

Практическая криптография в Java. Защита электронной почты

Николай Жишкевич

1. Введение

Данная статья продолжает серию посвященную средствам и методам защиты информации в интернет и интранет, а также рассматривает их практические приемы реализации с помощью Java. Сегодня очередь дошла до средств электронной переписки. Не секрет, что та информация, которой обмениваются организации или подразделения этих организаций, является по определению секретной. К сожалению, так получилось, что электронная почта в начале своего пути не предоставляла никаких действительно серьезных средств по обеспечению конфиденциальности передаваемой информации. Протоколы POP3 и SMTP являются не безопасными т.к. обмен информацией идет в открытом виде (вы ведь не считаете base64 за шифрование). В качестве последней демонстрации я привожу результат соединения с почтовым сервером.

```
Telnet localhost
+OK Avirt Mail POP3 Server Ready
user vasya
+OK
pass vasya
+OK
stat
+OK 4 8758
retr 1
+OK 3001 octets
Received: from 127.0.0.1 <127.0.0.1>
        by comp via Avirt Mail smtp <4.0>
for <vasya>; Fri, 11 Apr 2003 18:05:55 GMT
Received: from 127.0.0.1 <127.0.0.1> by comp via Avirt Mail smtp <4.0>; Fri, 11
Apr 2003 17:58:11 GMT
Received: from 127.0.0.1 <127.0.0.1> by comp via Avirt Mail smtp <4.0>; Fri, 11
Apr 2003 21:11:25 GMT
Received: from 127.0.0.1 <127.0.0.1> by comp via Avirt Mail smtp <4.0>; Thu, 10
Apr 2003 21:11:25 GMT
Received: from 127.0.0.1 <127.0.0.1> by comp via Avirt Mail smtp <4.0>; Thu, 10
Apr 2003 20:59:56 GMT
Received: from 127.0.0.1 <127.0.0.1> by comp via Avirt Mail smtp <4.0>; Thu, 10
Apr 2003 20:59:56 GMT
Received: from 127.0.0.1 <127.0.0.1> by comp via Avirt Mail smtp <4.0>; Thu, 10
Apr 2003 20:48:45 GMT
Received: from 127.0.0.1 <127.0.0.1> by comp via Avirt Mail smtp <4.0>; Thu, 10
Apr 2003 20:48:45 GMT
Received: from 127.0.0.1 <127.0.0.1> by comp via Avirt Mail smtp <4.0>; Thu, 10
Apr 2003 20:21:27 GMT
From: "kolya" <kolya@comp.con>
To: <vasya@comp.con.by>
Subject: Baaaaa
Date: Thu, 10 Apr 2003 23:21:27 +0300
Message-ID: <000a01c2ff9e$c3e92f70$0100007f@comp>
MIME-Version: 1.0
Content-Type: multipart/alternative;
        boundary="-----_NextPart_000_000B_01C2FFB7.E9366770"
X-Priority: 3 (Normal)
```

Здесь я воспользовался telnet и подключился к локальному почтовому серверу avirtmail на котором и провожу большую часть экспериментов. Если внимательно присмотреться к логу сообщений на экране, то вы увидите не только текст письма, и его служебные заголовки, но и пароль который я отправил серверу. Хотя, если говорить честно, то возможен другой способ аутентификации с почтовым клиентом, когда отправляется не пароль в открытом виде, а результат его преобразования с помощью хеш-функций (что-то похожее мы делали в начальных статьях серии), но сути дела это не очень изменяет. Почту надо защищать. Сейчас я кратко рассмотрю наличные решения, а потом посмотрим нельзя ли что-то еще усовершенствовать.

Замечание:

Для того чтобы подключиться к почтовому серверу и попробовать себя в роли его клиента достаточно набрать в командной строке "telnet your.mailserver.com 110" — для соединения с POP3 службой или "telnet

your.mailserver.com 25" — для соединения с SMTP-сервером. Ах, да и разумеется вам совсем неплохо было бы знать пару команд протокола smtp или pop3, например user, pass, stat, retr. За полным списком коих я отправляю вас к соответствующему RFC.

Итак, в чистом виде почтовые протоколы беззащитны, к счастью, в ряд приложений для работы с электронной почтой, такие как outlook или TheBat! встроены средства для работы с шифрованием и создания ЭЦП пересылаемых писем. В случае использования решений от microsoft вам следует получить сертификат на одном из партнерских сайтов (Microsoft настоятельно рекомендует VeriSign), а затем установив данный сертификат в хранилище можно пользоваться вышеперечисленными благами. Если же вы поклонник TheBat!, то тут возможностей еще больше, т.к. TheBat! поддерживает OpenPGP. В общем случае схема использования S\MIME такова: вы и ваш собеседник должны получить сертификаты от некоторой доверенной организации, в крайнем случае в TheBat! можно создать сертификат и секретный ключ самостоятельно, или вы можете воспользоваться keytool с которым мы познакомились ранее. Правда в этом случае возникает вопрос доверия вашего собеседника к полученному им сертификату, ведь злоумышленники не спят и постараются перехватить сертификат который вы пошлете вашему собеседнику, а если созданный вами сертификат не будет иметь подписи известного CA, то его цена будет достаточно низка. Конечно, позволить себе получение сертификата от VeriSign или иного CA достаточно сложно для частных лиц, но я, например, потратил 5 минут времени и ради пробы в свое время получил пробный сертификат и пока срок не кончился некоторое время переписывался секретно. Пример подобного сертификата будет приведен ниже. Многие современные почтовые сервера и соответственно почтовые клиенты имеют дополнительную возможность устанавливая соединения по протоколу SSL или TLS обеспечивая надежность передаваемой информации и ее защиту от несанкционированного прочтения. Хотя надо сказать, что в сети интернет есть достаточно большое количество решений построения защищенной почты, через специальные программные продукты или даже без использования оных через веб-интерфейс браузера.

2. Пример приложения для отправки почты

Далее я хочу сосредоточиться на том, чтобы рассмотреть как именно можно реализовать функции работы с защищенной почтой в Java. Вообще то говоря, есть разные пути как получать или отправлять email, отличающиеся затратами времени на

реализацию и уровнем детализации. Можно пойти по пути создания очередного собственного велосипеда и реализовать работу по протоколу SMTP или POP3 путем отправки и приема текстовых команд, т.е. это самый низкий уровень. Можно воспользоваться решениями сторонних разработчиков, например, уважением пользуется пакет `ipworks`, правда за него надо платить денежки, а то в демонстрационном режиме постоянно выводятся сообщения с просьбой зарегистрироваться и т.д. Конечно, вывод в поток `stdout` или `stderr` лицензионных сообщений не порождает больших неудобств, но все таки. Есть и третий путь, основанный на использовании `JavaMail`. Далее я привожу простой пример работы данного API. В данном фрагменте кода я создаю класс почтового клиента, который читает параметры работы из файла XML в котором содержится информация о заданиях, в каждом задании содержится информация о том по какому адресу следует отправить письмо, какая тема и текст сообщения, а также есть возможность добавлять к письму вложения. В ходе работы клиент выводит на экран журнал событий, для проверки хода работы и исправления ошибок. Данный клиент вполне может быть использован как самостоятельное приложение, хотя и без графического интерфейса пользователя, но например для автоматизированной рассылки вполне применима. В данной статье я не ставлю перед собой задачу детального изложения особенностей работы с `JavaMail`, по этой теме есть соответствующие источники. Итак, вот исходный код примера почтового клиента.

```
package com.javable.www.columns.security;

import java.io.*;
import java.util.*;
import javax.activation.*;
import javax.mail.*;
import javax.mail.internet.*;

import org.xml.sax.*;

public class AbstractMailAgent {
    protected org.w3c.dom.Element elt_root;
    // корень дерева файла конфигурации почтовых заданий

    public AbstractMailAgent(String fName) throws IOException, SAXException {
        org.apache.xerces.parsers.DOMParser dp = new org.apache.xerces.parsers.
```

Практическая криптография в Java. Защита электронной почты

```
DOMParser();
dp.parse(new org.xml.sax.InputSource(new java.io.FileInputStream(
    fName)));

elt_root = dp.getDocument().getDocumentElement();
System.out.println(
    "Успешно загружен файл конфигурации, к выполнению заданий готов");
}

/**
 * функция выполняет непосредственно рассылку писем согласно плану заданий
 * @throws IOException – если чтонибудь случится
 * @throws MessagingException – если чтонибудь случится
 */

void startMailAgent() throws IOException, MessagingException {
    System.out.println("Приступаю к выполнению заданий");
    int i;
    for (i = 0; ; i++) {
        // получаем информацию о том кому шлем письмо,
        // если выполнить это не удалось то прекращаем рассылку
        // конечно в идеале следует делать подобные проверки для всех полей,
        // но это ведь только учебный пример-шаблон,
        // а не готовый коммерческий продукт
        String to = XTools.getXPathText("/Mail-projects/Mail-project[" + i +
            "]/To", elt_root);

        if (to == null) {
            break;
        }
        initSocketSubSystem (); // настройка транспортной системы
        String from = XTools.getXPathText("/Mail-projects/Mail-project[" + i +
            "]/From", elt_root);

        System.out.println("Чтение конфигурации для письма from: \"" + from +
            "\"" to "\"" + to + "\"");
        Vector vcc = new Vector();
        for (int j = 0; ; j++) {

            String cc = XTools.getXPathText(
                "/Mail-projects/Mail-project/CC-List/CC-entry[" + j + "]", elt_root);
```

```
    if (cc == null) {
        break;
    }
    vcc.add(cc);
}

Vector vbcc = new Vector();
for (int j = 0; ; j++) {

    String bcc = XTools.getXPathText(
        "/Mail-projects/Mail-project/BCC-List/BCC-entry[" + j + "]",
        elt_root);
    if (bcc == null) {
        break;
    }
    vbcc.add(bcc);
}

Properties props = System.getProperties();
props.put("mail.smtp.host",
    XTools.getXPathText("/Mail-projects/Mail-project[" + i
        + "]/Smtp-server-name", elt_root));
props.put("mail.smtp.port",
    XTools.getXPathText("/Mail-projects/Mail-project[" + i
        + "]/Smtp-server-port", elt_root));
// набор дополнительных шагов по подготовке свойств соединения
props = initProperties (props , XTools.getElementByXPath(
    "/Mail-projects/Mail-project[" + i
        + "]" , elt_root) );
Session session = Session.getInstance(props, null);
MimeMessage message = new MimeMessage(session);

message.setFrom(new InternetAddress(from));
message.setRecipient(MimeMessage.RecipientType.TO, new InternetAddress(to));
for (int j = 0; j < vcc.size(); j++) {
    message.setRecipient(MimeMessage.RecipientType.CC,
        new InternetAddress( (String) vcc.get(j)));
}
for (int j = 0; j < vbcc.size(); j++) {
    message.setRecipient(MimeMessage.RecipientType.BCC,
```

Практическая криптография в Java. Защита электронной почты

```
        new InternetAddress( (String) vbcc.get(j)));
    }
    message.setSubject(XTools.getXPathText(
        "/Mail-projects/Mail-project[" + i + "t", elt_root));

    Multipart mp = new MimeMultipart();
    MimeBodyPart txt = new MimeBodyPart();
    txt.setText(XTools.getXPathText(
        "/Mail-projects/Mail-project[" + i + "]/Message", elt_root));
    System.out.println(подготовлен текст письма,
        подготовка к добавлению приложений");
    mp.addBodyPart(txt);
    for (int j = 0; ; j++) {
        String fName = XTools.getXPathText(
            "/Mail-projects/Mail-project/Attachments/Attachment[" + j + "]",
            elt_root);
        if (fName == null) {
            break;
        }
        MimeBodyPart fileAtt = new MimeBodyPart();
        DataHandler fdf = new DataHandler(new FileDataSource(fName));
        fileAtt.setDataHandler(fdf);
        fileAtt.setFileName(fName);
        // внимание именно здесь кроется возможность дальнейшего расширения
        // приложения, за счет выполнения дополнительной обработки
        // частей сообщения перед непосредственно отправкой
        mp.addBodyPart(
            processPart (fileAtt ,
                XTools.getElementByXPath("/Mail-projects/Mail-project["+i+"]" ,
                    elt_root) ));
        System.out.println("Успешно добавлено вложение к письму, файл: " +
            fName);
    }
    message.setContent(mp);
    System.out.println(
        "Сформированное задание подано на вход средству доставки");
    Transport.send(message);
}
```

```
        System.out.println("Рассылка писем завершена: найдено записей " + i);
    }

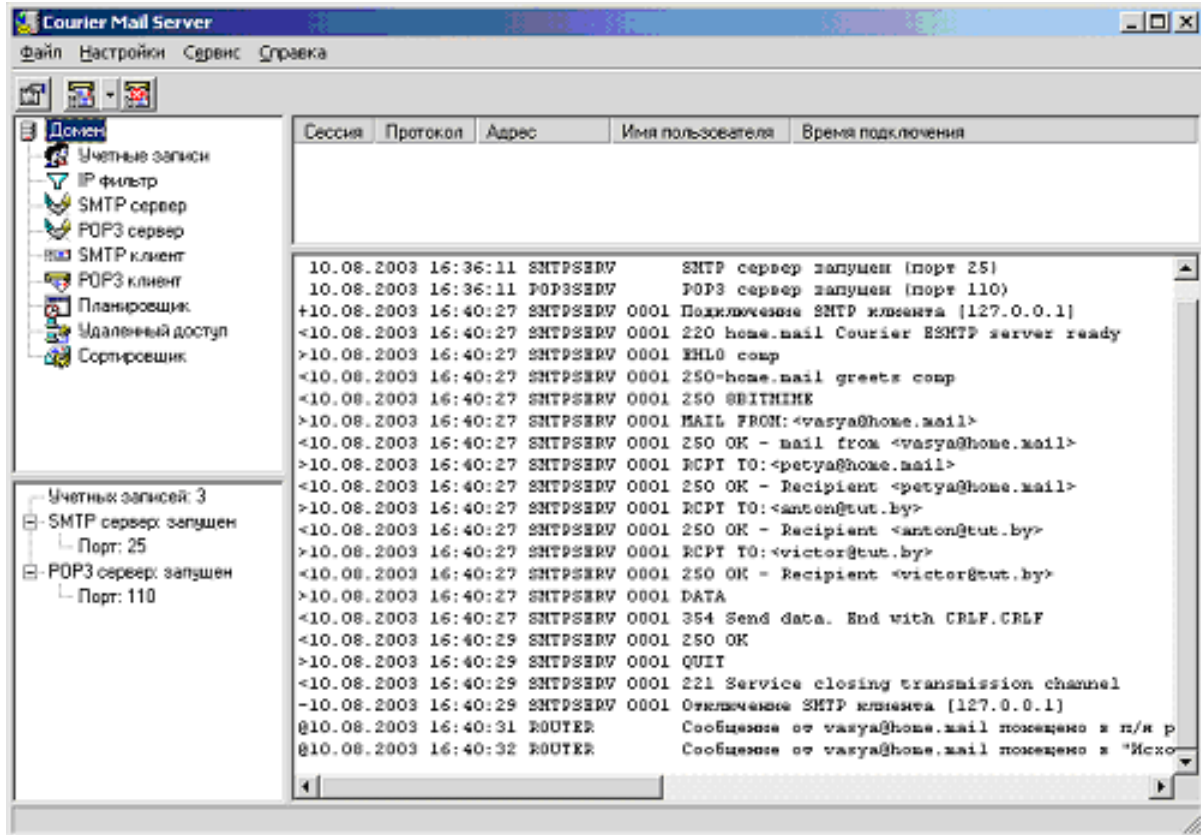
    /**
     * Внимание, именно в данной функции скрывается возможность
     * дальнейшего расширения и поддержки работы
     * с шифрованием данных или их ЭЦП, на
     * данный момент никакой обработки не выполняется, но в перспективе...
     * @param part – часть сообщения которая должна быть обработана
     * @param confElt – элемент дерева xml (файл конфигурации) в котором
     * должны содержаться более подробные сведения о том
     * что необходимо делать с данной частью сообщения
     * @return
     */
    protected MimeBodyPart processPart
    (MimeBodyPart part , org.w3c.dom.Element confElt ){
        return part;
    }
    /**
     * По аналогии с предыдущим методом данный код служит
     * только для целей дальнейшего расширения и предназначен для выполнения
     * специальных действий связанных с настройкой фабрик сокетов
     * для сетевых соединений устанавливаемых javamail api
     */
    protected void initSocketSubSystem (){
    }
    /**
     * По аналогии с предыдущим методом данная функция дает возможность
     * дальнейшего расширения возможностей почтового агента
     * путем настройки свойств
     * для соединения
     */
    protected Properties initProperties
    (Properties props , org.w3c.dom.Element conf){
        return props;
    }
}
```

Практическая криптография в Java. Защита электронной почты

Для работы приложению необходим специальный файл настроек. Который выглядит примерно так. Для анализа файла настроек используется вспомогательный класс XTools. Пример его был приведен ранее, за исходными кодами примера обращайтесь к предыдущим статьям серии.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v5 rel. 4 U (http://www.xmlspy.com) by Nikolai (Comp) -->
<Mail-projects>
  <Mail-project>
    <Smtп-server-name>comp</Smtп-server-name>
    <Smtп-server-port>25</Smtп-server-port>
    <From>vasya@home.mail</From>
    <To>petya@home.mail</To>
    <CC-List>
      <CC-entry>anton@tut.by</CC-entry>
    </CC-List>
    <BCC-List>
      <BCC-entry>victor@tut.by</BCC-entry>
    </BCC-List>
    <Subject>Hello From Java Mail</Subject>
    <Message>Hello, how are you</Message>
    <Attachments>
      <Attachment>H:\Docs_Books\mars\artifact.txt</Attachment>
      <Attachment>H:\Docs_Books\mars\bzg.txt</Attachment>
    </Attachments>
  </Mail-project>
</Mail-projects>
```

Для эксперимента я воспользовался почтовым сервером "Courier Mail Server", получить который можно с сайта courierms.narod.ru. Кстати, я настоятельно рекомендую данный продукт, он, конечно, уступает по возможностям почтовым серверам, вроде avirtmail (второго моего любимого подопытного кролика). Но для новичка он имеет преимущество в виде простого интерфейса несложного в настройке и использовании. Ниже я привожу пример экрана, на котором виден лог действий по приему писем.



3. Защищаем почту. Шифрование

Итак, подведем итог. Мы смогли отправить почту, однако, на данный момент никаких специальных действий по защите передаваемой информации мы не предпринимали. И сами письма и канал по которому информация передавалась на сервер ни коим образом не были защищены. Давайте исправим это. И начнем пожалуй с того что зашифруем содержимое письма при пересылке, каждое вложение в письмо будет защищено с помощью надежного симметричного алгоритма. Разумеется что на роль кандидата попал алгоритм blowfish, не только по тому что я рассказывал о нем в своей первой статье и сейчас мечтаю продолжить повествование о нем, но потому что он надежен и, что крайне важно, существует реализация данного алгоритма с помощью класса Cipher, и разумеется услуг определенных криптографических провайдеров. Если у вас JDK1.4 то по умолчанию все необходимое средства будут уже

Практическая криптография в Java. Защита электронной почты

интегрированы в него. Сначала я приведу пример исходного кода приложения, а потом и краткий разбор кода.

```
package com.javable.www.columns.security;

import javax.mail.*;
import javax.activation.*;
import javax.mail.internet.*;
import java.util.*;
import java.io.*;
import java.net.*;
import java.security.*;
import javax.net.*;
import javax.crypto.*;
import javax.crypto.spec.*;
import org.xml.sax.*;

public class CriptoMailAgent
    extends AbstractMailAgent {
    public CriptoMailAgent(String confName) throws
        IOException, SAXException {
        super(confName);
    }

    /**
     * А вот теперь как я и обещал я выполняю шифрование
     * данной части сообщения, в качестве средства
     * я возьму достаточно известный криптоалгоритм blowfish,
     * реализацию которого я приводил в предыдущей
     * статье, правда на этот раз я не буду пользоваться
     * собственной реализацией, а возьму стандартные средства java crypto api
     * @param part – часть сообщения для преобразования
     * @param confElt – элемент дерева xml соответствующий данному письму
     * @return – результат преобразования
     */
    protected MimeBodyPart processPart(MimeBodyPart part,
        org.w3c.dom.Element confElt) {
        try {
            String fName = part.getFileName();
            File ff = File.createTempFile("tmp_", "_mail.crypto");
```

```
ff.deleteOnExit();
// временный файл по завершению работы приложения должен быть уничтожен
MimeBodyPart npart = new MimeBodyPart();

System.out.println("Start Look for algName");
String algName = XTools.getXPathText("Mail-project/Algorithm", confElt);
System.out.println("Start Look for keyValue");
String keyValue = XTools.getXPathText("Mail-project/Secret-key", confElt);
if (algName == null || keyValue == null) {
    return part;
}
// не выполнять никакого преобразования раз не указан ни алгоритм ни ключ
Cipher ci = Cipher.getInstance(algName);

SecretKeySpec key = new SecretKeySpec(keyValue.getBytes("utf-16"),
                                       "blowfish");

ci.init(Cipher.ENCRYPT_MODE, key);
int flen = (int)new File(fName).length();
byte[] data = new byte[flen];
// честно несколько криво, однако я не думаю
// что вы будете пересылать файлы размером
// более чем несколько мегабайт
// в данном странном фрагменте кода я пытаюсь
// избежать необходимости выполнять
// чтение и шифрование данных блоками, дело в том,
// что в общем случае размер
// блока может варьироваться
// например в blowfish - 8 байт, а в алгоритме XYZ - t байт,
// более универсальным решением было бы
// решить одновременно и проблему некратности размера документа
// для шифрования и универсальности используемого алгоритма
// за счет одноразового преобразования материала
int rsize = 0;
byte[] buf = new byte[10000];

FileInputStream fin = new FileInputStream(fName);
FileOutputStream fout = new FileOutputStream(ff);
int posD = 0;
while ( (rsize = fin.read(data)) > 0) {
    System.arraycopy(buf, 0, data, posD, rsize);
```

Практическая криптография в Java. Защита электронной почты

```
        posD += rsize;
    }
    fout.write( (ci.doFinal(data)));
    fout.close();
    fin.close();
    npart.setFileName(ff.getAbsolutePath());
    npart.setDataHandler(new DataHandler(new FileDataSource(ff.
        getAbsolutePath())));

    return npart; // и возвращаем зашифрованную информацию
}
catch (Exception ex) {
    return part;
}
}
}
```

Файл конфигурации немножко изменился, что же приведем его новую версию.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v5 rel. 4 U (http://www.xmlspy.com) by Nikolai (Comp) -->
<Mail-projects>
    <Mail-project>
        <Secret-key>bigsecret</Secret-key>
        <Algorithm>blowfish</Algorithm>
        <Smtp-server-name>mail.tut.by</Smtp-server-name>
        <Smtp-server-port>25</Smtp-server-port>
        <From>x15@tut.by</From>
        <To>x14@tut.by</To>
        <CC-List>
            <CC-entry>x14@tut.by</CC-entry>
        </CC-List>
        <BCC-List>
            <BCC-entry>x15@tut.by</BCC-entry>
        </BCC-List>
        <Subject>Hello From Java Mail</Subject>
        <Message>Hello, how are you</Message>
        <Attachments>
            <Attachment>E:\Проба\1\boo.html</Attachment>
```

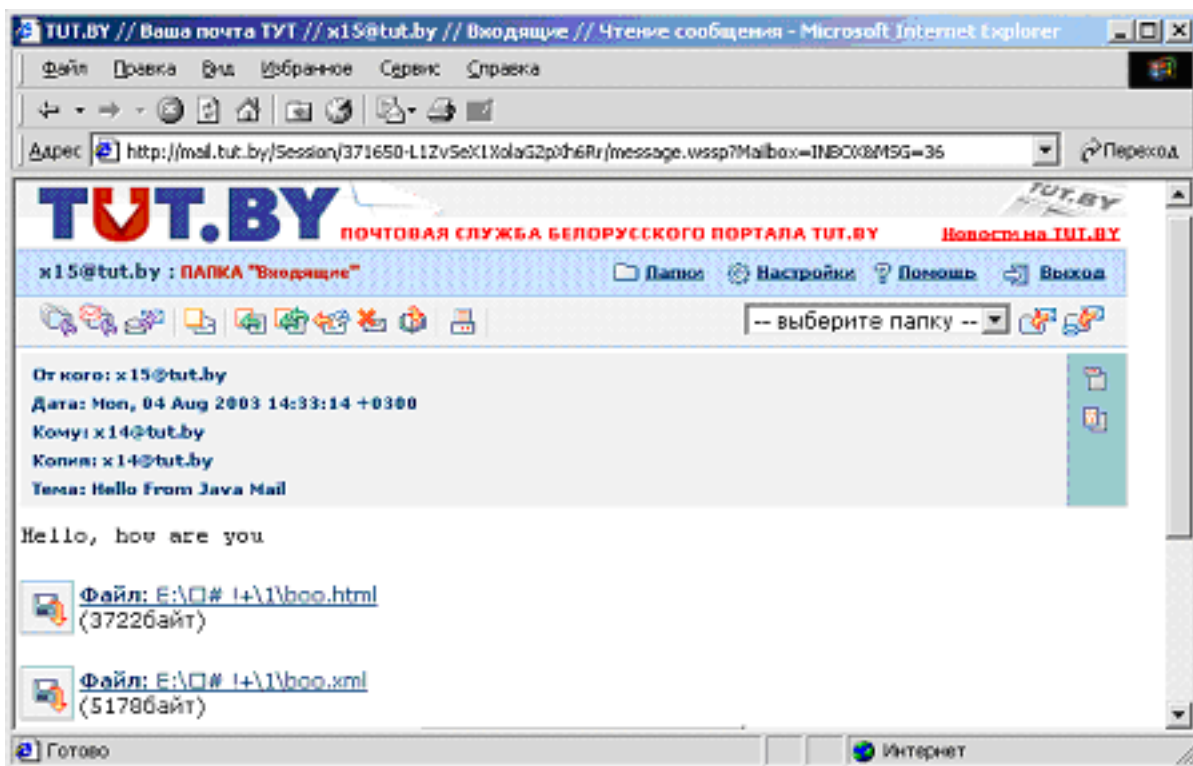
```
        <Attachment>E:\Проба\1\boo.xml</Attachment>
    </Attachments>
</Mail-project>
</Mail-projects>
```

А теперь как и обещал краткий анализ кода. В данном примере мы использовали следующий набор классов и интерфейсов Java Crypto API:

1. **Interface Key** — этот интерфейс играет роль родового и определяет как ключи используемые в симметричной криптографии так и ключи используемые в асимметричной криптографии. Ранее в предыдущих статьях мы уже встречались с интерфейсами `PrivateKey` и `PublicKey` от которых в свою очередь были порождены и интерфейсы для конкретных алгоритмов шифрования или подписи, например `DSAPrivateKey` или `DSAPrivateKey` для алгоритмов DSA согласно стандарту DSS. В нашем примере используется интерфейс `SecretKey` и конкретный класс `SecretKeySpec`, который служит для генерации ключей платформенно- и системно-независимым способом, на вход конструктору данного класса необходимо подать массив `char` которые и задают тот секрет, который знают участники нашего процесса взаимодействия. Класс `SecretKeySpec` определяет симметричный ключ шифрования/дешифрования.
2. **Cipher** — базовый класс для симметричного шифрования, по аналогии с остальным криптографическим API непосредственное создание экземпляра данного класса невозможно, необходимо пользоваться вызовом статического метода `getInstance`, в качестве параметров которому необходимо будет передать не только имя алгоритма, но и дополнительно режим работы блочного алгоритма, помните в первой статье серии я уже упоминал что существуют различные режимы преобразования. Не вдаваясь в избыточные подробности кратко напомним, что существенной дырой в системе безопасности является выполнение одинаковых преобразований над идентичными блоками исходного (открытого) текста, если размер блока был мал по сравнению с повторениями исходного документа, то можно было создать систему взлома на основе частотного анализа. Для решения этой проблемы используется сцепление блоков между собой, и конкретный способ данного сцепления задается при создании экземпляра класса `Cipher`. Таким образом строка передаваемая методу `getInstance` выглядит примерно так: "algorithm/mode/padding", последний, третий параметр задает способ дополнения последних некратных блоков до полного размера блока. Все эти

параметры являются важными для того чтобы текст, который вы зашифровали можно было расшифровать на другой стороне. Дело в том, что даже если ваш адресат знает ключ, но ошибся в способе совмещения блоков или алгоритме дополнения, то расшифровать сообщение он не сможет. В том коде, который привел я, второй и третий параметры не были указаны явно, и берутся по умолчанию. Для некоторых алгоритмов можно задавать также и размер блока для преобразования, например "DES/OFB32/PKCS5Padding".

Итак, подведем итог тому, что делает данный фрагмент кода. Он определяет простой почтовый клиент, который использует JavaMail API и отправляет письма с вложениями, каждое из которых может быть зашифровано симметричным алгоритмом. На стороне получателя приложенные файлы должны быть расшифрованы для прочтения. Если честно говорить, то возможности пока не впечатляют, той же функциональности можно было добиться используя внешние приложения для шифрования файлов. Однако давайте снова не будем торопиться, а задумаемся над тем как обеспечить действительно надежную пересылку информации с шифрованием и ЭЦП, да и при этом сделать так чтобы клиентам не пришлось модифицировать собственное программное обеспечение. Как я уже упоминал ранее в настоящий момент в качестве стандарта при пересылке и защите корреспонденции используется S\MIME и PGP. Хотя чисто внешних отличий в письме не будет но при попытке открытия приложенных файлов пользователь увидит абракадабру. Кстати, чтобы вы не думали что все получается без сложностей вот вам еще один пример картинки, обратите внимание во что превратились русские буквы. Надо, пожалуй, будет как-нибудь написать статью посвященную русским буквам и JavaMail, ну да ладно как-нибудь потом.



4. S/MIME и JavaMail

К сожалению, на момент написания данной статьи на сайте Sun в faq по JavaMail четко было сказано, что поддержки SSL и S/MIME нет, и соответственно следует воспользоваться услугами какого-нибудь из третьих поставщиков. Достаточно большой список которых вы можете найти по адресу java.sun.com/products/javamail/Third_Party.html

После недолго чтения вверительных грамот (т.е. рекламных проспектов) я выбрал продукт от компании "Wedgetail Communications Pty Ltd", а вот адрес ее сайта www.wedgetail.com/jcsi/smime

Ниже я привожу пример их рекламного проспекта.

JCSI offers an implementation of S/MIME (Secure/Multipurpose Internet Mail Extensions) version 3, an Internet standard (RFC 2633) for securing messages using public key

cryptography. S/MIME enables e-commerce by allowing companies and trading partners to encrypt and digitally sign communications, ensuring privacy and authenticity of data.

Следовательно, в числе возможностей данного продукта является шифрование писем, а также добавление к ним ЭЦП. Как раз того, что нам и требовалось. Загрузить библиотеки с сайта wedgetail не составит ни малейшего труда, библиотеки некоторой частью раздаются бесплатно, некоторой частью требуют регистрации и обещают отказаться работать через 30 дней, но нам для эксперимента этого срока хватит вполне, так что продолжаем. После того как вы загрузили библиотеку и примеры с документацией пойдём дальше. Прежде всего ознакомьтесь с примерами приложенными к пакету, они вполне исчерпывающие хотя и решают типовые задачи.

А теперь давайте попробуем создать приложение, использующее возможности пакета от "wedgetail". Однако перед тем как будет написана первая строчка кода давайте мы решим ещё один теоретический момент. Для выполнения операций подписи и шифрования необходим сертификат и закрытый ключ, алгоритму ЭЦП не принципиален алгоритм генерации ключа и то как мы его получим. Однако для нас-то разница есть. В предыдущей статье серии я упоминал о том, что есть возможность, создав собственный ключ и сертификат с помощью keytool или иной утилиты, выполнить заверение сертификата какой-нибудь серьёзной организацией, например, VeriSign. Также я упоминал о том, что VeriSign даёт возможность бесплатно получить на пробу и сертификат с ключом для работы с почтой и я думаю, что использовать лучше именно подобный сертификат, а не наш самопальный и, разумеется, не заслуживающий доверия. Если вы последуете моему примеру, то пройдя несложную процедуру регистрации на сайте VeriSign получите файл хранилища частной информации в формате pkcs12. При попытке выполнить просмотр или экспорт информации хранящейся в данном файле при помощи keytool я получил кучу ошибок, общий итог которых сводился к следующему "мы не знаем что такое pkcs12, мы дружим только с jks, и как бы ты не указывал через свойства -provider путь провайдера криптографических услуг, как бы не старался править файлы security, и не мучался с classpath и прочая и прочая, у тебя все равно ничего не получится". На этом я разозлился и решил написать самостоятельно маленькую программку по извлечению содержимого файла "*.pfx". Исходный код данного примера я привожу далее, равно как и результат ее работы. Далее я привожу копию экрана в котором открыт тот же сертификат который я извлек из хранилища с помощью данного фрагмента кода.

Последнее на что я обращаю ваше внимание, это то что необходимо будет добавить провайдера криптографических услуг в список используемых, либо на этапе выполнения программы или поместив это в файл "там где у вас установлена jre\lib\security\java.security".

```
#
# List of providers and their preference orders (see above):
#
security.provider.1=sun.security.provider.Sun
security.provider.2=com.sun.net.ssl.internal.ssl.Provider
security.provider.3=com.sun.rsa.jca.Provider
security.provider.4=com.sun.crypto.provider.SunJCE
security.provider.5=sun.security.jgss.SunProvider
security.provider.6=com.dstc.security.provider.DSTC
```

А вот и исходный код примера.

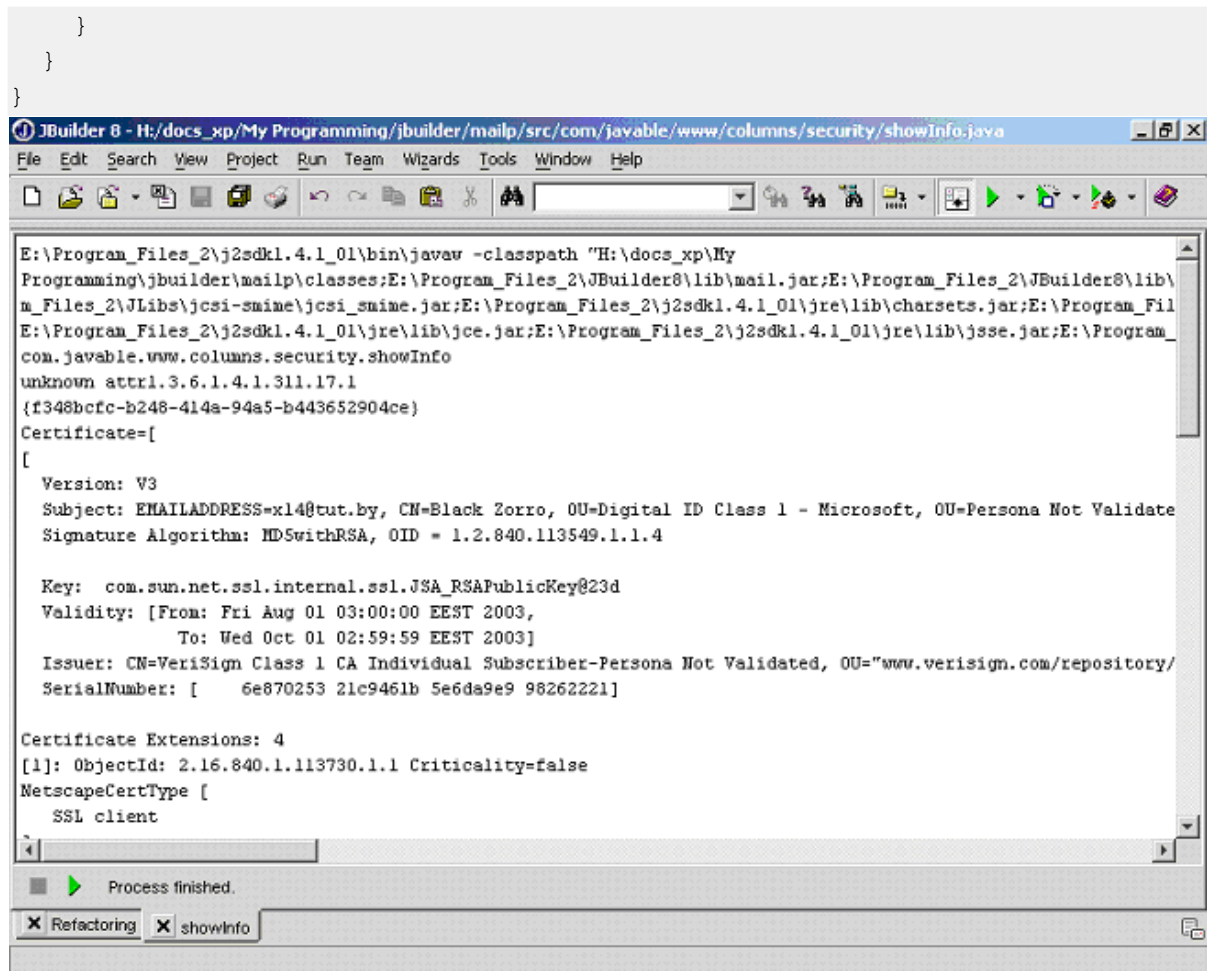
```
package com.javable.www.columns.security;
import java.io.*;
import java.math.*;
import java.security.*;
import java.util.*;
import java.security.cert.*;

import com.dstc.security.pkibase.*;

public class showInfo {
    public static void main(String[] args) throws Exception {
        FileInputStream fis = new FileInputStream("proba.pfx");
        KeyStore ks = KeyStore.getInstance("pkcs12");
        char[] pw = "1".toCharArray();
        // пароль я указал жестко, да и, честно говоря, данный код не
        // претендует на роль идеального
        ks.load(fis, pw);
        Enumeration E = ks.aliases();
        while (E.hasMoreElements())
        {
            Object ali = E.nextElement();
            System.out.println (ali);
            System.out.println ("Certificate="+ ks.getCertificate((String)ali));
            System.out.println ("Key "+ks.getKey((String)ali, "1".toCharArray()));
        }
    }
}
```

Практическая криптография в Java. Защита электронной почты

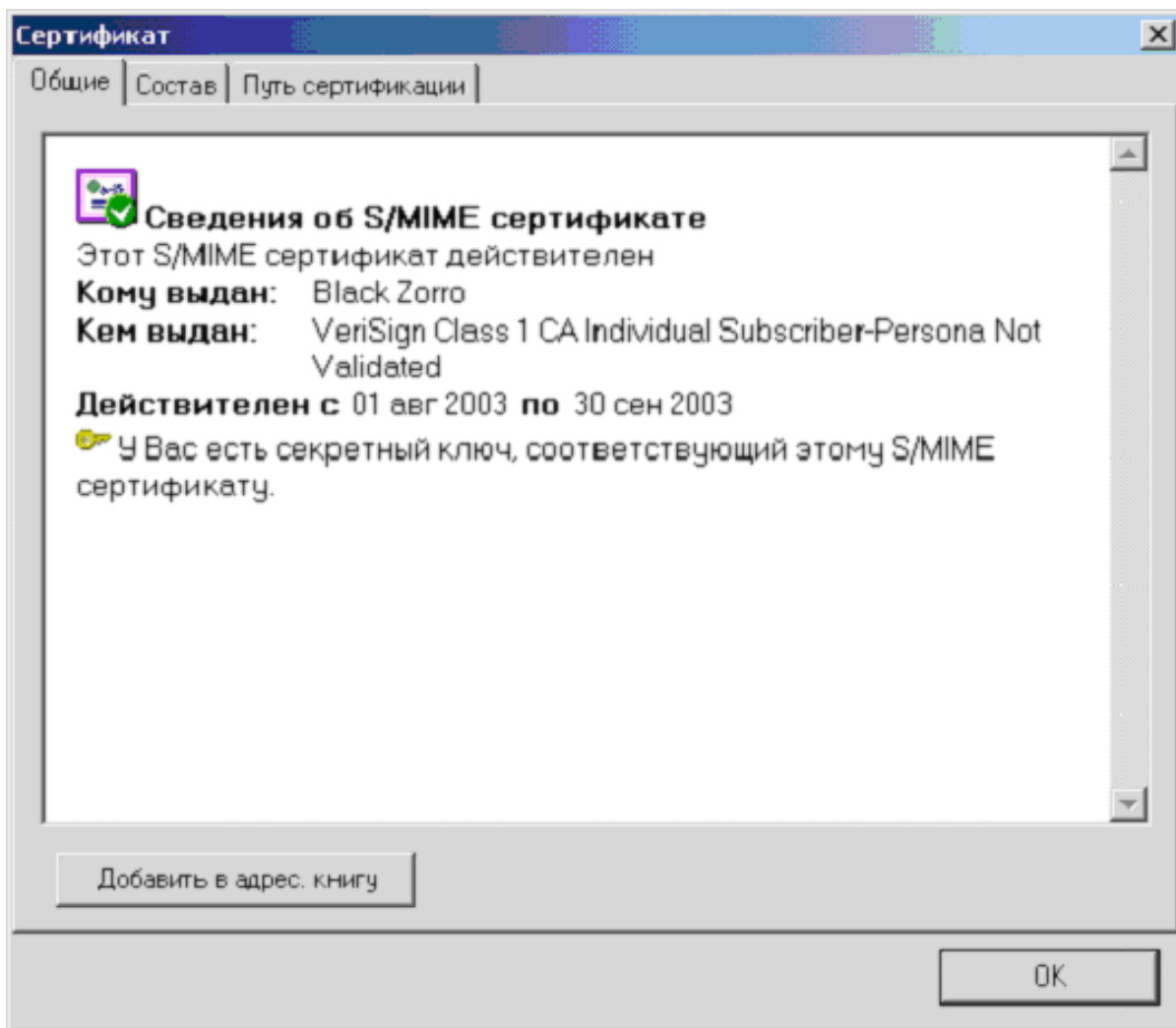
```
}  
}  
}  
}
```



```
E:\Program_Files_2\jdk1.4.1_01\bin\javaw -classpath "H:\docs_xp\My  
Programming\jbuilder\mailp\classes;E:\Program_Files_2\JBuilder8\lib\mail.jar;E:\Program_Files_2\JBuilder8\lib\  
m_Files_2\JLibs\jcsi-smime\jcsi_smime.jar;E:\Program_Files_2\jdk1.4.1_01\jre\lib\charsets.jar;E:\Program_Fil  
E:\Program_Files_2\jdk1.4.1_01\jre\lib\jce.jar;E:\Program_Files_2\jdk1.4.1_01\jre\lib\jsse.jar;E:\Program_  
com.javable.www.columns.security.showInfo  
unknown attr1.3.6.1.4.1.311.17.1  
{f348bcfc-b248-414a-94a5-b443652904ce}  
Certificate=[  
[  
  Version: V3  
  Subject: EMAILADDRESS=x14@tut.by, CN=Black Zorro, OU=Digital ID Class 1 - Microsoft, OU=Persona Not Validate  
  Signature Algorithm: MD5withRSA, OID = 1.2.840.113549.1.1.4  
  
  Key: com.sun.net.ssl.internal.ssl.JSA_RSAPublicKey@23d  
  Validity: [From: Fri Aug 01 03:00:00 EEST 2003,  
            To: Wed Oct 01 02:59:59 EEST 2003]  
  Issuer: CN=VeriSign Class 1 CA Individual Subscriber-Persona Not Validated, OU="www.verisign.com/repository/  
  SerialNumber: [ 6e870253 21c9461b 5e6da9e9 98262221]  
  
Certificate Extensions: 4  
[1]: ObjectId: 2.16.840.1.113730.1.1 Criticality=false  
NetscapeCertType [  
  SSL client  
]
```

Process finished.

Refactoring showInfo



Итак, мы научились извлекать из файла "*.pfx" и сертификат и ключи, давайте применим наши знания для непосредственно шифрования писем и добавления к ним ЭЦП. Правда сначала нам потребуется кратко описать основные классы реализованные компанией wedgetail.

5. Техническая информация о классах wedgetail

Класс `com.dstc.security.smime.SMIMESignature` – данный класс очень похож на уже знакомый нам класс `Signature` и служит для подписи сообщения

письма при его отправке и соответственно для того чтобы проверить действительность сообщения при получении. Все эти операции выполняются в точности с RFC 2633 "S/MIME Version 3 Message Specification". Данный класс обладает способностью выполнять подпись сообщений как "прозрачно" так и "не прозрачно". Отличия этих двух способов в том, что в когда используется прозрачный режим исходное сообщение обертывается другим сообщением, состоящим условно говоря из двух частей – оригинальное сообщение, а также часть в которой хранится ЭЦП. Такой способ имеет преимущество в том, что письма могут быть прочитаны даже теми почтовыми клиентами которые не умеют работать с S\MIME письмами.

Во втором же случае формируется новое письмо в формате S\MIME которое содержит как единое целое и исходное сообщение и результат его подписи, а следовательно прочитать письмо в почтовом клиенте без поддержки S\MIME будет невозможно. Так как данный класс реализован компанией wedgetail и не является полной частью Java Crypto API, то при создании объекта данного класса не следует, да и не возможно указать имя провайдера услуг. Создание экземпляра выполняется или с помощью `new` или вызова `getInstance` в качестве параметров данного метода можно передавать опции работы, например параметр `jcsl.mime.noencode` позволяет управлять кодированием письма после вычисления ЭЦП и ее присоединения. Как известно почта должна кодироваться с помощью `base64`, дело в том что некоторые почтовые сервера очень не дружат с сообщениями двоичные коды которых превосходят 127. Следовательно, если вы отправляете сообщение по сети то рекомендуется устанавливать данное свойство в значение "false", что в прочем делается автоматически.

После создания объекта типа "SMIMESignature" в зависимости от типа будущей работы должен быть вызван или метод `initSign` или метод `initVerify`. Для первого из них следует передать в качестве параметров имя используемого алгоритма подписи, частный ключ отправителя сообщения и цепочку сертификатов, начиная от сертификата отправителя (содержащего подтверждение о соответствии данного открытого ключа закрытому) а также как принято всю иерархию сертификатов СА удостоверяющих подлинность сертификата отправителя.

Если же нам необходимо выполнить процедуру проверки подлинности сообщения то вызываем метод `initVerify` а в качестве параметров передаем ему список

сертификатов которым мы доверяем. Затем нам следует указать какое именно сообщение мы хотим подписать или проверить, для этого следует вызвать метод "setMessage" и, наконец, для выполнения непосредственно операции подписи сообщения или его проверки вызываем метод "verify" или "sign".

Класс `com.dstc.security.smime.SMIMECipher` служит для выполнения операций шифрования и дешифрования почтовых сообщений согласно RFC 2633 "S/MIME Version 3 Message Specification". Создание класса выполняется с помощью `new` или через вызов фабричного метода `getInstance`, так же как и для предыдущего класса в качестве параметра для данного метода можно передать объект `Properties` с настройками поведения. Надо сказать что настройки в целом совпадают с теми, которые используются и в классе `SMIMESignature`.

После создания объекта следует указать для какой именно операции объект будет использоваться. Если вы хотите выполнить шифрование письма при его отправке, то необходимо вызвать метод `initEncrypt`. В качестве параметров которому следует передать имя алгоритма шифрования, в качестве допустимых используются "DESede", "RC2", "RC2/40" (40-bit RC2) (это в том случае если проживаете за пределами соединенных штатов, а уже тем более в одной из стран с неблагонадежным режимом) и "IDEA". Также среди параметров метода `initEncrypt` есть массив сертификатов. Эти сертификаты соответствуют списку тех лиц которые получают сообщение или его копии.

Надеюсь, вы помните, что общепринято ассиметричные алгоритмы использовать не для шифрования сообщений, а ключей (сеансовых ключей) для симметричных алгоритмов. Следовательно, при шифровании письма будет создан уникальный ключ для указанного вами симметричного алгоритма шифрования, а он в свою очередь будет зашифрован с помощью открытого ключа хранящегося в каждом из сертификатов. Таким образом, при получении сообщения получатель воспользуется своим закрытым ключом для того чтобы расшифровать симметричный ключ шифрования и потом с его помощью уже расшифровать и само сообщение.

Если вы хотите выполнять с помощью класса `SMIMECipher` операцию расшифровки полученных сообщений то следует вызвать метод `initDecrypt` в качестве параметра следует передать закрытый ключ и сертификат получателя сообщений. В том случае если у получателя есть несколько закрытых ключей, то в этом случае следует

воспользоваться перегруженной версией метода `initDecrypt`, в качестве параметров которому следует передать массив ключей и массив сертификатов. Следующим шагом будет привязка письма с помощью метода `setMessage` и, наконец, вызов метода `decrypt` или `encrypt`.

Для того чтобы проверить полученное сообщение на предмет того были ли они зашифрованы или подписаны следует воспользоваться классом `com.dstc.security.smime.SMIMEUtil`. Данный класс содержит методы: `isEncrypted` для проверки зашифровано ли сообщение; метод `isSigned` – для проверки было ли сообщение подписано. И, наконец, `getEmailAddress` для того чтобы извлечь из приложенного к сообщению сертификата электронный адрес отправителя и проверить его на совпадение с тем, что содержится в заголовках письма, говоря проще, чтобы проверить действительно ли сообщение было отправлено именно тем, чей сертификат был приложен к сообщению.

Для шифрования сообщений и их подписи может быть использован также набор классов работающих согласно RFC 2630. Это классы `com.dstc.security.cms.CMSSignature` и `com.dstc.security.cms.CMSCipher`. Схема использования данных классов практически идентична ранее рассмотренной. Отличия возникают только на этапе использования конкретного алгоритма для асимметричного шифрования и ЭЦП соответственно. Для данных двух классов существуют две схемы применения в зависимости от того используются ли ключи RSA или ключи Диффи-Хелмана.

6. Исходный код примера. Защищенный почтовый клиент

Что же пора привести и исходный код примера.

```
package com.javable.www.columns.security;

import java.io.*;
import java.security.*;
import java.security.cert.*;
import javax.mail.internet.*;

import org.xml.sax.*;
```

```
import com.dstc.security.smime.*;

public class SMIMEMailAgent extends AbstractMailAgent {
    public SMIMEMailAgent(String fName) throws IOException,
        SAXException {
        super (fName);
    }

    protected MimeMessage processMessage
        (MimeMessage mes ,org.w3c.dom.Element confElt ){
    try {
        System.out.println("Попытка преобразования сообщения");
        String keyStoreFName = XTools.getXPathText(
            "Mail-project/Keystore-path",confElt);
        // предпочительно работать с хранилищами ключей не в формате jks
        // а в чем-нибудь более широко известном и общепринятом
        // я использую хранилища личной информации pkcs12
        KeyStore ks = KeyStore.getInstance("pkcs12");
        ks.load(new FileInputStream(keyStoreFName),
            XTools.getXPathText("/Mail-project/Keystore-password", confElt).
                toCharArray());
        boolean isNeedSign = Boolean.valueOf(
            XTools.getXPathText("Mail-project/Need-Sign", confElt)).booleanValue();
        if (isNeedSign ){
            String aliasKeySender = XTools.getXPathText(
                "/Mail-project/Keystore-sender-alias", confElt);
            if (aliasKeySender == null)
                aliasKeySender = (String)ks.aliases().nextElement();
            // ключ с помощью которого мы можем подписать отправляемое сообщение
            Key k = ks.getKey(aliasKeySender , XTools.getXPathText(
                "/Mail-project/Keystore-sender-password", confElt).toCharArray());
            SMIMESignature ssig = new SMIMESignature();
            java.security.cert.Certificate [] senderCerts =
                ks.getCertificateChain(aliasKeySender);
            java.security.cert.X509Certificate senderX509Certs [] = new
                java.security.cert.X509Certificate [senderCerts.length];
            for (int i= 0; i < senderX509Certs.length; i++)
                senderX509Certs [i] = (X509Certificate) senderCerts [i];
            ssig.initSign("SHA-1",(PrivateKey) k, senderX509Certs, false);
        }
    }
}
```

```
        ssig.setMessage(mes);
        mes = ssig.sign();
        System.out.println("Операция подписи сообщения успешно завершена");
    }

    boolean isNeedCipher = Boolean.valueOf(
        XTools.getXPathText("Mail-project/Need-Cipher", confElt)
    ).booleanValue();
    if (isNeedCipher){
        String pathCertificate = XTools.getXPathText(
            "Mail-project/Receiver-Certificate-Path", confElt);
        CertificateFactory factory = CertificateFactory.getInstance("X509");
        X509Certificate certReceiver =
            (X509Certificate) factory.generateCertificate(
                new FileInputStream(pathCertificate));
        SMIMECipher ciph = new SMIMECipher();
        ciph.initEncrypt(new SecureRandom(), "DESede",
            new X509Certificate[] {certReceiver});
        ciph.setMessage(mes);
        mes = ciph.encrypt();
        System.out.println("Операция шифрования сообщения успешно завершена");
    }

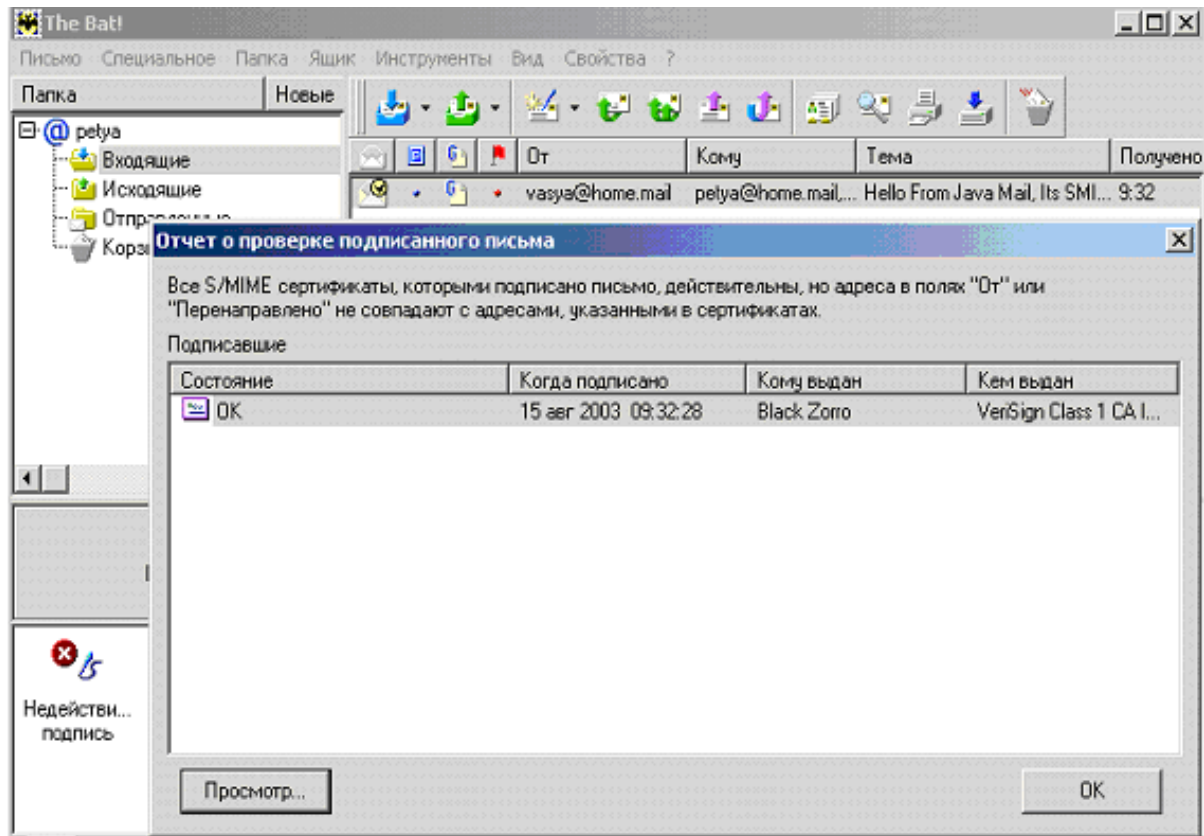
    return super.processMessage(mes, confElt);
}
catch (Exception ex) {
    ex.printStackTrace();
    return mes;
}
}
}
```

Файл конфигурации изменился также изменился в очередной раз. Приведем его снова. В новой редакции данного файла добавились новые разделы которые служат для хранения информации о местоположении хранилища ключей, паролей доступа для той сущности у которой в этом хранилище и находится частный (закрытый) ключ. Также добавился элемент с указанием местоположения того сертификата который принадлежит нашему адресату. И наконец, добавились два ключика включающих и соответственно отключающих определенные режимы работы почтового клиента

(шифровать, подписывать). Это ключи `<Need-Cipher>` и `<Need-Sign>`.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v5 rel. 4 U (http://www.xmlspy.com) by Kolya (Home) -->
<Mail-projects>
  <Mail-project>
    <Smtp-server-name>comp</Smtp-server-name>
    <Smtp-server-port>25</Smtp-server-port>
    <From>vasya@home.mail</From>
    <To>petya@home.mail</To>
    <CC-List>
      <CC-entry>anton@tut.by</CC-entry>
    </CC-List>
    <BCC-List>
      <BCC-entry>victor@tut.by</BCC-entry>
    </BCC-List>
    <Subject>Hello From Java Mail, Its SMIME Message</Subject>
    <Message>Hello, how are you</Message>
    <Keystore-path>
      H:\docs_xp\My Programming\jbuilder\mailp\get money1.pfx
    </Keystore-path>
    <Keystore-password>get money</Keystore-password>
    <Keystore-sender-password>get money</Keystore-sender-password>
    <Receiver-Certificate-Path>
      H:\docs_xp\My Programming\jbuilder\mailp\Niko_x15@tut.by.cer
    </Receiver-Certificate-Path>
    <Attachments>
      <Attachment>
        H:\Docs_Books\mars\artifact.txt
      </Attachment>
      <Attachment>
        H:\Docs_Books\mars\bzg.txt
      </Attachment>
    </Attachments>
    <Need-Cipher>False</Need-Cipher>
    <Need-Sign>True</Need-Sign>
  </Mail-project>
</Mail-projects>
```

А вот и примеры копий экрана как демонстрация того что все примеры кода, которые я привожу работают, что же на сегодня пожалуй хватит, давайте подводить итоги.



7. Аналоги и заключение

Если почитать информацию по "iForce Directories", то можно узнать, что с Sun сотрудничает более чем 6000 организаций, в том числе и в направлениях обеспечения приватности передаваемой информации. Интерес также представляют решения от "MailQube", направления ее работы связаны с обеспечением безопасности в вашей почтовой переписке. Решаемые задачи:

1. Гарантия того что сообщения которые вы получаете пришли именно от той организации или лица, который заявляет себя как отправителя.
2. Проверка целостности информации, которую вы получаете, т.е. защита от случайной или преднамеренной модификации сообщений.
3. И, разумеется, информация переписки не будет доступна третьим лицам. Информация будет зашифрована.

В основе данного решения также лежит принцип прозрачного преобразования запроса и использования средств HTTPS. Следовательно, клиенту нет необходимости выполнять дополнительные шаги по покупке специального ПО и он может использовать любой почтовый клиент.

На этом я, пожалуй, заканчиваю свое повествование, для тех, кого заинтересовали существующие решения в области безопасности и Java могу просто посоветовать покопаться в каталоге iForce.

8. Заключение

Да, сегодня мы поработали очень неплохо. В отличие от предыдущей статьи серии имевшей в основном характер теоретический, мы сегодня больше писали рабочий, а не перечисляли используемые классы и интерфейсы JavaMail. При написании материала я предполагал, что читатель обладает знаниями в области JavaMail API (а если нет, то можно найти достаточное количество специализированных учебников, очень хороший материал был на javaguru), так что основное внимание было обращено вопросу защиты передаваемой информации. С целью этого сначала была создана простейшая программная реализация почтового клиента который в последствии при развитии приложения обзавелся средствами шифрования приложенных файлов. Далее автор использовал для реализации поддержки S\MIME в JavaMail API решения от компании www.wedgetail.com. Не претендуя на детальное описание классов и интерфейсов которые предлагаются в данном пакете, нам удалось создать работающее приложение с возможностью средств шифрования и подписи электронных писем.

9. Ресурсы

1. www.javaworld.com/javatips/jw-javatip115.html — краткая, но емкая статья по применению SSL и JavaMail api.
2. java.sun.com/products/javamail/Third_Party.html — список поставщиков решений для JavaMail.
3. [/www.wedgetail.com/jcsi/smime](http://www.wedgetail.com/jcsi/smime) — Java и S\MIME. Продукты, документация.