DAFTAR PUSTAKA

Al Fikri, I. (2016). Aplikasi Navigasi Berbasis Perangkat Bergerak dengan MenggunakanPlatform Wikitude untuk Studi Kasus Lingkungan ITS. *Jurnal Teknik ITS*, *5*(1), 48–51. https://doi.org/10.12962/j23373539.v5i1.14511

Andriyani, S. (2016). Aplikasi Akademik Online Berbasis Mobile Android. *Jurnal SainsDan Teknologi Utama, Volume XI, Nomor 1, April 2016*, *XI*(152), 15–26.

Developer Training Team, G. (2016). Android Developer Fundamentals Course- Concept Reference. *CIRED - Open Access Proceedings Journal*, *2017*(July), 1–67.

http://www.eskom.co.za/CustomerCare/TariffsAndCharges/Documents/RSA DistributionTariff Code Vers 6.pdf%0Ahttp://www.nersa.org.za/

Eko Valentino, D., & Jodi Hardiansyah, M. (2020). PERANCANGAN VIDEOCOMPANY PROFILE PADA HOTEL de JAVA BANDUNG. *Tematik*,

7(1), 1–20. https://doi.org/10.38204/tematik.v7i1.285

Fathiha, V. A. (2020). *Blind Watermarking Pada Citra Digital Menggunakan DiscreteWavelet Transform (Dwt) Dan Singular Value Decomposition (Svd)*. *14*(0822062), 125–134.

Grueber, C. E., Nakagawa, S., Laws, R. J., Jamieson, I. G., Yamada, A., Flow, M.

F., Spinners, C., Bore, F., Meters, F., Meters, D. F., Format, L., Speed, C., Conventions, S., Principles, F. F., Tools, R. T., Logging, O. A., Activation, O., Example, L., Water, S., … Philpott, S. M. (2019). *Android Developer Fundamentals Course - PracticalWorkbook*.

Ikromina, F. I., & Ujianto, E. I. H. (2019). Invisible Watermarking Citra Digital Menggunakan Kombinasi Metode Discrete Cosine Transform Dan Discrete Wavelet Transform. *JANAPATI : Jurnal Nasional Pendidikan Teknik Informatika*, *8*, 261–271.

Imam Adli, HarunMukhtar, J. A. A. (2018). Perancangan dan pembuatan visual novelsejarah kh. ahmad dahlan sebagai media pembelajaran berbasisandroid. *RABIT (Jurnal Teknologi Dan Sistem Informasi Univrab)*, *3*(2), 69– 82.

Insani, O. J., & Abdullah, I. N. (2020). *Jurnal Teknologi Terpadu Journal of Integrated Technology OPTIMASI HYBRID INVISIBLE WATERMARKING RDWT-DCT-SVD MENGGUNAKAN ALGORITMA PSO PADA CITRA DIGITAL*. *6*(1), 1–10.

Lase, N. C. (2021). *Pengamanan Teks Terenkripsi Dengan Algoritma RC4 + DanSteganografi DCS Pada Citra Digital*. *9*(April), 232–243.

Maulani, G., Sasongko, N. J., & Mulyana, A. (2016). Pengembangan Media Promosi Pariwisata Kota Tangerang Dalam Bentuk Video Digital Pada Dinas Porparekraf.*ICITJournal*, *2*(2), 207–220. https://doi.org/10.33050/icit.v2i2.35

Pramadana, T. I., Soro, S., & Siswanto, R. D. (2019). Pengembangan Aplikasi BangunDatar Sederhana (Bandara) Matematika Berbasis Android Pada Materi Bangun Datar

Sederhana di Tingkat SMP. *Prosiding Seminar Nasional Teknoka*, *3*(2502), 13. https://doi.org/10.22236/teknoka.v3i0.2894

Sibarani, N. S. (2018). Analisis Performa Aplikasi Native Android Menggunakan Bahasa Pemrograman Java dan Kotlin. *Researchget*, *December*. https://www.researchgate.net/publication/329525878_Analisis_Performa_Aplikasi_Android_Pada_Bahasa_Pemrograman_Java_dan_Kotlin

Syawal, M. F., Fikriansyah, D. C., & Agani, N. (2016). Implementasi Teknik Steganografi Menggunakan Algoritma Vigenere Cipher Dan Metode LSB. *JurnalTICOM*, *4*(3), 91–99.

Verawati, & Liksha, P. D. (2018). Aplikasi Akuntansi Pengolahan Data Jasa Service PadaPt. Budi Berlian Motor Lampung. *Jurnal Sistem Informasi Akuntansi (JUSITA)*, *1*(1), 1–14.

Zebua, T. (2018). *Penerapan Metode Dynamic Cell Spreading (DCS) UntukMenyembunyikan Teks Tersandi Pada Citra*. *November*, 11–12. https://doi.org/10.31227/osf.io/ugzf9

LAMPIRAN

1. Listing Program

```
package com.shania.watermark.activities;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.ImageButton;

import com.shania.watermark.R;
import com.shania.watermark.error.ErrorManager;

public class AboutActivity extends Activity {

  private ImageButton _btnBack;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_about);

    _btnBack = (ImageButton) findViewById(R.id.btn_back_about);
    _btnBack.setOnClickListener(onClickListener);
  }

  @Override
  public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.about, menu);
    return true;
  }

  private OnClickListener onClickListener = new OnClickListener() {

    @Override
    public void onClick(View arg0) {
      switch (arg0.getId()) {
        case R.id.btn_back_about:
          finish();
          break;
        default:
          ErrorManager.getInstance().addErrorMessage("[About Activity] Click event not
known");
          ErrorManager.getInstance().displayErrorMessages(arg0.getContext());
```

```
            break;
        }
      }
    };

}

package com.shania.watermark.activities;

import android.app.Activity;
import android.content.Intent;
import  android.net.Uri;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;

import com.shania.watermark.R;
import com.shania.watermark.wifi.WifiActivity;

public class ChoiceActivity extends Activity {
Button share;
Button p2p;
    public String file_url;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.choice_send);
        share=(Button)findViewById(R.id.send_share);
        p2p=(Button)findViewById(R.id.send_p2p);
        if(getIntent().hasExtra("file_url")) {
            file_url = getIntent().getExtras().getString("file_url");
            Log.i("STEGA", "file url "+file_url);

        }
        share.setOnClickListener(onClickListener);
        p2p.setOnClickListener(onClickListener);
    }


    private View.OnClickListener onClickListener = new View.OnClickListener() {

        @Override
        public void onClick(View arg0) {
            switch (arg0.getId()) {
                case R.id.send_share:
                    Intent sendIntent = new Intent(Intent.ACTION_SEND);
                    sendIntent.setType("video/*");
                    sendIntent.putExtra(Intent.EXTRA_STREAM, Uri.parse(file_url));
                    sendIntent.addFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION);
```

```
                sendIntent.addFlags(Intent.FLAG_GRANT_WRITE_URI_PERMISSION);
                startActivity(Intent.createChooser(sendIntent,"Share video to: "));
                //finish();
                break;
            case R.id.send_p2p:
                Log.i("STEGA", "Start encode");
                Intent intent = new Intent(ChoiceActivity.this, WifiActivity.class);
                intent.putExtra("file_url",file_url);
                startActivity(intent);
                break;

        }
    }
  };
}


package com.shania.watermark.activities;

import  android.app.Activity;
import android.app.AlertDialog;
import android.app.ProgressDialog;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.support.v7.widget.CardView;
import android.text.Editable;
import android.text.TextWatcher;
import android.view.Menu;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.CompoundButton;
import android.widget.CompoundButton.OnCheckedChangeListener;
import android.widget.EditText;
import android.widget.ImageButton;
import android.widget.ImageView;
import android.widget.RadioButton;
import android.widget.TextView;
import android.widget.Toast;

import com.shania.watermark.R;
import com.shania.watermark.configuration.Preferences;
import com.shania.watermark.controller.DecodeParametersController;
import com.shania.watermark.directorydialog.ChoosenDirectoryListener;
import com.shania.watermark.directorydialog.DirectoryDialog;
import com.shania.watermark.error.ErrorManager;
import com.shania.watermark.parameters.DecodeParameters;
```

```
import com.shania.watermark.process.DecodeProcess;
import com.shania.watermark.tools.Utils;

public class DecodeActivity extends Activity {

   private final int CHOOSE_VIDEO_CONTAINER = 0;
   private final int SETTINGS_ACCESS = 1;
   private static final int ERROR_IN_PROCESS = 0;
   private static final int PROCESS_OK = 1;

   // Graphical components
   private ImageButton _btnBack;
   private ImageButton _btnSettings;
   private Button _btnSelectSourceVideo;
   private Button _btnDecode;
   private Button _btnSelectFileDestination;
   private RadioButton _checkBoxDisplayContent;
   private RadioButton _checkBoxSaveIntoFile;
   private RadioButton _radioCompressedTextLZW;
   private RadioButton _radioCompressedTextDeflate;
   private RadioButton _radioNoCompression;
   private TextView _sourcePath;
   private EditText _editTextCryptographyKeyDecode;
   private EditText _editTextFileExtension;
   private CardView _cardViewCryptographyKeyDecode;
   private CardView _cardViewSaveIntoFileDest;
   private CardView _cardViewSaveInoFileExt;
   private CardView _card_layout_decompression;


   private ProgressDialog _progressDialog;
   private DecodeParameters _decodeParameters;

   private Handler processHandler = new Handler()
   {
      @Override
      public void handleMessage(Message msg)
      {
         _progressDialog.cancel();
          if (msg.what == ERROR_IN_PROCESS)
          {
            displayProcessErrors();
          } else {
            displayProcessSuccess();
          }
      }
   };

   private void displayProcessErrors() {
     ErrorManager.getInstance().displayErrorMessages(this);
```

```
  }

  private void displayProcessSuccess() {
    String toastMsg = "Teks watermark telah di ekstrak.";

    if (_decodeParameters.getDisplay())
    {
      AlertDialog.Builder builder = new AlertDialog.Builder(this);
      builder.setTitle("TEKS WATERMARK PADA VIDEO");
      builder.setMessage(_decodeParameters.getDisplayText());
      builder.setPositiveButton("OK", null);
      builder.show();

    }
    Toast.makeText(this, toastMsg, Toast.LENGTH_LONG).show();
  }

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_decode);

      _sourcePath=(TextView)findViewById(R.id.sourcePath);
      _btnBack = (ImageButton) findViewById(R.id.btn_back_decode);
    _btnDecode = (Button) findViewById(R.id.btn_decode);
    _btnSelectSourceVideo = (Button)
findViewById(R.id.btn_select_video_source_decode);
    _btnSelectFileDestination = (Button)
findViewById(R.id.btn_select_file_destination_decode);
    _btnSettings = (ImageButton) findViewById(R.id.btn_settings_decode);
    _checkBoxDisplayContent = (RadioButton)
findViewById(R.id.chk_box_display_content);
    _checkBoxSaveIntoFile = (RadioButton)
findViewById(R.id.chk_box_save_into_file);
    _radioCompressedTextLZW = (RadioButton)
findViewById(R.id.lzwDecompressedText);
    _radioCompressedTextDeflate = (RadioButton)
findViewById(R.id.deflateDecompressedText);

    _editTextCryptographyKeyDecode = (EditText)
findViewById(R.id.edit_text_cryptography_key_decode);
    _editTextFileExtension = (EditText) findViewById(R.id.edit_text_file_extension);

_cardViewCryptographyKeyDecode=(CardView)findViewById(R.id.card_layout_crypto
graphy_key_decode) ;
    _cardViewSaveIntoFileDest = (CardView)
findViewById(R.id.card_view_save_into_file_destination);
    _cardViewSaveInoFileExt = (CardView)
findViewById(R.id.card_view_save_into_file_extension);
```

```
_card_layout_decompression=(CardView)findViewById(R.id.card_layout_decompression);

    _btnBack.setOnClickListener(onClickListener);
    _btnSettings.setOnClickListener(onClickListener);
    _btnSelectSourceVideo.setOnClickListener(onClickListener);
    _btnDecode.setOnClickListener(onClickListener);
    _btnSelectFileDestination.setOnClickListener(onClickListener);
    _checkBoxDisplayContent.setChecked(true);
    _checkBoxSaveIntoFile.setChecked(false);
    _radioCompressedTextLZW.setChecked(true);
    _radioCompressedTextDeflate.setChecked(false);

_checkBoxDisplayContent.setOnCheckedChangeListener(onCheckedChangeListener);
    _checkBoxSaveIntoFile.setOnCheckedChangeListener(onCheckedChangeListener);

_editTextCryptographyKeyDecode.addTextChangedListener(onTextChangedListenerCryptographyKey);

_editTextFileExtension.addTextChangedListener(onTextChangedListenerFileExtension);
    _cardViewSaveInoFileExt.setVisibility(View.GONE);
    _cardViewSaveIntoFileDest.setVisibility(View.GONE);
    _decodeParameters = new DecodeParameters();

    updateImageViews();
    updateLinearLayoutCryptographyVisibility();

  }

  @Override
  public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.decode, menu);
    return true;
  }

  private void updateLinearLayoutCryptographyVisibility() {
    if (Preferences.getInstance().getUseCryptography()) {
      _cardViewCryptographyKeyDecode.setVisibility(View.VISIBLE);
    } else {
      _cardViewCryptographyKeyDecode.setVisibility(View.GONE);
    }
  }

  private void updateImageViews() {
    DecodeParametersController controller = new DecodeParametersController(true);

    if (controller.controlSrcVideoPath(_decodeParameters)) {
      ((ImageView)
findViewById(R.id.img_view_valid_video_source_decode)).setImageResource(R.drawable.ic_check_circle_white_48dp);
```

```
    } else {
      ((ImageView)
findViewById(R.id.img_view_valid_video_source_decode)).setImageResource(R.drawab
le.ic_error_white_48dp);
    }

    if (controller.controlDestFilePath(_decodeParameters)) {
      ((ImageView)
findViewById(R.id.img_view_valid_video_destination_decode)).setImageResource(R.dr
awable.ic_check_circle_white_48dp);
    } else {
      ((ImageView)
findViewById(R.id.img_view_valid_video_destination_decode)).setImageResource(R.dr
awable.ic_error_white_48dp);
    }


    if (controller.controlCryptographyKey(_decodeParameters)) {
      ((ImageView)
findViewById(R.id.img_view_valid_key_length_decode)).setImageResource(R.drawable.
ic_check_circle_white_48dp);
    } else {
      ((ImageView)
findViewById(R.id.img_view_valid_key_length_decode)).setImageResource(R.drawable.
ic_error_white_48dp);
    }
  }

  private void showFileChooser(int code) {
    Intent intent = new Intent(Intent.ACTION_PICK);

    if (code == CHOOSE_VIDEO_CONTAINER) {
      intent.setType("video/*");
    } else {
      intent.setType("*/*");
    }

    try {
      startActivityForResult(Intent.createChooser(intent,
getResources().getString(R.string.destination_string)), code);
    } catch (android.content.ActivityNotFoundException ex) {
      Toast.makeText(this,
getResources().getString(R.string.error_file_manager_string),
Toast.LENGTH_SHORT).show();
    }
  }

  @Override
  public void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
```

```
    switch (requestCode) {
      case CHOOSE_VIDEO_CONTAINER:
        callbackFileChooserVideoContainer(requestCode, resultCode, data);
        break;
      case SETTINGS_ACCESS:
        updateLinearLayoutCryptographyVisibility();
        updateImageViews();
        break;
      default:
        ErrorManager.getInstance().addErrorMessage("[Decode Activity] Activity result
not known");
        ErrorManager.getInstance().displayErrorMessages(this);
        break;
    }
  }

  private void callbackFileChooserVideoContainer(int requestCode, int resultCode, Intent
data) {
    Uri selectedVideoLocation;

    if (resultCode == Activity.RESULT_OK) {
      selectedVideoLocation = data.getData();
        _decodeParameters.setVideoPath(Utils.getRealPathFromUri(this,
selectedVideoLocation));
      if (_decodeParameters.getDestinationVideoDirectory() == null ||
_decodeParameters.getDestinationVideoDirectory().isEmpty()) {

_decodeParameters.setDestinationVideoDirectory(Utils.getBasenameFromPath(_decode
Parameters.getVideoPath()));
          _sourcePath.setText(_decodeParameters.getDestinationVideoDirectory());
        }
      updateImageViews();
    }
  }

  private void showDirectoryChooser() {
    DirectoryDialog dialog = new DirectoryDialog(this);
    dialog.setChoosenDirectoryListener(onChoosenDirectoryListener);
    dialog.show();
  }

  private void process() {
    _progressDialog = ProgressDialog.show(this, "Loading", "Ekstraksi teks
watermark...");

    new Thread(new Runnable() {
      public void run() {
        Message res = processHandler.obtainMessage(PROCESS_OK);
```

```
          DecodeParametersController controller;
          _decodeParameters.setCompressLZW(_radioCompressedTextLZW.isChecked());

_decodeParameters.setCompressDeflate(_radioCompressedTextDeflate.isChecked());

_decodeParameters.setUseAudioChannel(Preferences.getInstance().getUseAudioChannel
());

_decodeParameters.setUseVideoChannel(Preferences.getInstance().getUseVideoChannel
());
          controller = new DecodeParametersController(false);
          if (controller.controlAllData(_decodeParameters)){
            DecodeProcess process = new DecodeProcess();
            if (!process.decode(_decodeParameters)) {
              ErrorManager.getInstance().addErrorMessage("[Decode Activity] Impossible
to find something in this file");
              res = processHandler.obtainMessage(ERROR_IN_PROCESS);
            }
          } else {
            res = processHandler.obtainMessage(ERROR_IN_PROCESS);
          }

          processHandler.sendMessage(res);
        }
      }).start();

  }

  private OnClickListener onClickListener = new OnClickListener() {

    @Override
    public void onClick(View arg0) {
      switch (arg0.getId()) {
        case R.id.btn_back_decode:
          finish();
          break;
        case R.id.btn_settings_decode:
          Intent intent = new Intent(getApplicationContext(), SettingsActivity.class);
          startActivityForResult(intent, SETTINGS_ACCESS);
          break;
        case R.id.btn_select_video_source_decode:
          showFileChooser(CHOOSE_VIDEO_CONTAINER);
          break;
        case R.id.btn_select_file_destination_decode:
          showDirectoryChooser();
          break;
        case R.id.btn_decode:
          process();
          break;
        default:
```

```
            ErrorManager.getInstance().addErrorMessage("[Decode Activity] Click event
not known");
            ErrorManager.getInstance().displayErrorMessages(arg0.getContext());
            break;
        }
      }
    };

    private ChoosenDirectoryListener onChoosenDirectoryListener = new
ChoosenDirectoryListener() {

      @Override
      public void onChoosenDir(String directory) {
        _decodeParameters.setDestinationVideoDirectory(directory);
        updateImageViews();
      }
    };

    private OnCheckedChangeListener onCheckedChangeListener = new
OnCheckedChangeListener() {

      @Override
      public void onCheckedChanged(CompoundButton arg0, boolean arg1) {
        if (arg0.getId() == R.id.chk_box_display_content && arg1) {
          _cardViewSaveInoFileExt.setVisibility(View.GONE);
          _cardViewSaveIntoFileDest.setVisibility(View.GONE);
          _checkBoxSaveIntoFile.setChecked(false);
          _card_layout_decompression.setVisibility(View.VISIBLE);
          _decodeParameters.setDisplay(true);
        } else if (arg0.getId() == R.id.chk_box_display_content) {
          _cardViewSaveInoFileExt.setVisibility(View.VISIBLE);
          _cardViewSaveIntoFileDest.setVisibility(View.VISIBLE);
          _card_layout_decompression.setVisibility(View.GONE);
          _checkBoxSaveIntoFile.setChecked(true);
          _decodeParameters.setDisplay(false);
        } else if (arg0.getId() == R.id.chk_box_save_into_file && arg1) {
          _cardViewSaveInoFileExt.setVisibility(View.VISIBLE);
          _cardViewSaveIntoFileDest.setVisibility(View.VISIBLE);
          _card_layout_decompression.setVisibility(View.GONE);
          _checkBoxDisplayContent.setChecked(false);
          _decodeParameters.setDisplay(false);
        } else {
          _cardViewSaveInoFileExt.setVisibility(View.GONE);
          _cardViewSaveIntoFileDest.setVisibility(View.GONE);
          _card_layout_decompression.setVisibility(View.VISIBLE);
          _checkBoxDisplayContent.setChecked(true);
          _decodeParameters.setDisplay(true);
        }
        updateImageViews();
      }
```

```
  };

  private TextWatcher onTextChangedListenerCryptographyKey = new TextWatcher() {

    @Override
    public void onTextChanged(CharSequence s, int start, int before, int count) {
    }

    @Override
    public void beforeTextChanged(CharSequence s, int start, int count, int after) {
    }

    @Override
    public void afterTextChanged(Editable s) {
      _decodeParameters.setCryptographyKey(s.toString());
      updateImageViews();
    }
  };

  private TextWatcher onTextChangedListenerFileExtension = new TextWatcher() {

    @Override
    public void onTextChanged(CharSequence s, int start, int before, int count) {
    }

    @Override
    public void beforeTextChanged(CharSequence s, int start, int count, int after) {
    }

    @Override
    public void afterTextChanged(Editable s) {
      updateImageViews();
    }
  };
}

package com.shania.watermark.activities;

import  android.app.Activity;
import android.app.ProgressDialog;
import android.content.Intent;
import android.graphics.Color;
import android.net.Uri;
import android.os.Build;
import android.os.Bundle;
import android.os.Environment;
import android.os.Handler;
import android.os.Message;
import android.provider.MediaStore;
import android.support.design.widget.Snackbar;
```

```
import android.support.v7.widget.CardView;
import android.text.Editable;
import android.text.TextWatcher;
import android.view.Menu;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.CompoundButton;
import android.widget.CompoundButton.OnCheckedChangeListener;
import android.widget.EditText;
import android.widget.ImageButton;
import android.widget.ImageView;
import android.widget.LinearLayout;
import android.widget.RadioButton;
import android.widget.TextView;
import android.widget.Toast;

import com.shania.watermark.R;
import com.shania.watermark.configuration.Preferences;
import com.shania.watermark.controller.EncodeParametersController;
import com.shania.watermark.directorydialog.ChoosenDirectoryListener;
import com.shania.watermark.directorydialog.DirectoryDialog;
import com.shania.watermark.error.ErrorManager;
import com.shania.watermark.parameters.EncodeParameters;
import  com.shania.watermark.process.EncodeProcess;
import com.shania.watermark.tools.Utils;

import java.io.File;


public class EncodeActivity  extends Activity{

   private final int CHOOSE_FILE_CONTENT = 0;
   private final int CHOOSE_VIDEO_CONTAINER = 1;
   private final int RECORD_VIDEO = 2;
   private final int SETTINGS_ACCESS = 3;
   private static final int ERROR_IN_PROCESS = 0;
   private static final int PROCESS_OK = 1;

   // Graphical components
   private ImageButton _btnBack;
   private ImageButton _btnCamera;
   private ImageButton _btnSettings;
   private Button _btnSelectSourceVideo;
   private Button _btnSelectVideoDestination;
   private Button _btnSelectFileToHide;
   private Button _btnEncode;
   private  RadioButton _checkBoxFileToHide;
   private RadioButton _checkBoxTextToHide;
   private RadioButton _noCompressio;
```

```
  private RadioButton _radioCompressTextLZW;
  private RadioButton _radioCompressTextDeflate;
  private EditText _editTextContentToHide;
  private EditText _editTextCryptographyKeyEncode;
  private TextView _sourcePath;
  private TextView _destinationPath;
  private TextView _filePath;
  private LinearLayout _linearLayoutEncode;
  private CardView _cardLayoutCryptographyKeyEncode;
  private CardView _cardLayoutCompressionText;


  private ProgressDialog _progressDialog;
  private EncodeParameters _encodeParameters;

  private Handler _processHandler = new Handler()
  {
     @Override
     public void handleMessage(Message msg)
     {
       _progressDialog.cancel();
        if (msg.what == ERROR_IN_PROCESS)
        {
          displayProcessErrors();
        } else {
          displayProcessSuccess();
        }
     }
  };

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_encode);

    _sourcePath=(TextView)findViewById(R.id.sourcePath);
    _destinationPath=(TextView)findViewById(R.id.destinationPath);
    _filePath=(TextView)findViewById(R.id.filePath);
    _btnBack = (ImageButton) findViewById(R.id.btn_back_encode);
    _btnCamera = (ImageButton) findViewById(R.id.btn_camera);
    _btnSettings = (ImageButton) findViewById(R.id.btn_settings_encode);
    _btnSelectSourceVideo = (Button)
findViewById(R.id.btn_select_video_source_encode);
    _btnSelectVideoDestination = (Button)
findViewById(R.id.btn_select_video_destination_encode);
    _btnSelectFileToHide = (Button) findViewById(R.id.btn_select_file_to_hide);
    _btnEncode = (Button) findViewById(R.id.btn_encode);
    _checkBoxFileToHide = (RadioButton) findViewById(R.id.chk_box_file_to_hide);
    _checkBoxTextToHide = (RadioButton) findViewById(R.id.chk_box_text_to_hide);
    _radioCompressTextLZW = (RadioButton) findViewById(R.id.lzwCompressText);
```

```
    _radioCompressTextDeflate = (RadioButton)
findViewById(R.id.deflateCompressText);
    _noCompressio=(RadioButton)findViewById(R.id.radio_no_compression);
    _editTextContentToHide = (EditText)
findViewById(R.id.edit_text_content_to_hide);
    _editTextCryptographyKeyEncode = (EditText)
findViewById(R.id.edit_text_cryptography_key_encode);
    _linearLayoutEncode = (LinearLayout) findViewById(R.id.linearLayoutEncode);
    _cardLayoutCryptographyKeyEncode=(CardView)
findViewById(R.id.card_layout_cryptography_key_encode);

_cardLayoutCompressionText=(CardView)findViewById(R.id.card_layout_compression
);
    _btnBack.setOnClickListener(onClickListener);
    _btnCamera.setOnClickListener(onClickListener);
    _btnSettings.setOnClickListener(onClickListener);
    _btnSelectSourceVideo.setOnClickListener(onClickListener);
    _btnSelectVideoDestination.setOnClickListener(onClickListener);
    _btnSelectFileToHide.setOnClickListener(onClickListener);
    _btnEncode.setOnClickListener(onClickListener);
    _checkBoxFileToHide.setChecked(false);
      _checkBoxTextToHide.setChecked(true);
    _radioCompressTextLZW.setChecked(true);
    _radioCompressTextDeflate.setChecked(false);
    _checkBoxFileToHide.setOnCheckedChangeListener(onCheckedChangeListener);
      _checkBoxTextToHide.setOnCheckedChangeListener(onCheckedChangeListener);

_editTextCryptographyKeyEncode.addTextChangedListener(onTextChangedListenerCry
ptographyKey);

_editTextContentToHide.addTextChangedListener(onTextChangedListenerTextToHide);
    _encodeParameters = new EncodeParameters();

    updateImageViews();
    updateLinearLayoutCryptographyVisibility();


  }

  @Override
  public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.encode, menu);
    return true;
  }

  private void displayProcessErrors() {
    ErrorManager.getInstance().displayErrorMessages(this);
  }

  private void displayProcessSuccess() {
```

```
      final Uri contentUri;
      if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.KITKAT) {
        final Intent scanIntent = new
Intent(Intent.ACTION_MEDIA_SCANNER_SCAN_FILE);
        contentUri = Uri.fromFile(new File("file://" +
Environment.getExternalStorageDirectory()));
        scanIntent.setData(contentUri);
        sendBroadcast(scanIntent);
      } else {
        final Intent intent = new Intent(Intent.ACTION_MEDIA_MOUNTED,
Uri.parse("file://" + Environment.getExternalStorageDirectory()));
        sendBroadcast(intent);
      }

      Snackbar snackbar = Snackbar.make(_linearLayoutEncode, "Video berwatermark
telah disimpan.", Snackbar.LENGTH_LONG);

      snackbar.setActionTextColor(Color.parseColor("#008f30"));
      snackbar.show();
    }

    private void updateLinearLayoutCryptographyVisibility() {
      if (Preferences.getInstance() != null) {
        if (Preferences.getInstance().getUseCryptography()) {
          _cardLayoutCryptographyKeyEncode.setVisibility(View.VISIBLE);
        } else {
          _cardLayoutCryptographyKeyEncode.setVisibility(View.GONE);
        }
      }
    }

    private void updateImageViews() {
      EncodeParametersController controller = new EncodeParametersController(true);

      if (controller.controlSrcVideoPath(_encodeParameters)) {
        ((ImageView)
findViewById(R.id.img_view_valid_video_source_encode)).setImageResource(R.drawab
le.ic_check_circle_white_48dp);
      } else {
        ((ImageView)
findViewById(R.id.img_view_valid_video_source_encode)).setImageResource(R.drawab
le.ic_error_white_48dp);
      }

      if (controller.controlDestVideoPath(_encodeParameters)) {
        ((ImageView)
findViewById(R.id.img_view_valid_video_destination_encode)).setImageResource(R.dr
awable.ic_check_circle_white_48dp);
      } else {
        ((ImageView)
```

```
findViewById(R.id.img_view_valid_video_destination_encode)).setImageResource(R.dr
awable.ic_error_white_48dp);
    }

    if (controller.controlContentToHide(_encodeParameters)) {
      ((ImageView)
findViewById(R.id.img_view_valid_content_to_hide_encode)).setImageResource(R.dra
wable.ic_check_circle_white_48dp);
    } else {
    ((ImageView)
findViewById(R.id.img_view_valid_content_to_hide_encode)).setImageResource(R.dra
wable.ic_error_white_48dp);
    }

    if (controller.controlCryptographyKey(_encodeParameters)) {
      ((ImageView)
findViewById(R.id.img_view_valid_key_length_encode)).setImageResource(R.drawable.
ic_check_circle_white_48dp);
    } else {
      ((ImageView)
findViewById(R.id.img_view_valid_key_length_encode)).setImageResource(R.drawable.
ic_error_white_48dp);
    }
  }

  private void recordVideo() {
    Intent takeVideoIntent = new Intent(MediaStore.ACTION_VIDEO_CAPTURE);

     if (takeVideoIntent.resolveActivity(getPackageManager()) != null) {
        startActivityForResult(takeVideoIntent, RECORD_VIDEO);
     }
  }

  private void showDirectoryChooser() {
    DirectoryDialog dialog = new DirectoryDialog(this);
    dialog.setChoosenDirectoryListener(onChoosenDirectoryListener);
    dialog.show();
  }

  private void showFileChooser(int code) {
     Intent intent = new Intent(Intent.ACTION_PICK);
     //intent.addCategory(Intent.CATEGORY_OPENABLE);

     if (code == CHOOSE_VIDEO_CONTAINER) {
       intent.setType("video/*");
     } else {
       intent.setType("*/*");
     }
      try {
        startActivityForResult(Intent.createChooser(intent,
```

```
getResources().getString(R.string.destination_string)), code);
      } catch (android.content.ActivityNotFoundException ex) {
         Toast.makeText(this,
getResources().getString(R.string.error_file_manager_string),
Toast.LENGTH_SHORT).show();
      }
  }

  @Override
  public void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);

    switch (requestCode) {
      case CHOOSE_VIDEO_CONTAINER:
        callbackFileChooserVideoContainer(requestCode, resultCode, data);
        break;
      case CHOOSE_FILE_CONTENT:
        callbackFileChooserFileToHide(requestCode, resultCode, data);
        break;
      case SETTINGS_ACCESS:
        updateLinearLayoutCryptographyVisibility();
        updateImageViews();
        break;
      case RECORD_VIDEO:
        if (resultCode != 0)
          Toast.makeText(this, "Video saved", Toast.LENGTH_LONG).show();
        break;
      default:
        ErrorManager.getInstance().addErrorMessage("[Encode Activity] Activity result
not known");
        ErrorManager.getInstance().displayErrorMessages(this);
        break;
    }
  }

  private void callbackFileChooserVideoContainer(int requestCode, int resultCode, Intent
data) {
    Uri selectedVideoLocation;
     if (resultCode == Activity.RESULT_OK) {
        selectedVideoLocation = data.getData();
        if (selectedVideoLocation != null) {
          _encodeParameters.setSourceVideoPath(Utils.getRealPathFromUri(this,
selectedVideoLocation));
          _sourcePath.setText(_encodeParameters.getSourceVideoPath());
          if (_encodeParameters.getDestinationVideoDirectory() == null ||
_encodeParameters.getDestinationVideoDirectory().isEmpty()) {

_encodeParameters.setDestinationVideoDirectory(Utils.getBasenameFromPath(_encode
Parameters.getSourceVideoPath()));
          _destinationPath.setText(_encodeParameters.getDestinationVideoDirectory());
```

```
            }
            } else {
               _encodeParameters.setSourceVideoPath("");
            }
            updateImageViews();
        }
    }

    private void callbackFileChooserFileToHide(int requestCode, int resultCode, Intent
data) {
        Uri selectedFileLocation;

        if (resultCode == Activity.RESULT_OK) {
          selectedFileLocation = data.getData();
          if (selectedFileLocation != null) {
            _encodeParameters.setFileToHidePath(Utils.getRealPathFromUri(this,
selectedFileLocation));
          } else {
            _encodeParameters.setFileToHidePath("");
          }
          _filePath.setText(_encodeParameters.getFileToHidePath());
          updateImageViews();
        }
    }

    private void process() {
        _progressDialog = ProgressDialog.show(this, "Loading", "Watermarking video...");

        new Thread(new Runnable() {
          public void run() {


            Message res = _processHandler.obtainMessage(PROCESS_OK);

            EncodeParametersController controller;
            String textToHide=_editTextContentToHide.getText().toString();

            _encodeParameters.setCompressLZW(_radioCompressTextLZW.isChecked());

_encodeParameters.setCompressDeflate(_radioCompressTextDeflate.isChecked());
            _encodeParameters.setTextToHide(textToHide);
            controller = new EncodeParametersController(false);
            if (controller.controlAllData(_encodeParameters)){
              EncodeProcess process = new EncodeProcess();
              if (!process.encode(_encodeParameters)) {
                res = _processHandler.obtainMessage(ERROR_IN_PROCESS);
              }
            } else {
              res = _processHandler.obtainMessage(ERROR_IN_PROCESS);
            }
```

```
            _processHandler.sendMessage(res);

          }
        }).start();
    }

    private ChoosenDirectoryListener onChoosenDirectoryListener = new
ChoosenDirectoryListener() {

      @Override
      public void onChoosenDir(String directory) {
        _encodeParameters.setDestinationVideoDirectory(directory);
        _destinationPath.setText(_encodeParameters.getDestinationVideoDirectory());
        updateImageViews();
      }
    };

    private OnCheckedChangeListener onCheckedChangeListener = new
OnCheckedChangeListener() {

      @Override
      public void onCheckedChanged(CompoundButton arg0, boolean arg1) {
        if (arg0.getId() == R.id.chk_box_file_to_hide && arg1) {
          _editTextContentToHide.setVisibility(View.GONE);
          _cardLayoutCompressionText.setVisibility(View.GONE);
          _btnSelectFileToHide.setVisibility(View.VISIBLE);
          _filePath.setVisibility(View.VISIBLE);
          _checkBoxTextToHide.setChecked(false);
          _encodeParameters.setHidingText(false);
        } else if (arg0.getId() == R.id.chk_box_file_to_hide) {
          _editTextContentToHide.setVisibility(View.VISIBLE);
          _cardLayoutCompressionText.setVisibility(View.VISIBLE);
          _btnSelectFileToHide.setVisibility(View.GONE);
          _filePath.setVisibility(View.GONE);
          _checkBoxTextToHide.setChecked(true);
          _encodeParameters.setHidingText(true);
        } else if (arg0.getId() == R.id.chk_box_text_to_hide && arg1) {
          _btnSelectFileToHide.setVisibility(View.GONE);
          _filePath.setVisibility(View.GONE);
          _editTextContentToHide.setVisibility(View.VISIBLE);
          _cardLayoutCompressionText.setVisibility(View.VISIBLE);
          _checkBoxFileToHide.setChecked(false);
          _encodeParameters.setHidingText(true);
        } else {
          _filePath.setVisibility(View.VISIBLE);
          _btnSelectFileToHide.setVisibility(View.VISIBLE);
          _editTextContentToHide.setVisibility(View.GONE);
          _cardLayoutCompressionText.setVisibility(View.GONE);
          _checkBoxFileToHide.setChecked(true);
```

```
            _encodeParameters.setHidingText(false);
        }
        updateImageViews();
    }
  };

  private OnClickListener onClickListener = new OnClickListener() {

    @Override
    public void onClick(View arg0) {
      switch (arg0.getId()) {
        case R.id.btn_back_encode:
          finish();
          break;
        case R.id.btn_camera:
          recordVideo();
          break;
        case R.id.btn_settings_encode:
          Intent intent = new Intent(getApplicationContext(), SettingsActivity.class);
          startActivityForResult(intent, SETTINGS_ACCESS);
          break;
        case R.id.btn_select_video_source_encode:
          showFileChooser(CHOOSE_VIDEO_CONTAINER);
          break;
        case R.id.btn_select_video_destination_encode:
          showDirectoryChooser();
          break;
        case R.id.btn_select_file_to_hide:
          showFileChooser(CHOOSE_FILE_CONTENT);
          break;
        case R.id.btn_encode:
          process();
          break;
        default:
          ErrorManager.getInstance().addErrorMessage("[Encode Activity] Click event
not known");
          ErrorManager.getInstance().displayErrorMessages(arg0.getContext());
          break;
      }
    }
  };

  private TextWatcher onTextChangedListenerCryptographyKey = new TextWatcher() {

    @Override
    public void onTextChanged(CharSequence s, int start, int before, int count) {
    }

    @Override
    public void beforeTextChanged(CharSequence s, int start, int count, int after) {
```

```
    }

    @Override
    public void afterTextChanged(Editable s) {
      if (s != null) {
        _encodeParameters.setCryptographyKey(s.toString());
      }
      updateImageViews();
    }
  };

  private TextWatcher onTextChangedListenerTextToHide = new TextWatcher() {

    @Override
    public void onTextChanged(CharSequence s, int start, int before, int count) {
    }

    @Override
    public void beforeTextChanged(CharSequence s, int start, int count, int after) {
    }

    @Override
    public void afterTextChanged(Editable s) {
      if (s != null) {
        _encodeParameters.setTextToHide(s.toString());
      }
      updateImageViews();
    }
  };

}


package com.shania.watermark.activities;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import android.app.Activity;
import android.graphics.Point;
import android.os.Bundle;
import android.view.Display;
import android.view.Menu;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemSelectedListener;
import android.widget.ArrayAdapter;
```

```
import android.widget.CheckBox;
import android.widget.CompoundButton;
import android.widget.CompoundButton.OnCheckedChangeListener;
import android.widget.ImageButton;
import android.widget.Spinner;

import com.shania.watermark.R;
import com.shania.watermark.algorithms.IDataAlgorithm;
import
com.shania.watermark.algorithms.data.SteganographyAlgorithmData.SteganographyCha
nnelType;
import com.shania.watermark.configuration.Configuration;
import com.shania.watermark.configuration.Preferences;
import com.shania.watermark.error.ErrorManager;

public class SettingsActivity extends Activity {

  // Graphical components
  private Spinner       _spinAudioAlrogithm;
  private Spinner       _spinVideoAlrogithm;
  private Spinner       _spinCryptographyAlgorithm;
  private CheckBox  _chkboxAudioChannel;
  private CheckBox   _chkboxVideoChannel;
  private CheckBox   _chkboxCryptography;
  private ImageButton _btnBack;

  // Private attributes
  private Map<String, String>    _mapClasses;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_settings);

    _chkboxAudioChannel = (CheckBox) findViewById(R.id.chk_box_audio_channel);
    _chkboxVideoChannel = (CheckBox) findViewById(R.id.chk_box_video_channel);
    _chkboxCryptography = (CheckBox) findViewById(R.id.chk_box_cryptography);

    _spinAudioAlrogithm = (Spinner) findViewById(R.id.spinner_audio_algorithm);
    _spinVideoAlrogithm = (Spinner) findViewById(R.id.spinner_video_algorithm);
    _spinCryptographyAlgorithm = (Spinner)
findViewById(R.id.spinner_cryptography_algorithm);

    _btnBack = (ImageButton) findViewById(R.id.btn_back_settings);
    _btnBack.setOnClickListener(onClickListener);

    _mapClasses = new HashMap<String, String>();

    this.initCheckboxes();
    this.initSpinnerContentFromList(this._spinAudioAlrogithm,
```

```
Configuration.getInstance().getSteganographyAlgorithmByType(SteganographyChannel
Type.AUDIO),
            Preferences.getInstance().getAudioAlgorithm());
      this.initSpinnerContentFromList(this._spinVideoAlrogithm,

Configuration.getInstance().getSteganographyAlgorithmByType(SteganographyChannel
Type.VIDEO),
            Preferences.getInstance().getVideoAlgorithm());

      this.initSpinnerContentFromList(this._spinCryptographyAlgorithm,
            Configuration.getInstance().getCryptographyAlgorithms(),
            Preferences.getInstance().getCryptographyAlgorithm());
      this.actualizeSpinners();

      Display display = getWindowManager().getDefaultDisplay();
      Point size = new Point();
      display.getSize(size);
    }

    private void initCheckboxes() {

_chkboxAudioChannel.setChecked(Preferences.getInstance().getUseAudioChannel());
      _chkboxAudioChannel.setOnCheckedChangeListener(onCheckedChangeListener);


_chkboxVideoChannel.setChecked(Preferences.getInstance().getUseVideoChannel());
      _chkboxVideoChannel.setOnCheckedChangeListener(onCheckedChangeListener);


      _chkboxCryptography.setChecked(Preferences.getInstance().getUseCryptography());
      _chkboxCryptography.setOnCheckedChangeListener(onCheckedChangeListener);
    }

    private void initSpinnerContentFromList(Spinner spinner, List<IDataAlgorithm> list,
String defaultValue) {
      ArrayAdapter<String> adaptater;
      int idx = 0;

      if (_mapClasses == null)
        _mapClasses = new HashMap<String, String>();

      adaptater = new ArrayAdapter<String>(this, android.R.layout.simple_spinner_item,
new ArrayList<String>());

adaptater.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item)
;
      for (int i = 0; i < list.size(); ++i) {
        IDataAlgorithm algorithm = list.get(i);
        _mapClasses.put(algorithm.getDisplayName(), algorithm.getPath());
```

```
        adaptater.add(algorithm.getDisplayName());
        if (algorithm.getPath().equals(defaultValue)) {
          idx = i;
        }
      }
      spinner.setAdapter(adaptater);
      spinner.setSelection(idx);
      spinner.setOnItemSelectedListener(onItemSelectedListener);
    }

    private void actualizeSpinners() {
      if (!_chkboxAudioChannel.isChecked() && _spinAudioAlrogithm.isEnabled()) {
        _spinAudioAlrogithm.setEnabled(false);
        _spinAudioAlrogithm.setVisibility(View.GONE);
      } else if (_chkboxAudioChannel.isChecked() && !_spinAudioAlrogithm.isEnabled())
{
        _spinAudioAlrogithm.setEnabled(true);
        _spinAudioAlrogithm.setVisibility(View.VISIBLE);
      }

      if (!_chkboxVideoChannel.isChecked() && _spinVideoAlrogithm.isEnabled()) {
        _spinVideoAlrogithm.setEnabled(false);
        _spinVideoAlrogithm.setVisibility(View.GONE);

      } else if (_chkboxVideoChannel.isChecked() && !_spinVideoAlrogithm.isEnabled())
{
        _spinVideoAlrogithm.setEnabled(true);
        _spinVideoAlrogithm.setVisibility(View.VISIBLE);
      }



      if (!_chkboxCryptography.isChecked() &&
_spinCryptographyAlgorithm.isEnabled()) {
        _spinCryptographyAlgorithm.setEnabled(false);
        _spinCryptographyAlgorithm.setVisibility(View.GONE);
      } else if (_chkboxCryptography.isChecked() &&
!_spinCryptographyAlgorithm.isEnabled()) {
        _spinCryptographyAlgorithm.setEnabled(true);
        _spinCryptographyAlgorithm.setVisibility(View.VISIBLE);
      }
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
      getMenuInflater().inflate(R.menu.settings, menu);
      return true;
    }

    private OnCheckedChangeListener onCheckedChangeListener = new
```

```
OnCheckedChangeListener() {

    @Override
    public void onCheckedChanged(CompoundButton arg0, boolean arg1) {
      switch (arg0.getId()) {
        case R.id.chk_box_audio_channel:
          Preferences.getInstance().setUseAudioChannel(arg1);
          break;
        case R.id.chk_box_video_channel:
          Preferences.getInstance().setUseVideoChannel(arg1);
          break;
        case R.id.chk_box_cryptography:
          Preferences.getInstance().setUseCryptography(arg1);
          break;
        default:
          ErrorManager.getInstance().addErrorMessage("[About Activity] Checked
change event not known");
          ErrorManager.getInstance().displayErrorMessages(arg0.getContext());
          break;
      }
      actualizeSpinners();
    }
  };

  private OnItemSelectedListener onItemSelectedListener = new
OnItemSelectedListener() {

    @Override
    public void onItemSelected(AdapterView<?> arg0, View arg1, int position, long
arg3) {
      String key;

      switch (arg0.getId()) {
        case R.id.spinner_audio_algorithm:
          key = (String) _spinAudioAlrogithm.getSelectedItem();
          if (_mapClasses.containsKey(key))
            Preferences.getInstance().setAudioAlgorithm(_mapClasses.get(key));
          else
            Preferences.getInstance().setAudioAlgorithm(null);
          break;
        case R.id.spinner_video_algorithm:
          key = (String) _spinVideoAlrogithm.getSelectedItem();
          if (_mapClasses.containsKey(key))
            Preferences.getInstance().setVideoAlgorithm(_mapClasses.get(key));
          else
            Preferences.getInstance().setVideoAlgorithm(null);
          break;

        case R.id.spinner_cryptography_algorithm:
          key = (String) _spinCryptographyAlgorithm.getSelectedItem();
```

```
            if (_mapClasses.containsKey(key)) {
              System.out.println("Save the crypto: " + _mapClasses.get(key));
              Preferences.getInstance().setCryptographyAlgorithm(_mapClasses.get(key));
            }
            else
              Preferences.getInstance().setCryptographyAlgorithm(null);
            break;
          default:
            ErrorManager.getInstance().addErrorMessage("[About Activity] Spinner event
not known");
            ErrorManager.getInstance().displayErrorMessages(arg0.getContext());
            break;
        }
      }

      @Override
      public void onNothingSelected(AdapterView<?> arg0) {
        return;
      }
    };

    private OnClickListener onClickListener = new OnClickListener() {

      @Override
      public void onClick(View arg0) {
        switch (arg0.getId()) {
          case R.id.btn_back_settings:
            finish();
            break;
          default:
            ErrorManager.getInstance().addErrorMessage("[Settings Activity] Click event
not known");
            ErrorManager.getInstance().displayErrorMessages(arg0.getContext());
            break;
        }
      }
    };

}
```

algorithms

compression

```
package com.shania.watermark.algorithms.compression;
```

```
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import  java.nio.charset.StandardCharsets;
import java.util.zip.DataFormatException;
import java.util.zip.Deflater;
import java.util.zip.Inflater;

public class Deflate {
   public static byte[] compress(String text) {
      byte[] data = text.getBytes(StandardCharsets.UTF_8);
      Deflater deflater = new Deflater();
      deflater.setInput(data);
      ByteArrayOutputStream outputStream = new ByteArrayOutputStream(data.length);
      deflater.finish();
      byte[] buffer = new byte[1024];
      while (!deflater.finished()) {
         int count = deflater.deflate(buffer);
         outputStream.write(buffer, 0, count);
      }
      try {
         outputStream.close();
      } catch (IOException e) {
         e.printStackTrace();
      }
      return outputStream.toByteArray();
   }

   public static String decompress(byte[] data) {
      Inflater inflater = new Inflater();
      inflater.setInput(data);
      ByteArrayOutputStream outputStream = new ByteArrayOutputStream(data.length);
      byte[] buffer = new byte[1024];
      while (!inflater.finished()) {
         int count = 0;
         try {
            count = inflater.inflate(buffer);
         } catch (DataFormatException e) {
            e.printStackTrace();
         }
         outputStream.write(buffer, 0, count);
      }
      try {
         outputStream.close();
      } catch (IOException e) {
         e.printStackTrace();
      }
      byte[] output = outputStream.toByteArray();
      return new String(output, StandardCharsets.UTF_8);
   }
}
```

```
package com.shania.watermark.algorithms.compression;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.StringTokenizer;

public class LZW {
  private static double MAX_TABLE_SIZE = Math.pow(2, 8);

  public static String compress(String text) {
    double table_Size = 255;
    Map<String, Integer> TABLE = new HashMap<String, Integer>();
    for (int i = 0; i < 255; i++)
      TABLE.put("" + (char) i, i);
    String initString = "";
    List<Integer> encoded_values = new ArrayList<Integer>();
    for (char symbol : text.toCharArray()) {
      String Str_Symbol = initString + symbol;
      if (TABLE.containsKey(Str_Symbol))
        initString = Str_Symbol;
      else {
        encoded_values.add(TABLE.get(initString));

        if (table_Size < MAX_TABLE_SIZE)
          TABLE.put(Str_Symbol, (int) table_Size++);
        initString = "" + symbol;
      }
    }

    if (!initString.equals(""))
      encoded_values.add(TABLE.get(initString));
    StringBuilder sb = new StringBuilder();
    Iterator<Integer> listIterator = encoded_values.listIterator();
    while (listIterator.hasNext()) {
      sb.append(listIterator.next());
      if (listIterator.hasNext())
        sb.append(" ");
    }
    return sb.toString();

  }

  public static String decompress(String compressedText) {
    List<Integer> get_compress_values = new ArrayList<Integer>();
    int table_Size = 255;
    StringTokenizer st = new StringTokenizer(compressedText);
```

```
    while (st.hasMoreTokens()) {
      get_compress_values.add(Integer.parseInt(st.nextToken()));
    }
    Map<Integer, String> TABLE = new HashMap<Integer, String>();
    for (int i = 0; i < 255; i++)
      TABLE.put(i, "" + (char) i);
    String Encode_values = "" + (char) (int) get_compress_values.remove(0);
    StringBuffer decoded_values = new StringBuffer(Encode_values);
    String get_value_from_table = null;
    for (int check_key : get_compress_values) {
      if (TABLE.containsKey(check_key))
        get_value_from_table = TABLE.get(check_key);
      else if (check_key == table_Size)
        get_value_from_table = Encode_values + Encode_values.charAt(0);
      decoded_values.append(get_value_from_table);
      if (table_Size < MAX_TABLE_SIZE)
        TABLE.put(table_Size++, Encode_values + get_value_from_table.charAt(0));
      Encode_values = get_value_from_table;
    }
    return decoded_values.toString();
  }

}
```

data

```
package com.shania.watermark.algorithms.data;

import com.shania.watermark.algorithms.IDataAlgorithm;

public class CryptographyAlgorithmData implements IDataAlgorithm {

  private String _displayName;
  private int _keyLength;
  private String _path;

  public CryptographyAlgorithmData() {
    _displayName = "";
    _keyLength = 0;
    _path = "";
  }

  public CryptographyAlgorithmData(String displayName, int keylength, String path) {
    _displayName = displayName;
    _keyLength = keylength;
    _path = path;
  }

  public void setKeyLength(int keylength) {
    _keyLength = keylength;
```

```
  }

  public int getKeyLength() {
    return _keyLength;
  }

  public void setDisplayName(String displayName) {
    _displayName = displayName;
  }

  public String getDisplayName() {
    return _displayName;
  }

  public void setPath(String path) {
    _path = path;
  }

  public String getPath() {
    return _path;
  }
}


package com.shania.watermark.algorithms.data;

import com.shania.watermark.algorithms.IDataAlgorithm;

public class SteganographyAlgorithmData implements IDataAlgorithm {

  public enum SteganographyChannelType{
    AUDIO,
    VIDEO,
    METADATA,
    NONE
  }

  private String _displayName;
  private String _path;
  private SteganographyChannelType _steganographyChannel;

  public SteganographyAlgorithmData() {
    _displayName = "";
    _path = "";
    _steganographyChannel = SteganographyChannelType.NONE;
  }

  public SteganographyAlgorithmData(String displayName, String path,
SteganographyChannelType steganographyChannel) {
    _displayName = displayName;
    _path = path;
```

```
    _steganographyChannel = steganographyChannel;
  }

  public void setDisplayName(String displayName) {
    _displayName = displayName;
  }

  public String getDisplayName() {
    return _displayName;
  }

  public void setPath(String path) {
    _path = path;
  }

  public String getPath() {
    return _path;
  }

  public void setSteganographyChannelType(SteganographyChannelType
steganographyChannelType) {
    _steganographyChannel = steganographyChannelType;
  }

  public SteganographyChannelType getSteganographyChannelType() {
    return _steganographyChannel;
  }

}
```

Steganografi

Data

Vidio

```
package com.shania.watermark.algorithms.steganography.video;

import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;
import java.nio.ByteBuffer;

import com.coremedia.iso.IsoTypeReaderVariable;
import  com.coremedia.iso.boxes.mdat.SampleList;
import com.googlecode.mp4parser.DataSource;
import com.googlecode.mp4parser.FileDataSourceImpl;
```

```
import com.googlecode.mp4parser.authoring.Sample;
import com.shania.watermark.algorithms.ISteganographyContainer;
import com.shania.watermark.h264.NaluParser;
import com.shania.watermark.h264.PictureParameterSetParser;
import com.shania.watermark.h264.SeqParameterSetParser;
import com.shania.watermark.h264.SliceParser;
import com.shania.watermark.mp4.MP4MediaReader;
import com.shania.watermark.mp4.SteganosMemoryDataSourceImpl;

public class H264SteganographyContainer implements ISteganographyContainer {

  private final int MAX_SIZE_BUFFERING = 7000000; // 10 Mo
  private final String FILE_STORAGE_NAME = "h264.tmp";

  protected SeqParameterSetParser _seqParameterSetParser;
  protected PictureParameterSetParser _pictureParameterSetParser;

  protected OutputStream _content;
  protected DataSource _dataSource;
  protected SampleList _sampleList;
  protected String _fileStreamDirectory;
  protected int _sampleLengthSize;
  protected int _sampleListPosition;
  protected int _subSampleIdx;
  protected int _subSampleOffset;

  protected byte[] _unHideData;

  public H264SteganographyContainer() {
    _content = null;
    _dataSource = null;
    _sampleList = null;
    _fileStreamDirectory = null;
    _sampleLengthSize = 0;
    _sampleListPosition = 0;
    _subSampleIdx = 0;
    _subSampleOffset = 0;

    _unHideData = null;
  }

  // Interface methods
  @Override
  public boolean loadData(MP4MediaReader mediaReader) {
    byte[] sps;
    byte[] pps;
    NaluParser naluParser;

    if (mediaReader != null) {
      _content = new ByteArrayOutputStream();
```

```
        _sampleList = mediaReader.getVideoSampleList();
        _sampleLengthSize = mediaReader.getVideoSampleLengthSize() + 1;
        _sampleListPosition = 0;
        _subSampleIdx = 0;
        _subSampleOffset = 0;

        sps = mediaReader.getSequenceParameterSets();
        if (sps != null && sps.length > 0) {
          this.addData(new byte[]{0, 0, 0, 1});
          this.addData(sps);
          naluParser = new NaluParser();
          _seqParameterSetParser = new SeqParameterSetParser();
          naluParser.parseNaluData(sps);
          _seqParameterSetParser.parseSeqParameterSetData(naluParser.getRbsp());
        }

        pps = mediaReader.getPictureParameterSets();
        if (pps != null && pps.length > 0) {
          this.addData(new byte[]{0, 0, 0, 1});
          this.addData(pps);
          naluParser = new NaluParser();
          _pictureParameterSetParser = new
PictureParameterSetParser(_seqParameterSetParser);
          naluParser.parseNaluData(pps);
          _pictureParameterSetParser.parsePictureParameterSet(naluParser.getRbsp());
        }
        return true;
      }
      return false;
    }

  public void writeRemainingSamples() {
      Sample currentSample;
      ByteBuffer currentSampleBuffer;
      int currentSampleLength = -1;
      byte[] dataToWrite;

      if (_sampleList == null || _sampleListPosition >= _sampleList.size()) {
        return;
      }

      for (; _sampleListPosition < _sampleList.size(); _sampleListPosition++) {
        currentSample = _sampleList.get(_sampleListPosition);
        currentSampleBuffer = currentSample.asByteBuffer();

        if (_subSampleIdx > 0) {
          for (int i = 0; i < _subSampleIdx; ++i) {
            currentSampleLength = (int)
IsoTypeReaderVariable.read(currentSampleBuffer, _sampleLengthSize);
            currentSampleBuffer.position(currentSampleBuffer.position() +
```

```
currentSampleLength);
          }
        }

        while (currentSampleBuffer.hasRemaining()) {
          currentSampleLength = (int) IsoTypeReaderVariable.read(currentSampleBuffer,
_sampleLengthSize);
          currentSampleBuffer.position(currentSampleBuffer.position() +
_subSampleOffset);
          dataToWrite = new byte[currentSampleLength - _subSampleOffset];
          currentSampleBuffer.get(dataToWrite);
          if (_subSampleOffset == 0) {
            this.addData(new byte[]{0x00, 0x00, 0x01});
          }
          this.addData(dataToWrite);
          _subSampleOffset = 0;
          _subSampleIdx++;
        }
        _subSampleIdx = 0;
      }
    }

    @Override
    public void hideData(byte[] content) {
    }

    @Override
    public void unHideData() {
    }

    @Override
    public long getMaxContentToHide() {
      long ret = 0;

      if (_sampleList != null) {
        for (Sample s : _sampleList) {
          ret += s.getSize();
        }
      }
      return ret;
    }

    @Override
    public byte[] getUnHideData() {
      return _unHideData;
    }

    public DataSource getDataSource() {
      if (_dataSource != null) {
        cleanDataSource();
```

```
      }
      if (_content != null) {
        if (_content instanceof ByteArrayOutputStream) {
          _dataSource = new
SteganosMemoryDataSourceImpl(((ByteArrayOutputStream)_content).toByteArray());
        } else {
          try {
            _content.close();
            _content = null;
            System.gc();
            _dataSource = new FileDataSourceImpl(new File(_fileStreamDirectory +
FILE_STORAGE_NAME));
          } catch (FileNotFoundException e) {
            System.err.println("[H264 Steganography container]: Unable to get data source:
" +  e.getMessage());
          } catch (IOException e) {
            System.err.println("[H264 Steganography container]: Unable to get data source:
" +  e.getMessage());
          }
        }
      }
      return _dataSource;
    }

    public void setFileStreamDirectory(String directory) {
      _fileStreamDirectory = directory;
    }

    public String getFileStreamDirectory() {
      return _fileStreamDirectory;
    }

    // Specific methods
    protected void addData(byte[] content) {
      switchOutputStreamToFile();
      if (_content != null) {
        try {
          _content.write(content);
        } catch (IOException e) {
          System.err.println("[H264 Steganography container]: Unable to add data: " +
e.getMessage());
        }
      }
    }

    protected void addData(byte data) {
      byte content[] = new byte[1];

      content[0] = data;
      addData(content);
```

```
  }

  protected void addData(ByteBuffer content) {
    byte tmp[];

    if (_content != null) {
      tmp = new byte[content.remaining()];
      content.get(tmp);
      addData(tmp);
    }
  }

  public void cleanUpResources() {
    cleanDataSource();
    cleanContentStream();
  }

  //Private methods
  private void cleanDataSource() {
    if (_dataSource != null) {
      try {
        _dataSource.close();
        _dataSource = null;
      } catch (IOException e) {
        System.err.println("[H264 Steganography container]: Unable to clean data source:
" +  e.getMessage());
      }
    }
    System.gc();
  }

  private void cleanContentStream() {
    if (_content != null) {
      try {
        _content.close();
        _content = null;
      } catch (IOException e) {
        System.err.println("[H264 Steganography container]: Unable to clean content
stream: " +  e.getMessage());
      }
    }
    System.gc();
    File file = new File(_fileStreamDirectory + FILE_STORAGE_NAME);
    file.delete();
  }

  protected int getSliceLayerWithoutPartitioningIdrDataOffset(ByteBuffer sample) {
    NaluParser parser = new NaluParser();
    byte tmp[] = new byte[sample.remaining()];
```

```
      sample.get(tmp);
      parser.parseNaluData(tmp);
      // 5 = slice layer without partitioning IDR
      if (parser.getNalUnitType() == 5) {
        SliceParser sp = new SliceParser(_seqParameterSetParser,
_pictureParameterSetParser, parser.getNalUnitType(), parser.getNalRefIdc());
        sp.parseSlice(parser.getRbsp());
        return (sp.getSliceDataOffset() + parser.getNaluHeaderSize());
      }
      return -1;
    }

    private void switchOutputStreamToFile() {
      if (_content != null && _content instanceof ByteArrayOutputStream
          && ((ByteArrayOutputStream)_content).size() >= MAX_SIZE_BUFFERING) {
        FileOutputStream fos;
        try {
          fos = new FileOutputStream(new File(_fileStreamDirectory +
FILE_STORAGE_NAME));
          ((ByteArrayOutputStream)_content).writeTo(fos);
          _content.close();
          _content = null;
          System.gc();
          _content = fos;
        } catch (FileNotFoundException e) {
          System.err.println("[H264 Steganography container]: Unable to switch content
stream: " +  e.getMessage());
        } catch (IOException e) {
          System.err.println("[H264 Steganography container]: Unable to switch content
stream: " +  e.getMessage());
        }
      }
    }


}



package com.shania.watermark.algorithms.steganography.video;

import java.io.ByteArrayOutputStream;
import java.nio.ByteBuffer;

import com.coremedia.iso.IsoTypeReaderVariable;
import com.googlecode.mp4parser.authoring.Sample;
import com.shania.watermark.dcs.DCSDecode;
import com.shania.watermark.dcs.DCSEncode;

public class H264SteganographyContainerLsb extends H264SteganographyContainer {
```

```
  private final int BYTE_SIZE = 8;

  protected int _nbBitToHideInOneByte;

  private long _maxContentToHide;

  public H264SteganographyContainerLsb() {
    super();
    _nbBitToHideInOneByte = 1;
    _maxContentToHide = -1;
  }

  // Parent methods
  @Override
  public void hideData(byte[] dataToHide) {
    DCSEncode encoder;
    ByteBuffer currentSampleBuffer;
    byte sample[];
    int currentSampleLength;
    int sliceDataOffset;

    if (_sampleList == null || dataToHide == null) {
      return;
    }

    encoder = new DCSEncode(dataToHide, _nbBitToHideInOneByte);
    for (Sample s : _sampleList) {
      currentSampleBuffer = s.asByteBuffer();
      while (currentSampleBuffer.hasRemaining()) {
        currentSampleLength = (int) IsoTypeReaderVariable.read(currentSampleBuffer,
_sampleLengthSize);
        this.addData(new byte[] {0x00, 0x00, 0x01});
        sliceDataOffset =
this.getSliceLayerWithoutPartitioningIdrDataOffset((ByteBuffer)
currentSampleBuffer.slice().limit(currentSampleLength));

        if (sliceDataOffset == -1) {
          this.addData((ByteBuffer)
currentSampleBuffer.slice().limit(currentSampleLength));
          currentSampleBuffer.position(currentSampleBuffer.position() +
currentSampleLength);
          continue;
        }

        this.addData((ByteBuffer) currentSampleBuffer.slice().limit(sliceDataOffset));
        currentSampleBuffer.position(currentSampleBuffer.position() + sliceDataOffset);

        // Sample
        sample = new byte[currentSampleLength - sliceDataOffset];
```

```
            currentSampleBuffer.get(sample);
            encoder.encodeNextFrame(sample);
            sample = insertEscapeSequence(sample);
            this.addData(sample);
            _subSampleIdx++;
          }
          _subSampleIdx = 0;
          _sampleListPosition++;
        }
      }

      @Override
      public void unHideData() {
        DCSDecode decoder;
        ByteBuffer currentSampleBuffer;
        byte[] sample;
        int currentSampleLength;
        int sliceDataOffset;

        if (_sampleList == null) {
          return;
        }
        decoder = new DCSDecode();
        for (Sample s : _sampleList) {
          currentSampleBuffer = s.asByteBuffer();
          while (currentSampleBuffer.hasRemaining()) {
            currentSampleLength = (int) IsoTypeReaderVariable.read(currentSampleBuffer,
_sampleLengthSize);
            sliceDataOffset =
this.getSliceLayerWithoutPartitioningIdrDataOffset((ByteBuffer)
currentSampleBuffer.slice().limit(currentSampleLength));

            if (sliceDataOffset == -1) {
              currentSampleBuffer.position(currentSampleBuffer.position() +
currentSampleLength);
              continue;
            }

            currentSampleBuffer.position(currentSampleBuffer.position() + sliceDataOffset);

            // Sample
            sample = new byte[currentSampleLength - sliceDataOffset];
            currentSampleBuffer.get(sample);
            sample = removeEscapeSequence(sample);
            _unHideData = decoder.decodeFrame(sample);
            if (_unHideData != null) {
              return;
            }
          }
        }
```

```
  }

  @Override
  public long getMaxContentToHide() {
    if (_maxContentToHide == -1) {
      reckonMaxContentToHide();
    }
    return _maxContentToHide;
  }

  // Private methods
  private byte[] insertEscapeSequence(byte sample[]) {
    ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();

    if (sample == null) {
      return byteArrayOutputStream.toByteArray();
    }

    for (int i = 0; i < sample.length; ++i) {
      if (i + 2 < sample.length && sample[i] == 0x00 && sample[i + 1] == 0x00 &&
sample[i + 2] == 0x01) {
        byteArrayOutputStream.write(sample[i]);
        byteArrayOutputStream.write(sample[i + 1]);
        byteArrayOutputStream.write(0x03);
        byteArrayOutputStream.write(sample[i + 2]);
        i += 2;
      } else {
        byteArrayOutputStream.write(sample[i]);
      }
    }
    return byteArrayOutputStream.toByteArray();
  }

  private byte[] removeEscapeSequence(byte sample[]) {
    ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();

    if (sample == null) {
      return byteArrayOutputStream.toByteArray();
    }
    for (int i = 0; i < sample.length; ++i) {
      if (i + 2 < sample.length && sample[i] == 0x00 && sample[i + 1] == 0x00 &&
sample[i + 2] == 0x03) {
        byteArrayOutputStream.write(sample[i]);
        byteArrayOutputStream.write(sample[i + 1]);
        i += 2;
      } else {
        byteArrayOutputStream.write(sample[i]);
      }
    }
    return byteArrayOutputStream.toByteArray();
```

```
  }

  private void reckonMaxContentToHide() {
    ByteBuffer currentSampleBuffer;
    int currentSampleLength;
    int sliceDataOffset;
    long ret = 0;
    float sizeNeededToHideOneByte = (float) Math.ceil((float) BYTE_SIZE /
_nbBitToHideInOneByte);

    if (_sampleList != null) {
      for (Sample s : _sampleList) {
        currentSampleBuffer = s.asByteBuffer();
        while (currentSampleBuffer.hasRemaining()) {
          currentSampleLength = (int)
IsoTypeReaderVariable.read(currentSampleBuffer, _sampleLengthSize);
          sliceDataOffset =
this.getSliceLayerWithoutPartitioningIdrDataOffset((ByteBuffer)
currentSampleBuffer.slice().limit(currentSampleLength));
          currentSampleBuffer.position(currentSampleBuffer.position() +
currentSampleLength);

          if (sliceDataOffset == -1) {
            continue;
          }
          ret += (currentSampleLength - sliceDataOffset);
        }
      }
    }
    ret -= (8 * sizeNeededToHideOneByte);
    ret /= Math.floor(sizeNeededToHideOneByte);
    _maxContentToHide = ret;
  }
}



package com.shania.watermark.algorithms.steganography.video;

public final class H264SteganographyContainerLsb1Bit extends
H264SteganographyContainerLsb{

  private final int BITS_TO_HIDE_IN_ONE_BYTE = 1;

  public H264SteganographyContainerLsb1Bit() {
    super();
    _nbBitToHideInOneByte = BITS_TO_HIDE_IN_ONE_BYTE;
  }
}
```

```
package com.shania.watermark.algorithms.steganography.video;

public final class H264SteganographyContainerLsb2Bits extends
H264SteganographyContainerLsb {

  private final int BITS_TO_HIDE_IN_ONE_BYTE = 2;

  public H264SteganographyContainerLsb2Bits() {
    super();
    _nbBitToHideInOneByte = BITS_TO_HIDE_IN_ONE_BYTE;
  }
}
```

```
package com.shania.watermark.algorithms.steganography.video;

public final class H264SteganographyContainerLsb3Bits extends
H264SteganographyContainerLsb {

  private final int BITS_TO_HIDE_IN_ONE_BYTE = 3;

  public H264SteganographyContainerLsb3Bits() {
    super();
    _nbBitToHideInOneByte = BITS_TO_HIDE_IN_ONE_BYTE;
  }

}
```

```
package com.shania.watermark.algorithms.steganography.video;

public final class H264SteganographyContainerLsb4Bits extends
H264SteganographyContainerLsb {

  private final int BITS_TO_HIDE_IN_ONE_BYTE = 4;

  public H264SteganographyContainerLsb4Bits() {
    super();
    _nbBitToHideInOneByte = BITS_TO_HIDE_IN_ONE_BYTE;
  }
}
```

```
package com.shania.watermark.algorithms.steganography.video;

/**
 * Created by simone on 22/03/18.
 */

import com.coremedia.iso.IsoTypeReaderVariable;
import com.googlecode.mp4parser.authoring.Sample;
import com.shania.watermark.msb.MSBDecode;
import com.shania.watermark.msb.MSBEncode;

import java.io.ByteArrayOutputStream;
import java.nio.ByteBuffer;

public class H264SteganographyContainerMsb extends H264SteganographyContainer {

    private final int BYTE_SIZE = 8;

    protected int _nbBitToHideInOneByte;

    private long _maxContentToHide;

    public H264SteganographyContainerMsb() {
        super();
        _nbBitToHideInOneByte = 1;
        _maxContentToHide = -1;
    }

    // Parent methods
    @Override
    public void hideData(byte[] dataToHide) {
        MSBEncode encoder;
        ByteBuffer currentSampleBuffer;
        byte sample[];
        int currentSampleLength;
        int sliceDataOffset;

        if (_sampleList == null || dataToHide == null) {
            return;
        }

        encoder = new MSBEncode(dataToHide, _nbBitToHideInOneByte);
        for (Sample s : _sampleList) {
            currentSampleBuffer = s.asByteBuffer();
            while (currentSampleBuffer.hasRemaining()) {
                currentSampleLength = (int)
IsoTypeReaderVariable.read(currentSampleBuffer, _sampleLengthSize);
                this.addData(new byte[] {0x00, 0x00, 0x01});
                sliceDataOffset =
this.getSliceLayerWithoutPartitioningIdrDataOffset((ByteBuffer)
```

```
currentSampleBuffer.slice().limit(currentSampleLength));

            if (sliceDataOffset == -1) {
               this.addData((ByteBuffer)
currentSampleBuffer.slice().limit(currentSampleLength));
               currentSampleBuffer.position(currentSampleBuffer.position() +
currentSampleLength);
               continue;
            }

            this.addData((ByteBuffer) currentSampleBuffer.slice().limit(sliceDataOffset));
            currentSampleBuffer.position(currentSampleBuffer.position() +
sliceDataOffset);

            // Sample
            sample = new byte[currentSampleLength - sliceDataOffset];
            currentSampleBuffer.get(sample);
            encoder.encodeNextFrame(sample);
            sample = insertEscapeSequence(sample);
            this.addData(sample);
            _subSampleIdx++;
         }
         _subSampleIdx = 0;
         _sampleListPosition++;
      }
   }

   @Override
   public void unHideData() {
      MSBDecode  decoder;
      ByteBuffer currentSampleBuffer;
      byte[] sample;
      int currentSampleLength;
      int sliceDataOffset;

      if (_sampleList == null) {
         return;
      }
      decoder = new MSBDecode();
      for (Sample s : _sampleList) {
         currentSampleBuffer = s.asByteBuffer();
         while (currentSampleBuffer.hasRemaining()) {
            currentSampleLength = (int)
IsoTypeReaderVariable.read(currentSampleBuffer, _sampleLengthSize);
            sliceDataOffset =
this.getSliceLayerWithoutPartitioningIdrDataOffset((ByteBuffer)
currentSampleBuffer.slice().limit(currentSampleLength));

            if (sliceDataOffset == -1) {
               currentSampleBuffer.position(currentSampleBuffer.position() +
```

```
currentSampleLength);
                continue;
            }

            currentSampleBuffer.position(currentSampleBuffer.position() +
sliceDataOffset);

            // Sample
            sample = new byte[currentSampleLength - sliceDataOffset];
            currentSampleBuffer.get(sample);
            sample = removeEscapeSequence(sample);
            _unHideData = decoder.decodeFrame(sample);
            if (_unHideData != null) {
                return;
            }
        }
    }
}

@Override
public long getMaxContentToHide() {
    if (_maxContentToHide == -1) {
        reckonMaxContentToHide();
    }
    return _maxContentToHide;
}

// Private methods
private byte[] insertEscapeSequence(byte sample[]) {
    ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();

    if (sample == null) {
        return byteArrayOutputStream.toByteArray();
    }

    for (int i = 0; i < sample.length; ++i) {
        if (i + 2 < sample.length && sample[i] == 0x00 && sample[i + 1] == 0x00 &&
sample[i + 2] == 0x01) {
            byteArrayOutputStream.write(sample[i]);
            byteArrayOutputStream.write(sample[i + 1]);
            byteArrayOutputStream.write(0x03);
            byteArrayOutputStream.write(sample[i + 2]);
            i += 2;
        } else {
            byteArrayOutputStream.write(sample[i]);
        }
    }
    return byteArrayOutputStream.toByteArray();
}
```

```
    private byte[] removeEscapeSequence(byte sample[]) {
        ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();

        if (sample == null) {
            return byteArrayOutputStream.toByteArray();
        }
        for (int i = 0; i < sample.length; ++i) {
            if (i + 2 < sample.length && sample[i] == 0x00 && sample[i + 1] == 0x00 &&
sample[i + 2] == 0x03) {
                byteArrayOutputStream.write(sample[i]);
                byteArrayOutputStream.write(sample[i + 1]);
                i += 2;
            } else {
                byteArrayOutputStream.write(sample[i]);
            }
        }
        return byteArrayOutputStream.toByteArray();
    }

    private void reckonMaxContentToHide() {
        ByteBuffer currentSampleBuffer;
        int currentSampleLength;
        int sliceDataOffset;
        long ret = 0;
        float sizeNeededToHideOneByte = (float) Math.ceil((float) BYTE_SIZE /
_nbBitToHideInOneByte);

        if (_sampleList != null) {
            for (Sample s : _sampleList) {
                currentSampleBuffer = s.asByteBuffer();
                while (currentSampleBuffer.hasRemaining()) {
                    currentSampleLength = (int)
IsoTypeReaderVariable.read(currentSampleBuffer, _sampleLengthSize);
                    sliceDataOffset =
this.getSliceLayerWithoutPartitioningIdrDataOffset((ByteBuffer)
currentSampleBuffer.slice().limit(currentSampleLength));
                    currentSampleBuffer.position(currentSampleBuffer.position() +
currentSampleLength);

                    if (sliceDataOffset == -1) {
                        continue;
                    }
                    ret += (currentSampleLength - sliceDataOffset);
                }
            }
        }
        ret -= (8 * sizeNeededToHideOneByte);
        ret /= Math.floor(sizeNeededToHideOneByte);
        _maxContentToHide = ret;
```

```
    }
}
```

```
package com.shania.watermark.algorithms.steganography.video;

/**
 * Created by simone on 22/03/18.
 */

public class H264SteganographyContainerMsb1Bit extends
H264SteganographyContainerMsb{

    private final int BITS_TO_HIDE_IN_ONE_BYTE = 1;

    public H264SteganographyContainerMsb1Bit() {
        super();
        _nbBitToHideInOneByte = BITS_TO_HIDE_IN_ONE_BYTE;
    }

}
```

```
package com.shania.watermark.algorithms.steganography.video;

/**
 * Created by simone on 22/03/18.
 */

public class H264SteganographyContainerMsb2Bits extends
H264SteganographyContainerMsb{

    private final int BITS_TO_HIDE_IN_ONE_BYTE = 2;

    public H264SteganographyContainerMsb2Bits() {
        super();
        _nbBitToHideInOneByte = BITS_TO_HIDE_IN_ONE_BYTE;
    }
}
```

```
package com.shania.watermark.algorithms.steganography.video;

/**
 * Created by simone on 22/03/18.
 */

public class H264SteganographyContainerMsb3Bits extends
H264SteganographyContainerMsb{
```

```
    private final int BITS_TO_HIDE_IN_ONE_BYTE = 3;

    public H264SteganographyContainerMsb3Bits() {
        super();
        _nbBitToHideInOneByte = BITS_TO_HIDE_IN_ONE_BYTE;
    }
}
```

```
package com.shania.watermark.algorithms.steganography.video;

/**
 * Created by simone on 22/03/18.
 */

public class H264SteganographyContainerMsb4Bits extends
H264SteganographyContainerMsb{

    private final int BITS_TO_HIDE_IN_ONE_BYTE = 4;

    public H264SteganographyContainerMsb4Bits() {
        super();
        _nbBitToHideInOneByte = BITS_TO_HIDE_IN_ONE_BYTE;
    }
}
```

```
package com.shania.watermark.algorithms;

import com.shania.watermark.error.ErrorManager;
public class AlgorithmFactory {

   public static ISteganographyContainer
getSteganographyContainerInstanceFromName(String name) {
      ISteganographyContainer algorithm = null;
      Class<?> algorithmClass;
      ErrorManager errorManager = ErrorManager.getInstance();

      try {
         algorithmClass = Class.forName(name);
         algorithm = (ISteganographyContainer) algorithmClass.newInstance();
      } catch (ClassNotFoundException e) {
        errorManager.addErrorMessage("[Algorithm Factory]: Unable to find class " +
name);
      } catch (InstantiationException e) {
        errorManager.addErrorMessage("[Algorithm Factory]: Failed to instantiate class " +
name);
      } catch (IllegalAccessException e) {
        errorManager.addErrorMessage("[Algorithm Factory]: Illegal access to " + name);
```

```
    }
    return (algorithm);
  }

  public static ICryptographyAlgorithm
getCryptographyAlgorithmInstanceFromName(String name) {
    ICryptographyAlgorithm algorithm = null;
    Class<?> algorithmClass;
    ErrorManager errorManager = ErrorManager.getInstance();

    try {
      algorithmClass = Class.forName(name);
      algorithm = (ICryptographyAlgorithm) algorithmClass.newInstance();
    } catch (ClassNotFoundException e) {
      errorManager.addErrorMessage("[Algorithm Factory]: Unable to find class " +
name);
    } catch (InstantiationException e) {
      errorManager.addErrorMessage("[Algorithm Factory]: Failed to instantiate class " +
name);
    } catch (IllegalAccessException e) {
      errorManager.addErrorMessage("[Algorithm Factory]: Illegal access to " + name);
    }
    return (algorithm);
  }
}
```

Dcs

DcsDecode

```
package com.shania.watermark.dcs;

import com.shania.watermark.tools.Utils;

public class DCSDecode {

  private static final int BYTE_SIZE      = 8;
  private static final int INT_SIZE     = BYTE_SIZE * 4;

  private int _to_unhide_byte_length     = 0;
  private int _to_unhide_bit_length  = 0;
  private int _nbBitToDecodeInOneByte    = 1;
  private int _cursor                = 0;
  private int _get_int_cursor          = 0;

  public byte[] _unhide_content     = null;

  private String _intRepr            = "";
```

```
  public byte[] decodeFrame(byte[] frame) {
    if (_cursor > _to_unhide_bit_length)
      return _unhide_content;

    for (int i = 0; i < frame.length; i++) {
      if (_get_int_cursor < INT_SIZE * 2) {
        _intRepr += Utils.getBit(frame[i], 0);
        _get_int_cursor++;
        try {

          if (_get_int_cursor == INT_SIZE) {
            _to_unhide_byte_length = Integer.parseInt(_intRepr, 2);
            _to_unhide_bit_length = _to_unhide_byte_length * BYTE_SIZE;
            if (_to_unhide_byte_length > Utils.MAX_BYTE_TO_HIDE)
              return new byte[0];
            _unhide_content = new byte[_to_unhide_byte_length];
            _intRepr = "";
          } else if (_get_int_cursor == INT_SIZE * 2) {
            _nbBitToDecodeInOneByte = Integer.parseInt(_intRepr, 2);
            if (_nbBitToDecodeInOneByte > 4)
              return new byte[0];
          }

        } catch (NumberFormatException e) {
          return new byte[0];
        }
      } else {
        for (int j = 0; j < _nbBitToDecodeInOneByte; j++) {
          if (_cursor >= _to_unhide_bit_length) {
            return _unhide_content;
          }
          int bitValue = Utils.getBit(frame[i], j);
          _unhide_content = Utils.setBitInByteArray(_unhide_content, bitValue,
_cursor++);
        }
      }
    }

    return null;
  }
}
```

DcsEncode

```
package com.shania.watermark.dcs;

import com.shania.watermark.tools.Utils;
```

```
public class DCSEncode {

  private static final int BYTE_SIZE       = 8;
  private static final int INT_SIZE     = BYTE_SIZE * 4;

  private int _to_hide_byte_length = 0;
  private int _to_hide_bit_length    = 0;
  private int _nbBitToHideInOneByte = 1;
  private int _cursor              = 0;

  private byte[] _content           = null;
  public byte[] _to_hide         = null;


  public DCSEncode(byte[] content, int nbBitToHideInOneByte) {
    _content          = content;
    _to_hide_byte_length  = content.length;
    _nbBitToHideInOneByte  = nbBitToHideInOneByte;

    constructToHide();
  }


  public void constructToHide() {
    byte[] one = Utils.intToByteArray(_to_hide_byte_length);
    byte[] two = Utils.intToByteArray(_nbBitToHideInOneByte);
    byte[] combined = new byte[one.length + two.length];

    System.arraycopy(one,0,combined,0        ,one.length);
    System.arraycopy(two,0,combined,one.length,two.length);

    _to_hide = new byte[combined.length + _content.length];
    System.arraycopy(combined,0,_to_hide,0        ,combined.length);
    System.arraycopy(_content,0,_to_hide,combined.length,_content.length);

    // re compute the length
    _to_hide_byte_length  = _to_hide.length;
    _to_hide_bit_length    = _to_hide_byte_length * BYTE_SIZE;
  }


  public byte[] encodeNextFrame(byte[] frame) {
    if (_cursor > _to_hide_bit_length)
      return frame;

    for (int i = 0; i < frame.length; i++) {
      if (_cursor < INT_SIZE * 2) {
        int bitValue = Utils.getBitInByteArray(_to_hide, _cursor++);
        frame[i] = Utils.setSpecificBit(frame[i], bitValue, 0);
```

```
        } else {
          for (int j = 0; j <
            _nbBitToHideInOneByte; j++) {if
            (_cursor >= _to_hide_bit_length)
              return frame;
            int bitValue = Utils.getBitInByteArray(_to_hide,
            _cursor++);frame[i] = Utils.setSpecificBit(frame[i],
            bitValue, j);
          }
        }
      }
      return frame;
    }
}
```