Password Repository

Prof. Dr. W. Kowalk

Univeristät Oldenburg

kowalk@informatik.uni-oldenburg.de

This report describes a program to store encrypted data like passwords etc.

The program is written in Java and can be opened from the archive

```
PasswordRepository.jar
```

The source code is appended to this document, so anybody who does not trust the author (and why should you?) can inspect and compile the executional code himself.

How to use Password Repository

To use Password Repository the first time see below.

To use Password Repository, start the program and insert your actual password. The program will ask for a file that holds the encrypted data. Then it will display those data in clear text.

Press the *New Password* button to change the password.

Press the *Save* button to encrypt the data and save it in a file, which will be asked for.

Press the *Exit* button to close the program.

How to use Password Repository the first time

To use Password Repository the first time, start the program and press OK with an empty input field of the password mask. You can now give in a *new password* (twice the same). Then an empty text area will be displayed. Data to be stored can be saved by the *Save* button (see above).

Ecnryption Algorithm

Encryption is done by DES, which is relatively weak. So, feel free to use another available encryption algorithm, which can be changed as a parameter of `SecretKeyFactory`.

Encrypted data are stored in a test file coded base64. This means, you get a printable file. Of course, this must not be changed manually, since this would destroy the data.

Extensions

There are no extensions planned, since the program should be as simple as possible. However, you may change what you want, e.g. alter the interactive language or add a more comfortable access technique to the password repository.

Warranty

There is no warranty for

- correct functionality,
- safety against data loss or data decryption,
- legacy of use.

Appendix A

The source code.

```
package password;
/**
 * This program can be used as given.
 * It can be used to encrypt and decrypt data,
 *  e.g. as password repository.
 * Encryption is done by DES, but might be improved
 *  by other algorithms in class DesEncrypter.
 * There is no warranty for
 *  - correct funtion,
 *  - safety against data loss,
 *  - legacy of use.
 * Users can use and change this program ad libitum.
 * (C) Prof. Dr. W. Kowalk,
 *  Uni Oldenburg
 *  kowalk@informatik.uni-oldenburg.de
 */

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.GridLayout;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;

import javax.swing.JButton;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.SwingConstants;

public class MainPassword {

  /**
   * This method initializes jFrameAll
   *     jFrameAll is the Frame that contains the interface
   * @return javax.swing.JFrame
   */
  private JFrame getJFrameAll() {
    if (jFrameAll == null) {
      jFrameAll = new JFrame();
      jFrameAll.setSize(new Dimension(657, 289));
      jFrameAll.setContentPane(getJContentPaneALL());
      jFrameAll.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
      jFrameAll.setContentPane(getJContentPaneALL());
    }
    return jFrameAll;
  }

  /**
   * This method initializes jContentPaneALL
   *     is the Pane that contains the interface
   * It is added to JFrameAll
   * @return javax.swing.JPanel
   */
  private JPanel getJContentPaneALL() {
    if (jContentPaneALL == null) {
      jContentPaneALL = new JPanel();
      jContentPaneALL.setLayout(new BorderLayout());
      jContentPaneALL.add(getJScrollPaneAll(), BorderLayout.CENTER);
      jContentPaneALL.add(getJPanelButtons(), BorderLayout.SOUTH);
      jContentPaneALL.add(getJPanelLabe(), BorderLayout.NORTH);
    }
    return jContentPaneALL;
  }

  /**
   * This method initializes jScrollPaneAll
   *      To scroll the text
   * @return javax.swing.JScrollPane
   */
  private JScrollPane getJScrollPaneAll() {
    if (jScrollPaneAll == null) {
      jScrollPaneAll = new JScrollPane();
      jScrollPaneAll.setViewportView(getJTextAreaAll());
    }
    return jScrollPaneAll;
  }

  private boolean textHasBeenChanged = false;

  /**
   * This method initializes jTextAreaAll
   *      to hold the text to be displayed
   * @return javax.swing.JTextArea
   */
  private JTextArea getJTextAreaAll() {
    if (jTextAreaAll == null) {
      jTextAreaAll = new JTextArea();
      jTextAreaAll.addKeyListener(new java.awt.event.KeyAdapter() {
        public void keyTyped(java.awt.event.KeyEvent e) {
          textHasBeenChanged = true;
        }
      });
    }
    return jTextAreaAll;
  }

  /**
   * This method initializes getJPanelButtons
   *      to hold the buttons
   * @return javax.swing.JPanel
   */
  private JPanel getJPanelButtons() {
    if (jPanelAll == null) {
      GridLayout gridLayout = new GridLayout();
      gridLayout.setRows(1);
      jPanelAll = new JPanel();
      jPanelAll.setLayout(gridLayout);
      jPanelAll.add(getJButtonSave(), null);
      jPanelAll.add(getJButtonPhrase(), null);
      jPanelAll.add(getJButtonExit(), null);
    }
    return jPanelAll;
  }

  /**
   * This method initializes jButtonSave
```

```
     *      this button is to save the data after encrytpion
     * @return javax.swing.JButton
     */
    private JButton getJButtonSave() {
      if (jButtonSave == null) {
        jButtonSave = new JButton();
        jButtonSave.setText("Save");
        jButtonSave.setToolTipText("Encrypt actual data and save in file");
        jButtonSave.addActionListener(new java.awt.event.ActionListener() {
          public void actionPerformed(java.awt.event.ActionEvent e) {
            closeData();
          }
        });
      }
      return jButtonSave;
    }

    /**
     * This method initializes jPanelLabe
     *      the panel that holds the "Password Repository" Titel
     * @return javax.swing.JPanel
     */
    private JPanel getJPanelLabe() {
      if (jPanelLabe == null) {
        GridLayout gridLayout1 = new GridLayout();
        gridLayout1.setRows(1);
        gridLayout1.setColumns(1);
        jLabelTitel = new JLabel();
        jLabelTitel.setText("Password Repository");
        jLabelTitel.setFont(new Font("Dialog", Font.BOLD, 24));
        jLabelTitel.setHorizontalAlignment(SwingConstants.CENTER);
        jPanelLabe = new JPanel();
        jPanelLabe.setLayout(gridLayout1);
        jPanelLabe.add(jLabelTitel, null);
      }
      return jPanelLabe;
    }

    /**
     * This method initializes jButtonExit
     *      the button to exit the program
     * @return javax.swing.JButton
     */
    private JButton getJButtonExit() {
      if (jButtonExit == null) {
        jButtonExit = new JButton();
        jButtonExit.setText("Exit");
        jButtonExit.setToolTipText("Exit Program! Save before exit!");
        jButtonExit.addActionListener(new java.awt.event.ActionListener() {
          public void actionPerformed(java.awt.event.ActionEvent e) {
            if(!textHasBeenChanged) System.exit(0);
            int n = JOptionPane.showConfirmDialog(
                jFrameAll, "Data has been altered!\nSave data before exit?",
"Confirm!",
                JOptionPane.YES_NO_CANCEL_OPTION);
            if(n==JOptionPane.YES_OPTION)
              if(closeData()) System.exit(0);
            if(n==JOptionPane.NO_OPTION)
              System.exit(0);
          }
        });
      }
      return jButtonExit;
    }
```

```
  /**
   * This method initializes jButtonPhrase
   *      to alter the pass phrase
   * @return javax.swing.JButton
   */
  private JButton getJButtonPhrase() {
    if (jButtonPhrase == null) {
      jButtonPhrase = new JButton();
      jButtonPhrase.setText("New Passphrase");
      jButtonPhrase.setToolTipText("Change Pass Phrase. All data to be saved
will be encryted by the new Pass Phrase!");
      jButtonPhrase.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent e) {
          setNewPassPhraseAndEncrypter("Change Password!\n");
        }
      });
    }
    return jButtonPhrase;
  }

  public static void main(String[] x) {
//    System.out.println("Start: Password Repository");
    new MainPassword();
  }
  private DesEncrypter encrypter;  //  @jve:decl-index=0:
  MainPassword() {
    System.out.println("Create: Password Repository");
    String passPhrase = getPassPhrase("Insert actual password");
    String decrypted;
    if(passPhrase!=null && passPhrase.length()>0) {
      String s = loadData();
      // Create encrypter/decrypter class
      encrypter = new DesEncrypter(passPhrase);
      if(!encrypter.setupOkay) {
        JOptionPane.showMessageDialog(
            null, "Error in set up decrypter!\nI give up!",
            "Decryption error!", JOptionPane.ERROR_MESSAGE);
        System.exit(0);
      }
      // Decrypt
      decrypted = encrypter.decrypt(s);
      if(!encrypter.decryptOkay) {
        JOptionPane.showMessageDialog(
            null, "Error in decryption!\nI give up!",
            "Decryption error!", JOptionPane.ERROR_MESSAGE);
        System.exit(0);
      }
    } else {
      decrypted = "New Data";
      setNewPassPhraseAndEncrypter("Set up new Password Repository!\n");
      if(encrypter==null) return;
      if(!encrypter.setupOkay) {
        JOptionPane.showMessageDialog(
            null, "Error in set up decrypter!\nI give up!",
            "Decryption error!", JOptionPane.ERROR_MESSAGE);
        System.exit(0);
      }
    }
    // Now Display data and wait for another event
    displayData(decrypted);
  }
  /**
   * Method loadData
   * @return loaded Data as a string
   */
  String loadData() {
    try {
      File selectedFile = inputDialog();
      BufferedReader in = new BufferedReader(new FileReader(
          selectedFile));
      String s = "";
      char cbuf[] = new char[1024];
      int n;
      while((n=in.read(cbuf))>0) // read chunks and append to string s
        s += new String(cbuf,0,n);
      in.close();
      textHasBeenChanged = false; // text isn't changed
      return s;
    } catch (Exception ee) {
      System.out.println(
          "Error reading file: Error Message = "
          + ee.getMessage());}
    return "";
  }
  /**
   * Method saveDate
   * save Data String s. Data should be encrypted!
   * @param s the String of data to be saved in a file
   */
  boolean saveDate(String s) {
    int returnVal = fc.showSaveDialog(jFrameAll);
    File selectedFile = null;
    if (returnVal == JFileChooser.APPROVE_OPTION) {
      selectedFile = fc.getSelectedFile();
      if(selectedFile.exists()) { // don't overwrite, if not appropriate
        int n = JOptionPane.showConfirmDialog(
            jFrameAll, "File "+selectedFile.getName()+ " exits!\nOverwrite?",
"Confirm!",
            JOptionPane.YES_NO_OPTION);
        if(n==JOptionPane.NO_OPTION) return false;
      }
      try {
        BufferedWriter out = new BufferedWriter(new FileWriter(selectedFile));
        out.write(s);
        out.close();
        return true;
      } catch (Exception ex) {
        System.err.println("Couldn't write to "
            + selectedFile.getAbsolutePath() + "\nError Message = "
            + ex.getMessage());
        return false;
      }
    }
    return false;
  }
  /**
   * Method getPassPhrase
   * reads a pass phrase from input mask
   * @return the pass phrase String given by the user
   */
  String getPassPhrase(String s) {
    String p = JOptionPane.showInputDialog(
        null, s, "Password Input Dialog", JOptionPane.OK_OPTION);
    return p;
  }
  /**
   * Method setNewPassPhraseAndEncrypter
```

```
    * sets a new passphrase by input dialog.
    * Also generates a new encrypter with that new pass phrase!
    * @param s
    */
   void setNewPassPhraseAndEncrypter(String s) {
     String passPhrase = getPassPhrase(s+"Insert new password!");
     if(passPhrase==null) return;
     String passPhrase2 = getPassPhrase("Repeat new password!");
     if(passPhrase!=null && passPhrase2!=null
         && passPhrase.equals(passPhrase2))
      // Create encrypter/decrypter class
       encrypter = new DesEncrypter(passPhrase);
     else
       JOptionPane.showMessageDialog(jFrameAll, "pass phrases differ!\nReenter
 again ", "Pass Phrase Dialog", JOptionPane.ERROR_MESSAGE);
     return;
   }
   /**
    * Method displayData
    * displays data in Text Area
    * @param s is displayed in a textArea
    */
   void displayData(String s) {
     getJFrameAll();
     jFrameAll.setVisible(true);
     jTextAreaAll.setText(s);
   }
   /**
    * Method closeDate
    * encrypts the data by the current pass phrase and stores encrypted
    * data into a given file
    */
   boolean closeData() {
     String encrypted = encrypter.encrypt(jTextAreaAll.getText());
     return saveDate(encrypted);
   }
   JFileChooser fc = new JFileChooser(
   "C:\\Dokumente und Einstellungen\\Administrator\\Eigene
 Dateien\\Bilder\\passwd.txt");  //
   private JFrame jFrameAll = null;  //
   private JPanel jContentPaneALL = null;
   private JScrollPane jScrollPaneAll = null;
   private JTextArea jTextAreaAll = null;
   private JPanel jPanelAll = null;
   private JButton jButtonSave = null;
   private JPanel jPanelLabe = null;
   private JLabel jLabelTitel = null;
   private JButton jButtonExit = null;
   private JButton jButtonPhrase = null;

   /**
    * Method inputDialog
    * @return the File for data input
    */
   File inputDialog() {
     fc.setSelectedFile(new File("passwd.txt"));
     int returnVal = fc.showOpenDialog(jFrameAll);
     if (returnVal == JFileChooser.APPROVE_OPTION) {
       return fc.getSelectedFile();
     }
     return null;
   }

 }
```

```
////////////////////////////////////////
///*********************************//
package password;

import java.io.UnsupportedEncodingException;
import java.security.spec.AlgorithmParameterSpec;
import java.security.spec.KeySpec;

import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.SecretKey;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.PBEKeySpec;
import javax.crypto.spec.PBEParameterSpec;

public class DesEncrypter {
  Cipher ecipher;
  Cipher dcipher;

  // 8-byte Salt
  byte[] salt = {
      (byte)0xA9, (byte)0x9B, (byte)0xC8, (byte)0x32,
      (byte)0x56, (byte)0x35, (byte)0xE3, (byte)0x03
  };

  // Iteration count
  int iterationCount = 19;

  DesEncrypter(String passPhrase) {
    setupOkay = true;
    try {
      // Create the key
      KeySpec keySpec = new PBEKeySpec(passPhrase.toCharArray(), salt,
          iterationCount);
      SecretKey key = SecretKeyFactory.getInstance("PBEWithMD5AndDES")
          .generateSecret(keySpec);
      ecipher = Cipher.getInstance(key.getAlgorithm());
      dcipher = Cipher.getInstance(key.getAlgorithm());

      // Prepare the parameter to the ciphers
      AlgorithmParameterSpec paramSpec = new PBEParameterSpec(salt,
          iterationCount);

      // Create the ciphers
      ecipher.init(Cipher.ENCRYPT_MODE, key, paramSpec);
      dcipher.init(Cipher.DECRYPT_MODE, key, paramSpec);
      setupOkay = true;
    } catch (java.security.InvalidAlgorithmParameterException e) {
    } catch (java.security.spec.InvalidKeySpecException e) {
    } catch (javax.crypto.NoSuchPaddingException e) {
    } catch (java.security.NoSuchAlgorithmException e) {
    } catch (java.security.InvalidKeyException e) {
    }
  }

  public boolean setupOkay = true;
  public boolean encryptOkay = true;
  public boolean decryptOkay = true;

  public String encrypt(String str) {
    encryptOkay = false;
    try {
      // Encode the string into bytes using utf-8
      byte[] utf8 = str.getBytes("UTF8");
      // Encrypt
      byte[] enc = ecipher.doFinal(utf8);

      // Encode bytes to base64 to get a string
      String s = new sun.misc.BASE64Encoder().encode(enc);
      encryptOkay = true;
      return s;
    } catch (javax.crypto.BadPaddingException e) {
      return "Encrypt: BadPaddingException: Error Message = " + e.getMessage();
    } catch (IllegalBlockSizeException e) {
      return "Encrypt: IllegalBlockSizeException: Error Message = "
          + e.getMessage();
    } catch (UnsupportedEncodingException e) {
      return "Encrypt: UnsupportedEncodingException: Error Message = "
          + e.getMessage();
    } catch (java.io.IOException e) {
      return "Encrypt: IOException: Error Message = " + e.getMessage();
    }
  }

  public String decrypt(String str) {
    decryptOkay = false;
    try {
      // Decode base64 to get bytes
      byte[] dec = new sun.misc.BASE64Decoder().decodeBuffer(str);

      // Decrypt
      byte[] utf8 = dcipher.doFinal(dec);

      // Decode using utf-8
      String s = new String(utf8, "UTF8");
      decryptOkay = true;
      return s;
    } catch (javax.crypto.BadPaddingException e) {
      return "Decrypt: BadPaddingException: Error Message = " + e.getMessage();
    } catch (IllegalBlockSizeException e) {
      return "Decrypt: IllegalBlockSizeException: Error Message = "
          + e.getMessage();
    } catch (UnsupportedEncodingException e) {
      return "Decrypt: UnsupportedEncodingException: Error Message = "
          + e.getMessage();
    } catch (java.io.IOException e) {
      return "Decrypt: IOException: Error Message = " + e.getMessage();
    }
  }
}
```