikai trokšņa uzlīkšanu un konteinera daļas zaudējumu. Izturības pakāpei jābūt regulējamai.	
Ievaddati	
Konteineris, ziņa, izturības pakāpe.	
Kļūda:	
1. Konteiners ir pārāk mazs tāda izmēra ziņai. 2. Nav norādīta ziņa.	

3.1.2. Izturīgā ziņas slēpšana

Identifikators	Embed.2.Robust
Nosaukums	Izturīgā ziņas slēpšana
Ievads	
Nodrošina iespēju slēpt konteinerī ziņu tā, lai ziņa saglabātos pie vairākām konteinera izmaiņām.	
Ievaddati	
Konteineris, ziņa, izturības pakāpe.	
Apraksts	
Starp izmaiņām kuras ziņai jāpārdzīvo jābūt: 1. pārveidojumam formātā ar saspiešanu ar zudumiem (*lossy compression*); 2. trokšņa uzlikšana; 3. konteinera daļas zaudējums.	
Kļūda:	
1. Konteiners ir pārāk mazs tāda izmēra ziņai. 2. Nav norādīta ziņa.	

3.1.3. Ziņas izņemšana no konteinera

Identifikators	Extract.1
Nosaukums	Ziņas izņemšanas no konteinera
Ievads	
Modulim jāpiedāvā iespēja izņemt ziņu ārā no konteinera.	
Ievaddati	
Konteineris, ziņa, parole, izmantota metode.	
Apraksts	
Lietotājam jāpiedāvā iespēja izņemt ziņu no konteinera zinot tikai slēpšanas metodi un pareizo paroli.	
Kļūda	
1. Ziņa konteinerī ir neatgriezeniski bojāta, vai parole nav pareiza.	
Izsaucamās prasības	
Identifikators	**Nosaukums**
Security.1.Crypt	Ziņas šifrēšana

3.2. Prasības drošībai

3.2.1. Ziņas šifrēšana

Identifikators	Security.1.Crypt
Nosaukums	Ziņas šifrēšana
Ievads	
Jāpastāv iespēja aizsargāt ziņas saturu no neautorizētām pusēm, tomēr lai programmu neaiztiktu eksporta ierobežojumi.	
Ievaddati	
Ziņa, parole.	
Apraksts	
Modulim jāaizsargā ziņas saturu no triviālas lasīšanas, bet tomēr tas izplatīšana nedrīkst būt ierobežota ar ieroču eksporta regulējumiem, precīzāk programmai jāpakļaujas *Wassenaar Arrangement on Export Controls for Conventional Arms and Dual-Use Goods and Technologies* vienošanām.	
Izvaddati	
Šifrēta vai atšifrēta ziņa.	

4. Prasības lietotāja saskarnei

Lietotāja saskarnei jāpiedāvā sekojošas iespējas, informāciju konteinera slēpjot:

- izvēlēties ziņu (norādīt datni vai ievadīt tekstu);
- izvēlēties paslēpšanas metodi;
- norādīt slēpjamās informācijas izturības pakāpi;
- ievadīt paroli;
- novērtēt, cik pie dota slēpjamas informācijas apjoma (un izturības), izmainīsies konteiners (subjektīvs novērtējums; nepamānāmi, maz, stipri, tā ka paliek tikai troksnis utt.);
- cik daudz informācijas (bitos un baitos vai *Unicode* simbolu) var paslēpt pie dotām konteineri un izturības pakāpi.

Un izņemot ārā no konteinera:

- ievadīt paroli;
- iepriekš apskatīt ziņu;
- saglabāt ziņu datnē (vajag norādīt faila vārdu un ceļu);
- pateikt vai paslēptajā informācijā ir bojājumi un vai izdevās tās koriģēt;
- pateikt ka konteiners ir tukšs.

5. Nefunkcionālās prasības

5.1. Prasības izpildes videi

Programma tiks darbināta uz *Java VM* versijas 1.6.0_20, kas strādā zem *Windows XP SP3 (32-bit)*. Minimāla monitora izšķirtspēja ir 1024x768.

5.2. Prasības ātrdarbībai

Programmai jāiekodē 100000 bitu garo ziņu 10 sekunžu laikā ar ātro neizturīgo metodi Embed.1.Capacious un 15 minūšu laikā ar izturīgo Embed.2.Robust. Atkarība starp ziņas garumu un iekodēšanas laiku abos gadījumos ir lineāra.

6. Mijējdarbība ar «AttēluApstrade»

Moduļa funkcijas tiks izsauktas no izvēlnes *Steganography* galvenajā izvēļņu joslā. Izvēlnē *Steganography* jābūt diviem izvēlnes elementiem *Embed Message* un *Extract Message* attiecīgi ziņas slēpšanai un izņemšanai.

Ziņas izņemšanas un slēpšanas dialoglodziņi kā vecāko (*parent*) logu jālieto «AttēluApstrādes» klases `gui.MainFrame` objekts `gui.MFAListener.parentFrame`.

Slēpšanas rezultātu priekšskatījumam jālieto klases `gui.ImageViewer` `gui.MFAListener.iv`.

Moduļa Steganography projektējuma apraksts

1. Ievads

1.1. Nolūks

Šī dokumenta nolūks ir definēt moduļa *Steganography* zemā līmeņa projektējumu. Dokumenta nav cītīgo moduļa klašu metožu aprakstu, tāpēc kā šīm mērķiem tika lietots rīks *JavaDoc*. Šīs dokuments ir paredzēts izstrādātājiem.

1.2. Darbības sfēra

Moduļa *Steganography* funkcijas ir informācijas slēpšana digitālajos attēlos, kā arī paslēptas informācijas izņemšanai.

1.3. Definīcijas un saīsinājumi

YCbCr	Krāsu telpa, kas ir lietotā formātā *JPEG*.

1.4. Saistība ar citiem dokumentiem

Šis dokuments ir izstrādāts balstoties uz «Moduļa *Steganography* prasību specifikāciju».

Šis dokuments ir izstrādāts saskaņā ar Latvijas Valsts Standarta 72:1996 «Ieteicamā prakse programmatūras projektējuma aprakstīšanai» standarta prasībām.

Šī dokumenta sastāvdaļa ir Andreja Žmakina kvalifikācijas darba «Informācijas paslēpšana digitālajos attēlos» Pielikums 1 – «Steganogrāfijas algoritmu salīdzināšana».

2. SISTĒMAS INTERFEISA APRAKSTS

Ziņas iekodēšanas dialogam jābūt realizētām ar klases `zhmakin.steganography.gui.EmbedmentDialog` objektu, konstruktoram nododot sekojošos parametrus:

`EmbedmentDialog( Frame owner, ImageViewer viewport, RasterImage image ),`

kur `owner` – programmas galvenais logs;

`viewport` – `ImageViever` klases objekts, kas lietots parametra `owner` nodotajā logā, lai radīt attēlu;

`image` – `RasterImage` klases objekts, kas objekta `EmbedmentDialog` veidošanas laikā tiek radīts programmas galvenajā logā (tajā tiks iekodēta ziņa).

Ziņas izņemšanas dialogam jābūt realizētām kā klases `zhmakin.steganography.gui.ExtractionDialog` objektu, konstruktoram nododot sekojošos parametrus:

`EmbedmentDialog( Frame owner, RasterImage image ),`

kur `owner` – programmas galvenais logs;

`image` – `RasterImage` klases objekts, kas objekta `EmbedmentDialog` veidošanas laikā tiek radīts programmas galvenajā logā (no tās mēģināsim dabūt ziņu ārā).

3. Pamatprogrammas «AttēluApstrāde» izsaucamās funkcijas

Programma «AttēluApstrāde» nodrošina sekojošo funkcionalitāti: konteinera izvēli, atvēršanu, apskati, saglabāšanu.

Modulis izmanto sekojošus programmas «AttēluApstrāde» objektus:

`Util.RasterImage` – operācijām ar konteineri ir lietotas sekojošas metodes:

- `public boolean isValid( int row, int col )`
  - pārbauda, vai eksistē pikselis ar koordinātēm (`col`;`row`);
- `public Color get( int row, int col )`
  - atgriež pikseļa (`col`;`row`) krāsu kā `java.awt.Color` objektu;
- `public Color get( int index )`
  - atgriež `index`-tā (skat. zemāk) pikseļa krāsu kā `java.awt.Color` objektu;
- `public void set( int index, Color color )`
  - uzstāda `index`-tā (skat. zemāk) pikseļa krāsu uz parametra `color` norādīto;
- `public void set( int row, int col, Color color )`
  - uzstāda pikseļa ar koordinātēm (`col`;`row`) krāsu uz parametra `color` norādīto;
- `public int getRGB( int row, int col )`
  - atgriež pikseļa ar koordinātēm (`col`;`row`) krāsas RGB vērtību formā 0x00RRGGBB;
- `public void setRGB( int row, int col, int rgb )`
  - uzstāda pikseļa ar koordinātēm (`col`;`row`) krāsas RGB vērtību uz parametra `rgb` norādīto; krāsas vērtības formāts ir 0x00RRGGBB;
- `public void set( RasterImage image )`
  - saglabā tekošā objektā `image` kopiju;
- `public RasterImage makeCopy()`
  - veido tekošā objekta kopiju;

kā arī lauki

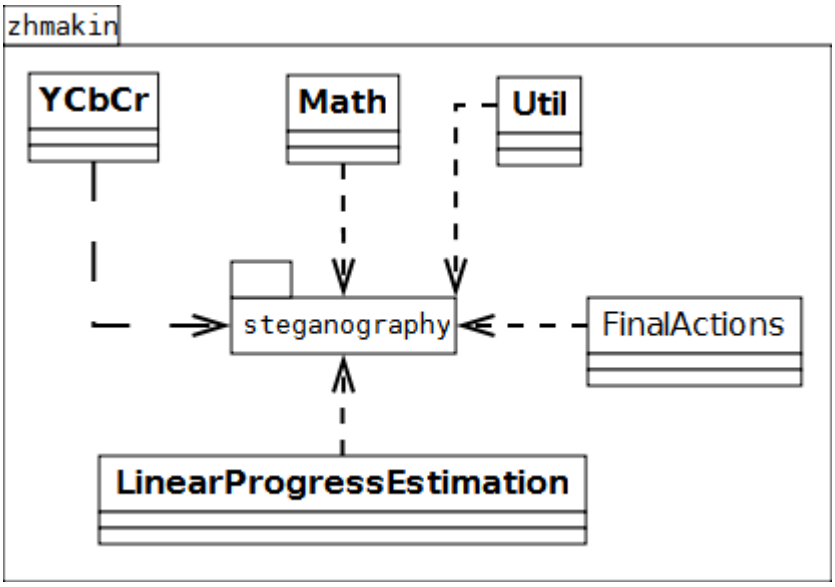
- `width` – attēla platums (tikai lasa);
- `height` – attēla garums (tikai lasa).

Klases `gui.MainFrame` objekts `gui.MFAListener.parentFrame` lietots, kā vecākais (*parent*) moduļa logiem.

Klases `gui.ImageViewer` objekts `gui.MFAListener.iv` lietots rezultāta apskatei, tām nolūkam ir lietota klases `gui.ImageViewer` metode `public void setImage(RasterImage rImage)`.

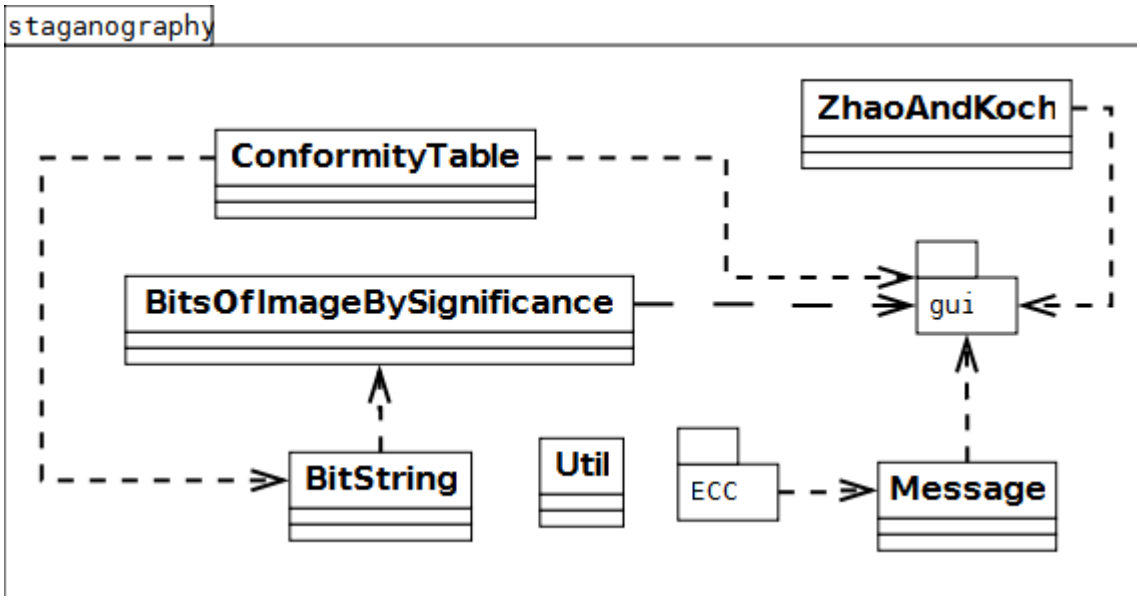
4. Dekompozīcijas apraksts

4.1. Pakotņu diagramma



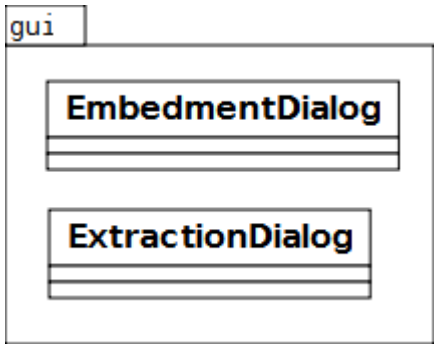
4.2. Klašu diagrammas

4.2.1. Pakotne `steganography`



4.2.2. Pakotne `ECC`

4.2.3. Pakotne `gui`



4.3. Klašu apraksts

4.3.1. Pakotne `zhmakin`

`YCbCr`	Klase reprezentē krāsu *YCbCr* krāsu telpā.
`Math`	Klase satur dažas matemātiskās funkcijas.
`Util`	Klase satur dažas statiskās palīgmetodes.
`LinearProgressEstimation`	Klase novērtē laiku kas paliek līdz procesa beigām.
`FinalActions`	Abstraktā klase, kas piedāvā standartu saskarni darbībām, kas jāveic procesam savu darbību beidzot «normālā» gadījumā, ar kļūdām un kad process ir pārtrauks to ārienes.
`steganography`	Apakšpakotne, kur ir realizēta moduļa pamatfunkcionalitāte.

4.3.2. Pakotne `staganography`

`ConformityTable`	Klase realizē gadījumā permutāciju.
`BitsOfImageBySignificance`	Klase realizē metodi *Least Significant Bits*, arī realizē saskarni, kas ļauj nolasīt un mainīt atsevišķus attēla bitus.
`BitString`	Klase realizē bitu virkni.
`Util`	Klase satur dažas statiskās palīgmetodes.
`Message`	Klase kas ļauj kodēt un atkodēt zemā līmeņa ziņu.
`ZhaoAndKock`	Klase realizē vienkāršoto ziņas slēpšanas (un arī izņemšanas) metodi, kas piedāvāja *Zhao* un *Koch*.
`ECC`	Apakšpakotne, kas satur kļūdu korekciju kodu algoritmu realizācijas.
`gui`	Pakotnes, kura realizē moduļa *Steganography* grafisko saskarni.

4.3.3. Pakotne `ECC`

`ReedSolomon`	Klase realizē Rida-Solomona kodējumu.

4.3.4. Pakotne `gui`

`EmbedmentDialog`	Klase realizē ziņas slēpšanas dialoglodziņu.
`ExtractionDialog`	Klase realizē ziņas izņemšanas dialoglodziņu.

5. Detalizēts komponenšu apraksts

5.1. Algoritmu izvēle

5.1.1. Informācijas slēpšanas algoritmu apskate

Pētot literatūru un zinātniskās publikācijas bija atrasti vairāki algoritmi informācijas slēpšanai attēlos. (Skat. Pielikumu 1). Tika konstatēts, ka prasībām Embed.1.Capacious un Embed.2.Robust atbilst attiecīgi Informācijas slēpšana jaunākajos bitos un *Zhao* un *Koch* vienkāršotais algoritms. Nevienam nepiemīt izturība pret konteinera daļas zaudējumu, bet tāda īpašība parādīsies, ja ziņu papildināt ar kļūdu korekcijas kodiem. Modulim *Steganography* tika izvēlēti Rīda-Solomona kodi.

5.1.2. Šifrēšanas metodes izvēle

Ar abiem algoritmiem ir organiski savietojama vienkārša šifrēšanas metode: ziņas bitu indeksu gadījuma permutācija. No paroles dabūsim graudu pseidogadījuma skaitļu ģeneratoram. Ar ģeneratora palīdzību dabūsim gadījuma permutāciju no veseliem skaitļiem no 0 līdz ziņas garums – 1. Lietosim tos kā indeksus piekļuvei ziņas bitiem. Ja lietot kriptogrāfiski nedrošu gadījuma skaitļu ģeneratoru, algoritms nebūs aizskarts ar *Wassenaar* ierobežojumiem. Šinī projektā tika lietots *Java* standarts `java.util.Random`, kas turklāt lieto tikai 48 bitu garo graudu, kaut *Wassenaar* atļuaj lietot līdz pat 56 bitu garas atslēgas.[1,2]

5.2. Algoritmu projektējums

5.2.1. Slēpšana jaunākajos bitos

Viss konteineris ir uztverts kā viens bitu masīvs ar indeksiem no 0 līdz «garums» × «platums» × 3 × 8 - 1; biti ir numurēti pēc zemāk aprakstītiem noteikumiem.

Konteiners ir sadalīts astoņos slāņos: kārto n -to slāni veido visu pikseļu visu trīs komponenšu n -tie biti. Par komponenti c sauksim sarkanai, zilai un zaļai krasam atvēlētos astoņus bitus. Par konteinera pikseļa ar koordinātēm $(x; y)$ indeksu sauksim skaitļi $i = y \times \text{platums} + x$.

Līdz ar to n , i un c dabūjamās no indeksa konteinera j pēc sekojošām formulām:

$n = j \operatorname{div} (\text{platums} \times \text{garums} \times 3)$,

$i = (j \bmod (\text{platums} \times \text{garums} \times 3)) \operatorname{div} 3$,

$c = i \bmod 3$, kur div ir dalīšana veselajos skaitļos un mod ir atlikums veselo skaitļu no dalīšanās.

Zinot n , i un c mēs varam nolasīt un mainīt jebkuru konteinera bitu.

5.2.2. Slēpšana ar vienkāršoto Zhao un Koch metodi

Šī metode ir paņemta no [3].

Konteineris ir dalāms blokos izmēra 8x8 pikseli. Izmaiņas ir veiktas ar *YCbCr* spīduma komponenti. Katrs bloks glābj tieši vienu ziņas bitu. Katram blokam b_i ir pielietota divdimensiju diskrētā kosinusa transformācija $B_i = D\{b_i\}$:

$$S(u,v) = \frac{2}{N} C(u) C(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} s(x,y) \cos\left(\frac{\pi u (2x+1)}{2N}\right) \cos\left(\frac{\pi v (2y+1)}{2N}\right) ,$$

kur $C(u) = \begin{cases} u = 0, & \frac{1}{\sqrt{(2)}} \\ u \neq 0, & 1 \end{cases}$.

Pirms ziņas pārraides jāizvēlas divi indeksi $(u_1; v_1)$ un $(u_2; v_2)$. Tiem jābūt vidējos frekvencēs, ja mazākas frekvences nesaglabājas pārveidojumos, bet lielie stipri ietekmē konteinera izskatu. Avots [3] piedāvā izvēlēties $(4;1)$ un $(3;2)$, lai dabūtu maksimālo izturību pret *JPEG* saspiešanu.

Ja bloks iekodē «1», tad $B_i(u_1, v_1) > B_i(u_2, v_2)$, un tieši pretēji, ja ir iekodētā «0». Vērtības jāmaina vietām, lai tie atbilstu nosacījumam. Pēc tām vērtības jākoriģē tā, lai $|(B_i(u_1, v_1) - B_i(u_2, v_2))| \geq x$, kur x ir iepriekš uzdotais kvocients. Jo lielāks x , ja lielāka izturība, bet līdz ar to ietekme uz konteineri.

Kad izmaiņas ir veiktas, jaunās b_i' vērtības ir dabūjamās ar apgriezto transformāciju $b_i' = D^{-1}\{B_i\}$:

$$s(x,y) = \frac{2}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} C(u) C(v) \cos\left(\frac{\pi u (2x+1)}{2N}\right) \cos\left(\frac{\pi v (2y+1)}{2N}\right)$$

Ziņas slēpšanas algoritms pseidokodā

Ievade:

$m_1, \ldots, m_N$ - ziņas biti no pirmā līdz N-tām.

$b_1, \ldots, b_M$ - konteinera bloki no pirmā līdz M-tām ($M > N$)

```
no i = 1 līdz M
    B_i = D{b_i}
  ja  m_i = 0
     ja  B_i(u_1, v_1) > B_i(u_2, v_2)
        mainām vietām  B_i(u_1, v_1)  un  B_i(u_2, v_2)
  citād
     ja  B_i(u_1, v_1) < B_i(u_2, v_2)
        mainām vietām  B_i(u_1, v_1)  un  B_i(u_2, v_2)
  pielāgojam abas vērtības tā, lai  |(B_i(u_1, v_1) - B_i(u_2, v_2))| ≥ x
    b_i' = D^{-1}{B_i}
veidojam jauno konteineri no visiem  b_i  -tiem
```

Ziņas izņemšanas algoritms

Ievade:

$m_{1,}\ldots, m_N$ - ziņas biti no pirmā līdz N-tām.

Izvade:

$b_{1,}\ldots, b_M$ - konteinera bloki no pirmā līdz M-tām ($M > N$)

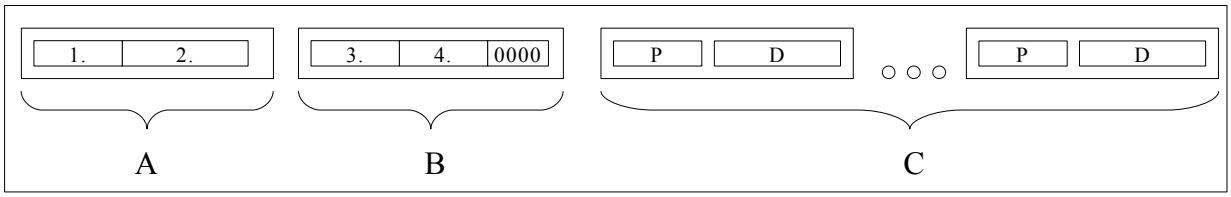
```
no i = 1 līdz M
```
$B_i = D\{b_i\}$
```
ja
```
$B_i(u_{1,}v_2) \le B_i(u_{2,}v_2)$
$m_i = 0$
```
citād
```
$m_i = 1$

5.2.3. Rīda-Solomona kļūdu korekciju kodi

Tika pieņemts lēmums paņemt gatavo realizāciju no [4]. Tas bija pārrakstīta *Java* programmēšanas valodā.

5.2.4. Ziņas struktūra un tas atkodēšanas algoritms

Ziņai sastāv no trim daļām:



Daļa A glabā stiprumu Rīda-Solomona kodiem, kas lietoti ziņas uzglabāšanai. Apakšdaļa 2. sastāv no 9 deviņiem 4 bitu garajiem, kopā 36 biti. Katrā blokā glabājas viens skaitlis no 0 līdz 10, pieļauti tikai pāra skaitļi. Skaitļa jaunākie biti pa priekšu. Apakšdaļā 1. glabājas Rīda-Solomona RS(15,9) kodi daļai 2. Apakšdaļas 1. garums ir 24 biti. Rīda-Solomona kodu stiprums ir vitāla informācija un tāpēc tā ir tik cītīgi saglabāta. Ja tomēr ar Rīda-Solomona kodiem šo informāciju atjaunot neizdevās, tad skatāmies vai starp tiem deviņiem blokiem ir sakrītošie 7 biti katrā pozīcijā. Ja ir, tad atjaunojam stiprumu. Pretējā gadījuma konstatējam ziņas nesalabojamo bojājumu jeb (mums tas ir viens un tas pats) ka ziņas tur nemaz nav.

Daļa B glabā nekodētas un «nepolsterētas» ziņas garumu. Apakšdaļa 4. sastāv no 32 bitiem un glabā minēto garumu (skaitļi no 0 līdz $2^{32}-1$); jaunākie biti pa priekšu. Pēdējie četri biti vienmēr nulles. Apakšdaļā 3. glabājās Rīda-Solomona kodi RS(15;9) daļai 4., nulles ieskaitot.

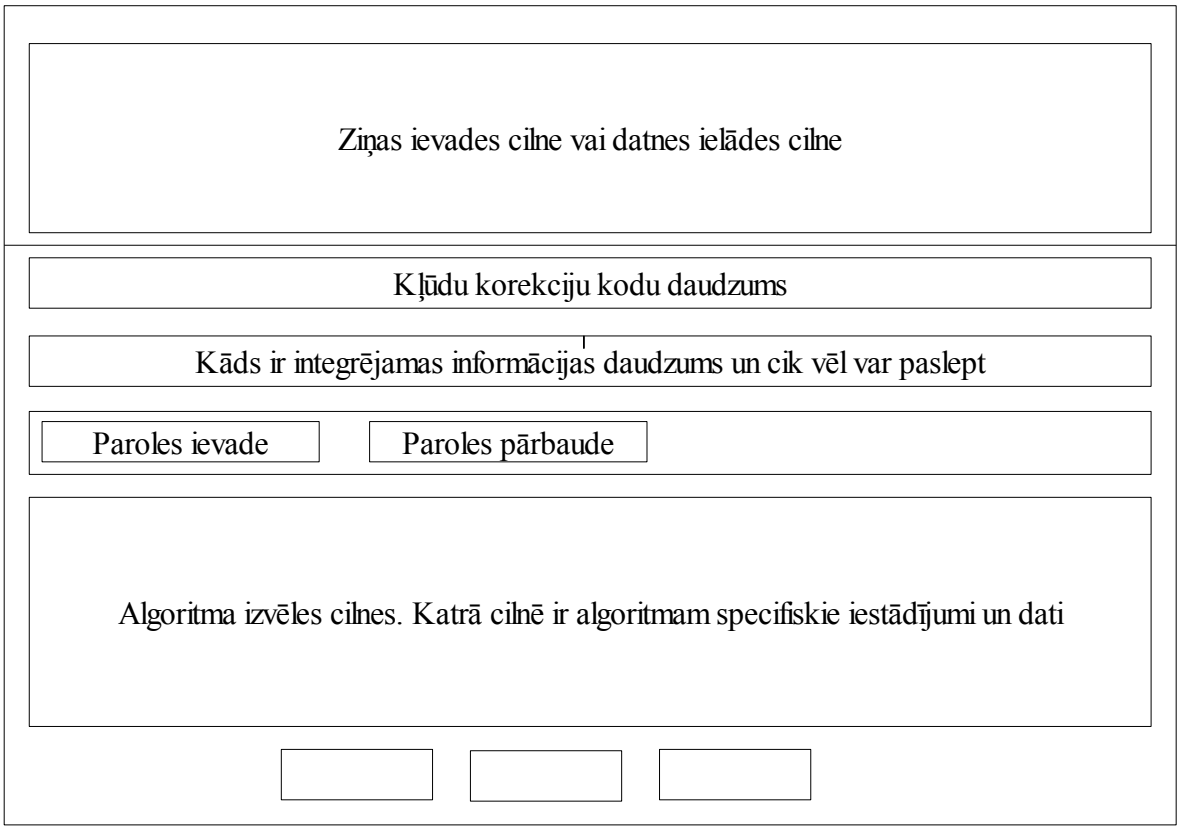
Daļa C glabā ziņu. Ja Rīda-Solomona kodu stiprums ir nulle, tad nekodēta ziņa ir vienkārši satur ziņas bitus pēc kārtas.

Ja RS kodu stiprums ir lielāks par nulli, tad ziņas kodēšanai ir lietots Rīda-Solomona kodu modelis RS(N, K), kur N ir vienmēr 15, bet K = 15 - «RS kodu stiprums no daļas A». Ziņa jāsadalā blokos pa $K \times 4$ biti katrā. Lai sadalītu ziņu veselajā bloku skaitā, papildināsim jeb «polsterēsim» to ar nulles bitiem. Katram blokam izrēķināsim RS kļūdu korekcijas kodus. Kodētai ziņai pierakstīsim vispirms bloka RS kodus, pēc tām pašu bloku. Ja atkodējot ziņu kaut vienā blokā ir atrastas labojamas kļūdas, kļūdas jāizlabo un lietotājam jāpaziņo ka ziņā bija un ir izlabotas kļūdas. Ja kaut vienu bloku atjaunot neizdevās, lietotājam jāpaziņo, ka ziņā ir nelabojamās kļūdas, bloki jāuztver un lietotājam jārada «kā tie ir»; šī gadījumā par labotājiem blokiem (ja tādi ir) lietotājam nav jāpaziņo.

6. Lietotāja saskarnes dizains

6.1. Ziņas slēpšanas dialogs

Ziņas slēpšanas dialogam jāizskatās sekojoši:



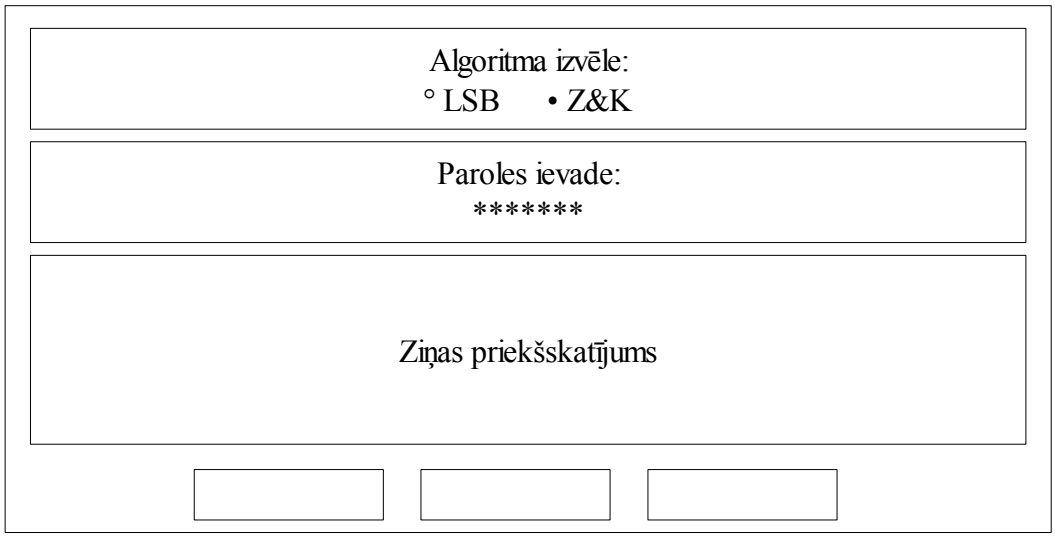
Dialoga augšējā daļā jābūt ziņas ievades rīkiem: cilnei «*Unicode Text*» jāpiedāvā iespēja paslēpt tekstu, cilnē «*File*» jābūt faila ielādes rīkiem.

Dialogā apakšējā daļā jābūt rīkiem, kas attiecas uz ziņas slēpšanas procedūru:

1. Divi paroles ievades lauki. Viens ievadei un otrais paroles pārbaudei. Ziņu var paslēpt tikai ja lauku saturs sakrīt.
2. Divas cilnes, kārtā dati, kas attiecas uz konkrēto paslēpšanas algoritmu. Ziņa būs paslēpta ar to algoritmu, kurai attiecīga cilne ir atvērta.
3. Trīs pogas:
    1. *Preview* – slēpj ziņu, attēls ar paslēpto ziņu parādās galvenajā loga, bet logu aizverot izmaiņas nesaglabājas.
    2. *Apply* – slēpj ziņu, attēls ar paslēpto ziņu parādās galvenajā loga un izmaiņas tūlīt saglabājas.
    3. *Cancel* – aizver logu, visas nesaglabātas izmaiņas izzust.

6.2. Ziņas izņemšanas dialogs

Ziņas izņemšanas dialogam jāizskatās sekojoši:



Dialoga augšējā daļā atrodas algoritma izvēlēs un paroles ievades elementi. Apakšējā daļā atrodas ziņas priekšskatījuma lauks.

Ja nospiest pogu Preview ziņas priekšapskates laukā parādīsies ziņas teksts vai paziņojums par kļūdu: ka ziņa ir bojāta vai tās tur vienkārši nav.

Jebkurā laikā var nospiest pogu Save un saglābāt ziņu failā. (Būs piedāvāta iespēja izvēlēties faila vārdu un ceļu.)

Nospiežot pogu Cancel, dialogs aizvērsies.

Moduļa Steganography testēšanas plāns

1. Ievads

Šīs plāns ir moduļa *Steganography* testēšanas plāns. Testēšana tika veikta izmantojot tehnoloģiju *JUnit* un izstrādes vides *IntelliJ IDEA 9.0.2* iespējas.

2. Saistītie dokumenti

Šī plāna veidotie testi balstās uz moduļa *Steganography* prasību specifikācijā izvirzītām prasībām un projektējuma aprakstu.

3. Testējamie vienumi

Pakotne	Klase	Vai tiks testēts?	Statuss
`zhmakin`	`FinalActions`	Nē	
	`LinearProgressEstimation`	Jā	Notestēts
	`Math`	Nē	
	`Util`	Nē	
	`YCbCr`	Jā	Notestēts
`zhmakin.steganography`	`BitsOfImageBySignificance`	Jā	Notestēts
	`BitString`	Jā	Notestēts
	`ConformityTable`	Jā	Notestēts
	`Message`	Jā	Notestēts
	`Util`	Nē	
	`ZhaoAndKoch`	Jā	Notestēts
`zhmakin.steganography.ECC`	`ReedSolomon`	Nē	

4. Testējamās raksturiezīmes

Jāpārbauda, vai testējamo vienību realizētas funkcijas atbilst moduļa prasību specifikācijai un *JavaDoc* aprakstītām. Atbilstoši prasību specifikācijai jānotestē ātrdarbība. Jānotestē metodes darbība robežgadījumos un pie robežvērtībām.

5. Netestējamās raksturiezīmes

Netikts testēta slodze uz sistēmu, tajā skaitā atmiņas patēriņš.

6. Testēšanas pieēja

Lietojama testēšanas pieeja ir tā saucamā baltas kastes testēšana. Tika veikta svarīgāko klašu vienībtestēšana izmantojot tehnoloģiju *JUnit* un izstrādes vides *IntelliJ IDEA* iespējas. Katrai testējamai klasei bija izveidota viena atbilstoša testu klase.

7. Testējamā vienuma novērtēšanas kritēriji

Testēto klašu vairumam *JUnit* veic visu darbu automātiski un paziņo, ja kāds no nosacījumiem neizpildās. Dažos gadījumos, piemērām testējot ātrdarbību, testa klase drukā apkopotu testa statistiku (ātrdarbības gadījumā tas ir iekodēto vai nolasīto bitu skaits sekundē) un gala lēmumu par testa izpildes rezultātu pieņem testētājs.

8. Atlikšanas kritēriji un atsākšanas prasības

Testēšanas vienumi tika izstrādāti viena pēc otrās. Izstrādē neturpinājās kamēr visas zināmās kļūdas bija izlabotas. Šāda pieeja bija izvēlētā tāpēc kā svarīgākie testējamie vienumi ir savstarpēji stipri atkarīgi.

9. Testēšanas nodevumi

1. Moduļa *Steganography* testēšanas plāns
2. Moduļa *Steganography* testa piemēru specifikācija
3. Moduļa *Steganography* testēšanas žurnāls

10. Testēšanas uzdevumi

1. Jāsagatavo testēšanas dokumentu melnrakstu veidlapas;
2. jāpārliecinās kā integrēta izstrādes vide *IntelliJ IDEA* ir pareizi nokonfigurēta (īpaši tas attiecas uz moduļa *JUnit* pieslēgšanu);
3. jādabū pēdēja strādājoša pirmkoda versija no SVN repozitorija un jāatrisina visi versiju konflikti, ja rodas.

11. Vides vajadzības

Testēšanas procesam ir nepieciešamas sekojoša programmatūra un dati:

1. pilns programmas «AttēluApstrāde» pirmkods;
2. *IntelliJ IDEA* (versijas 9.0.1, 9.0.2 vai savietojama);
3. *Java Development Kit* (versija 1.6.0_18 vai savietojama);
4. bibliotēka *JUnit* (versija 4.8.2 vai savietojama).

Moduļa Steganography testa piemēru specifikācija

1. Klase zhmakin.LinearProgressEstimation

1.1. Testpiemēru kopa

1.1.1. Testpiemērs 1

Testpiemēra ID:	`s1t1`
Apraksts:	Novērtēsim atlikušo izpildes laiku procesam, kas sākas 0-tajā milisekundē un kuram ir 16 posmi no 0. līdz 15. 20. milisekundē mēs tikam līdz 5. posmam (1). Trīsdesmitajā sekundē mēs jau esam 10. posmā. (2)
Sagaidāmais rezultāts:	(1) Šeit novērtētājām atlikušajam laikam jābūt vienādam ar 40 milisekundēm. (2) Novērtētājs beigu laiks ir 15 milisekundes.

1.1.2. Testpiemērs 2

Testpiemēra ID:	`s1t2`
Apraksts:	Novērtēsim atlikušo izpildes laiku procesam, kas sākas 10. milisekundē un kuram ir 41 posms no 0. līdz 40. Desmitajā milisekundē kopš procesa sākumam mēs tikam līdz 20. posmam (1) . Pēc tam secinājām, ka kļūdījāmies mūsu novērtējumos 20. milisekundē kopš procesa sakumam mēs esam tikai 10. posmā (2).
Sagaidāmais rezultāts:	(1) Līdz galam paliek 10 milisekundes. (2) Līdz galam paliek 60 milisekundes.

1.1.3. Testpiemērs 3

Testpiemēra ID:	`s1t3`
Apraksts:	Pārbaudīsim robežgadījumus. Novērtēsim atlikušo izpildes laiku procesam, kas sākas 0. milisekundē un kuram ir 11 posmi no 0. līdz 10. Uzreiz pēc sākšanas mēs esam 0. posmā. Kāds nebūtu laiks līdz procesa beigām paliek (1). Procesa beigās sasniedzot, līdz procesa beigām paliek (2).
Sagaidāmais rezultāts:	(1) «Bezgalība», `Integer.Max_VALUE` (2) Nekas nepaliek, 0 milisekundes.

2. Klase zhmakin.YCbCr

2.1. Testpiemēru kopa 1

2.1.1. Testpiemērs 1

Testpiemēra ID:	`s1t1`
Apraksts:	Klase tika inicializēta no `java.awt.Color` tipa objektiem ar *RGB* vērtībām, kuriem attiecīgas *YCbCr* vērtības ir iepriekš izrēķinātas.
Sagaidāmais rezultāts:	Vērtībām jāsakrīt ar rēķinātām.

2.1.2. Testpiemērs 2

Testpiemēra ID:	`s1t2`
Apraksts:	Tika apstaigāta visa *RGB* krāsu telpa (16,8 milj. krāsu), ar kārtu krāsu tika inicializēts `java.awt.Color` tipa objekts, kas pēc tām konvertēts *YCbCr* krāsu telpā un atpakaļ `java.awt.Color`. Sākotnējs objekts salīdzināms ar iegūto, ja nesakrīt metām izņēmumu.
Sagaidāmais rezultāts:	Testa veiksmīga izpildīšanās.

3. Klase zhmakin.steganography.BitsOfImageBySignificance

3.1. Testpiemēru kopa 1

3.1.1. Testpiemērs 1

Testpiemēra ID:	`s1t1`
Nepieciešami faili:	`3D_Graphics_Su_25_512x512.png` `Photos_Lenna_512x512.png`
Apraksts:	Katrā konteinera bitā rakstām sākuma «0» un uzreiz pārbaudām vai ir «0», pēc tām «1» un pārbaudām vai bitā ir «1».
Sagaidāmais rezultāts:	Pārbaudījumi izpildās veiksmīgi.

3.1.2. Testpiemērs 2

Testpiemēra ID:	`s1t2`
Nepieciešami faili:	`3D_Graphics_Su_25_512x512.png` `Photos_Lenna_512x512.png`
Apraksts:	Katru konteinera bitu ar pāra indeksu uzstādām un «0», katru nepāra bitu uzstādām uz «1». Saglabājam konteineru formātā PNG. Nolasām to un pārbaudām vai pāra bitos ir «0» un nepāra bitos ir «1».
Sagaidāmais rezultāts:	Pārbaudījumi izpildās veiksmīgi.

3.2. Testpiemēru kopa 2

3.2.1. Testpiemērs 1

Testpiemēra ID:	`timeTrialEmbedmentRaw`
Nepieciešami faili:	`3D_Graphics_Su_25_512x512.png` `Photos_Lenna_512x512.png`
Apraksts:	Rakstam katrā konteinera bitā «0» un rēķinām, cik daudz laika aizņem visu bitu aizpildīšana.
Sagaidāmais rezultāts:	Nomērītais ātrums ir vienāds vai pārsniedz PPS izvirzītās prasības ātrdarbībai.

3.2.2. Testpiemērs 2

Testpiemēra ID:	`timeTrialExtractionRaw`
Nepieciešami faili:	`3D_Graphics_Su_25_512x512.png` `Photos_Lenna_512x512.png`
Apraksts:	Lasām visus konteinera bitus un rēķinām, cik daudz laika aizņem visa konteinera bitu vērtību nolasīšana.
Sagaidāmais rezultāts:	Nomērītais ātrums ir vienāds vai pārsniedz PPS izvirzītās prasības

	ātrdarbībai.

4.1.1. Testpiemērs 1

Testpiemēra ID:	`s1t1`
Apraksts:	Tika veidotas divas divas bitu virknes – vienā garumā desmit, otrā garumā nulle. Tās bija aizpildītās ar sekojošu saturu: 0101010101.
Sagaidāmais rezultāts:	Virknēm ir vienāds garums un saturs.

4.1.2. Testpiemērs 2

Testpiemēra ID:	`s1t2`
Apraksts:	Veidotas divas bitu virknes `a` un `b` ar vienādu saturu: 0101010101. Virknei `a` ar metodi `attach()` tika piekabināta virkne `b`, un pēc tām virknei `a` piekabināta paša virkne `a`.
Sagaidāmais rezultāts:	Virkne `a` ir četras reizes lielāka par virkni `b`, un tas garums ir 40 biti. Visi `a` virknes pāra biti (skaitot no nulles) ir «0», un nepāra biti - «1».

4.1.3. Testpiemērs 3

Testpiemēra ID:	`s1t3`
Apraksts:	Tika veidota 101 simbolu virkne ar saturu «ABCABCABCA...» garumā no nulles līdz 100. Katra virkne bija pārveidota par bitu virkni, bet bitu virkne pārveidota atpakaļ par simbolu virkni.
Sagaidāmais rezultāts:	Visām simbolu virknēm sākuma un pārveidojumu rezultātiem jāsakrīt.

4.1.4. Testpiemērs 4

Testpiemēra ID:	`s1t4`
Apraksts:	Tika uzģenerētās bitu virknes skaitā 1000. Virknes garums auga no 0 līdz 999 bitiem. Virknes bija inicializētas no pseidogadījuma skaitļu ģeneratora (grauds ir konstants). Katra virkne bija saglabāta datnē un pēc tām nolasīta ar divām metodēm: `loadFile( String filename )` un `loadFile( String filename, BitString result, java.awt.Container owner, FinalActions actions ).`

Sagaidāmais rezultāts:	Abu nolasīšanu rezultātu saturiem jāsakrīt ar to kas bija ierakstīts, un pierakstīto nuļļu skaitam jābūt prognozējamam pēc likuma:
	`padding = 8 - i % 8;`
	`if ( padding == 8 ) padding = 0;`

5. Klase zhmakin.steganography.ConformityTable

5.1. Testpiemēru kopa 1

5.1.1. Testpiemērs 1 – nenegatīva permutācija

Testpiemēra ID:	`s1t1`
Apraksts:	Veidojam 100 permutācijas $(0;i)$, kur $i \in [1..100]$. Katrai i -tai permutācija pārbaudām, ka katrs skaitlis no diapazona $[0;i]$ sastopas tieši vienu reizi.
Sagaidāmais rezultāts:	Pārbaude notiek veiksmīgi.

5.1.2. Testpiemērs 2 – pozitīva permutācija

Testpiemēra ID:	`s1t2`
Apraksts:	Veidojam 100 permutācijas $(i;i \times 10)$, kur $i \in [1..100]$. Katrai i -tai permutācija pārbaudām, ka katrs skaitlis no diapazona $[i;i \times 10]$ sastopas tieši vienu reizi.
Sagaidāmais rezultāts:	Pārbaude notiek veiksmīgi.

5.1.3. Testpiemērs 3 – permutācija no veseliem skaitļiem

Testpiemēra ID:	`s1t3`
Apraksts:	Veidojam 100 permutācijas $(-i;i \times 2)$, kur $i \in [1..100]$. Katrai i -tai permutācija pārbaudām, ka katrs skaitlis no diapazona $[-i;i \times 2]$ sastopas tieši vienu reizi.
Sagaidāmais rezultāts:	Pārbaude notiek veiksmīgi.

6. Klase zhmakin.steganography.Message

6.1. Testpiemēru kopa 1

6.1.1. Testpiemērs 1

Testpiemēra ID:	`s1t1`
Apraksts:	Tiek veidota zemā līmeņa ziņa ar sekojošām raksturiezīmēm: Saturs: «`Copyright © by Andrey Zhmakin`», garums 464 biti, kļūdu korekciju kodu stiprums 20%. Pēc tām ziņa ir atkodētai.
Sagaidāmais rezultāts:	Atkodētai ziņai jāsakrīt ar oriģinālo.

6.1.2. Testpiemērs 2

Testpiemēra ID:	`s1t2`
Apraksts:	Veidojam 4000 ziņas garumā no nulles līdz 3999 bitiem. Katru iekodējam 20 reizēs ar izturības pakāpēm no 0% līdz 100% (ar sili 5%), Un atkodējam atpakaļ.
Sagaidāmais rezultāts:	Atkodētai ziņai jāsakrīt ar oriģinālo.

7. Klase zhmakin.steganography.ZhaoAndKoch

7.1. Testpiemēru kopa 1

7.1.1. Testpiemērs 1

Testpiemēra ID:	`s1t1`
Apraksts:	Pārbaudām vai vērtības kvantizācijas tabulas šūnās $(u_1; v_1)$ un $(u_2; v_2)$ ir vienādas.
Sagaidāmais rezultāts:	Vērtībām jābūt vienādām.

7.1.2. Testpiemērs 2

Testpiemēra ID:	`s1t2`
Nepieciešami faili:	`3D_Graphics_Su_25_512x512.png` `Photos_Lenna_512x512.png`
Apraksts:	Ar katru izturības vērtību no 0% līdz 100% ar soli 10% (ti 0%, 10%, 20%, utt.) veicam sekojošo: Katru konteinera bitu uzstādām uz «0», pārbaudām vai visi biti ir uzstādīti uz «0».
Sagaidāmais rezultāts:	Pārbaude notiek veiksmīgi.

7.1.3. Testpiemērs 3

Testpiemēra ID:	`s1t3`
Nepieciešami faili:	`3D_Graphics_Su_25_512x512.png` `Photos_Lenna_512x512.png`
Apraksts:	Ar katru izturības vērtību no 0% līdz 100% ar soli 10% (ti 0%, 10%, 20%, utt.) veicam sekojošo: Katru konteinera bitu uzstādām uz «1», pārbaudām vai visi biti ir uzstādīti uz «1».
Sagaidāmais rezultāts:	Pārbaude notiek veiksmīgi.

7.1.4. Testpiemērs 4

Testpiemēra ID:	`s1t4`
Nepieciešami faili:	`3D_Graphics_Su_25_512x512.png` `Photos_Lenna_512x512.png`
Apraksts:	Ar katru izturības vērtību no 0% līdz 100% ar soli 10% (ti 0%, 10%, 20%, utt.) veicam sekojošo: Katru bitu ar pāra indeksu uzstādām uz «0», ar nepāra indeksu uz «1». Pēc visu bitu uzstādīšanas pārbaudām, vai pāra bitos ir «0» un nepāra bitos «1».
Sagaidāmais rezultāts:	Pārbaude notiek veiksmīgi.

7.2. Testpiemēru kopa 2

7.2.1. Testpiemērs 1 – rakstīšanās ātrums

Testpiemēra ID:	`timeTrialEmbedmentRaw`
Nepieciešami faili:	`3D_Graphics_Su_25_512x512.png` `Photos_Lenna_512x512.png`
Apraksts:	Visos konteinera pāra bitus konsekventi uzstādām uz «0», nepāra – uz «1» un mērām cik daudz laika tas aizņems.
Sagaidāmais rezultāts:	Nomērītais ātrums ir vienāds vai pārsniedz PPS izvirzītās prasības ātrdarbībai.

7.2.2. Testpiemērs 2 – lasīšanās ātrums

Testpiemēra ID:	`timeTrialExtractionRaw`
Nepieciešami faili:	`3D_Graphics_Su_25_512x512.png` `Photos_Lenna_512x512.png`
Apraksts:	Konsekventi lasām visus konteinera bitus un mērām cik daudz laika tas aizņems.
Sagaidāmais rezultāts:	Nomērītais ātrums ir vienāds vai pārsniedz PPS izvirzītās prasības ātrdarbībai.

Moduļa Steganography testēšanas žurnāls

1. Ievads

Šinī dokumentā ir parādīti moduļa *Steganography* testēšanas rezultāti. Testēšana tika veikta saskaņā ar moduļa testēšanas plānu.

2. Saistītie dokumenti

Moduļa *Steganography* testēšanas plāns

Moduļa *Steganography* prasību specifikācija un projektējuma apraksts

Pielikums 1: *JavaDoc*

Klase	Testpiemēru ID	Problēmas un risinājumi
`zhmakin.LinearProgressEstimation`	`s1t1` – `s1t3`	
`zhmakin.YCbCr`	`s1t1` – `s1t2`	
`zhmakin.steganography.BitsOfImageBySignificance`	`s1t1` – `s1t2`, `timeTrialEmbedmentRaw` `timeTrialExtractionRaw`	
`zhmakin.steganography.BitString`	`s1t1` – `s1t4`	PRB01
`zhmakin.steganography.ConformityTable`	`s1t1` – `s1t3`	
`zhmakin.steganography.Message`	`s1t1` – `s1t2`	
`zhmakin.steganography.ECC.ReedSolomon`		
`zhmakin.steganography.ZhaoAndKoch`	`s1t1` – `s1t4` `timeTrialEmbedmentRaw` `timeTrialExtractionRaw`	PRB02

3. Testēšanas gaitā atklātās problēmas un risinājumi

3.1. Problēma 1 (PRB01)

Klase:	`zhmakin.steganography.BitString`
Testpiemēra ID:	`s1t2`
Apraksts:	Testēšanas gaitā atklājās, ka izsaucot metodi `attach()` forma `a.attach(a)`, t.i. mēģinot bitu virknes piekabināt tās saturu rodas nopietnā kļūda - *Java* virtuālas mašīnas kaudzes pārpildījums ar izņēmumu: `java.lang.OutOfMemory`.
Risinājums:	Kļūdas labošana metodē `attach()`.
Statuss:	Izlabots.

3.2. Problēma 2 (PRB02)

Klase:	`zhmakin.steganography.ZhaoAndKoch`
Testpiemēra ID:	`s1t1`
Apraksts:	Norādot minimālo izturības pakāpi, neizdevās saglabāt bita vērtību dažos specifiskajos blokos.
Risinājums:	Eksperimentālā viedā tika noteikts un pieskaņots minimāls izturības slieksnis, labojamā konstante `ReedSolomon.MINIMAL_X`.
Statuss:	Izlabots.

Projekta organizācija

Projekta sakumā notika iepazīšanās ar Java programmēšanas valodas pamatiem un programmas «AttēluApstrāde» projektējumu. Pēc tām konsekventi bija izstrādāti moduļa prasību specifikācija un projektējuma apraksts. Vienībtestēšana katrai klasei bija veikta tās izstrādēs laikā.

Kvalitātes nodrošināšana

Lai nodrošinātu izstrādāta moduļa kvalitāti tika veikti sekojoši pasākumi:

1. bija uzrakstītā moduļa dokumentācija: «Prasību specifikācija» un «Projektējuma apraksts», tas bija darīts ievērojot Latvijas un starptautisko standartu prasības;
2. bija pēc iespējas pilnībā izmantots programmēšanas valodas Java potenciāls, lietotas progresīvas tehnikas, kā piemēram *assertion programming*;
3. bija lietotās vispārīgas labas programmēšanas prakses: vienošanas par kodēšanas stilu, koda komentēšana un strukturēšana pakotnēs, klasēs un metodēs, kods bija labi komentēts.

KONFIGURĀCIJU PĀRVALDĪBA

Konfigurāciju pārvaldībai tika izmantota konfigurāciju vadības sistēma *Subversion*. Repozitorijs glabājas bezmaksas virtuālās mitināšanas projektā *Google Code*. Konfigurāciju vadības sistēma *Subversion* ļāva komandai no četriem izstrādātājiem kopā strādāt par vienu programmu cits citam traucējot minimāli. Turklāt *Subversion* automātiski saglabā izmaiņu vēsturi, kas ļauj nesāpīgi veikt izmaiņas un atkāpties uz jebkuras datnes vecāko versiju.

Darbietilpības novērtējums

Darbietilpības novērtēšanai ir lietots *COCOMO* modelis:

SF1	4,0	Nav pieredzes šādu uzdevumu veikšanā
SF2	0,5	Nedaudz ietekmē *Java VM* ierobežojumi
SF3	1,0	Daži lēmumi par programmas uzbūvi bija pieņemti izstrādes laikā
SF4	0	Komunikācija nebija vajadzīga
SF5	4,0	Izstrādēs procesa organizācija bija diezgan laba
B =	1,11	
EM1	1,0	Izstrādātāja spējas samērā augstās
EM2	1,0	Uzsvars uz dokumentēšanu bija neliels
	0,9	Neliela produkta sarežģītība
	0,8	Datu bāzes nav
EM3	1,1	Atkārtotās pielietojamības prasības bija vidējas
EM4	1,0	Ierobežojumi uz ātrdarbību nebija īpaši strikti
EM5	1,1	Izstrādes vide nav īpaši stabila
	0,8	Izstrādes vide ir ļoti integrēta
	1,1	Atbalsts komandā bija vājš
EM6	1,5	Izstrādes kalendārais plāns bija saspiests
E = 1,15		
S = 3600 / 3 = 1200 funkcionālas koda rindas		
PM = 3,52 (personmēneši)		

Ka redzams, izstrādāta produkta darbietilpība atbilst prasītiem 3 personmēnešiem.

Rezultāti un secinājumi

Prakses laikā tika izpētīti, uzlaboti un realizēti divi informācijas slēpšanas algoritmi. Tie veido programmas svarīgāko daļu. Programmai bija izveidota draudzīga lietotāja saskarne. Programmas svarīgākie komponenti bija notestēti un programmai bija uzrakstīta dokumentācija. Daļa no prakses laika bija veltītā arī programmas «AttēluApstrāde» uzlabošanai, *Java* programmēšanas valodas, progresīvu programmēšanas tehniku (dokumentācijas ģenerēšanas no koda, *assertion programming* utt.) un citu tehnoloģiju apguvei.

Modulis *Steganography* principā ir pilnvērtīgs pabeigts produkts, bet ir iespējamas arī papildus uzlabošanas. Piemēram, Rīda-Solomona kodu vietā var izmantot *Viberti* vai *Turbo* kodus. Dažviet arī vajadzētu uzlabot ātrdarbību un lietot atmiņu taupīgāk. Var uzlabot arī slēpšanas algoritmus.

Izmantotā literatūra

1. Pārskats par *Wassenaar* vienošanām: http://rechten.uvt.nl/koops/cryptolaw/cls2.htm#Wassenaar
2. Par Java iebūvēto `java.util.Random`: http://java.sun.com/j2se/1.4.2/docs/api/java/util/Random.html
3. Stefan Katzenbeisser and Fabien A. P. Petitcolas. Information Hiding Techniques for Steganography and Digital Watermarking, ISBN 978-1-58053-035-4, pages 56-61, 2000.
4. http://www.eccpage.com/rs.c – Implementation of Reed-Solomon codes in C by Simon Rockliff.

Pielikums 1 – Steganogrāfijas algoritmu salīdzināšana

Nr.	Algoritms	Konteinera apjoms, bitos	Ietekme konteineri	Ātrdarbība	Izturība
1.	Slēpšana jaunākajos bitos	garums×platums×24	Atkarīga no ziņas izmēra	Ļoti augsta	Zema, zaudējot pikseļus, tiks zaudēti attiecīgie ziņas biti.
2.	*Zhao&Koch* vienkāršotais	garums×platums/64	Vidēja, regulējama.	Vidēja	Zema, zaudējot pikseļus, tiks zaudēti attiecīgie ziņas biti.
3.	*Zhao&Koch*	Mazāks pār (garums×platums/64)	Mazāka par 2., regulējama.	Vidēja	Ļoti zema, zaudējot pikseļus, tiks zaudēti attiecīgie ziņas biti.
4.	*Pereira&Pun*	Ap 100 bitiem.	Vidēja, regulējama.	Ļoti zema	Ļoti izturīgs

Avoti:

[2]. «Information Hiding Techniques For Steganography And Digital Watermark», Stefan Katzenbeisser and Fabien A. Page 59.

[3]. Zhao, J., and E. Koch, "Embedding Robust Labels into Images for Copyright Protection," in *Proceedings of the International Conference on Intellectual Property Rights for Information, Knowledge and New Techniques,*München, Wien: Oldenbourg Verlag, 1995, pp. 242–251.

[4]. «Fast Robust Template Matching for Affine Resistant Image Watermarks» by Pereira , Shelby and Pun, Thierry.

Pielikums 2 – JavaDoc

ZHMAKIN.STEGANOGRAPHY

Class BitsOfImageBySignificance

java.lang.Object
  └ **zhmakin.steganography.BitsOfImageBySignificance**

---

public class **BitsOfImageBySignificance**extends java.lang.Object

---

Nested Class Summary

class	**BitsOfImageBySignificance.BrokenMessageException**
class	**BitsOfImageBySignificance.ImageTooSmallToHoldDataException**
class	**BitsOfImageBySignificance.IndexOutOfBoundsException**

Field Summary

static int	**NUM_OF_LAYERS**

Constructor Summary

BitsOfImageBySignificance(RasterImage image) Creates an object of class BitsOfImageBySignificance linked to object of class util.RasterImage; You still have to call initConformityTables(long key) before using anything.	
BitsOfImageBySignificance(RasterImage image, long key) Creates an object of class BitsOfImageBySignificance linked to object of class util.RasterImage; Every instance is ready to use with encryption key key.	

Method Summary

BitString	**decode_trivially**() Extracts message encoded with encode_trivially(BitString data).
void	**encode_trivially**(BitString data) Encodes a message in container in the most trivial way.
boolean	**get**(int index)

	Used to determinate the state of the `index`-th bit of the container.
`boolean`	**`get`**`(int layer, int indexInImage, int component)` Used to determinate the state of the certain bit of the container.
`int`	**`getComponent`**`(int index)` Used to determinate color component `index`-th bit corresponds to.
`int`	**`getIndexInImage`**`(int index)` See description of the return value.
`int`	**`getIndexInLayer`**`(int index)` Returns the index in layer.
`int`	**`getLayer`**`(int index)` Used to determinate the layer `index`-th bit corresponds to.
`int`	**`getLayerSize`**`()` Gets size of image layer.
`int`	**`getSize`**`()` Gets size of bit container.
`void`	**`initConformityTables`**`(long key)` Initializes conformity tables for message encryption.
`void`	**`set`**`(int index, boolean state)` Used to change the state of the `index`-th bit of the container.
`void`	**`set`**`(int layer, int indexInImage, int component, boolean state)` Used to change the state of the certain bit of the container.

Methods inherited from class java.lang.Object
`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Field Detail

1.1. NUM_OF_LAYERS

`public static final int` **`NUM_OF_LAYERS`**

See Also:

Constant Field Values

Constructor Detail

1.2. BitsOfImageBySignificance

`public` **`BitsOfImageBySignificance`**`(RasterImage image)`

Creates an object of class `BitsOfImageBySignificance` linked to object of class

util.RasterImage; You still have to call initConformityTables(long key) before using anything.

Parameters:

image - A non-null object of class util.RasterImage.

1.3. BitsOfImageBySignificance

```
public BitsOfImageBySignificance(RasterImage image,
                                 long key)
```

Creates an object of class BitsOfImageBySignificance linked to object of class util.RasterImage; Every instance is ready to use with encryption key key.

Parameters:

image - Image watermark to be embedded in.
key - Encryption key.

Method Detail

1.4. initConformityTables

```
public void initConformityTables(long key)
```

Initializes conformity tables for message encryption.

Parameters:

key - Encryption key.

1.5. getSize

```
public int getSize()
```

Gets size of bit container.

Returns:

Returns size of bit container.

1.6. getLayerSize

```
public int getLayerSize()
```

Gets size of image layer.

Returns:
Returns size of image layer.

1.7. getIndexInLayer

```
public int getIndexInLayer(int index)
                    throws BitsOfImageBySignificance.IndexOutOfBoundsException
```

Returns the index in layer.

Parameters:
`index` - Index in container.
Returns:
Returns index in layer.
Throws:
`java.lang.IndexOutOfBoundsException` - Throws if `index` is out of layer's bounds.
`BitsOfImageBySignificance.IndexOutOfBoundsException`

1.8. getLayer

```
public int getLayer(int index)
             throws BitsOfImageBySignificance.IndexOutOfBoundsException
```

Used to determinate the layer `index`-th bit corresponds to.

Parameters:
`index` - Index in the bit container.
Returns:
Returns index of the layer [0..7] corresponding to the bit at the `index`-th position.
Throws:
`java.lang.IndexOutOfBoundsException` - Exception is thrown if `index` is out of [0..`getSize()` - 1] bounds.
`BitsOfImageBySignificance.IndexOutOfBoundsException`

1.9. getComponent

```
public int getComponent(int index)
                 throws BitsOfImageBySignificance.IndexOutOfBoundsException
```

Used to determinate color component `index`-th bit corresponds to.

Parameters:
`index` - Index in the bit container.

Returns:

Returns 0, 1 or 2 if index-th bit corresponds to red, green or blue component respectively.

Throws:

`java.lang.IndexOutOfBoundsException` - Exception is thrown if `index` is out of [0..`getSize()` - 1] bounds.
`BitsOfImageBySignificance.IndexOutOfBoundsException`

1.10. getIndexInImage

```
public int getIndexInImage(int index)
                    throws BitsOfImageBySignificance.IndexOutOfBoundsException
```

See description of the return value.

Parameters:

`index` - Index in the bit container.

Returns:

Returns index of the `util.RasterImage.pixels` array element corresponding to the `index`-th bit on the container.

Throws:

`java.lang.IndexOutOfBoundsException` - Exception is thrown if `index` is out of [0..`getSize()` - 1] bounds.
`BitsOfImageBySignificance.IndexOutOfBoundsException`

1.11. get

```
public boolean get(int layer,
                   int indexInImage,
                   int component)
            throws BitsOfImageBySignificance.IndexOutOfBoundsException
```

Used to determinate the state of the certain bit of the container.

Parameters:

`layer` - Refers to the bit [0..7] of the color component.
`indexInImage` - Index in the `util.RasterImage.pixels` array.
`component` - Is 0, 1 or 2 which stands for red, green or blue respectively.

Returns:

Returns state on the corresponding bit of container.

Throws:

`java.lang.IndexOutOfBoundsException` - Exception is thrown if `index` is out of [0..`getSize()` - 1] bounds.
`BitsOfImageBySignificance.IndexOutOfBoundsException`

1.12. set

```
public void set(int layer,
                int indexInImage,
                int component,
                boolean state)
         throws BitsOfImageBySignificance.IndexOutOfBoundsException
```

Used to change the state of the certain bit of the container.

Parameters:

`layer` - Refers to the bit [0..7] of the color component.
`indexInImage` - Index in the `util.RasterImage.pixels` array.
`component` - Is 0, 1 or 2 which stands for red, green or blue respectively.
`state` - The in which the `index`-th bit currently is.

Throws:

`java.lang.IndexOutOfBoundsException` - Exception is thrown if `index` is out of [0..`getSize()` - 1] bounds.
`BitsOfImageBySignificance.IndexOutOfBoundsException`

1.13. get

```
public boolean get(int index)
            throws BitsOfImageBySignificance.IndexOutOfBoundsException
```

Used to determinate the state of the `index`-th bit of the container.

Parameters:

`index` - Index in the bit container.

Returns:

Returns state of the `index`-th bit in the bit container.

Throws:

`java.lang.IndexOutOfBoundsException` - Exception is thrown if `index` is out of [0..`getSize()` - 1] bounds.
`BitsOfImageBySignificance.IndexOutOfBoundsException`

1.14. set

```
public void set(int index,
                boolean state)
         throws BitsOfImageBySignificance.IndexOutOfBoundsException
```

Used to change the state of the `index`-th bit of the container.

Parameters:

`index` - Index of the bit in the bit container.
`state` - State to be assigned to the `index`-th bit in the bit container.

Throws:

`java.lang.IndexOutOfBoundsException` - Exception is thrown if `index` is out of [0..`getSize()` - 1] bounds.
BitsOfImageBySignificance.IndexOutOfBoundsException

1.15. encode_trivially

```
public void encode_trivially(BitString data)
                      throws
BitsOfImageBySignificance.ImageTooSmallToHoldDataException
```

Encodes a message in container in the most trivial way.

Parameters:

`data` - Message to hide.

Throws:

BitsOfImageBySignificance.ImageTooSmallToHoldDataException - If the image is too small for such a large message.

1.16. decode_trivially

```
public BitString decode_trivially()
                             throws
BitsOfImageBySignificance.ImageTooSmallToHoldDataException,
                                   BitsOfImageBySignificance.BrokenMessageExcepti
on
```

Extracts message encoded with encode_trivially(BitString data).

Returns:

Returns contents of the message, if found.

Throws:

BitsOfImageBySignificance.ImageTooSmallToHoldDataException - Thrown if image is too small to hold message of any size.
BitsOfImageBySignificance.BrokenMessageException - If there is no message inside or message is broken.

2. ZHMAKIN.STEGANOGRAPHY

CLASS BITSTRING

```
java.lang.Object
  └ java.util.BitSet
      └ zhmakin.steganography.BitString
```

All Implemented Interfaces:

java.io.Serializable, java.lang.Cloneable

```
public class BitStringextends java.util.BitSet
```

See Also:

Serialized Form

Constructor Summary	
BitString() Creates a new object of class BitString with initial size of 0 (zero) bits.	
BitString(java.util.BitSet rhr) Creates a replica of rhr object with exactly the same size.	
BitString(int size) Creates an object of class BitString with initial size of size bits.	

Method Summary	
void	**attach**(BitString bits) Attaches the contents of bits to this.
static java.lang.String	**decodeString**(BitString bits) Decodes an object of class BitString as String.
static BitString	**encodeString**(java.lang.String string) Creates an object of class BitString on the basis of class String.
boolean	**equals**(BitString bits) Compares this with other BitString object.
BitString	**get**(int fromIndex, int toIndex) Create object of class BitString and fills it with the content of this's [fromIndex..toIndex-1].
static BitString	**loadFile**(java.lang.String filename) The basic mean to load a file to BitString.
static void	**loadFile**(java.lang.String filename,

	BitString result, java.awt.Container owner, FinalActions actions) A more sophisticated mean to load a file.
void	**saveToFile**(java.lang.String filename) Saves the content of BitString object to file; the string is padded with zero bits to be placed in a file consisting of 8 bit bytes.
void	**set**(boolean value, int index) Set the index bit to the state of value.
void	**set**(int index) Sets the index-th bit of BitString to true; If index >= this.size(), BitString is extended automatically.
void	**setSize**(int size) Changes size BitString object.
int	**size**() Returns current size of BitString object.
java.lang.String	**toString**() Converts BitString to the string of "1"-s and "0"-s.

Methods inherited from class java.util.BitSet
and, andNot, cardinality, clear, clear, clear, clone, equals, flip, flip, get, hashCode, intersects, isEmpty, length, nextClearBit, nextSetBit, or, set, set, set, xor

Methods inherited from class java.lang.Object
finalize, getClass, notify, notifyAll, wait, wait, wait

Constructor Detail

2.1. BitString

public **BitString**()

Creates a new object of class BitString with initial size of 0 (zero) bits.

2.2. BitString

public **BitString**(int size)

Creates an object of class BitString with initial size of size bits.

Parameters:

size - Initial size of BitString.

2.3. BitString

```
public BitString(java.util.BitSet rhr)
```

Creates a replica of `rhr` object with exactly the same size.

Parameters:
`rhr` - Object to be replicated.

Method Detail

2.4. encodeString

```
public static BitString encodeString(java.lang.String string)
```

Creates an object of class `BitString` on the basis of class `String`.

Parameters:
`string` - `String` object to be converted.
Returns:
Object of class `BitString` created from `String`; bit order is little-endian.

2.5. decodeString

```
public static java.lang.String decodeString(BitString bits)
```

Decodes an object of class `BitString` as String. `Bit order is assumed to be little-endian. BitString is padded with zero bits to be integer dividend of 16.`

Parameters:
`bits` - `BitString` object to be converted.
Returns:
String composed from `bits` object.
See Also:
`BitString.encodeString( String string )`

2.6. loadFile

```
public static BitString loadFile(java.lang.String filename)
                          throws java.io.IOException
```

The basic mean to load a file to `BitString`.

Parameters:

`filename` - File to load.

Returns:

Content of the file; Bit order in bytes is assumed to be little-endian.

Throws:

`java.io.IOException` - Thrown if IO error occurs.

2.7. loadFile

```
public static void loadFile(java.lang.String filename,
                            BitString result,
                            java.awt.Container owner,
                            FinalActions actions)
```

A more sophisticated mean to load a file.

Parameters:

`filename` - Name of the file to be loaded.

`result` - Object of class `BitString` to receive the content of the file.

`owner` - Parent frame for ProgressMonitor window.

`actions` - Actions to perform on different ending scenarios.

See Also:

`zhmakin.FinalActions`, `ProgressMonitor`

2.8. saveToFile

```
public void saveToFile(java.lang.String filename)
                throws java.io.IOException
```

Saves the content of `BitString` object to file; the string is padded with zero bits to be placed in a file consisting of 8 bit bytes.

Parameters:

`filename` - Specifies name of the file the `BitString` object should be saves to.

Throws:

`java.io.IOException` - An exception of class `IOException` is thrown is case of failure.

2.9. set

```
public void set(boolean value,
                int index)
```

Set the `index bit to the state of value.`

Parameters:

`value` - Value to be assigned.

`index` - Index of the bit to get a new value.

2.10. set

```
public void set(int index)
```

Sets the `index`-th bit of `BitString` to true; If `index` >= `this.size()`, `BitString` is extended automatically.

Overrides:

`set` in class `java.util.BitSet`

Parameters:

`index` - Index of the bit to be set to true.

2.11. get

```
public BitString get(int fromIndex,
                     int toIndex)
```

Create object of class `BitString` and fills it with the content of this's [fromIndex..toIndex-1].

Overrides:

`get` in class `java.util.BitSet`

Parameters:

`fromIndex` - Start index to cut out.

`toIndex` - Index of the first behind the copied segment.

Returns:

Segment of size (toIndex - fromIndex) with the content of this's [fromIndex..toIndex-1].

2.12. size

```
public int size()
```

Returns current size of `BitString` object.

Overrides:

`size` in class `java.util.BitSet`

Returns:

Current size of `BitString` object.

2.13. setSize

`public void` **`setSize`**`(int size)`

Changes size `BitString` object.

Parameters:
`size` - New size of the object.

2.14. equals

`public boolean` **`equals`**`(BitString bits)`

Compares `this` with other `BitString` object.

Parameters:
`bits` - `BitString` object `this` should be compared with.
Returns:
Returns `true` iff `this` and `bits` are of equal size and contents; `false` otherwise.

2.15. attach

`public void` **`attach`**`(BitString bits)`

Attaches the contents of `bits` to `this`.

Parameters:
`bits` - `BitString` to attach.

2.16. toString

`public java.lang.String` **`toString`**`()`

Converts `BitString` to the string of "1"-s and "0"-s.

Overrides:
`toString` in class `java.util.BitSet`
Returns:
`String`

3. ZHMAKIN.STEGANOGRAPHY

CLASS CONFORMITYTABLE

```
java.lang.Object
  └ zhmakin.steganography.ConformityTable
```

```
public class ConformityTableextends java.lang.Object
```

Constructor Summary

ConformityTable(int fromIndex, int toIndex) Creates an array of elements.	
ConformityTable(int fromIndex, int toIndex, long seed) Creates random permutation of integers.	

Method Summary

int	**get**(int index) Returns an element corresponding to `index` in permutation.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

3.1. ConformityTable

```
public ConformityTable(int fromIndex,
                       int toIndex)
```

Creates an array of elements. Element value is equal to its position.

Parameters:

`fromIndex` - The lowest integer to be included.
`toIndex` - The first integer not included.

3.2. ConformityTable

```
public ConformityTable(int fromIndex,
                       int toIndex,
                       long seed)
```

Creates random permutation of integers.

Parameters:

`fromIndex` - The lowest integer to be included in permutation.
`toIndex` - The first integer not included in permutation.
`seed` - Random seed.

Method Detail

3.3. get

```
public int get(int index)
```

Returns an element corresponding to `index` in permutation.

Parameters:

`index` - Index in array (fromIndex <= `index` < toIndex).

Returns:

Random value in range [fromIndex;toIndex[.

PIELIKUMS 3 – PROGRAMMAS KODS

```
/**
 * Object of class <code>BitString</code> implements a string of bits with
strictly defined size.
 *
 * @author Andrey Zhmakin
 *
 */

package zhmakin.steganography;

import zhmakin.FinalActions;
import zhmakin.LinearProgressEstimation;

import javax.swing.*;
import java.io.*;
import java.text.DecimalFormat;
import java.util.BitSet;



public class BitString extends BitSet
{

    /**
     * Creates a new object of class <code>BitString</code> with initial size of
0 (zero) bits.
     *
     */

    public BitString()
    {
        super();

        this.size = 0;
    }



    /**
     * Creates an object of class <code>BitString</code> with initial size of
<code>size</code> bits.
     *
     * @param size Initial size of <code>BitString</code>.
     *
     */

    public BitString( int size )
    {
        super( size );

        this.size = size;
    }



    /**
     * Creates a replica of <code>rhr</code> object with exactly the same size.
     *
```

```
     * @param rhr Object to be replicated.
     *
     */

    public BitString( BitSet rhr )
    {
        super();

        for ( int i = 0; i < rhr.length(); i++ )
        {
            if ( rhr.get( i ) )
            {
                this.set( i );
            }
        }

        this.size = rhr.size();
    }



    /**
     * Creates an object of class <code>BitString</code> on the basis of class
<code>String</code>.
     *
     * @param string <code>String</code> object to be converted.
     *
     * @return Object of class <code>BitString</code> created from
<code>String</code>; bit order is little-endian.
     *
     */

    public static BitString encodeString( String string )
    {
        BitString result = new BitString( string.length() << 4 );

        int k = 0;
        for ( int i = 0; i < string.length(); i++ )
        {
            for ( int j = 0; j < 16; j++ )
            {
                if ( ((string.charAt(i) >> j) & 1) == 1 ) { result.set   ( k++ );
}
                else                                      { result.clear( k++ );
}
            }
        }

        return result;
    }



    /**
     * Decodes an object of class <code>BitString</code> as </code>String<code>.
     * Bit order is assumed to be little-endian. BitString is padded with zero
bits to be integer dividend of 16.
     *
     * @see BitString BitString.encodeString( String string )
     *
     * @param bits <code>BitString</code> object to be converted.
```

```
     *
     * @return String composed from <code>bits</code> object.
     *
     */

    public static String decodeString( BitString bits )
    {
        char array[] = new char[bits.size() >> 4];

        for ( int b = 0, c = 0, limit = bits.size() & 0xFFFFFFF0; b < limit; )
        {
            char ch = 0;

            for ( int i = 0; i < 16; i++, b++ )
            {
                if ( bits.get(b) )
                {
                    ch |= 1 << i;
                }
            }

            array[c++] = ch;
        }

        return new String( array );
    }



    /**
     * The basic mean to load a file to <code>BitString</code>.
     *
     * @param filename File to load.
     *
     * @return Content of the file; Bit order in bytes is assumed to be little-
endian.
     *
     * @throws IOException Thrown if IO error occurs.
     *
     */

    public static BitString loadFile( String filename )
        throws IOException
    {
        InputStream in = new FileInputStream( filename );

        int fileSize = in.available();

        BitString result = new BitString( fileSize << 3 );

        for ( int i = 0, j = 0; j < fileSize; j++ )
        {
            int v = in.read();

            for ( int k = 0; k < 8; i++, k++ )
            {
                if ( ((v >> k) & 1) == 1 ) { result.set  ( i ); }
              //else                       { result.clear( i ); }
            }
        }
```

```
        in.close();

        return result;
    }



    /**
     * A more sophisticated mean to load a file.
     *
     * @see zhmakin.FinalActions
     * @see ProgressMonitor
     *
     * @param filename Name of the file to be loaded.
     * @param result   Object of class <code>BitString</code> to receive the
content of the file.
     * @param owner    Parent frame for ProgressMonitor window.
     * @param actions  Actions to perform on different ending scenarios.
     */

    public static void loadFile( String filename, BitString result,
java.awt.Container owner, FinalActions actions )
    {
        ProgressMonitor progressMonitor = new ProgressMonitor( owner, "Loading
message from file", "", 0, 100 );
        progressMonitor.setMillisToPopup( 0 );
        progressMonitor.setMillisToDecideToPopup( 0 );

        FileLoader fileLoader = new FileLoader( filename, result,
progressMonitor, actions );
        fileLoader.start();
    }



    /**
     * File loading thread; A helper class for loadFile( String filename,
BitString result,
     *                                                    java.awt.Container
owner, FinalActions actions ).
     *
     * @see static void BitString.loadFile( String filename, BitString result,
java.awt.Container owner, FinalActions actions )
     *
     */

    private static class FileLoader extends Thread
    {
        public FileLoader( String filename, BitString result, ProgressMonitor
pm, FinalActions actions )
        {
            this.result     = result;
            this.pm         = pm;
            this.actions    = actions;
            this.filename   = filename;
        }



        public void run()
        {
```

```
            try
            {
                InputStream in = new FileInputStream( this.filename );
                int fileSize = in.available();

                this.result.setSize( fileSize << 3 );

                this.pm.setMinimum( 0 );
                this.pm.setMaximum( fileSize );

                this.progress = new LinearProgressEstimation( 0, fileSize );

                for ( int i = 0, j = 0; j < fileSize; j++ )
                {
                    // update progress and estimate completion time on every 128
bytes read
                    if ( (j & 127) == 0 )
                    {
                        this.updateProgress( j );
                    }

                    if ( this.pm.isCanceled() )
                    {
                        in.close();
                        this.pm.close();

                        this.result.setSize(0);
                        this.actions.doOnAbort();
                        return;
                    }

                    int v = in.read();

                    for ( int k = 0; k < 8; i++, k++ )
                    {
                        if ( ((v >> k) & 1) == 1 ) { this.result.set  ( i ); }
                        else                       { this.result.clear( i ); }
                    }
                }

                this.updateProgress( fileSize );

                in.close();
                this.pm.close();

                this.actions.doOnSuccess();
            }
            catch( IOException e )
            {
                this.actions.doOnFailure();
            }
        }



        private void updateProgress( int position )
        {
            progress.setProgress( position );

                    this.pm.setProgress( position );
                    this.pm.setNote( "Progress: " +
```

```
FileLoader.formatProgress( this.progress.getProgress() )
                                        + ". Estimated time left: "
                                        +
LinearProgressEstimation.toTime( this.progress.estimateTimeLeft() )
                                    );
        }


        private final static DecimalFormat progressFormat = new
DecimalFormat( "0.0%" );

        private static String formatProgress( double progress )
        {
            return FileLoader.progressFormat.format( progress );
        }


        private LinearProgressEstimation progress;

        private ProgressMonitor pm;
        private BitString       result;
        private String          filename;
        private FinalActions    actions;
    }



    /**
     * Saves the content of <code>BitString</code> object to file; the string is
padded with zero bits to be placed
     * in a file consisting of 8 bit bytes.
     *
     * @param filename Specifies name of the file the <code>BitString</code>
object should be saves to.
     *
     * @throws IOException An exception of class <code>IOException</code> is
thrown is case of failure.
     *
     */

    public void saveToFile( String filename )
            throws IOException
    {
        OutputStream out = new FileOutputStream( filename );

        int fileSize = ( this.size() >> 3 ) + ( ((this.size() & 7) != 0) ? 1 : 0
);

        for ( int i = 0, j = 0; j < fileSize; j++ )
        {
            int buffer = 0;

            for ( int k = 0; k < 8; k++, i++ )
            {
                if ( this.get(i) )
                {
                    buffer |= 1 << k;
                }
            }

            out.write( buffer );
```

```
        }

        out.close();
    }



    /**
     * Set the <code>index<code> bit to the state of <code>value</code>.
     *
     * @param value Value to be assigned.
     * @param index Index of the bit to get a new value.
     *
     */

    public void set( boolean value, int index )
    {
        if ( index >= this.size )
        {
            this.size = index + 1;
        }

        if ( value )
        {
            super.set( index );
        }
        else if ( index < super.length() )
        {
            super.clear( index );
        }
    }



    /**
     * Sets the <code>index</code>-th bit of <code>BitString</code> to true;
     * If <code>index</code> >= <code>this.size()</code>, <code>BitString</code>
is extended automatically.

     *
     * @param index Index of the bit to be set to true.
     *
     */

    public void set( int index )
    {
        if ( index >= this.size )
        {
            this.size = index + 1;
        }

        super.set( index );
    }



    /**
     * Create object of class <code>BitString</code> and fills it with the
content of this's [fromIndex..toIndex-1].
     *
     *
```

```
     * @param fromIndex Start index to cut out.
     * @param toIndex   Index of the first behind the copied segment.
     *
     * @return Segment of size (toIndex - fromIndex) with the content of this's
[fromIndex..toIndex-1].
     *
     */

    public BitString get( int fromIndex, int toIndex )
    {
        BitString result = new BitString( super.get( fromIndex, toIndex ) );

        result.size = toIndex - fromIndex;

        return result;
    }



    /**
     * Returns current size of <code>BitString</code> object.
     *
     * @return Current size of <code>BitString</code> object.
     *
     */

    public int size()
    {
        return this.size;
    }



    /**
     * Changes size <code>BitString</code> object.
     *
     * @param size New size of the object.
     *
     */

    public void setSize( int size )
    {
        this.size = size;
    }



    /**
     * Compares <code>this</code> with other <code>BitString</code> object.
     *
     * @param bits <code>BitString</code> object <code>this</code> should be
compared with.
     *
     * @return Returns <code>true</code> iff <code>this</code> and
<code>bits</code> are of equal size and contents;
     *         <code>false</code> otherwise.
     *
     */

    public boolean equals( BitString bits )
    {
```

```
        if ( this.size != bits.size )
        {
            return false;
        }

        for ( int i = 0; i < this.size; i++ )
        {
            if ( this.get(i) != bits.get(i) )
            {
                return false;
            }
        }

        return true;
    }


    /**
     * Attaches the contents of <code>bits</code> to <code>this</code>.
     * @param bits <code>BitString</code> to attach.
     */

    public void attach( BitString bits )
    {
        for ( int i = bits.size, j = this.size; --i >= 0; )
        {
            this.set( bits.get( i ), i + j );
        }
    }



    /**
     * Converts <code>BitString</code> to the string of "1"-s and "0"-s.
     *
     * @return <code>String</code>
     */
    public String toString()
    {
        char[] buffer = new char[this.size];

        for ( int i = 0; i < this.size; i++ )
        {
            buffer[i] = this.get(i) ? '1' : '0';
        }

        return new String( buffer );
    }



    private int size;
}




/**
 * Conformity table for indexes in range [fromIndex..toIndex-1]; a unique number
in the range is given for each member of the range.
 *
```

```
 * @author Andrey Zhmakin
 *
 * Created on Mar 26, 2010 1:00:23 AM
 *
 */

package zhmakin.steganography;

import java.util.Random;



public class ConformityTable
{
    /**
     * Creates an array of elements. Element value is equal to its position.
     *
     * @param fromIndex The lowest integer to be included.
     * @param toIndex   The first integer not included.
     */

    public ConformityTable( int fromIndex, int toIndex )
    {
        int size = toIndex - fromIndex;

        this.table = new int[size];

        for ( int i = 0; i < size; i++ )
        {
            this.table[i] = i + fromIndex;
        }

        this.fromIndex = fromIndex;
        this.toIndex   = toIndex;
    }


    /**
     * Creates random permutation of integers.
     *
     * @param fromIndex The lowest integer to be included in permutation.
     * @param toIndex   The first integer not included in permutation.
     * @param seed      Random seed.
     */

    public ConformityTable( int fromIndex, int toIndex, long seed )
    {
        if ( toIndex <= fromIndex )
        {
            throw new IndexOutOfBoundsException();
        }

        int size = toIndex - fromIndex;

        Random generator = new Random( seed );

        this.table = new int[size];

        for ( int i = size; --i >= 0; )
        {
            this.table[i] = fromIndex + i;
```

```
        }

        for ( int i = size; --i >= 1; )
        {
            int swapped         = generator.nextInt(i);
            int temp            = this.table[swapped];
            this.table[swapped] = this.table[i];
            this.table[i]       = temp;
        }

        this.fromIndex = fromIndex;
        this.toIndex   = toIndex;
    }



    /**
     * Returns an element corresponding to <code>index</code> in permutation.
     *
     * @param index Index in array (fromIndex <= <code>index</code> < toIndex).
     * @return Random value in range [fromIndex;toIndex[.
     */

    public int get( int index )
    {
        if ( ! ( index >= this.fromIndex && index < this.toIndex ) )
        {
            throw new IndexOutOfBoundsException();
        }

        return this.table[ index - this.fromIndex ];
    }



    private int fromIndex;
    private int toIndex;

    private int table[];
}




/**
 * <code>FinalActions</code> provides standard methods to be called on abortion,
successful ending of continuous process
 * or its failure.
 *
 * @author Andrey Zhmakin
 *
 * Created on May 14, 2010 11:29:49 PM
 *
 */

package zhmakin;



public abstract class FinalActions
{
```

```
    /**
     * Call this method when process ended successfully.
     */

    public abstract void doOnSuccess();



    /**
     * Call this method when process ended with errors.
     */

    public abstract void doOnFailure();



    /**
     * Call this method when process is aborted somewhere in the middle.
     */

    public abstract void doOnAbort();
}




/**
 * Class for providing access to the bits of image by their significance, least
significant first.
 *
 * @author Andrey Zhmakin
 *
 */

package zhmakin.steganography;

import util.RasterImage;

import java.awt.*;



public class BitsOfImageBySignificance
{
    /**
     * Creates an object of class <code>BitsOfImageBySignificance</code> linked
to object of class <code>util.RasterImage</code>;
     * You still have to call initConformityTables( long key ) before using
anything.
     *
     * @param image A non-null object of class <code>util.RasterImage</code>.
     *
     */

    public BitsOfImageBySignificance( RasterImage image )
    {
        assert ( image != null );

        this.image = image;
        this.key   = null;
        this.table = null;
```

```
    }


    /**
     * Creates an object of class <code>BitsOfImageBySignificance</code> linked
to object of class <code>util.RasterImage</code>;
     * Every instance is ready to use with encryption key <code>key</code>.
     *
     * @param image             Image watermark to be embedded in.
     * @param key               Encryption key.
     *
     */

    public BitsOfImageBySignificance( RasterImage image, long key )
    {
        assert ( image != null );

        this.image = image;

        this.initConformityTables( key );
    }



    /**
     * Initializes conformity tables for message encryption.
     *
     * @param key Encryption key.
     *
     */

    public void initConformityTables( long key )
    {
        if ( this.key != null && key == this.key )
        {
            return;
        }

        this.key = key;

        this.table = new ConformityTable[NUM_OF_LAYERS];

        for ( int i = 0; i < NUM_OF_LAYERS; i++ )
        {
            this.table[i] = new ConformityTable( 0, this.getLayerSize(), key + i
);
        }
    }



    /**
     * Gets size of bit container.
     *
     * @return Returns size of bit container.
     *
     */

    public int getSize()
    {
```

```
        return image.width * image.height * 24;
    }



    /**
     * Gets size of image layer.
     *
     * @return Returns size of image layer.
     *
     */

    public int getLayerSize()
    {
        return image.width * image.height * 3;
    }



    /**
     * Returns the index in layer.
     *
     * @param index Index in container.
     * @return Returns index in layer.
     * @throws IndexOutOfBoundsException Throws if <code>index</code> is out of
layer's bounds.
     */

    public int getIndexInLayer( int index )
        throws IndexOutOfBoundsException
    {
        if ( index < 0 || index >= getSize() )
        {
            throw new IndexOutOfBoundsException();
        }

        int result = index % getLayerSize();

        assert ( result >= 0 && result < getLayerSize() ) : "Index in layer is
out of bounds!";

        return result;
    }



    /**
     * Used to determinate the layer <code>index</code>-th bit corresponds to.
     *
     * @param index Index in the bit container.
     *
     * @return Returns index of the layer [0..7] corresponding to the bit at the
<code>index</code>-th position.
     *
     * @throws IndexOutOfBoundsException Exception is thrown if
     *                                   <code>index</code> is out of
[0..<code>getSize()</code> - 1] bounds.
     *
     */

    public int getLayer( int index )
```

```
        throws IndexOutOfBoundsException
    {
        if ( index < 0 || index >= getSize() )
        {
            throw new IndexOutOfBoundsException();
        }

        int layer = index / getLayerSize();

        assert (layer >= 0 && layer < NUM_OF_LAYERS) : "Internal error: Layer
index calculated incorrectly!";

        return layer;
    }



    /**
     * Used to determinate color component <code>index</code>-th bit corresponds
to.
     *
     * @param index Index in the bit container.
     *
     * @return Returns 0, 1 or 2 if index-th bit corresponds to red, green or
blue component respectively.
     *
     * @throws IndexOutOfBoundsException Exception is thrown
 if <code>index</code> is out of [0..<code>getSize()</code> - 1] bounds.
     *
     */

    public int getComponent( int index )
        throws IndexOutOfBoundsException
    {
        if ( index < 0 || index >= getSize() )
        {
            throw new IndexOutOfBoundsException();
        }

        int component = index % 3;

        assert ( component >= 0 && component <= 2 ) : "Not an RGB component!";

        return component;
    }



    /**
     * See description of the return value.
     *
     * @param index Index in the bit container.
     *
     * @return Returns index of the <code>util.RasterImage.pixels</code> array
element corresponding to the <code>index</code>-th bit on the container.
     *
     * @throws IndexOutOfBoundsException Exception is thrown if
<code>index</code> is out of [0..<code>getSize()</code> - 1] bounds.
     *
     */
```

```
    public int getIndexInImage( int index )
        throws IndexOutOfBoundsException
    {
        if ( index < 0 || index >= getSize() )
        {
            throw new IndexOutOfBoundsException();
        }

        int indexInImage = ( index % getLayerSize() ) / 3;

        assert ( indexInImage >= 0 && indexInImage <= getLayerSize() ) : "You
are out of image bounds!";

        return indexInImage;
    }



    /**
     * Used to determinate the state of the certain bit of the container.
     *
     * @param layer Refers to the bit [0..7] of the color component.
     *
     * @param indexInImage Index in the <code>util.RasterImage.pixels</code>
array.
     *
     * @param component Is 0, 1 or 2 which stands for red, green or blue
respectively.
     *
     * @return Returns state on the corresponding bit of container.
     *
     * @throws IndexOutOfBoundsException Exception is thrown if
<code>index</code> is out of [0..<code>getSize()</code> - 1] bounds.
     *
     */

    public boolean get( int layer, int indexInImage, int component )
        throws IndexOutOfBoundsException
    {
        if ( layer     < 0 || layer     >= NUM_OF_LAYERS ) throw new
IndexOutOfBoundsException();
        if ( component < 0 && component >= 3             ) throw new
IndexOutOfBoundsException();

        try
        {
            Color c = image.get( indexInImage );

            boolean state = false;

            switch ( component )
            {
                case 0: state = ((c.getRed  () >>> layer) & 1) == 1; break;
                case 1: state = ((c.getGreen() >>> layer) & 1) == 1; break;
                case 2: state = ((c.getBlue () >>> layer) & 1) == 1; break;
            }

            return state;
        }
        catch ( RasterImage.AttemptToAccessOutOfImageBoundsException e )
        {
```

```
            throw new IndexOutOfBoundsException();
        }
    }



    /**
     * Used to change the state of the certain bit of the container.
     *
     * @param layer Refers to the bit [0..7] of the color component.
     *
     * @param indexInImage Index in the <code>util.RasterImage.pixels</code>
array.
     *
     * @param component Is 0, 1 or 2 which stands for red, green or blue
respectively.
     *
     * @param state The in which the <code>index</code>-th bit currently is.
     *
     * @throws IndexOutOfBoundsException Exception is thrown if
<code>index</code> is out of [0..<code>getSize()</code> - 1] bounds.
     *
     */

    public void set( int layer, int indexInImage, int component, boolean state )
        throws IndexOutOfBoundsException
    {
        if ( layer     < 0 || layer     >= NUM_OF_LAYERS ) throw new
IndexOutOfBoundsException();
        if ( component < 0 && component >= 3             ) throw new
IndexOutOfBoundsException();

        int setter = state ? (1 << layer) : 0;
        int filter = 0x000000FF ^ (1 << layer);

        try
        {
            Color c = image.get( indexInImage );

            switch ( component )
            {
                case 0:
                    int r = (c.getRed() & filter) | setter;
                    c = new Color( r, c.getGreen(), c.getBlue() );
                    break;

                case 1:
                    int g = (c.getGreen() & filter) | setter;
                    c = new Color( c.getRed(), g, c.getBlue() );
                    break;

                case 2:
                    int b = (c.getBlue() & filter) | setter;
                    c = new Color( c.getRed(), c.getGreen(), b );
                    break;
            }

            image.set( indexInImage, c );
        }
        catch ( RasterImage.AttemptToAccessOutOfImageBoundsException e )
        {
```

```
            throw new IndexOutOfBoundsException();
        }
    }



    /**
     * Used to determinate the state of the <code>index</code>-th bit of the
container.
     *
     * @param index Index in the bit container.
     *
     * @return Returns state of the <code>index</code>-th bit in the bit
container.
     *
     * @throws IndexOutOfBoundsException Exception is thrown if
<code>index</code> is out of [0..<code>getSize()</code> - 1] bounds.
     *
     */

    public boolean get( int index )
        throws IndexOutOfBoundsException
    {
        int layer        = getLayer         ( index );

        int indexInLayer = this.table[layer].get( this.getIndexInLayer(index) );

        int indexInImage = getIndexInImage  ( indexInLayer );
        int component    = getComponent     ( indexInLayer );

        return get( layer, indexInImage, component );
    }



    /**
     * Used to change the state of the <code>index</code>-th bit of the
container.
     *
     * @param index Index of the bit in the bit container.
     *
     * @param state State to be assigned to the <code>index</code>-th bit in the
bit container.
     *
     * @throws IndexOutOfBoundsException Exception is thrown if
<code>index</code> is out of [0..<code>getSize()</code> - 1] bounds.
     *
     */

    public void set( int index, boolean state )
        throws IndexOutOfBoundsException
    {
        int layer        = getLayer         ( index );

        int indexInLayer = this.table[layer].get( this.getIndexInLayer(index) );

        int indexInImage = getIndexInImage  ( indexInLayer );
        int component    = getComponent     ( indexInLayer );

        set( layer, indexInImage, component, state );
    }
```

```
    public class ImageTooSmallToHoldDataException  extends Exception { }
    public class BrokenMessageException            extends Exception { }


    /**
     * Encodes a message in container in the most trivial way.
     *
     * @param data Message to hide.
     * @throws ImageTooSmallToHoldDataException If the image is too small for
such a large message.
     */
    public void encode_trivially( BitString data )
        throws ImageTooSmallToHoldDataException
    {
        if ( data.size() > this.getSize() )
        {
            throw new ImageTooSmallToHoldDataException();
        }

        BitString message = Util.concatenate( Util.convert(data.size()), data );

        try
        {
            for ( int i = 0; i < message.size(); i++ )
            {
                this.set( i, message.get(i) );
            }
        }
        catch ( IndexOutOfBoundsException e ) { assert false : "Index out of
bounds!"; }

    }



    /**
     * Extracts the whole contents of the container to <code>BitString</code>.
     *
     * @return Returns contents of the container.
     */
    private BitString toBitString()
    {
        BitString result = new BitString( this.getSize() );

        try
        {
            for ( int i = 0; i < this.getSize(); i++ )
            {
                if ( this.get( i ) )
                {
                    result.set( i );
                }
            }
        }
        catch ( IndexOutOfBoundsException e ) { assert false : "This could not
happen!"; }

        return result;
```

```
    }


    /**
     * Extracts message encoded with encode_trivially( BitString data ).
     *
     * @throws ImageTooSmallToHoldDataException Thrown if image is too small to
hold message of any size.
     * @throws BrokenMessageException           If there is no message inside or
message is broken.
     *
     * @return Returns contents of the message, if found.
     */

    public BitString decode_trivially()
        throws ImageTooSmallToHoldDataException,
               BrokenMessageException
    {
        if ( this.getSize() < 32 )
        {
            throw new ImageTooSmallToHoldDataException();
        }

        BitString message = toBitString();

        int dataSize = Util.toInt( message.get( 0, 31 ) );

        if ( dataSize < 0 || dataSize > this.getSize() - 32 )
        {
            throw new BrokenMessageException();
        }

        BitString result = message.get( 32, 32 + dataSize );

        assert (result.size() == dataSize) : "Wrong result size!";

        return result;
    }



    public class IndexOutOfBoundsException  extends Exception { }



    public final static int NUM_OF_LAYERS = 8;

    private Long            key;

    private RasterImage     image;
    private ConformityTable table[];
}




/**
 * Class <code>ZhaoAndKoch</code> implements a steganographic method described
by Zhao and Koch.
 *
```

```
 * @author Andrey Zhmakin
 *
 * Created on Apr 14, 2010 3:34:51 PM
 *
 */

package zhmakin.steganography;

import util.RasterImage;
import zhmakin.YCbCr;



public class ZhaoAndKoch
{
    /**
     * Creates an object used to get access to bits of container interpreted by
simplified method proposed by Zhao&Koch.
     *
     * @param image Image to be used as a container.
     *
     */

    public ZhaoAndKoch( RasterImage image )
    {
        assert ( image != null ) : "Provide a non-null image please!";

        this.image = image;
        this.robustness = 0.25;
    }



    /**
     * Returns the number of horizontal blocks in container.
     *

 * @return Number of horizontal blocks in container.
     */

    private int getHorizontalSquares()
    {
        return ( this.image.width / 8 );
    }



    /**
     * Returns the number of vertical blocks in container.
     *
     * @return Number of vertical blocks in container.
     */

    private int getVerticalSquares()
    {
        return ( this.image.height / 8 );
    }



    /**
```

```
     * Returns the number of bits the container is capable to store.
     *
     * @return Capacity of container in bits.
     */

    public int getSize()
    {
        return ( this.getVerticalSquares() * this.getHorizontalSquares() );
    }



    /**
     * Adjust the strength of the watermark.
     *
     * @param robustness Strength of the watermark in per cents ( 0 <=
robustness <= 1 ).
     */

    public void setRobustness( double robustness )
    {
        assert ( robustness >= 0.0 && robustness <= 1.0 ) : "Give robustness in
per cents!";

        this.robustness = robustness;
    }



    /**
     * Extracts the square of pixels of size 8x8 from image.
     * @param row Upper row to start extraction from.
     * @param col Left column to start extraction from.
     * @return Block of 8x8 YCbCr pixels.
     */

    private YCbCr[][] getSquare( int row, int col )
    {
        YCbCr result[][] = new YCbCr[8][8];

        int offsetY = row * 8,
            offsetX = col * 8;

        for ( int y = 0, i = 0; y < 8; y++ )
        {
            for ( int x = 0; x < 8; x++, i++ )
            {
                try
                {
                    result[y][x] = new YCbCr( this.image.getRGB( offsetY + y,
offsetX + x ) );
                }
                catch ( RasterImage.AttemptToAccessOutOfImageBoundsException e )
                {
                    assert false : "You're out of image bounds!";
                }
            }
        }

        return result;
    }
```

```
    /**
     * Replaces a block of 8x8 pixel in image with the provided block.
     *
     * @param square Block of 8x8 pixels.
     * @param row Upper row to start replacement from.
     * @param col Left column to start replacement from.
     */

    private void setSquare( YCbCr square[][], int row, int col )
    {
        int offsetY = row * 8,
            offsetX = col * 8;

        for ( int y = 0, i = 0; y < 8; y++ )
        {
            for ( int x = 0; x < 8; x++, i++ )
            {
                try
                {
                    this.image.setRGB( offsetY + y, offsetX + x, square[y]
[x].getRGB() );
                }
                catch ( RasterImage.AttemptToAccessOutOfImageBoundsException e )
                {
                    System.out.println( "You're out of image bounds!" );
                }
            }
        }
    }



    /**
     * Extracts luminance component from block.
     *
     * @param square Block of size 8x8 to extract luminance from.
     * @return Two-dimensional array of size 8x8 carrying luminance components
of the block.
     */

    private static double[][] extractLuminance( YCbCr square[][] )
    {
        double result[][] = new double[8][8];

        for ( int y = 0; y < 8; y++ )
        {
            for ( int x = 0; x < 8; x++ )
            {
                result[y][x] = square[y][x].Y;
            }
        }

        return result;
    }



    /**
```

```
     * Embeds luminance components to the block.
     *
     * @param block Block of 8x8 YCbCr color pixels.
     *
     * @param luminance Two-dimensional array of size 8x8 carrying luminance
components to embed.
     *
     */

    private static void embedLuminance( YCbCr block[][], double luminance[][] )
    {
        for ( int y = 0; y < 8; y++ )
        {
            for ( int x = 0; x < 8; x++ )
            {
                block[y][x].Y = luminance[y][x];
            }
        }
    }


    /**
     * Returns the state of the <code>index</code>-th bit of the container.
     * @param index Index in the container (0 <= <code>index</code> <
<code>this.getSize()</code>).
     * @return State of the <code>index</code>-th bit of the container.
     */

    public boolean get( int index )
    {
        assert ( index >= 0 && index < this.getSize() );

        int row = index / this.getHorizontalSquares();
        int col = index % this.getHorizontalSquares();

        YCbCr square[][] = this.getSquare( row, col );

        double block[][] = transform_DCT( extractLuminance( square ) );

        return block[V_1][U_1] > block[V_2][U_2];
    }

    //public double min = Double.POSITIVE_INFINITY, max =
Double.NEGATIVE_INFINITY;

    public void set( boolean value, int index )
    {
        assert ( index >= 0 && index < this.getSize() );

        int row = index / this.getHorizontalSquares();
        int col = index % this.getHorizontalSquares();

        YCbCr square[][] = this.getSquare( row, col );

        double block[][] = transform_DCT( extractLuminance( square ) );

        if ( value )
        {
            if ( block[V_1][U_1] < block[V_2][U_2] )
            {
                double t = block[V_1][U_1];
```

```
            block[V_1][U_1] = block[V_2][U_2];
            block[V_2][U_2] = t;
        }

        //if ( block[V_2][U_2] < min ) min = block[V_2][U_2];
        //if ( block[V_1][U_1] > max ) max = block[V_1][U_1];
    }
    else
    {
        if ( block[V_1][U_1] > block[V_2][U_2] )
        {
            double t = block[V_1][U_1];
            block[V_1][U_1] = block[V_2][U_2];
            block[V_2][U_2] = t;
        }

        //if ( block[V_1][U_1] < min ) min = block[V_1][U_1];
        //if ( block[V_2][U_2] > max ) max = block[V_2][U_2];
    }


    double x = MINIMAL_X + MAXIMAL_X * this.robustness;

    double delta = Math.abs( block[V_1][U_1] - block[V_2][U_2] );

    if ( delta <= x )
    {
        double corr = 0.5 * x;

        assert ( corr > 0.0 );

        if ( value )
        {
            block[V_1][U_1] += corr;
            block[V_2][U_2] -= corr;
        }
        else
        {
            block[V_1][U_1] -= corr;
            block[V_2][U_2] += corr;
        }
    }

    embedLuminance( square, inverse_DCT( block ) );

    this.setSquare( square, row, col );
}



/**
 * Performs discrete cosine transform on 8x8 block.
 *
 * @param block Block of 8x8 elements to be transformed.
 * @return Transformed block.
 */

private static double[][] transform_DCT( double block[][] )
{
    double result[][] = new double[N][N];
```

```
        for ( int u = 0; u < N; u++ )
        {
            for ( int v = 0; v < N; v++ )
            {
                result[v][u] = 0.0;

                for ( int x = 0; x < N; x++ )
                {
                    for ( int y = 0; y < N; y++ )
                    {
                        result[v][u] += block[y][x]
                                        * Math.cos( Math.PI * u * ( 2 * x +
1 ) / ( 2.0 * N ) )
                                        * Math.cos( Math.PI * v * ( 2 * y +
1 ) / ( 2.0 * N ) );
                    }
                }

                result[v][u] *= (2.0 / N) * C(u) * C(v);
            }
        }

        return result;
    }



    /**
     * Performs inverse discrete cosine transform on 8x8 block.
     *
     * @param s Block of 8x8 elements.
     * @return Inversely transformed block.
     */

    private static double[][] inverse_DCT( double s[][] )
    {
        double square[][] = new double[8][8];

        for ( int x = 0; x < N; x++ )
        {
            for ( int y = 0; y < N; y++ )
            {
                square[y][x] = 0.0;

                for ( int u = 0; u < N; u++ )
                {
                    for ( int v = 0; v < N; v++ )
                    {
                        square[y][x] += C(u) * C(v) * s[v][u]
                                        * Math.cos( Math.PI * u * ( 2 * x + 1 ) /
( 2.0 * N ) )
                                        * Math.cos( Math.PI * v * ( 2 * y + 1 ) /
( 2.0 * N ) );
                    }
                }

                square[y][x] *= ( 2.0 / N );
            }
        }

        return square;
```

```
    }


    /**
     * Help method for transform_DCT and inverse_DCT.
     *
     * @param u 0 <= i <= 7.
     * @return If u = 0, C(u) = 1 / sqrt(2), C(u) = 1 othewise.
     */

    private static double C( int u )
    {
        return ( u == 0 ) ? 0.70710678118654752440084436210485 /* = 2 ^ -0.5
*/ : 1.0;
    }



    /**
     * Returns the quantization values exploited be JPEG format.
     *
     * @param u  0 <= u <= 8. See description for details.
     * @param v  0 <= v <= 8. See description for details.
     *
     * @return JPEG luminance quantization value for pair (u,v).
     */

    public static int Q( int u, int v )
    {
        assert ( u >= 0 && u <= 7 );
        assert ( v >= 0 && v <= 7 );

        int t[][] =
                {
                    { 16,  11,  10,  16,  24,  40,  51,  61 },
                    { 12,  12,  14,  19,  26,  58,  60,  55 },
                    { 14,  13,  16,  24,  40,  57,  69,  56 },
                    { 14,  17,  22,  29,  51,  87,  80,  62 },
                    { 18,  22,  37,  56,  68, 109, 103,  77 },
                    { 24,  35,  55,  64,  81, 104, 113,  92 },
                    { 49,  64,  78,  87, 103, 121, 120, 101 },
                    { 72,  92,  95,  98, 112, 100, 103,  99 }
                };

        return t[u][v];
    }


    // These constants determinate minimal and maximal strength of watermark
allowed respectively.
    // MINIMAL_X is considered to be infinitely small.
    private final static double MINIMAL_X =  10.0;
    private final static double MAXIMAL_X = 500.0;


    public final static int N = 8;


  public final static int U_1 = 4;
    public final static int V_1 = 1;
```

```
    public final static int U_2 = 3;
    public final static int V_2 = 2;

    /*
    public final static int U_1 = 1;
    public final static int V_1 = 2;
    public final static int U_2 = 3;
    public final static int V_2 = 0;
    */


    private double robustness;


    private RasterImage image;
}




/**
 * <code>EmbedmentDialog</code> implements message embedment dialog.
 *
 * @author Andrey Zhmakin
 *
 * Created on May 12, 2010 11:41:28 PM
 *
 */

package zhmakin.steganography.gui;

import gui.ImageViewer;
import util.RasterImage;
import zhmakin.FinalActions;
import zhmakin.steganography.*;
import zhmakin.steganography.ZhaoAndKoch;

import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import java.text.DecimalFormat;
import java.util.Arrays;



public class EmbedmentDialog extends JDialog implements ActionListener,
AncestorListener
{
    /**
     * Provides dialog for convenient embedment of message into image.
     *
     * @param owner     Parent frame.
     * @param viewer    ImageViewer used to visualize preview.
     * @param image     Container image.
     */

    public EmbedmentDialog( Frame owner, ImageViewer viewer, RasterImage image )
    {
        super( owner, "Embed message", Dialog.ModalityType.APPLICATION_MODAL );
```

```
        assert ( owner != null ) : "Provide a non-null owner, please!";

        if ( image == null )
        {
            JOptionPane.showMessageDialog( owner,
                                           "Load image, please!", "Error",
                                           JOptionPane.ERROR_MESSAGE, null );
            this.dispose();
            return;
        }

        this.owner = owner;
        this.imageViewer = viewer;

        this.originalImage = image;
        this.previewImage  = image.makeCopy();

        this.zhaoAndKoch = new ZhaoAndKoch               ( this.previewImage );
        this.bitsOfImage = new BitsOfImageBySignificance( this.previewImage );

        this.setSize( 600, 700 );
        this.setMinimumSize( new Dimension( 600, 700 ) );

        this.addWindowListener( new WindowAdapter()
                                    {
                                        public void windowClosing( WindowEvent e
)
                                        {
                                            // TODO: Optimize this
                                            imageViewer.setImage( originalImage
);
                                        }
                                    }
                                );

        this.placeControls();

        this.setVisible( true );
    }



    /**
     * Places controls in the dialog.
     *
     */

    private void placeControls()
    {
        JTabbedPane tabDataChoice = new JTabbedPane();
        JPanel pnlMessagePanel = new JPanel();

        pnlMessagePanel.setLayout( new BorderLayout() );
        pnlMessagePanel.setBorder( BorderFactory.createTitledBorder( "Select
message:" ) );


        this.lblMessageSize = new JLabel();
        this.notifyAboutMessageSize( 0 );
        pnlMessagePanel.add( this.lblMessageSize, BorderLayout.SOUTH );
```

```
        this.txtMessageText = new JTextArea( "Copyright B© by Andrey Zhmakin",
8, 0 );
        this.txtMessageText.getDocument().addDocumentListener( new
MessageTextUpdateListener() );

        MessageSourceListener messageSourceListener = new
MessageSourceListener();

        this.panTextInput = new JScrollPane( this.txtMessageText );
        this.panTextInput.addComponentListener( messageSourceListener );
        tabDataChoice.addTab( "Unicode Text", null, this.panTextInput, "Conceal
text" );
        tabDataChoice.setMnemonicAt( 0, KeyEvent.VK_1 );

        this.pnlFileChoicePanel = new JPanel();
        this.pnlFileChoicePanel.addComponentListener( messageSourceListener );


        this.pnlFileChoicePanel.setLayout( new
BoxLayout( this.pnlFileChoicePanel, BoxLayout.X_AXIS ) );

        this.txtMessageFilePath      = new JTextField( "", 1 );
        this.cmdBrowseForMessageFile = new JButton( "Browse" );
        this.cmdLoadMessageFile      = new JButton( "Load" );

        this.cmdBrowseForMessageFile.addActionListener( this );
        this.cmdLoadMessageFile     .addActionListener( this );


        this.txtMessageFilePath.setMaximumSize( new
Dimension( Integer.MAX_VALUE, 25 ) );

        this.pnlFileChoicePanel.add( Box.createRigidArea( new Dimension( 5,
5 ) ) );
        this.pnlFileChoicePanel.add( this.txtMessageFilePath );
        this.pnlFileChoicePanel.add( Box.createRigidArea( new Dimension( 5,
5 ) ) );
        this.pnlFileChoicePanel.add( this.cmdBrowseForMessageFile );
        this.pnlFileChoicePanel.add( Box.createRigidArea( new Dimension( 5,
5 ) ) );
        this.pnlFileChoicePanel.add( this.cmdLoadMessageFile );
        this.pnlFileChoicePanel.add( Box.createRigidArea( new Dimension( 5,
5 ) ) );

        tabDataChoice.addTab( "File", null, pnlFileChoicePanel, "Conceal entire
file" );
        tabDataChoice.setMnemonicAt( 1, KeyEvent.VK_2 );
        pnlMessagePanel.add( tabDataChoice );


        JPanel pnlSteganography = new JPanel();
        pnlSteganography.setLayout( new BoxLayout( pnlSteganography,
BoxLayout.PAGE_AXIS ) );


        JPanel pnlErrorCorrection = new JPanel();
        pnlErrorCorrection.setLayout( new BoxLayout( pnlErrorCorrection,
BoxLayout.PAGE_AXIS ) );
        pnlErrorCorrection.setBorder( BorderFactory.createTitledBorder( "Provide
strength of error correction:" ) );
```

```
        JLabel lblErrorCorrection = new JLabel();
        lblErrorCorrection.setAlignmentX( (float)0.5 );
        pnlErrorCorrection.add( lblErrorCorrection );

        this.sldErrorCorrection = new JSlider( JSlider.HORIZONTAL, 0, 5, 1 );

        this.sldErrorCorrection.setMajorTickSpacing( 1 );
        this.sldErrorCorrection.setMinorTickSpacing( 0 );
        this.sldErrorCorrection.setPaintTrack( true );
        this.sldErrorCorrection.setPaintLabels( true );
        pnlErrorCorrection.add( this.sldErrorCorrection );
        this.sldErrorCorrection.addChangeListener( new
SliderToLabelWriterListener( this.sldErrorCorrection,

lblErrorCorrection,

new DecimalFormat( "Level #" ),

new Actions()

{

public void perform()

{

updateCapacityStatistics();

}

}

)
                                                    );

        pnlSteganography.add( pnlErrorCorrection );

        JPanel pnlCapacitySummary = new JPanel();
        pnlCapacitySummary.setLayout( new GridLayout( 0, 1 ) );
        pnlCapacitySummary.setBorder( BorderFactory.createTitledBorder( "Capacit
y statistics:" ) );

        this.lblStatistics_Service      = new JLabel();
        this.lblStatistics_Message      = new JLabel();
        this.lblStatistics_ECC          = new JLabel();
        this.lblStatistics_Available    = new JLabel();

        pnlCapacitySummary.add( this.lblStatistics_Service );
        pnlCapacitySummary.add( this.lblStatistics_Message );
        pnlCapacitySummary.add( this.lblStatistics_ECC );
        pnlCapacitySummary.add( this.lblStatistics_Available );

        pnlSteganography.add( pnlCapacitySummary );


        JPanel pnlPassword = new JPanel();

        pnlPassword.setBorder( BorderFactory.createTitledBorder( "Enter
password:" ) );

        pnlPassword.setLayout( new FlowLayout() );
```

```
        pnlPassword.add( new JLabel( "Password: " ) );
        this.txtPassword = new JPasswordField( RECOMMENDED_PASSWORD_LENGTH );
        pnlPassword.add( this.txtPassword );
        pnlPassword.add( new JLabel( "Confirm password: " ) );
        this.txtConfirmPassword = new
JPasswordField( RECOMMENDED_PASSWORD_LENGTH );
        pnlPassword.add( this.txtConfirmPassword );

        pnlSteganography.add( pnlPassword );


        JPanel pnlAlgorithmPanel = new JPanel();
        pnlAlgorithmPanel.setLayout( new BorderLayout() );
        pnlAlgorithmPanel.setBorder( BorderFactory.createTitledBorder( "Choose
algorithm:" ) );

        this.tabAlgorithmChoiceTabs = new JTabbedPane();

        JComponent LsbInputPanel = new JPanel();

        this.lblLsbInfluence = new JLabel();
        LsbInputPanel.add( this.lblLsbInfluence );

        LsbInputPanel.add( new JLabel( "This method provides you with maximum "
                                        + this.bitsOfImage.getSize()
                                        + " bit(s) to conceal your message."
) );


        tabAlgorithmChoiceTabs.addTab( "LSB", null, LsbInputPanel, "Conceal in
Least Significant Bits of image" );
        tabAlgorithmChoiceTabs.setMnemonicAt( 0, KeyEvent.VK_1 );


        JPanel pnlZnKInput = new JPanel();
        //pnlZnKInput.setLayout( new GridLayout( 0, 1 ) );
        pnlZnKInput.setLayout( new BoxLayout( pnlZnKInput, BoxLayout.PAGE_AXIS )
);

        JPanel pnlZnKRobustness = new JPanel();
        pnlZnKRobustness.setLayout( new BoxLayout( pnlZnKRobustness,
BoxLayout.PAGE_AXIS ) );
        pnlZnKRobustness.setBorder( BorderFactory.createTitledBorder(
 "Set level of robustness:" ) );

        JLabel lblZnKRobustness = new JLabel();
        lblZnKRobustness.setAlignmentX( (float)0.5 );
        pnlZnKRobustness.add( lblZnKRobustness );

        sldZnKRobustness = new JSlider( JSlider.HORIZONTAL, 0, 100, 25 );
        sldZnKRobustness.setMajorTickSpacing( 10 );
        sldZnKRobustness.setMinorTickSpacing(  5 );
        sldZnKRobustness.setPaintTrack( true );
        sldZnKRobustness.setPaintTicks( true );
        sldZnKRobustness.setPaintLabels( true );
        pnlZnKRobustness.add( sldZnKRobustness );

        sldZnKRobustness.addChangeListener( new
SliderToLabelWriterListener( sldZnKRobustness,
                                                                        lbl
```

```
ZnKRobustness,
                                                                    new
DecimalFormat( "#'%'" )
                                                                    )
                                            );

        pnlZnKInput.add( pnlZnKRobustness );

        pnlZnKInput.add( new JLabel( "This method provides you with maximum "
                                     + this.zhaoAndKoch.getSize()
                                     + " bit(s) to conceal your
message." ) );

        tabAlgorithmChoiceTabs.addTab( "Zhao & Koch", null, pnlZnKInput,
                                    "Conceal according to method proposed by
Zhao & Koch" );
        tabAlgorithmChoiceTabs.setMnemonicAt( 1, KeyEvent.VK_2 );

        tabAlgorithmChoiceTabs.addChangeListener( new ChangeListener()
                                                  {
                                                      public void
stateChanged( ChangeEvent e )
                                                      {
                                                          updateCapacityStatis
tics();
                                                      }
                                                  }
                                                );

        pnlAlgorithmPanel.add( tabAlgorithmChoiceTabs );


        pnlSteganography.add( pnlAlgorithmPanel );


        JPanel pnlCommandPanel = new JPanel();
        pnlCommandPanel.setLayout( new FlowLayout()   );

        this.cmdPreview  = new JButton( "Preview" );
        this.cmdApply    = new JButton( "Apply"   );
        this.cmdCancel   = new JButton( "Cancel"  );

        pnlCommandPanel.add( this.cmdPreview );
        pnlCommandPanel.add( this.cmdApply   );
        pnlCommandPanel.add( this.cmdCancel  );

        this.cmdPreview .addActionListener( this );
        this.cmdApply   .addActionListener( this );
        this.cmdCancel  .addActionListener( this );

        this.getContentPane().setLayout( new BorderLayout() );

        JSplitPane splInputPane = new JSplitPane( JSplitPane.VERTICAL_SPLIT,
true, pnlMessagePanel, pnlSteganography );

        splInputPane.setDividerLocation( 0.5 );

        this.getContentPane().add( splInputPane );
        this.getContentPane().add( pnlCommandPanel, BorderLayout.SOUTH );
    }
```

```
    public void actionPerformed( ActionEvent e )
    {
        // Three buttons in the bottom of my dialog
        if ( e.getSource() == this.cmdCancel )
        {
            // TODO: Set condition of restoration
            this.imageViewer.setImage( this.originalImage );

            this.dispose();
        }
        else if ( e.getSource() == this.cmdApply )
        {
            // TODO: "preview"

            this.originalImage.set( this.previewImage );
            this.imageViewer.setImage( this.originalImage );
        }
        else if ( e.getSource() == this.cmdPreview )
        {
            //this.imageViewer.setImage( this.originalImage );
            this.previewImage.set( this.originalImage );

            assert ( this.previewImage != null );

            if ( this.message == null )
            {
                JOptionPane.showMessageDialog( this,
                                               "Load a file, please!",
                                               "Error",
                                               JOptionPane.ERROR_MESSAGE,
                                               null );
                return;
            }

            // read and check password

            if ( !Arrays.equals( this.txtPassword.getPassword(),
this.txtConfirmPassword.getPassword() ) )
            {
                JOptionPane.showMessageDialog( this,
                                               "Password doesn't match its
confirmation!",
                                               "Error",
                                               JOptionPane.ERROR_MESSAGE,
                                               null );
                this.txtPassword.requestFocus();
                return;
            }

            long key =
Util.getKeyFromPassword( this.txtPassword.getPassword() );

            // TODO: Determinate algorithm, initialize it and choose its metrics

            String algorithmTabTitle =
this.tabAlgorithmChoiceTabs.getTitleAt( this.tabAlgorithmChoiceTabs.getSelectedI
ndex() );

            int errorCorrectionLevel = this.sldErrorCorrection.getValue() * 2 *
```

```
10;

            BitString msg = Message.produceLowLevelMessage( this.message,
errorCorrectionLevel );

            if ( algorithmTabTitle.equals( "LSB" ) )
            {
                // TODO: Move to separate thread
                {
                    assert ( msg.size() <= this.bitsOfImage.getSize() ) :
"Message too long!";

                    // TODO: Optimize for number of tables to initialize
                    this.bitsOfImage.initConformityTables( key );

                    try
                    {
                        for ( int i = 0; i < msg.size(); i++ )
                        {
                            this.bitsOfImage.set( i, msg.get(i) );
                        }
                    }
                    catch ( BitsOfImageBySignificance.IndexOutOfBoundsException
x )
                    {
                        // Do nothing, as this couldn't happen!
                    }

                    // show preview
                    this.imageViewer.setImage( this.previewImage );
                }
            }
            else if ( algorithmTabTitle.equals( "Zhao & Koch" ) )
            {
                // TODO: Move to separate thread
                {
                    assert ( msg.size() <= this.zhaoAndKoch.getSize() ) :
"Message too long!";

                    ConformityTable table = new ConformityTable( 0,
this.zhaoAndKoch.getSize(), key );

                    this.zhaoAndKoch.setRobustness( this.sldZnKRobustness.getVal
ue() * 0.01 );

                    for ( int i = 0; i < msg.size(); i++ )
                    {
                        this.zhaoAndKoch.set( msg.get(i), table.get(i) );
                    }

                    // show preview
                    this.imageViewer.setImage( this.previewImage );
                }
            }
            else
            {
                assert false : "Choice of algorithm failed!";
            }
        }
        // Buttons at File tab of message input section
        else if ( e.getSource() == this.cmdBrowseForMessageFile )
```

```
        {
            final JFileChooser fc = new JFileChooser();
            fc.setFileSelectionMode( JFileChooser.FILES_ONLY );

            if ( fc.showOpenDialog( this.owner ) ==
JFileChooser.APPROVE_OPTION )
            {
                this.txtMessageFilePath.setText( fc.getSelectedFile().getAbsolut
ePath() );
            }
        }
        else if ( e.getSource() == this.cmdLoadMessageFile )
        {
            this.setEnabled( false );

            String filename = this.txtMessageFilePath.getText();
            this.message = new BitString( 0 );
            BitString.loadFile( filename, this.message, this, new
RecoilOfFileLoading(this) );
        }
    }


    /**
     * Actions to do when the file is loaded.
     */

    private class RecoilOfFileLoading extends FinalActions
    {
        RecoilOfFileLoading( EmbedmentDialog belovedDialog )
        {
            this.belovedDialog = belovedDialog;
        }



        public void doOnSuccess()
        {
            this.belovedDialog.notifyAboutMessageSize( this.belovedDialog.messag
e.size() );
            this.belovedDialog.updateCapacityStatistics();

            this.belovedDialog.setEnabled( true );
            this.belovedDialog.bringOnTop();
        }



        public void doOnFailure()
        {
            JOptionPane.showMessageDialog( this.belovedDialog,
                                           "Filename you provided is
wrong!\nMessage not loaded!",
                                           "Error",
                                           JOptionPane.ERROR_MESSAGE,
                                           null );

            this.belovedDialog.message = null;
            this.belovedDialog.notifyAboutMessageSize( null );

            this.belovedDialog.setEnabled( true );
```

```
            this.belovedDialog.bringOnTop();
        }


        public void doOnAbort()
        {
            this.belovedDialog.message = null;
            this.belovedDialog.lblMessageSize.setText( "Loading aborted! No
message loaded!" );

            this.belovedDialog.setEnabled( true );
            this.belovedDialog.bringOnTop();
        }

        private EmbedmentDialog belovedDialog;
    }



    private void updateCapacityStatistics()
    {
        if ( !this.isVisible() )
        { // We need all elements to be shown
            return;
        }

        if ( this.message != null )
        {
            int capacity;

            String algorithmTabTitle
                    =
this.tabAlgorithmChoiceTabs.getTitleAt( this.tabAlgorithmChoiceTabs.getSelectedI
ndex()
 );

            if      ( algorithmTabTitle.equals( "LSB" ) )           capacity =
this.bitsOfImage.getSize();
            else if ( algorithmTabTitle.equals( "Zhao & Koch" ) )   capacity =
this.zhaoAndKoch.getSize();
            else                                                    capacity =
-1;

            int eccOverhead = Message.predictEccOverhead( this.message.size(),
this.sldErrorCorrection.getValue() * 2 * 10 );


            this.lblStatistics_Service  .setText( "Size of service data is "
                                                  + Message.SERVICE_DATA_SIZE +
" bit(s)" );

            this.lblStatistics_Message  .setText( "Size of your message is " +
this.message.size() + " bit(s)" );

            int total = Message.SERVICE_DATA_SIZE + this.message.size() +
eccOverhead;

            this.lblStatistics_ECC      .setText( "Error correction adds " +
eccOverhead + " bit(s) to this, bringing "
                                                    + "total data size to "
                                                    + total + " bit(s)" );
```

```
            int available = capacity - total;

            if ( available >= 0 )
            {
                this.lblStatistics_Available.setText( available + " bit(s)
available." );
                this.lblStatistics_Available.setForeground( (new
JLabel()).getForeground() );
                this.cmdApply  .setEnabled( true );
                this.cmdPreview.setEnabled( true );
            }
            else
            {
                this.lblStatistics_Available.setText( "Message exceeds maximum
size for " + (-available) + " bit(s)." );
                this.lblStatistics_Available.setForeground( Color.RED );
                this.cmdApply  .setEnabled( false );
                this.cmdPreview.setEnabled( false );
            }

            int layersOccupied = total / this.bitsOfImage.getLayerSize();

            String influence;

            if      ( layersOccupied <= 3 ) influence = "Negligible";
            else if ( layersOccupied <= 5 ) influence = "Severe";
            else if ( layersOccupied <= 7 ) influence = "Devastating";
            else                            influence = "N/A";

            this.lblLsbInfluence.setText( "Under this amount of data, influence
on the image will be: " + influence );
        }
        else
        {
            this.lblStatistics_Service  .setText( "   " );
            this.lblStatistics_Message  .setText( "   " );
            this.lblStatistics_ECC      .setText( "   " );
            this.lblStatistics_Available.setText( "   " );

            this.lblLsbInfluence.setText( "Under this amount of data, influence
on the image will be: N/A" );

            this.cmdApply  .setEnabled( false );
            this.cmdPreview.setEnabled( false );
        }
    }



    private void notifyAboutMessageSize( Integer size )
    {
        if ( size != null )
        {
            this.lblMessageSize.setText( "Message size is " + size + "
bit(s)" );
            this.lblMessageSize.setForeground( (new JLabel()).getForeground() );
        }
        else
        {
            this.lblMessageSize.setText( "No message available! Load file,
```

```
please!" );
            this.lblMessageSize.setForeground( Color.RED );
        }
    }



    public void bringOnTop()
    {
        this.setAlwaysOnTop( true );
        this.setAlwaysOnTop( false );
    }



    public void ancestorAdded  ( AncestorEvent event ) {}
    public void ancestorMoved  ( AncestorEvent event ) {}
    public void ancestorRemoved( AncestorEvent event ) {}



    RasterImage originalImage,
                previewImage;


    ZhaoAndKoch               zhaoAndKoch;
    BitsOfImageBySignificance bitsOfImage;


    private Frame owner;

    private ImageViewer imageViewer;

    private BitString message;

    private JTabbedPane     tabAlgorithmChoiceTabs;

    private JPanel          pnlFileChoicePanel;
    private JScrollPane     panTextInput;

    private JLabel          lblMessageSize;

    private JTextArea       txtMessageText;


    private JPasswordField  txtPassword,
                            txtConfirmPassword;

    private JSlider         sldErrorCorrection;

    private JLabel          lblStatistics_Service;
    private JLabel          lblStatistics_Message;
    private JLabel          lblStatistics_ECC;
    private JLabel          lblStatistics_Available;

    private JLabel          lblLsbInfluence;

    private JSlider         sldZnKRobustness;

    private JButton         cmdPreview,
                            cmdApply,
```

```
                        cmdCancel;

private JTextField      txtMessageFilePath;

private JButton         cmdBrowseForMessageFile,
                        cmdLoadMessageFile;


public final static int RECOMMENDED_PASSWORD_LENGTH = 16;



class MessageTextUpdateListener implements DocumentListener
{
    public MessageTextUpdateListener()
    {
        this.doMyBusiness();
    }

    public void insertUpdate ( DocumentEvent e ) { this.doMyBusiness(); }
    public void removeUpdate ( DocumentEvent e ) { this.doMyBusiness(); }
    public void changedUpdate( DocumentEvent e ) { this.doMyBusiness(); }

    private void doMyBusiness()
    {
        message = BitString.encodeString( txtMessageText.getText() );
        notifyAboutMessageSize( message.size() );

        updateCapacityStatistics();
    }
}



class MessageSourceListener implements ComponentListener
{
    public void componentShown(ComponentEvent e)
    {
        if ( e.getSource() == panTextInput )
        {
            message = BitString.encodeString( txtMessageText.getText() );
            notifyAboutMessageSize( message.size() );
        }
        else if ( e.getSource() == pnlFileChoicePanel )
        {
            notifyAboutMessageSize( null );
            message = null;
        }

        updateCapacityStatistics();
    }

    public void componentHidden ( ComponentEvent e ) { }
    public void componentMoved  ( ComponentEvent e ) { }
    public void componentResized( ComponentEvent e ) { }
}



abstract class Actions
{
```

```
        abstract public void perform();
    }



    class SliderToLabelWriterListener implements ChangeListener
    {
        public SliderToLabelWriterListener( JSlider slider, JLabel result,
DecimalFormat format )
        {
            this.slider  = slider;
            this.result  = result;
            this.format  = format;
            this.actions = null;

            this.updateResult();
        }


        public SliderToLabelWriterListener( JSlider slider, JLabel result,
DecimalFormat format, Actions actions )
        {
            this.slider  = slider;
            this.result  = result;
            this.format  = format;
            this.actions = actions;

            this.updateResult();
        }

        public void stateChanged( ChangeEvent e )
        {
            this.updateResult();
        }

        private void updateResult()
        {
            this.result.setText( format.format( this.slider.getValue() ) );

            if ( this.actions != null )
            {
                this.actions.perform();
            }
        }

        private JSlider         slider;
        private JLabel          result;
        private DecimalFormat   format;
        private Actions         actions;
    }
}




/**
 * <code>ExtractionDialog</code> implements message extraction dialog.
 *
 * @author Andrey Zhmakin
 *
 * Created on May 19, 2010 12:09:07 PM
```

```
 *
 */

package zhmakin.steganography.gui;

import util.RasterImage;
import zhmakin.steganography.*;
import zhmakin.steganography.ECC.ReedSolomon;
import zhmakin.steganography.ZhaoAndKoch;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.IOException;


public class ExtractionDialog extends JDialog implements ActionListener
{
    public ExtractionDialog( Frame owner, RasterImage image )
    {
        super( owner, "Extract message",
Dialog.ModalityType.APPLICATION_MODAL );

        if ( image == null )
        {
            JOptionPane.showMessageDialog( owner,
                                           "Load image, please!", "Error",
                                           JOptionPane.ERROR_MESSAGE, null );
            this.dispose();
            return;
        }

        this.message = null;

        this.image = image;

        this.setSize( 600, 500 );
        this.setMinimumSize( new Dimension( 600, 500 ) );

        this.addWindowListener(
                (owner == null) ? new WindowAdapter()
                                  {
                                      public void windowClosing( WindowEvent e
)
                                      {
                                          System.exit( 0 );
                                      }
                                  }
                                : new WindowAdapter()
                                  {
                                      public void windowClosing( WindowEvent e
) { }
                                  }
        );

        this.placeControls();

        this.setVisible( true );
```

```
    }


    private void placeControls()
    {
        //this.getContentPane().setLayout( new BoxLayout( this.getContentPane(),
BoxLayout.Y_AXIS ) );
        this.getContentPane().setLayout( new BorderLayout() );

        JPanel pnlOptions = new JPanel();
        pnlOptions.setLayout( new BoxLayout( pnlOptions, BoxLayout.Y_AXIS ) );

        JPanel pnlAlgorithm = new JPanel();
        pnlAlgorithm.setBorder( BorderFactory.createTitledBorder(
 "Choose algorithm:" ) );
        pnlOptions.add( pnlAlgorithm );


        ButtonGroup algorithms = new ButtonGroup();

        this.optLSB = new JRadioButton( "Least Significant Bits" );
        this.optLSB.setSelected( true );
        algorithms.add( this.optLSB );

        this.optZhaoAndKoch = new JRadioButton( "Zhao & Koch" );
        algorithms.add( this.optZhaoAndKoch );

        pnlAlgorithm.add( this.optLSB );
        pnlAlgorithm.add( this.optZhaoAndKoch );


        JPanel pnlDecryption = new JPanel();
        pnlDecryption.setBorder( BorderFactory.createTitledBorder( "Decryption:"
) );

        pnlDecryption.add( new JLabel( "Password: " ) );

        this.txtPassword = new JPasswordField( "",
zhmakin.steganography.gui.EmbedmentDialog.RECOMMENDED_PASSWORD_LENGTH );
        pnlDecryption.add( this.txtPassword );

        pnlOptions.add( pnlDecryption );


        JPanel pnlPreview = new JPanel();
        pnlPreview.setLayout( new BorderLayout() );
        pnlPreview.setBorder( BorderFactory.createTitledBorder( "Message
preview:" ) );

        this.txtPreview = new JTextArea( "", 8, 0 );
        this.txtPreview.setEditable( false );
        this.txtPreview.setLineWrap( true );

        pnlPreview.add( new JScrollPane( this.txtPreview ) );

        JSplitPane splMetrics = new JSplitPane( JSplitPane.VERTICAL_SPLIT, true,
pnlOptions, pnlPreview );
        splMetrics.setOneTouchExpandable( true );
        splMetrics.setDividerLocation( Integer.MIN_VALUE/*Integer.MAX_VALUE*/ );
        this.getContentPane().add( splMetrics );
```

```
        JPanel pnlCommand = new JPanel();

        this.cmdPreview = new JButton( "Preview" );
        this.cmdPreview.addActionListener( this );
        pnlCommand.add( this.cmdPreview );

        this.cmdSave = new JButton( "Save" );
        this.cmdSave.addActionListener( this );
        pnlCommand.add( this.cmdSave );

        this.cmdCancel = new JButton( "Cancel" );
        this.cmdCancel.addActionListener( this );
        pnlCommand.add( this.cmdCancel );

        this.getContentPane().add( pnlCommand, BorderLayout.SOUTH );
    }



    public void actionPerformed( ActionEvent e )
    {
        if ( e.getSource() == this.cmdPreview )
        {
            int method;

            if      ( this.optLSB.isSelected() )            method = METHOD_LSB;
            else if ( this.optZhaoAndKoch.isSelected() )    method = METHOD_ZNK;
            else
            {
                method = 0;
                assert false : "GUI error!";
            }

            long key =
Util.getKeyFromPassword( this.txtPassword.getPassword() );

            BitsOfImageBySignificance bitsOfImage = new
BitsOfImageBySignificance( this.image, key );
            ZhaoAndKoch               zhaoAndKoch = new
ZhaoAndKoch( this.image );
            ConformityTable table;

            BitString strengthClause = new
BitString( Message.SIZE_OF_ECC_STRENGTH );
            BitString sizeClause     = new
BitString( Message.SIZE_OF_SIZE_CLAUSE  );

            int containerMaxSize;

            if ( method == METHOD_LSB )
            {
                table = null; // just to prevent error
                containerMaxSize = bitsOfImage.getSize();

                try
                {
                    for ( int i = 0; i < Message.SIZE_OF_ECC_STRENGTH; i++ )
                    {
                        strengthClause.set( bitsOfImage.get(i), i );
```

```
                    }

                    for ( int i = 0; i < Message.SIZE_OF_SIZE_CLAUSE; i++ )
                    {
                        sizeClause.set( bitsOfImage.get(Message.SIZE_OF_ECC_STRE
NGTH + i), i );
                    }

                }
                catch ( BitsOfImageBySignificance.IndexOutOfBoundsException x )
                {
                    assert false : "This should not happen!";
                }
            }
            else// if ( method == METHOD_ZNK )
            {
                containerMaxSize = zhaoAndKoch.getSize();

                table = new ConformityTable( 0, zhaoAndKoch.getSize(), key );

                for ( int i = 0; i < Message.SIZE_OF_ECC_STRENGTH; i++ )
                {
                    strengthClause.set( zhaoAndKoch.get(table.get(i)), i );
                }

                for ( int i = 0; i < Message.SIZE_OF_SIZE_CLAUSE; i++ )
                {
                    sizeClause.set( zhaoAndKoch.get(table.get(Message.SIZE_OF_EC
C_STRENGTH + i)), i );
                }
            }

            int strengthOfEcc  = Message.getEccStrength( strengthClause );

            if ( strengthOfEcc == -1 )
            {
                JOptionPane.showMessageDialog( this,
                                               "Container is either empty or
message is significantly damaged!",
                                               "Error",
                                               JOptionPane.ERROR_MESSAGE,
null );
                this.message = null;
                this.txtPreview.setText( "" );

                return;
            }

            int rawMessageSize = Message.decodeMessageSize( sizeClause );

            int codedSize = rawMessageSize +
Message.predictEccOverhead( rawMessageSize, strengthOfEcc * 10 );

            if ( codedSize + Message.SERVICE_DATA_SIZE > containerMaxSize )
            {
                codedSize = containerMaxSize - Message.SERVICE_DATA_SIZE;
            }



            BitString body = new BitString( codedSize );
```

```
            if ( method == METHOD_LSB )
            {
                int i = Message.SERVICE_DATA_SIZE;

                for ( int j = 0; j < codedSize; i++, j++ )
                {
                    try
                    {
                        body.set( bitsOfImage.get(i), j );
                    }
                    catch ( BitsOfImageBySignificance.IndexOutOfBoundsException
x )
                    {
                        assert false : "This couldn't happen!";
                    }
                }
            }
            else// if ( method == METHOD_ZNK )
            {
                int i = Message.SERVICE_DATA_SIZE;

                for ( int j = 0; j < codedSize; i++, j++ )
                {
                    body.set( zhaoAndKoch.get(table.get(i)), j );
                }
            }


            this.message = new BitString();
            int diagnosis = Message.decodeBody( body, strengthOfEcc,
rawMessageSize, this.message );

            switch ( diagnosis )
            {
                case ReedSolomon.DIAGNOSIS_HEALTHY:
                    break;

                case ReedSolomon.DIAGNOSIS_REPAIRED:
                    JOptionPane.showMessageDialog( this,
                                                   "Container was altered!\n"
                                                        + "Though it was
possible to restore all the data!",
                                                   "Error",
                                                    JOptionPane.ERROR_MESSAGE,
null );
                    break;

                case ReedSolomon.DIAGNOSIS_UNREPAIRABLE:
                    JOptionPane.showMessageDialog( this,
                                                   "Container was altered!\n"
                                                        + "It was impossible to
restore all the data!",
                                                   "Error",
                                                    JOptionPane.ERROR_MESSAGE,
null );
                    break;
            }

            this.txtPreview.setText( Util.toString( this.message ) );
        }
```

```
        else if ( e.getSource() == this.cmdSave )
        {
            if ( this.message == null )
            {
                 JOptionPane.showMessageDialog( this,
                                                "Preview message, please!",
"Error",
                                                JOptionPane.ERROR_MESSAGE,
null );
                return;
            }

            final JFileChooser fc = new JFileChooser();
            fc.setFileSelectionMode( JFileChooser.FILES_ONLY );

            if ( fc.showSaveDialog( this ) == JFileChooser.APPROVE_OPTION )
            {
                try
                {
                    Util.save( fc.getSelectedFile().getAbsolutePath(),
this.message );
                }
                catch ( IOException x )
                {
                    JOptionPane.showMessageDialog( this,
                                                "Can\'t save the file!",
"Error",
                                                JOptionPane.ERROR_MESSAGE,
null );
                }
            }
        }
        else if ( e.getSource() == this.cmdCancel )
        {
            this.dispose();
        }
    }



    private BitString message;

    private RasterImage image;

    private JPasswordField  txtPassword;

    private JRadioButton    optLSB,
                            optZhaoAndKoch;


    private JTextArea       txtPreview;

    private JButton         cmdPreview,
                            cmdSave,
                            cmdCancel;

    private final static int METHOD_LSB = 1;
    private final static int METHOD_ZNK = 2;
}
```

DOKUMENTĀRĀ LAPA

Kvalifikācijas darbs „Informācijas paslēpšana digitālajos attēlos” izstrādāts LU Datorikas fakultātē.

Ar savu parakstu apliecinu, ka kvalifikācijas darbs veikts patstāvīgi un iesniegtā darba elektroniskā kopija atbilst izdrukai. Piekrītu sava darba publicēšanai internetā.

Autors: _____
(Andrejs Žmakins)

Ar savu parakstu apliecinu, ka esmu lasījis augšminēto kvalifikācijas darbu un atzīstu to par piemērotu/nepiemērotu (nevajadzīgo svītrot) aizstāvēšanai Latvijas Universitātes studiju programmas «Programmēšana un datortīklu administrēšana» kvalifikācijas pārbaudījumu komisijas sēdē.

Darba vadītājs: _____
(Vadītāja paraksts)

Darbs iesniegts Datorikas fakultātē _____.
(Iesniegšanas datums)

Ar šo es apliecinu, ka darba elektroniskā versija ir augšupielādēta LU informatīvajā sistēmā.

Metodiķe: _____.
(Metodiķes paraksts)

Recenzents_____
Darbs aizstāvēts kvalifikācijas pārbaudījumu komisijas sēdē
_____ prot. Nr. _____, vērtējums
_____
(Darba aizstāvēšanas datums)

Komisijas sekretārs: _____
(Sekretāra paraksts)