

# Системы безопасности для клиентов. Создание доверенного апплета с использованием криптосистем открытого ключа

Григорий Печеницын

В первой статье из серии мы рассказывали о создании доверенного (trusted) апплета наделенного правом доступа к локальной файловой системе. Настоящая статья является продолжением цикла и в ней рассматривается одна из специфических задач требующая реализации алгоритма шифрования MD5 with RSA/SHA1 with DSA ("RSA/DSA open key crypto system") на стороне клиента и дешифрования сообщения клиента на стороне сервера, которая практически не может быть реализована java апплетом без доступа к файлам на машине клиента. Примером из практики с которым вы можете столкнуться уже в ближайшее время (достаточно вспомнить проект "Электронная Россия" недавно принятый правительством РФ) является случай заверения удаленным клиентом платежного документа или договора выставяемого ему владельцем сайта, например банком, или любой другой финансовой организацией.

Хочу напомнить, что данная схема подписи апплетов пригодна для клиентов которые согласны установить плагин jdk1.3 который можно скачать с <http://java.sun.com/getjava/download.html> или аналогичный для работы с бизнес приложениями и используют IE4 и выше или NN4.75 и более поздние версии. Для клиентов не имеющих возможности работы с jdk1.3 следует использовать схему доверения реализованную на базе JDK1.1 с использованием javakey. В данной статье не опущены некоторые прикладные аспекты, например взаимодействие апплета и сервлета или создание документа для отсылки клиенту, их реализация достаточно банальна, и не должна вызывать затруднения — соответственно, она не является предметом

рассмотрения для данной статьи.

## **1. С чего следует начать работу с криптосистемами открытого ключа?**

— с общего описания криптосистем: сама по себе тема шифрования настолько объемна и сложна что не имеет смысла рассматривать ее в рамках даже отдельной статьи, поэтому я постараюсь ограничиться минимально достаточным обзором, и по возможности касаться только тех аспектов вопроса, которые имеют самое непосредственное отношение к задаче которую нам предстоит решить.

В криптографической системе с открытым ключом каждый имеет два связанных взаимно однозначно ключа: публикуемый открытый ключ и секретный ключ. Каждый из них дешифрует код, сделанный с помощью другого. Знание открытого ключа не позволяет вам вычислить соответствующий секретный ключ. Открытый ключ может публиковаться и широко распространяться через коммуникационные сети. Такой протокол обеспечивает секретность без необходимости использовать специальные каналы связи, необходимые для стандартных криптографических систем.

Кто угодно может использовать открытый ключ получателя, чтобы зашифровать сообщение ему, а получатель использует его собственный соответствующий секретный ключ для расшифровки сообщения. Никто, кроме получателя, не может расшифровать его, потому что никто больше не имеет доступа к секретному ключу. Даже тот, кто шифровал сообщение, не будет иметь возможности расшифровать его.

Кроме того, обеспечивается также установление подлинности сообщения. Собственный секретный ключ отправителя может быть использован для шифровки сообщения, таким образом "подписывая" его. Так создается электронная подпись сообщения, которую получатель (или кто-либо еще) может проверять, используя открытый ключ отправителя для расшифровки. Это доказывает, что отправителем был действительно создатель сообщения и что сообщение впоследствии не изменялось кем-либо, так как отправитель — единственный, кто обладает секретным ключом, с помощью которого была создана подпись. Подделка подписанного сообщения невозможна, и отправитель не может впоследствии изменить свою подпись.

Эти два процесса могут быть объединены для обеспечения и секретности, и

## *Системы безопасности для клиентов. Создание доверенного апплета с использованием криптосистем открытого ключа*

установления подлинности: сначала подписывается сообщение вашим собственным секретным ключом, а потом шифруется уже подписанное сообщение открытым ключом получателя. Получатель делает наоборот: расшифровывает сообщение с помощью собственного секретного ключа, а затем проверяет подпись с помощью вашего открытого ключа. Эти шаги выполняются автоматически с помощью программного обеспечения получателя.

Открытые ключи хранятся в виде "сертификатов ключей", которые включают в себя идентификатор пользователя владельца ключа (обычно это имя пользователя) , временную метку, которая указывает время генерации пары ключей, и собственно ключи. Сертификаты открытых ключей содержат открытые ключи, а сертификаты секретных ключей — секретные. Каждый секретный ключ также шифруется с отдельным паролем. Файл ключей, или каталог ключей keystore в нашем случае содержит один или несколько таких сертификатов. В каталогах открытых ключей хранятся сертификаты открытых ключей, а в каталогах секретных — сертификаты секретных ключей.

## **2. Сценарии работы**

Примерно так выглядят канонические пояснения по поводу сущности криптосистем открытого ключа. На самом деле — часть описанного процесса вам уже довелось реализовывать, если вы создавали trusted апплет — для этого вы создали пару ключей в вашем локальном keystore и которыми заверили ваш апплет — принадлежность подписи апплета (если заняться детальной проверкой) однозначно определяется как принадлежность паре ключей хранящейся в вашем keystore. В сети можно найти и более упрощенные формулировки — одну совершенно восхитительную формулировку 'на пальцах' мне удалось найти на [astu.secna.ru/](http://astu.secna.ru/) — похоже что доступнее уже некуда, однако автор совершенно справедливо отмечает проблемы связанные с использованием открытых ключей — основная из которых — это вопросы доверения, которые возникают, когда нет возможности проверить подпись отправителя, и этот пунктик наверняка был отмечен вами если вы уже имели дело с доверенными апплетами или просматривали материалы в предыдущей статье посвященной апплетам. Проще говоря, никто не мешает кому угодно подписаться кем угодно, и тогда весь вопрос заключается только в том, насколько вы доверяете источнику

## *Системы безопасности для клиентов. Создание доверенного апплета с использованием криптосистем открытого ключа*

получения сертификата — ведь сам факт наличия сертификата без возможности его проверки не говорит ни о чем, кроме, пожалуй, достаточной квалификации его издателя.

Что же касается заварения сертификатов третьей стороной например VeriSign или Thawte, когда проблема в конечном итоге сводится к вопросу доверия, но уже не одной организации, а двум или более, однако, это уже несколько отдельная тема, и не хотелось бы пытаться рассмотреть все аспекты проблемы в рамках одной статьи, тем более что сама тема весьма объемна.

Если же вы имеете сертификат pkcs12, \*.cer или аналогичный с помощью которого можно проверить, совпадает ли он с сертификатом стороны, подписавшей сообщение то это сразу меняет дело, поскольку появляется возможность сразу же определить подлинность и целостность сообщения и однозначно идентифицировать отправителя. Таким образом, решается основная задача стоящая перед нами - это однозначное определение личности человека подписавшего сообщение, задача вполне актуальная для любой финансовой группы ведущей электронный бизнес.

На практике это могло бы выглядеть следующим образом: доверенное лицо фирмы обладающее правом заверения документа получает или берет тщательно хранимую (лучше если в сейфе, как печати или другие платежные документы фирмы) дискету с секретным ключом, и заходит на сайт банка за документом. После логинизации в системе банка, если он действительно является доверенным лицом обладающим правом доступа к документу, то только тогда он вправе затребовать документ для просмотра и, возможно, заверения. В ответ на его запрос сервер генерирует документ для этого пользователя, исходя из его данных логинизации. Если после ознакомления пользователь согласен с документом, документ может быть им заверен при следующих условиях:

- лицо, заверяющее документ должно иметь дискету с ключами
- он должен знать alias(имя) которым может быть заверен данный документ (их может быть несколько на дискете, но только одним можно заверить данный документ)
- лицо, заверяющее документ обладает паролем для алиаса (имени) в keystore хранящимся на дискете, и правомочным для заверения данного документа.

Встречаются подход, когда псевдоним в системе может совпадать с псевдонимом для

## *Системы безопасности для клиентов. Создание доверенного апплета с использованием криптосистем открытого ключа*

электронной подписи — и тогда alias (имя) можно передавать как параметр сессии, но выбор между удобством и безопасностью всегда определяется конкретным случаем.

При нормальном учете выдаваемых дискет и имен доступа, все это практически однозначно позволяет идентифицировать человека, заверившего документ. На практике могут встречаться случаи, когда есть риск копирования keystore вкупе с передачей паролей и имен третьим лицам, и хотя это уже скорее из области криминологии, чем программирования, однако даже и в этой ситуации источник утечки данных легко найти. Когда же правом заверения обладает один человек, то только при условии публикации логина, пароля, системного имени и свободной раздачи дискеты всем желающим он имеет неплохие шансы на подделку электронной подписи, поскольку при отсутствии хотя бы одного из компонентов задача подделки подписи уже значительно усложняется.

### **3. Детали реализации**

Давайте перейдем к схеме которую мы собираемся реализовать и для начала просто рассмотрим основные моменты:

- Мы имеем тонкого web-клиента, обращающегося из браузера на сервер за документом (договором или платежкой), у клиента заранее есть его персональный секретный ключ (хранимый на дискете, keystore), с его личным паролем.
- Сервер, принимающий обращение клиента, идентифицирующий его по кукам или тракингу сессии или другим способом, и имеющий в БД файл сертификата (keystore) содержащий публичный ключ клиента для его проверки на принадлежность.
- После запроса авторизованного клиента, по его запросу, сервер загружает в браузер клиента документ и ожидает от клиента тот же документ, но уже заверенный цифровой подписью данного авторизованного клиента.
- Если клиент заверяет документ, то сервер получив заверенный документ проверяет, действительно ли данный документ заверен секретным ключом данного клиента, сверяя его с открытым ключом клиента хранящимся в БД.
- В случае, если цифровая подпись верна — документ признается заверенным, после чего сам документ и цифровая подпись заносятся в БД и остаются в ней на хранение на нужный срок.

## *Системы безопасности для клиентов. Создание доверенного апплета с использованием криптосистем открытого ключа*

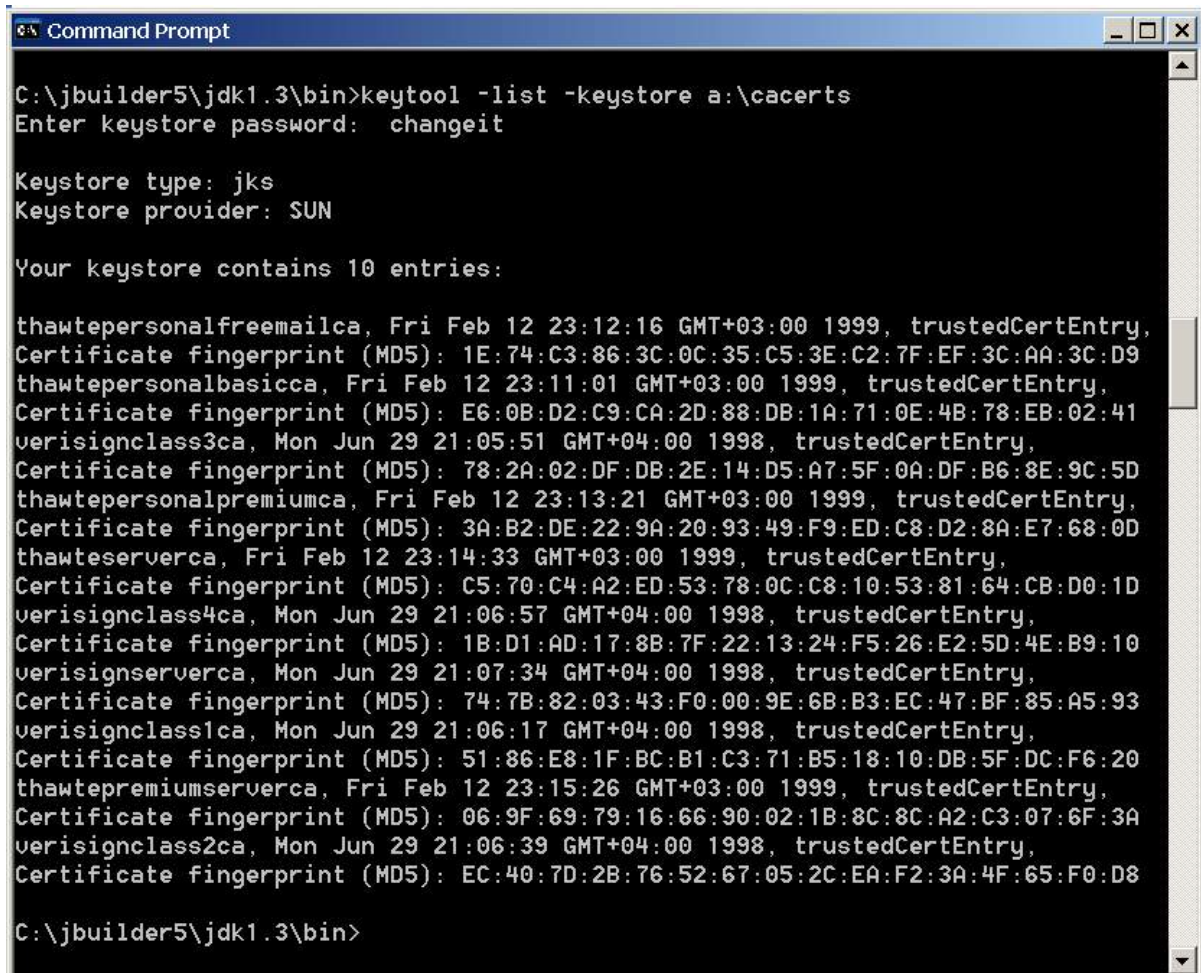
Весь процесс напоминает заверение апплета апплетом с точностью до наоборот по существу, нужно поместить на машину клиента секретные ключи для подписи пересланного файла и переслать его обратно для проверки на стороне сервера. Очевидно, что мы не сможем избежать обращения к файлу на клиентской машине содержащему секретные ключи для подписи — именно для этого нам понадобится доступ к локальной файловой системе клиента, и соответственно, именно доверенный апплет может быть тем программным обеспечением, которое способно выполнить цифровую подпись без установки чего-либо еще на машине клиента. Хочу заметить, что ключи, которыми вы подписываете апплет никак не связаны с ключами которыми клиент подписывает сообщение вам (серверу) — лучше сразу мысленно разделить это на два процесса для удобства восприятия, в конце концов вполне возможна ситуация когда заверять апплет будете не вы, а какая-то третья сторона. Сама по себе процедура создания доверенного апплета не сложна и описана в [предыдущей статье](#) из цикла.

Начать следует с создания мобильного keystore — с процессом создания локального keystore из приложения командной строки вы уже знакомы по предыдущей статье где мы использовали его для подписания доверенного апплета. Процесс создания несколько отличается от описанного ранее, с добавлением некоторых весьма существенных деталей:

- вы должны создать мобильный keystore для подписи в формате "JKS"
- для упрощения keystore должен быть пригоден для подписи сообщения и верификации одновременно.
- возможно на стороне сервера получать ваше сообщение будет вовсе не java приложение, и потому следует предусмотреть возможность унификации сертификатов и ключей согласно криптографическим стандартам.

Поскольку мы не хотим программно создавать keystore с самого начала можно оставить эту затею любителям димедрола — то практически проще всего взять уже существующий и приспособить под свои нужды. Взять его можно в плагине jdk1.3 в директории JavaSoft\JRE\1.3.1\_02\lib\security\cacerts или jdk1.3\jre\lib\security\cacerts и для удобства работы лучше скопировать его на диск А. Он уже содержит сертификаты изначально заложенные в него при создании, и открывается стандартным паролем changeit который следует сменить при первой возможности, а заодно имеет смысл почистить его для приспособления под наши задачи и удобства. Первым делом давайте посмотрим что же мы имеем, и для этого посмотрим содержимое keystore'a:

*Системы безопасности для клиентов. Создание доверенного апплета с использованием криптосистем открытого ключа*



```
Command Prompt
C:\jbuilder5\jdk1.3\bin>keytool -list -keystore a:\cacerts
Enter keystore password: changeit

Keystore type: jks
Keystore provider: SUN

Your keystore contains 10 entries:

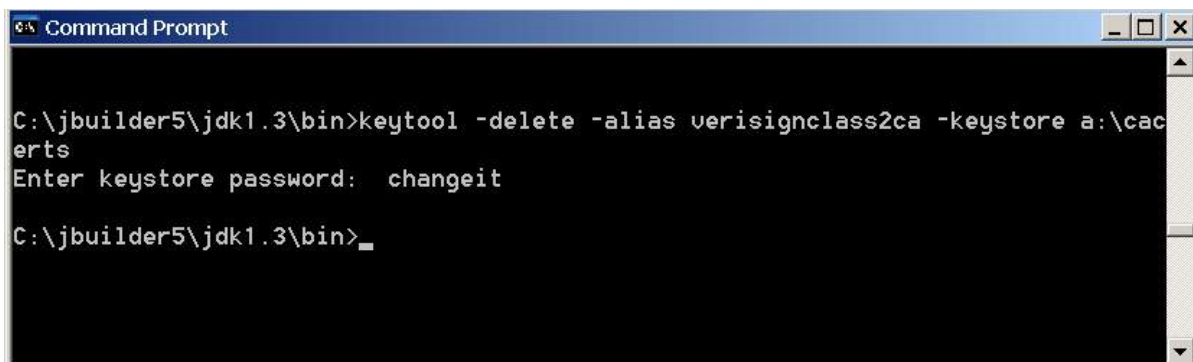
thawtepersonalfreemailca, Fri Feb 12 23:12:16 GMT+03:00 1999, trustedCertEntry,
Certificate fingerprint (MD5): 1E:74:C3:86:3C:0C:35:C5:3E:C2:7F:EF:3C:AA:3C:D9
thawtepersonalbasicca, Fri Feb 12 23:11:01 GMT+03:00 1999, trustedCertEntry,
Certificate fingerprint (MD5): E6:0B:D2:C9:CA:2D:88:DB:1A:71:0E:4B:78:EB:02:41
verisignclass3ca, Mon Jun 29 21:05:51 GMT+04:00 1998, trustedCertEntry,
Certificate fingerprint (MD5): 78:2A:02:DF:DB:2E:14:D5:A7:5F:0A:DF:B6:8E:9C:5D
thawtepersonalpremiumca, Fri Feb 12 23:13:21 GMT+03:00 1999, trustedCertEntry,
Certificate fingerprint (MD5): 3A:B2:DE:22:9A:20:93:49:F9:ED:C8:D2:8A:E7:68:0D
thawteserverca, Fri Feb 12 23:14:33 GMT+03:00 1999, trustedCertEntry,
Certificate fingerprint (MD5): C5:70:C4:A2:ED:53:78:0C:C8:10:53:81:64:CB:D0:1D
verisignclass4ca, Mon Jun 29 21:06:57 GMT+04:00 1998, trustedCertEntry,
Certificate fingerprint (MD5): 1B:D1:AD:17:8B:7F:22:13:24:F5:26:E2:5D:4E:B9:10
verisignserverca, Mon Jun 29 21:07:34 GMT+04:00 1998, trustedCertEntry,
Certificate fingerprint (MD5): 74:7B:82:03:43:F0:00:9E:6B:B3:EC:47:BF:85:A5:93
verisignclass1ca, Mon Jun 29 21:06:17 GMT+04:00 1998, trustedCertEntry,
Certificate fingerprint (MD5): 51:86:E8:1F:BC:B1:C3:71:B5:18:10:DB:5F:DC:F6:20
thawtepremiumserverca, Fri Feb 12 23:15:26 GMT+03:00 1999, trustedCertEntry,
Certificate fingerprint (MD5): 06:9F:69:79:16:66:90:02:1B:8C:8C:A2:C3:07:6F:3A
verisignclass2ca, Mon Jun 29 21:06:39 GMT+04:00 1998, trustedCertEntry,
Certificate fingerprint (MD5): EC:40:7D:2B:76:52:67:05:2C:EA:F2:3A:4F:65:F0:D8

C:\jbuilder5\jdk1.3\bin>
```

**Рисунок 1. Просмотр содержимого keystore.**

Поскольку нам эти сертификаты не нужны, можно убрать их командой `-delete` следующего вида :

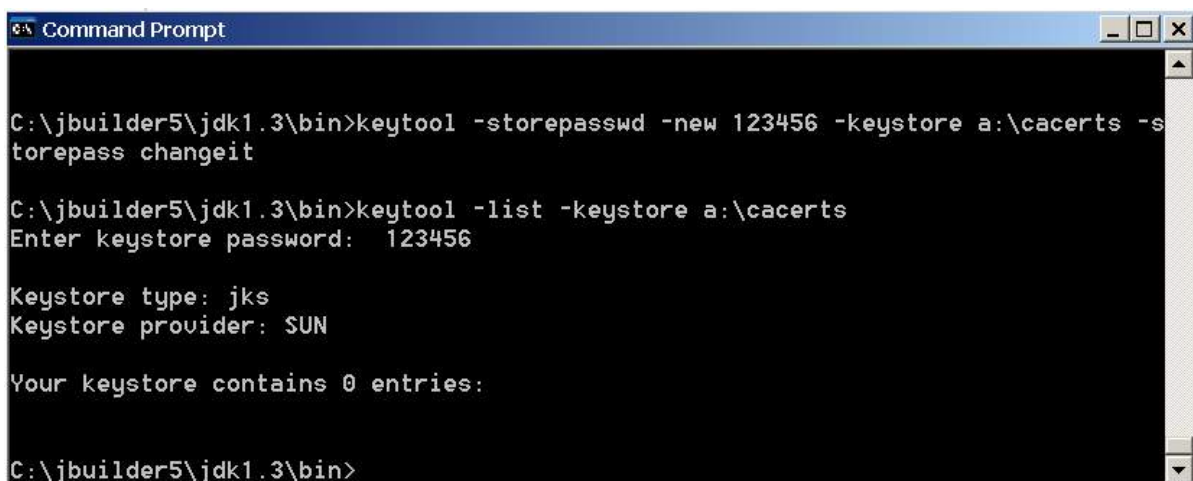
*Системы безопасности для клиентов. Создание доверенного апплета с использованием криптосистем открытого ключа*



```
Command Prompt
C:\jbuilder5\jdk1.3\bin>keytool -delete -alias verisignclass2ca -keystore a:\cacerts
Enter keystore password: changeit
C:\jbuilder5\jdk1.3\bin>
```

**Рисунок 2. Удаление сертификатов.**

вызывая ключи по алиасам можно убрать их все, после чего поменять пароль

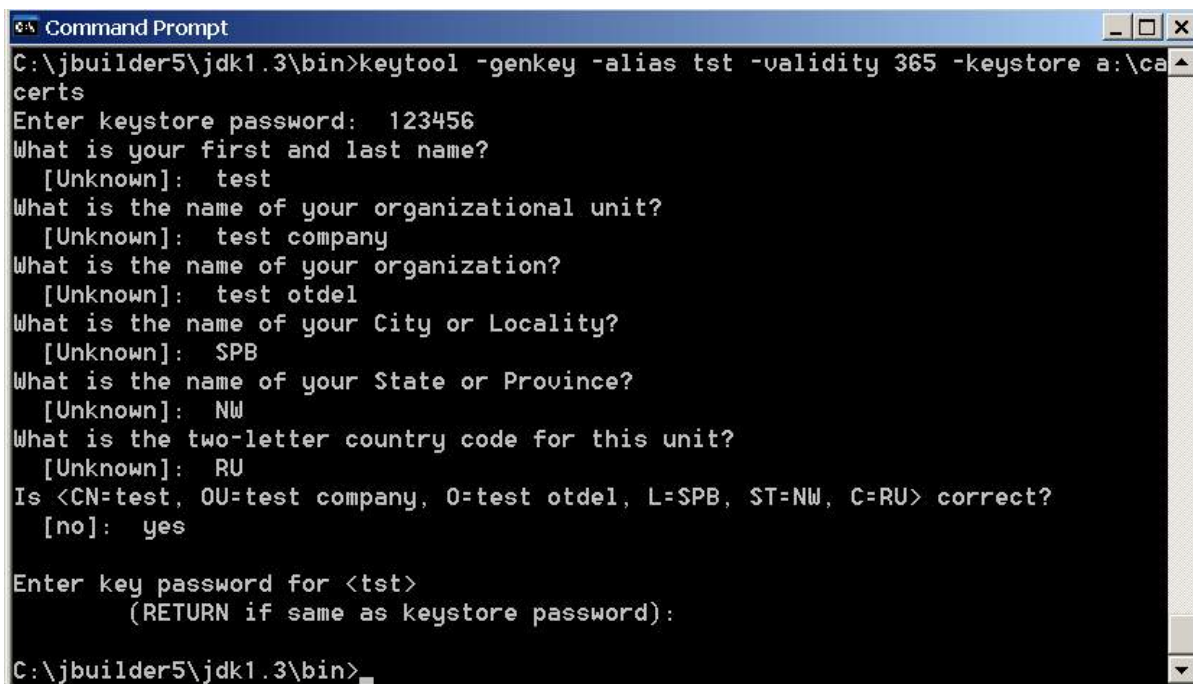


```
Command Prompt
C:\jbuilder5\jdk1.3\bin>keytool -storepasswd -new 123456 -keystore a:\cacerts -storepass changeit
C:\jbuilder5\jdk1.3\bin>keytool -list -keystore a:\cacerts
Enter keystore password: 123456
Keystore type: jks
Keystore provider: SUN
Your keystore contains 0 entries:
C:\jbuilder5\jdk1.3\bin>
```

**Рисунок 3. Смена паролей.**

Для простоты мы оставим единый пароль для keystore и alias'a — теперь можно создавать сам сертификат следующей командой:

## Системы безопасности для клиентов. Создание доверенного апплета с использованием криптосистем открытого ключа



```
Command Prompt
C:\jbuilder5\jdk1.3\bin>keytool -genkey -alias tst -validity 365 -keystore a:\cacerts
Enter keystore password: 123456
What is your first and last name?
 [Unknown]: test
What is the name of your organizational unit?
 [Unknown]: test company
What is the name of your organization?
 [Unknown]: test otdel
What is the name of your City or Locality?
 [Unknown]: SPB
What is the name of your State or Province?
 [Unknown]: NW
What is the two-letter country code for this unit?
 [Unknown]: RU
Is <CN=test, OU=test company, O=test otdel, L=SPB, ST=NW, C=RU> correct?
 [no]: yes

Enter key password for <tst>
 (RETURN if same as keystore password):

C:\jbuilder5\jdk1.3\bin>
```

Рисунок 4. Создание сертификата.

подсказка `jdk1.3\bin >` означает что алиас и его ключи успешно созданы. Сразу же вслед за созданием ключей имеет смысл создать и публичный сертификат для распространения в формате \*.cer — впоследствии с его помощью мы сможем сверять цифровую подпись клиента. Для этого создаем файл для сохранения сертификата, например на диске `A:\certificate.cer` и из приложения командной строки задаем команду следующего вида:



```
Command Prompt
C:\jbuilder5\jdk1.3\bin>keytool -export -alias tst -file a:\certificate.cer -keystore a:\cacerts
Enter keystore password: 123456
Certificate stored in file <a:\certificate.cer>

C:\jbuilder5\jdk1.3\bin>
```

Рисунок 5. Создание файла для сохранения сертификата.

которая в файл `A:\certificate.cer` импортирует публичные ключи вашего сертификата, и который может быть установлен на машину клиента Windows. Я использую упрощенный путь, когда для получения публичных ключей мы просто оставляем себе

## Системы безопасности для клиентов. Создание доверенного апплета с использованием криптосистем открытого ключа

копию клиентского keystore, что, хотя не совсем безупречно с правовой точки зрения, (хакеры просто отдыхают) бывает весьма удобно в практическом применении. Теперь, когда у вас есть все необходимые компоненты, можно приступить непосредственно к написанию кода апплета и сервлета для обработки его запроса.

Начнем с апплета, который должен решать следующие задачи:

- получать с сервера документ для подписи.
- демонстрировать его клиенту
- создавать цифровую подпись документа
- отсылать документ и подпись для проверки
- должен быть доверенным апплетом для доступа к `a:\cacerts`

Естественно, что любую задачу можно решить множеством способов, и какой из них лучше -- или хуже — следует рассматривать в контексте конкретной задачи, в нашем же случае, код апплета и HTML приспособлен для большей наглядности и мог бы быть приблизительно таким: HTML — форма для апплета:

```
<FORM name='sbm'> //пустая форма в которую javascript использует для
</FORM>           //отсылки на сервер — она позволит нам избежать открытия
                  //нового InputStream'a

<applet
MAYSCRIPT="MAYSCRIPT" //строка необходимая для выполнения javascript'a
codebase = ". "
ARCHIVE = "sign_applet.jar"
code = "sign_applet.class"
name = "TestApplet"
width = "600"
height = "450"
hspace = "0"
vspace = "0"
align = "middle"
>
<param name = "addressFrom" value =
"http://naprimer. ru/test. txt"> //параметр — адрес файла для подписи
<param name = "addressTo" value =
"http://naprimer. ru/yourServlet"> //параметр — адрес получателя-сервлета
```

и сам апплет:

## Системы безопасности для клиентов. Создание доверенного апплета с использованием криптосистем открытого ключа

```
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.lang.Byte;
import java.applet.*;
import java.io.*;
import java.net.*;

        //блок импорта для цифровой подписи
import java.security.spec.*;
import java.security.cert.*;
import java.security.*;
import java.security.interfaces.*;
import java.math.*;

        //блок импорта для JSObject
import netscape.javascript.*;

public class sign_applet extends Applet {

    KeyStore ks;
    InputStream is = null;
    DataInputStream dis = null;
    boolean isStandalone = false;
    byte b[]=null;
    TextArea ar;
    TextField tf;
    Button bt;
    String func = "func";
    String addressFrom = "";
    String addressTo = "";
    String buttonTxt ="send your signature to other side";

    public void init()
{
    //в котором получаем основные параметры
    //исходящий и входящий адреса
    //и создаем визуальную оболочку апплета
    setSize(600, 450);
    setBackground(Color.white);
}
```

**Системы безопасности для клиентов. Создание доверенного апплета с использованием криптосистем открытого ключа**

```
//для показа документа клиенту
    ar = new TextArea(25, 80);
//без права редактирования
    ar. setEditable(false);
    ar. setBackground(Color.white);
//для введения пароля
    tf= new TextField(40);
//для запуска процедуры подписи и отправки
    bt = new Button(buttonTxt);
    bt. setBackground(Color.white);
//попытка получить входящий и исходящий адреса...
    try{
        addressFrom = getParameter("addressFrom", "address not found");
        addressTo = getParameter("addressTo", "address not found");
    }
    catch(Exception e) {}
}

public void start()
{
    add(ar);//добавляем кнопки и поля в апплет
    add(tf);
    add(bt);
    try {

//програмно создаем keystore на клиенте
        ks = KeyStore.getInstance("JKS", "SUN");

//и берем FileInputStream из его keystore на дискете
        is = new FileInputStream("a:\\cacerts");

//обращаемся по адресу - за подписываемым файлом
        URL url = new URL(addressFrom);

//раз он есть - открываем URLConnection
        URLConnection urlConn = url.openConnection();

//из него получаем InputStream и заворачиваем его в BufferedInputStream
        BufferedInputStream bf =
```

## Системы безопасности для клиентов. Создание доверенного апплета с использованием криптосистем открытого ключа

```
        new BufferedInputStream((InputStream)
        urlConn.getInputStream() );

// ByteArrayOutputStream понадобится
// для заполнения байтового массива b[]
        ByteArrayOutputStream bas =
        new ByteArrayOutputStream();

//промежуточный счетчик
        int bit;

//пока поток не пуст, читаем из него и сразу пишем в bas
        while((bit = bf.read() ) != -1)
        {
            bas.write(bit);
        }

//поток пуст, все что в нем есть – кладем в массив байт
        b = bas.toByteArray();

//не изменяя массив байт b[]
//выводим строку клиенту для просмотра
        String a = new String(bas.toString("UTF-8") );

//Юникод обеспечит корректное отображение русских букв при условии что
//для плагина jdk1.3 установлена локализация RU и язык – русский. Кодировка
//на сервере также должна быть UTF-8 – для более подробной информации смотри
//статью С. Астахова 'Русские буквы и не только...'

        writer (a+"\r\n");
    }
    catch(Exception e) {e.toString();}
}

public void writer(String str)
    //просто добавляет строку в TextArea
    {
        ar.append(str+"\r\n");
    }
}
```

**Системы безопасности для клиентов. Создание доверенного апплета с использованием криптосистем открытого ключа**

```
// метод запуска процедуры подписи и отправки на
// сервер по нажатию кнопки - метод управления

    public boolean action(Event evt, Object obj)
    {
        String tx = "";
        if(buttonTxt.equals(obj) )
        {
//получить пароль клиента
            tx = tf.getText();

//подписать
            String sign = sign(b, tx);

//и отослать ее на сервер javascript'ом
            doScript(sign, new String(b) );
        }
        return true;
    }

//Самое интересное. На входе подаем заверяемый документ в виде массива
// байт b[] и пароль клиента который он ввел в поле апплета,
//на выходе получаем строку подписи

    public String sign(byte[]b, String client_mess)
    {
        String signdoc = "";
        try
        {

//is - поток из файла "a:\cacerts",
// client_mess.toCharArray() -это пароль =
        '1', '2', '3', '4', '5', '6';
            ks.load(is, client_mess.toCharArray() );

//Enumeration получает имена(алиасы) из keystore клиента
//в нашем случае - одно имя 'tst'
            Enumeration aliaslist = ks.aliases();
```

## Системы безопасности для клиентов. Создание доверенного апплета с использованием криптосистем открытого ключа

```
//RSAPrivateKey – ключ для подписи будет создан если в keystore
//клиента найдется алиас к которому подойдет пароль
//обратите внимание на тип DSAPrivateKey -для подписи типа SHA1withDSA
//по умолчанию, но возможен и тип RSAPrivateKey – для MD5withRSA
//дающий ошибку java.lang.ClassCastException:
//sun.security.provider.DSAPrivateKey

        RSAPrivateKey priv = (RSAPrivateKey) ks.getKey
            (aliaslist.nextElement().toString() , client_mess.toCharArray() );

//объект Signature – инстанциация цифровой подписи
        Signature sign = Signature.getInstance("MD5withRSA");

//с данным priv – уникальным ключом
        sign.initSign(priv);

//с массива байт b[]
        sign.update(b);

//byte[] realSig – он и есть по сути цифровая подпись для массива b[]
//при условии данного ключа, пароля и времени примерно такого вида
// 2 20 81 6 120 -122 60 -106 102 14 60 42 -35 51 105
        byte[] realSig = sign.sign();

//подпись получена, теперь ее можно отсылать для проверки на сервер
//к слову, получена она может быть только один раз – другая подпись
//потребуется повторной инициализации апплета
//мы не ищем проблем – поэтому сразу переводим ее в строку –
//из интовых значений – чтоб избежать проблем при пересылке. . .

        for(int z=0; z<realSig.length; z++)
            signdoc += " " + Byte.toString(realSig[z]);

        return signdoc;
    }catch(Exception e)
    {
        System.out.println(e.toString() );
        return "false";
    }
}
```

## Системы безопасности для клиентов. Создание доверенного апплета с использованием криптосистем открытого ключа

```
//метод отправки строки подписи и оригинала на сервер который
//на входе получает строку подписи и оригинальную строку и используя
//JavaScript и HTML форму 'sbm' отсылает их серверу...

    void doScript(String sig, String doc) {
//если все параметры переданы корректно, то скрипт должен сработать...
    try{
        JavaScript win = (JavaScript) JavaScript.getWindow(this);
        win.eval("f = document. forms['sbm']");
        win.eval("f.method = 'post'");
        win.eval("f.action =  '"+ addressTo +'");

        win.eval("inputTag = document.createElement('input' ");
        win.eval("inputTag.name = 'doc'");
        win.eval("inputTag.type = 'text'");
        win.eval("inputTag.value = '"+doc+"'");
        win.eval("f.appendChild(inputTag) ");

        win.eval("inputTag = document. createElement('input' ");
        win.eval("inputTag.name = 'sig'");
        win.eval("inputTag.type = 'text'");
        win.eval("inputTag.value = '"+sig+"'");
        win.eval("f.appendChild(inputTag) ");

        win.eval("f.submit() ");
        win = null;
    } catch(Exception t) {}
}

//типовой метод получения параметров из HTML

    public String getParameter(String key, String def) {
return isStandalone ? System. getProperty(key, def) :
    (getParameter(key) != null ? getParameter(key) : def);
}

    public void stop()
    { }
```

*Системы безопасности для клиентов. Создание доверенного апплета с использованием криптосистем открытого ключа*

```
public void destroy()
{ }

public String getAppletInfo()
{return ""; }

}
```

не забудьте, что перед загрузкой апплет должен быть заверен подписью (объявлен как trusted) локального keystore распространителя апплета. В результате у вас получится следующий апплет, где содержимое TextArea, естественно, будет зависеть от вашего файла на сервере:

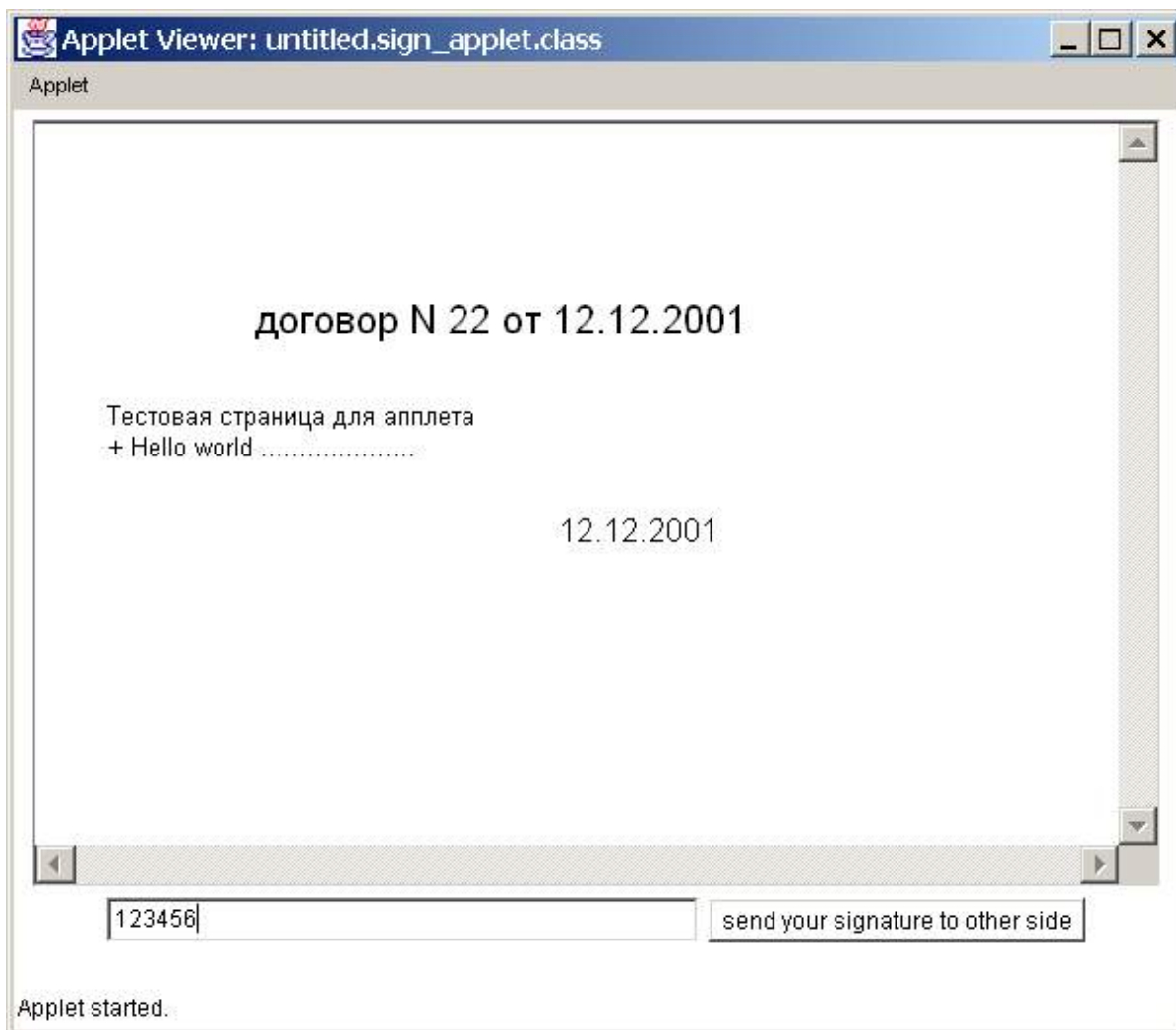


Рисунок 6. Тестовый апплет.

## 4. Серверная часть

На этом можно временно оставить апплет в покое и обратить внимание на сторону принимающую сообщение: как в апплете, так и на стороне сервера возможны разные варианты построения приложения — например можно упростить пересылку данных отправляя серверу только строку подписи — при условии что клиент не может изменять изначальный текст сообщения такой вариант вполне работоспособен, однако,

## *Системы безопасности для клиентов. Создание доверенного апплета с использованием криптосистем открытого ключа*

в реальном приложении возможны любые вариации -- поэтому мы постараемся рассмотреть наиболее универсальный способ.

Теперь нам нужен сервлет который бы решал нижеследующие задачи:

- мог бы распознавать обращающегося клиента и запрос от клиента
- фильтровать из request'a переменные отправленные нами из апплета
- обращаться к БД за публичными ключами данного авторизованного клиента
- на основании ключа и данных запроса-провести проверку сообщения на подлинность
- если проверка положительна — занести документ и подпись в БД

```
import javax.servlet.*;
import javax.servlet. http.*;
import java.io.*;
import java.util.*;

//загрузка пакетов для работы oracle с Blob
import java.sql.*;
import oracle.sql.*;
import oracle.jdbc.*;

//пакеты для верификации подписи
import sun.security.pkcs.PKCS10;
import java.security.spec.*;
import java.security.cert.*;
import java.security.*;
import java.security.interfaces.*;

public class verify_sign extends HttpServlet
{
    Connection conn;
    Blob keystore;
    private static final String CONTENT_TYPE =
        "text/html;charset=Windows-1251";
    HttpSession session;

    public void init(ServletConfig config)
        throws ServletException
```

**Системы безопасности для клиентов. Создание доверенного апплета с использованием криптосистем открытого ключа**

```
{
    super.init(config);
}

public void putsession(HttpSession ses) {
    session = ses;
}

// обработка request'a
public void service(HttpServletRequest
request, HttpServletResponse response)
throws ServletException, IOException
{
    String er = "";
    String sig = "";
    String doc = "";
    boolean verifies;
    try
    {

// получение цифровой подписи и документа из request'a
byte[] sign_from_client = new byte[1024];
sig = request.getParameter("sig");

// StringTokenizer нужен для разделения
// строки на подстроки и извлечения байтов
// которые укладываем в массив sign_from_client[i]
StringTokenizer stoken = new StringTokenizer(sig, " ");
int i = 0;
while(stoken.hasMoreTokens() )
{
    sign_from_client[i] = Byte.parseByte(stoken. nextToken() );
    i++;
}
doc = request. getParameter("doc");

//инициализируем keystore
KeyStore ks = KeyStore. getInstance("JKS", "SUN");
```

## **Системы безопасности для клиентов. Создание доверенного апплета с использованием криптосистем открытого ключа**

```
//получаем connection к БД из сессии --
//предполагается, что мы его передали
//в сессию в момент логинизации клиента
    conn = (Connection) session.getValue("connection");

//получаем логин клиента из connection --
//используем то, что логин клиента
// совпадает с его alias в keystore...
    String login = conn.getMetaData().getUserName();

//запрашиваем keystore клиента из БД
    Statement stmt = conn.createStatement();
    ResultSet rset = stmt.executeQuery(
        "SELECT KEY_STORE FROM users WHERE
        LOGIN_NAME = '"+login.toUpperCase()+"'");
    rset.next();

//получаем из БД BLOB - копию keystore
    BLOB blob = ((OracleResultSet) rset).
        getBLOB("KEY_STORE");

//а из него - InputStream
    InputStream is1 = blob.getBinaryStream();

//получаем пароль клиента из request'a
//уложенный туда ранее или при логинизации
    char[] pass = request.getParameter("password").toCharArray();

//загружаем созданный keystore
    ks.load(is1, pass);

//по логину клиента из keystore
//получаем его сертификат - публичный ключ
    java.security.cert.Certificate certif =
        (java.security.cert.Certificate)
            ks.getCertificate(login);

//создаем объект sign типа Signature
    Signature sign = Signature.getInstance("MD5withRSA");
```

## Системы безопасности для клиентов. Создание доверенного апплета с использованием криптосистем открытого ключа

```
//и инициализируем проверку подписи
    sign. initVerify(certif);
    sign. update(doc.getBytes() );
    verifies = sign.verify(sign_from_client);
    stmt.close();
}
catch(Exception e)
{
    er = "Error: " + e. toString();
    verifies = false;
}

//выдаем какой-то ответ клиенту...

response. setContentType(CONTENT_TYPE);
PrintWriter out = response. getWriter();
out. println("<html>");
out. println("<head><title>
    </title></head>");
out. println("<body>");
out. println("<p>Sign is " + verifies + er+ " "+
    doc+ "**** "+sig+"</p>");
out. println("</body></html>");
}

//освобождаем ресурсы машины
public void destroy()
{
}
}
```

Умышленно не включен в сервлет код который бы производил запись в БД и опущены малопринципиальные детали — скорее всего, вам самим придется разработать свою реализацию зависящую от конкретной задачи и БД. По большому счету, процедура проверки проста, и если абстрагироваться от сопровождающих деталей, то ее можно свести к одному методу:

```
// sig    — подпись клиента в виде строки
// doc    — заверяемый документ с сервера или из БД
// login  — алиас клиента в keystore
// pass   — пароль этого алиаса
```

## Системы безопасности для клиентов. Создание доверенного апплета с использованием криптосистем открытого ключа

```
public boolean check(String sig, String doc,
                    String login, String pass)
{
    boolean verifies = false;
    byte[] sign_from_client = new byte[1024];

// StringTokenizer stoken нужен в
// случае когда вы передаете не массив байт,
// а строку int'ов
    StringTokenizer stoken = new StringTokenizer(sig, " ");
    int q=0;
    while(stoken.hasMoreTokens() )
    {
        sign_from_client[q] = Byte.parseByte
            (stoken.nextToken() );
        q++;
    }

    int i=0;
    char [] password = pass.toCharArray();
    try{
// в случае с вашей БД FileInputStream
// is1 вам нужно будет заменить
// на InputStream из BLOB'a возвращенного БД

        FileInputStream is1 = new FileInputStream("a:\\cacerts");
        ks = KeyStore.getInstance("JKS", "SUN");
        ks.load(is1, password);
        java.security.cert.Certificate certif =
            (java.security.cert.Certificate) ks.getCertificate(login);
        Signature sign = Signature.getInstance("SHA1withDSA");
        sign.initVerify(certif);
        sign.update(doc.getBytes() );
        verifies = sign.verify(sign_from_client);
    }catch(Exception e)
        {System.out.println(e.toString() );}
    return verifies;
}
```

## 5. Заключение

Предлагаемая схема далеко не безупречна, но вполне работоспособна при использовании jdk1.3. Самым очевидным ее недостатком на мой взгляд является то, что на стороне сервера сохраняется копия keystore клиента, и это значит, что в принципе существует возможность злонамеренного использования его в случае взлома БД или кражи информации. Утешает только то, что при корректно измененном пароле keystore'a затраты времени на подбор будут достаточно велики. Использование сертификата в формате \*.cer в конечном итоге позволяет победить и этот недостаток.

И, наконец, в качестве примера также можно привести такую реализацию верификации полностью соответствующую стандарту подписи openSSL на основании сертификата certificate.cer когда на сервере не установлена jdk и сообщение получает скрипт на языке Python:

```
def sslverify(login, message, sign) :
    /*
    */

    result = getstatusoutput('openssl dgst -verify '
+ file_cer + ' -signature '
+ file_sig + ' ' + file_mes)

    /*
    */

    if rfind(result[1], 'OK') %gt;= 0:
        return 1
    else:
        return 0
```

## 6. Ресурсы

- Rivest, R., Shamir, A., and L. Adleman 'A Method for Obtaining Digital Signatures and Public Key Crypto — systems'
- [Криптология и ее основные понятия.](#)
- статья ['Работа с ключами'](#)
- С. Астахов статья ['Русские буквы и не только...'](#)