

Системы безопасности для клиентов. Создание доверенного апплета в JDK1.3

Григорий Печеницын

Создание trusted (доверенных) апплетов наделенных правом обращения к локальной файловой системе на машине клиента в стиле JDK1.3 было и остается актуальной темой в кругу прикладных программистов — ряд задач в силу своей специфики могут быть выполнены только на стороне клиента и при этом установка программного обеспечения на клиенте не всегда желательна или просто невозможна, а также нередка ситуация, когда пересылка данных на сервер недопустима по каким-либо причинам связанным с безопасностью пересылаемых данных. В предлагаемой вашему вниманию серии статей рассматривается ряд возможных решений в области защиты распространяемых апплетов и пересылаемых данных с использованием алгоритмов RSA/DSA реализованная на базе JDK1.3.

Хочу сразу оговорить, что данная схема подписи апплетов пригодна лишь для клиентов которые могут установить плагин jdk1.3 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. Не гарантируя универсальных средств, замечу, что большинство версий trusted апплетов неплохо исполняются под управлением плагина который можно скачать по адресу java.sun.com/getjava/download.html с названием j2re-1_3_1_02 от SUN.

1. Сперва немного предыстории

Во многом популярность java в версии 1.1 была обусловлена возможностями которые для разработчиков открывали java-апплеты — на тот момент весьма привлекательной

казалась идея перенести часть нагрузки на клиентские машины и таким образом избежать нагрузки на сервер, появились и были предложены схемы использования ресурсов клиентских машин на базе java-платформы для проведения распределенных вычислений и уже было слышно прогнозы о создании сетевой java-машины и нового поколения интернета. Первая эйфория довольно быстро прошла — и не в последнюю очередь тому способствовала строгость ограничений накладываемой моделью безопасности браузеров на апплеты. Новый прекрасный мир не состоялся — конец мечты.

После того, как эта точка зрения приобрела популярность в сетевом сообществе, та же часть идеологов идеи сетевых распределенных вычислений недавно певших оду java резво отвернулась от апплетов и отправилась воспевать иные технологии. К сожалению во многом к этому привела позиция разработчиков браузеров пытавшихся изо всех сил применить к апплетам java самую строгую модель безопасности из всех какие только возможны — именно эта модель и привела апплеты в sandbox и она же отчасти послужила причиной отказа от использования апплетов — слишком велики оказались ограничения — запрет обращения к любым частям системы. Проще говоря — менеджер безопасности браузера не допустит ничего за пределами браузера его выполняющего, и уж тем более для апплета не существует файловой системы и локальных программ к которым апплет мог бы обращаться.

Однако, на все есть свое решение — нашлось оно и для ограничений накладываемых на апплет — компромисс был предложен в виде создания подписных (trusted) апплетов, которые избавлены от части ограничений модели безопасности и должны были отчасти решить проблему обхода модели безопасности браузера. По замыслу разработчиков jdk1.1 пользователь мог импортировать на свою машину сертификат в формате X509 и с помощью утилиты `javakey` из командной строки объявить его доверенным сертификатом для данного апплета — после чего апплеты загружаемые с этого узла сети и ссылающиеся на данный сертификат становились trusted (доверенными) для данной машины и могли обращаться к локальным файлам и системным вызовам машины. Что же ответили на предложенную схему пользователи? — свое однозначное ‘До свидания’ — слишком сложной для них оказалась предложенная схема доверения с использованием утилиты командной строки, и в сочетании с тем, что большинство апплетов не несли реальной функциональности и служили лишь для украшения страниц это привело к тому что пользователи стали

Системы безопасности для клиентов. Создание доверенного апплета в JDK1.3

игнорировать выполнение апплетов или просто отказались от их загрузки — да и сама java в версии jdk 1.2.2 стала скорее серверной платформой чем языком для выполнения на стороне клиента. К тому же появились и конкуренты — тот же flash5 который может осуществлять на стороне клиента простейшие вычисления достаточные для придания flash — роликам функциональности.

Таким образом, на момент выпуска jdk 1.3 апплеты успели пережить свой взлет и стать темой хотя и не безынтересной, но уже и не новой, со своей сложившейся неоднозначной репутацией. Возможно, именно из-за этого выход в составе jdk1.3 утилит keytool и jarsigner предназначенных для подписи апплета, создания и заверения цифровой подписи и не вызвало в среде разработчиков особого внимания. Между тем, с их помощью создание trusted апплетов уже не представляет особой сложности и гораздо более удобно чем в более ранних версиях jdk.

Однако, если с общей картиной и так все понятно, то как быть, когда перед тобой поставлена руководством реальная задача — создать апплет, имеющий право доступа к машине клиента — и нет простых описаний на русском которые как-нибудь бы проясняли данную тему, а апплет нужен как всегда — вчера ? Ответ — здесь.

2. Познакомимся с инструментами

Помочь вам может поставляемые в составе jdk1.3 утилиты командной строки keytool.exe и jarsigner.exe, keytool — для создания и управления собственными парами public/private ключей и для администрирования файла keystore — хранилища ключей на вашей машине и jarsigner — для подписи jar-файла (апплета) и аутентификации цифровой подписи распространяемой вместе с апплетом.

Чтоб понять, кто есть кто в этой схеме, следует обратиться к документации jdk1.3, в частности, к страницам java.sun.com/j2se/1.3/docs/tooldocs/win32/jarsigner.html java.sun.com/j2se/1.3/docs/tooldocs/win32/keytool.html

Однако, в качестве короткого пояснения замечу лишь, что:

- keystore — хранилище сертификатов на вашей машине,
- keytool.exe — утилита для администрирования keystore и создания сертификатов.
- jarsigner.exe — утилита -инструмент для непосредственной подписи апплета

сертификатом созданным с помощью `keytool.exe` и хранящегося в `keystore`.

3. Последовательность действий

В общих чертах вам следует соблюсти следующую последовательность действий:

1. Создать `keystore` в вашей `C:\Documents and Settings\user name` (домашней) директории.
2. Создать в `keystore` Ваш персональный сертификат подписанный вашими данными.
3. Подписать апплет созданным сертификатом.

Что для этого следует иметь изначально?

1. На вашей машине должна быть установлена `jdk 1.3`. В директории `jdk 1.3/bin` находится утилита `keytool.exe` и `jarsigner.exe` — они-то и будут вашим главным инструментом.
2. У вас должен быть создан ваш файл `test_applet.jar` — файл который вы предполагаете заверить и распространять как доверенный апплет.
3. В вашей домашней директории `C:\Documents and Settings\имя пользователя` должен находиться файл `keystore` — именно он и является хранилищем ключей и сертификатов на вашей машине — с ним нам и предстоит работать. Если его не существует — не отчаивайтесь — создадим сами.

Процедура создания `*.jar` файла достаточно подробно и интуитивно понятно описана в документации `Jbuilder5/Archive Builder` и не является предметом рассмотрения данной заметки. Замечу только, что механизм цифровой подписи для файлов с расширением `*.class` не предусмотрен.

4. Итак, приступим

Войдя в режим командной строки из любой утилиты командной строки (`FAR`, `Norton`, `Command Prompt`) переместитесь в директорию `jdk 1.3/bin`. Из этой директории вам станут доступны утилиты `keytool` и `jarsigner`. В командной строке набираете команду `keytool` — в ответ получаете длинный список опций утилиты — это означает что утилита доступна для использования. Для полной уверенности можно повторить вызов для `jarsigner` — результат должен быть аналогичен — в ответ получаем список опций утилиты. Если `keystore` до сих пор не существует на вашей машине, то именно сейчас самое время его создать — и заодно и ваш персональный

Системы безопасности для клиентов. Создание доверенного апплета в JDK1.3

сертификат которым предполагается подписывать ваш апплет — делается это командой такого вида:

`keytool -genkey` со следующими опциями:

```
-genkey {-alias alias} {-keyalg keyalg} {-keysize keysize} {-sigalg sigalg} [-dname dname]
[-keypass keypass] {-validity valDays} {-storetype storetype} {-keystore keystore}
[-storepass storepass] [-provider provider_class_name] {-v} {-Jjavaoption}
```

Среди которых стоило бы отдельно пояснить следующие наиболее важные:

`{-alias alias}` — псевдоним (имя) под которым ваш сертификат будет помещен в `keystore` и по которому вы впоследствии будете к нему обращаться.

`{-keyalg keyalg}` — алгоритм шифрования вашей подписи — обычно по умолчанию это алгоритм SHA1 with DSA и его можно не указывать если не собираетесь изменять его, размер ключа при генерации DSA ключевой пары может быть от 512 до 1024 бит, но если вы хотите применить MD5 with RSA то укажите опцию `-keyalg "RSA"` Следует заметить что опция `-keyalg` обуславливает и опцию `-sigalg` — алгоритм подписи который будет использован по умолчанию для подписания jar-файла.

`[-keypass keypass]` — пароль для доступа к создаваемому вами сертификату — если опция опущена будет использован пароль используемый для `keystore`. Впоследствии в самом конце диалога вам будет предложено ввести его еще раз.

`{-validity valDays}` — срок годности вашего сертификата. По умолчанию это 180 дней, можно указать больше или меньше — по вашему усмотрению.

`{-keystore keystore}` — если не указано, то при вызове `-genkey` в директории `C:\Documents and Settings\имя пользователя` (являющейся `keystore`-директорией по умолчанию) будет создан новый `keystore`, в противном случае в существующий будет добавлен новый сертификат.

`{-storepass storepass}` — пароль для создаваемого `keystore`, не менее 6 СИМВОЛОВ.

`{-storetype storetype}` — по умолчанию `jks(JavaKeyStore)`, но может быть изменен на `pkcs12`.

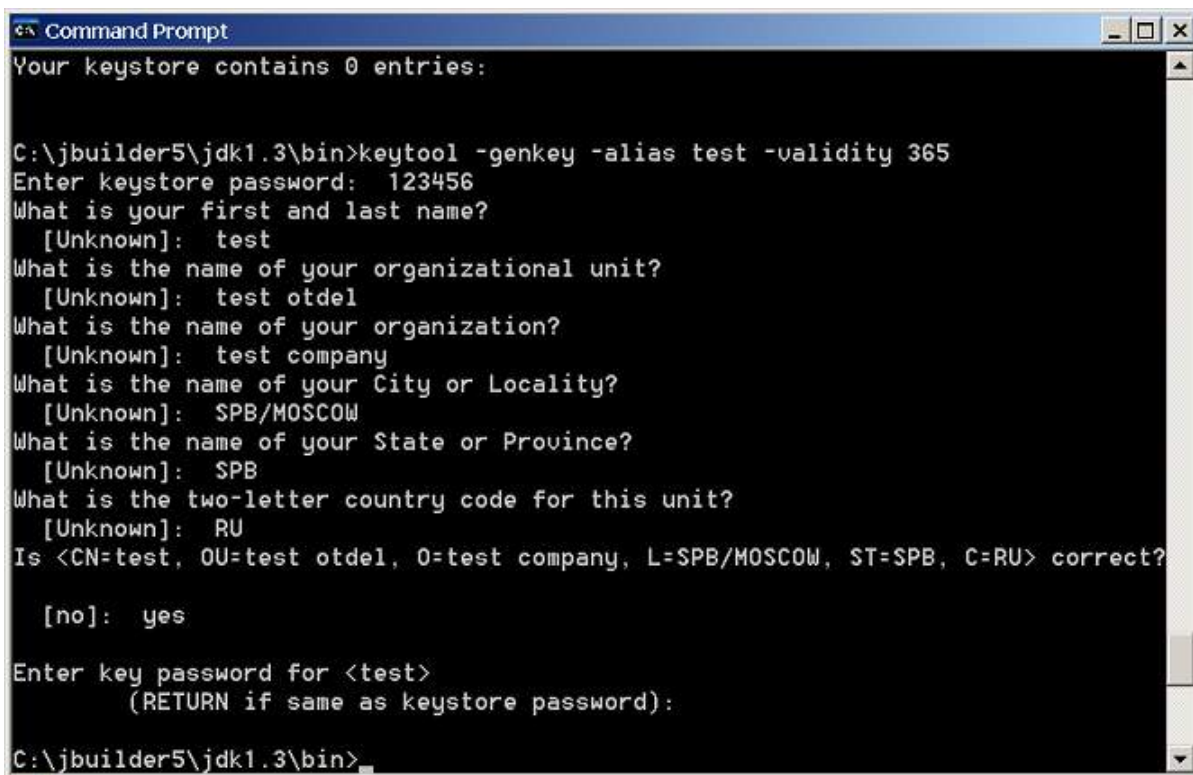
Системы безопасности для клиентов. Создание доверенного апплета в JDK1.3

В результате выбора опций в командной строке у вас будет набрано примерно следующее:

```
C:\jdk1.3\bin\keytool> -genkey -alias test -validity 365
```

 тогда все минимально необходимые опции будут запрошены и введены вами в диалоге сразу после введения команды — ваше диалоговое окно должно быть таким:

В первой строке диалога пароль нового keystore будет назначен; если keystore уже существует — естественно, прежний пароль.



```
Command Prompt
Your keystore contains 0 entries:

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

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

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

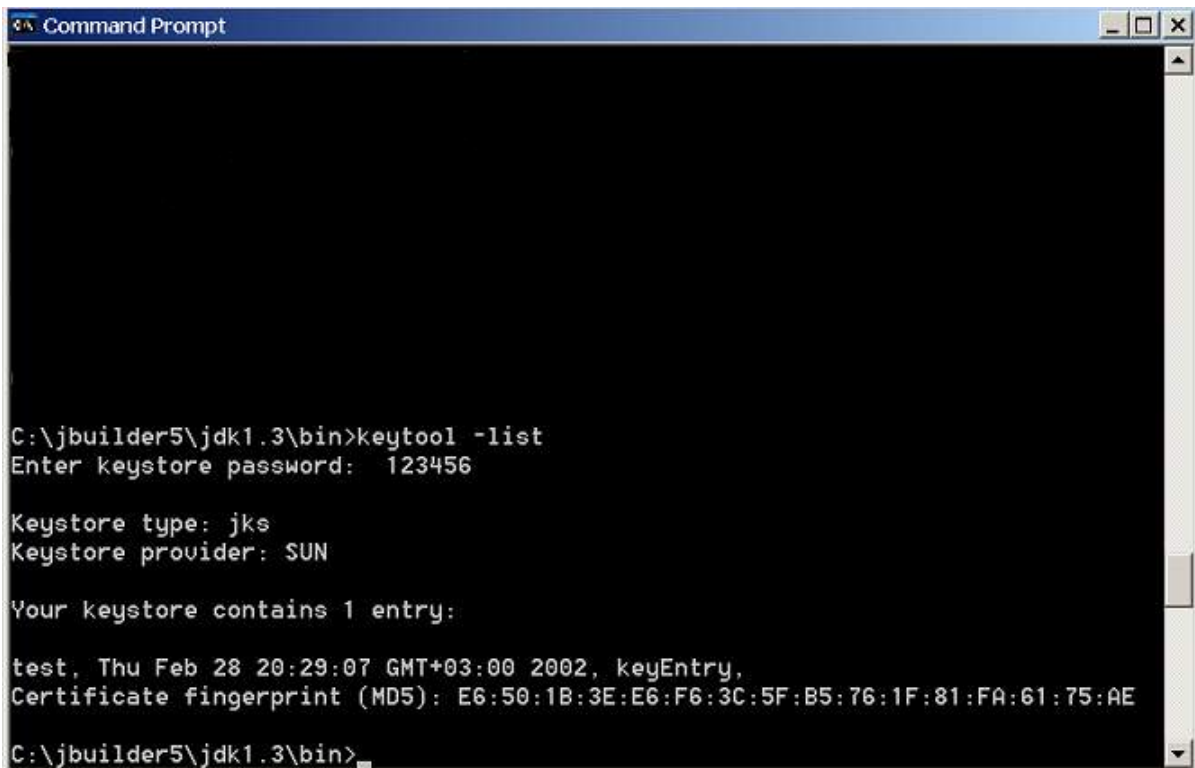
Рисунок 1. Создание сертификата с помощью keytool.

C:\jdk1.3\bin> — появление этой строки означает успешное создание сертификата. К слову говоря, хотелось бы предостеречь от бездумного присвоения паролей и данных вашему сертификату — это не будет способствовать безопасности и не повысит доверия пользователей к апплету. Для сравнения представьте свою собственную реакцию на строку с вопросом : "Do you want to install and run applet destributed by ХАКЕР.RU?". Тут-то и вспомнишь что "Не все йогурты одинаково

Системы безопасности для клиентов. Создание доверенного апплета в JDK1.3

полезны".

Для проверки содержимого keystore из командной строки следует набрать команду `keytool -list` которая выведет в окно диалога содержимое хранилища :



```
Command Prompt

C:\jbuilder5\jdk1.3\bin>keytool -list
Enter keystore password: 123456

Keystore type: jks
Keystore provider: SUN

Your keystore contains 1 entry:

test, Thu Feb 28 20:29:07 GMT+03:00 2002, keyEntry,
Certificate fingerprint (MD5): E6:50:1B:3E:E6:F6:3C:5F:B5:76:1F:81:FA:61:75:AE

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

Рисунок 2. Проверка содержимого keystore.

5. Подробнее о keytool

Следует заметить что `keytool.exe` обеспечивает полный набор для операций с сертификатами хранящимися в вашем keystore — список опций утилиты и описание можно вызвать уже упомянутой выше командой `keytool` без опций, однако упомяну наиболее часто употребляемые :

`keytool -genkey -alias (имя сертификата) [опции]` — более подробно описана выше, замечу лишь, что вызов `-import` или `-importkey` также создает новый keystore, если он не существовал ранее.

`-delete -alias (имя сертификата) [опции]` — уничтожает из keystore указанный в псевдониме(-alias'e) сертификат.

`-export -alias (имя сертификата) -file (имя файла) [опции]` — команда экспорта сертификата в файл с расширением *.cer для экспорта на другие машины.

`-import -alias (имя сертификата) -file (имя файла) [опции]` — команда импорта сертификата из файла с расширением *.cer или X509 сертификата, однако, естественно что никто вам не даст права (в данном случае — пары ключей) для подписи от своего имени — вам доступно только объявление данного импортированного сертификата как "trusted" для своей машины. Интересна опция `-trustcacerts` позволяющая заодно поместить мобильную копию импортированного сертификата в cacerts (будет рассмотрен позже).

`-storepasswd -new (нов.парсворд) -storepass (ст.парсворд)` — команда смены пароля для keystore

`-keypasswd -alias (имя сертификата) -keypass (прежний пароль) — new (новый пароль)` команда смены пароля для отдельного сертификата из keystore.

`-printcert -file (путь к файлу/имя файла)` — команда вывода на экран сертификата из указанного файла cer или X509.

`-sertreq -alias(имя сертификата) -file (имя файла)` — позволяет создать в указанном файле самоподписанный сертификат в формате pkcs#10 для отправки и заверения третьей стороной, такой как VeriSign или Thawte.

6. Подписываем jar-файл

Однако, вернемся к нашим подписным апплетам: мы уже успешно преодолели пол-дела — нами создан keystore и в нем находится наш сертификат — следующей простой задачей становится подписание апплета в форме jar-файла этим сертификатом. Для этого нам понадобится утилита jarsigner.exe. Если все описанные выше были выполнены корректно, то утилита командной строки которой вы пользуетесь находится в директории C:\jdk1.3\bin и jarsinger.exe доступен прямым вызовом jarsigner; если нет — то перейдите в директорию. Для удобства проще всего

Системы безопасности для клиентов. Создание доверенного апплета в JDK1.3

скопировать jar с апплетом прямо в директорию `jdk1.3\bin` и задать команду в виде

```
C:\jdk1.3\bin>jarsigner test_applet.jar test
```

где `test_applet.jar` — имя jar-файла вашего апплета `test` — алиас созданного вами сертификата в `keystore`

`C:\jdk1.3\bin>` на этот раз означает успешное подписание апплета.

Проверка `MANIFEST.MF` в jar'e показывает что он действительно подписан:

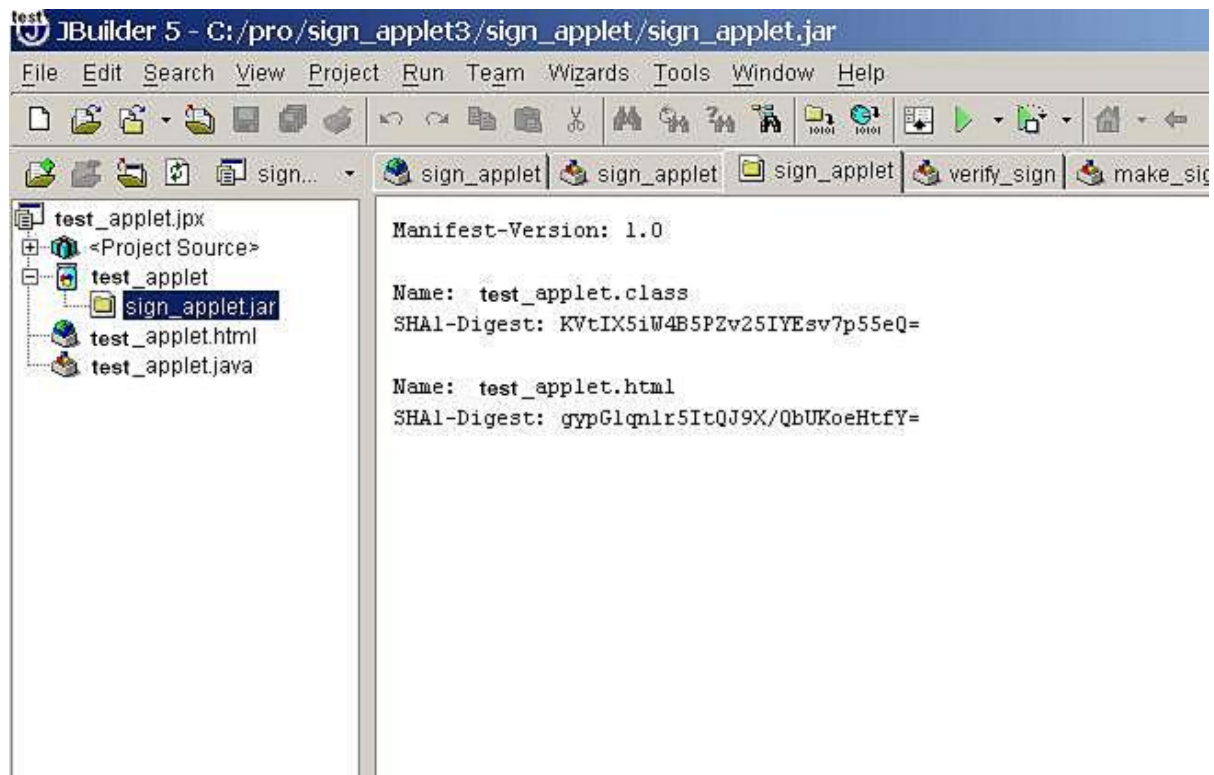


Рисунок 3. Проверка `MANIFEST.MF` в подписанном jar архиве.

В JAR директории папки `META-INF` появились файлы `test.dsa` и `test.sf` — мобильная цифровая подпись вашего апплета. Теперь при запуске апплета будет выводиться табличка с вопросом "Do you want to install and run applet destributed by...."? — собственно апплет готов. Какие права ему даны ? — в таблице документации java.sun.com/jdk1.3/docs/tooldocs/win32/jarsigner.html описаны права `trusted` апплетов в зависимости от версии `jdk` или `java`-плагила под управлением которого исполняется

апплет — там же можно найти исчерпывающее описание всех функций утилиты, а по ссылке java.sun.com/jdk1.3/docs/tooldocs/win32/keytool.html — описание функций keytool.exe — именно на базе этих описаний написана эта заметка.

7. Заключение

В следующей статье из серии будут описано практическое применение trusted апплета на примере банковского платежного документа выставляемого клиенту банком для заверения в котором используется процедура идентификации клиента на основании его персональной digital signature (цифровой подписи).