

# Практическая криптография в Java. Асимметричная криптография, методы безопасного обмена информацией

Николай Жишкевич

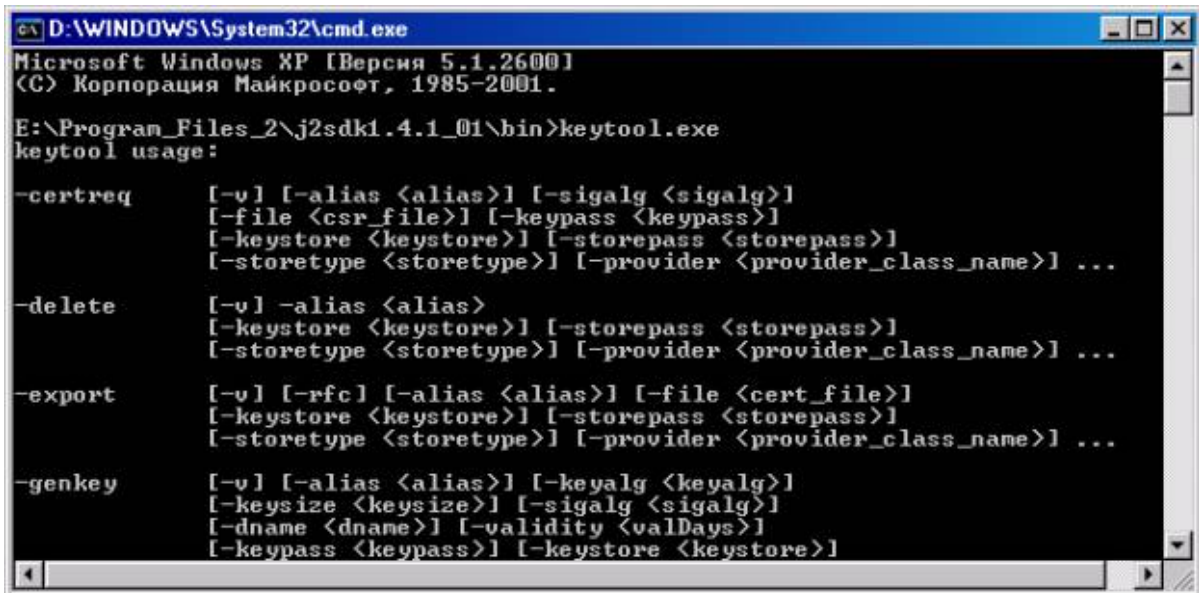
## 1. Создание сертификатов. Инструменты в составе JSDK и их использование

Сертификаты представляют собой цифровое «удостоверение личности», которое может быть использовано для установления соединения по протоколу SSL&TLS. Они также могут быть использованы для проверки подлинности клиента или сервера.

Мы поставили перед собой задачу получить сертификат для сервера и установить его туда. Серьезные приложения, например веб-сервер MIIIS, Apache-SSL и др. имеют инструменты для создания запросов на сертификацию. Хотя упомяну далее о схеме получения данного электронного документа на примере MIIIS 5.0&5.1, но раз данный материал посвящен в основном Java, то и инструментарий я рекомендую соответствующий.

Тем, кто дружит с Java, можно воспользоваться специальным специальной утилитой `keytool` для создания ключей и хранилища сертификатов. Разумеется, что просто созданный набор ключей не имеет практической ценности, пока он будет подписан соответствующим издателем. На момент написания данного материала на сайте VeriSign проводилась рекламная акция в рамках которой можно было получить сертификат для сервера бесплатно. Правда срок его действия ограничивался двумя неделями, но для того чтобы попробовать насколько это полезная вещь вполне хватает.

*Практическая криптография в Java. Асимметричная криптография, методы безопасного обмена информацией*



```
D:\WINDOWS\System32\cmd.exe
Microsoft Windows XP [Версия 5.1.2600]
(C) Корпорация Майкрософт, 1985-2001.

E:\Program_Files_2\jdk1.4.1_01\bin>keytool.exe
keytool usage:

-certreq      [-v] [-alias <alias>] [-sigalg <sigalg>]
              [-file <csr_file>] [-keypass <keypass>]
              [-keystore <keystore>] [-storepass <storepass>]
              [-storetype <storetype>] [-provider <provider_class_name>] ...

-delete       [-v] -alias <alias>
              [-keystore <keystore>] [-storepass <storepass>]
              [-storetype <storetype>] [-provider <provider_class_name>] ...

-export       [-v] [-rfc] [-alias <alias>] [-file <cert_file>]
              [-keystore <keystore>] [-storepass <storepass>]
              [-storetype <storetype>] [-provider <provider_class_name>] ...

-genkey       [-v] [-alias <alias>] [-keyalg <keyalg>]
              [-keysize <keysize>] [-sigalg <sigalg>]
              [-dname <dname>] [-validity <valDays>]
              [-keypass <keypass>] [-keystore <keystore>]
```

Теперь мы с помощью инструмента `keytool` создадим хранилище ключей и поместим туда несколько созданных ключей, которые мы выдадим гипотетическим организациям: “Terrana Inc.”, “Venera corp.”, “Mars ltd.”. Сначала мы просто запустим данную утилиту и получим список параметров, с помощью которых мы можем ее запускать для выполнения определенных задач.

```
E:\Program_Files_2\jdk1.4.1_01\bin>keytool.exe
```

Нас интересует опция `-genkey`. Именно с помощью ее мы сможем создать пару закрытого и открытого ключа и поместить в выбранное хранилище, если оно существовало до этого момента или если же его не было ранее, то оно будет автоматически создано. С помощью параметра `-keystore` мы укажем местоположение данного хранилища в файловой системе. Другим важным параметром будет `-alias` — именно с помощью его мы укажем псевдоним, под которым созданная пара ключей будет храниться в хранилище. Я подчеркиваю, что `alias` — это не имя или название организации, которой будет выдан данный сертификат, а только его псевдоним в данном хранилище и в общем случае не имеет никакого практического значения за пределами данного хранилища ключей.

Всегда следует помнить, что в основе асимметричной криптографии и, соответственно, ЭЦП ради которой мы сейчас попытаемся создать пару ключей лежат определенные

## *Практическая криптография в Java. Асимметричная криптография, методы безопасного обмена информацией*

математические зависимости. Наиболее распространенными алгоритмами на данный момент являются древний, но отлично работающий и широко применяемый алгоритм RSA, названный так в честь его создателей, являющихся одними из основателей всей математики двух ключевых криптопреобразований.

Также используется цифровая подпись Эль-Гамала и DSS (Digital Signature Standard) принятый институтом NIST как стандарт ЭЦП. Поэтому при создании пары ключей, следует указать с помощью какого именно алгоритма будет создана данная пара. Следующим параметром является указание на размер ключа, который будет создан. Размер ключа влияет на затраты времени необходимые для его вскрытия. Заметьте, я не говорю, что алгоритмы ЭЦП нельзя вскрыть или обмануть, теоретически задача компрометации криптосистемы решается, вопрос только в затратах времени. Необходимо при указании размера ключа помнить, что разница между сложностью взлома симметричного алгоритма и асимметричного есть и она не в пользу асимметричного.

Если для большинства применений симметричной криптографии длины ключа в сотню бит будет достаточно, то в случае асимметричной необходимые размеры исчисляются в тысячах. Принято для несерьезных применений использовать длину ключа 512, 768, а для коммерческих приложений необходима длина 1024 или 2048 бит. Если вы откроете любой сертификат и посмотрите его свойства, то среди них найдете и длину ключа и сможете убедиться в моих словах.

Следующий параметр `-sigalg` — задает алгоритм генерации ключа и именно цифровой подписи. Для того чтобы понять, зачем нужен следующий параметр, необходимо напомнить, что созданные нами ключи в будущем будут применяться для организации безопасного обмена информацией. Следует помнить также и о том, что нападения на цифровую подпись включают в себя не только попытки на основе известного открытого ключа вычислить закрытый, но и более опасные приемы, в которых идет попытка обмана или подмены.

Так для ЭЦП остается актуальной такая проблема как проблема «первого контакта». Я о ней уже упоминал, но считаю важным напомнить еще раз. Как один участник процесса общения может быть уверенным, что тот ключ, который ему прислал или опубликовал его собеседник действительно ему принадлежит. С целью решения данной проблемы открытые ключи собеседников публикуются в своеобразной обертке

— сертификате. В нем хранится не только ключ, но и информация о сроках в течении которых данный ключ действителен, а также информация о той организации, которая данный ключ выдала предъявителю и удостоверяет это соответствие, а в подтверждение этого подписывает своим закрытым ключом данный сертификат.

Итак, мы сказали что в сертификате, который хранит в себе открытый ключ некоторого лица также должна присутствовать информация о сроке в течение которого данный сертификат будет действителен. Хочу сразу предупредить, что сертификаты не выдаются задним числом. Поэтому мы можем указать только количество дней, в течение которых они будут действительны. Вся информация, которую мы помещаем в хранилище ключей должна быть закрыта, и если к открытым ключам это не столь важно, то для хранящихся в том же файле закрытых ключах должна быть применена более жесткая система безопасности. Пожалуй, говорить о том, что само хранилище и ключи в нем по отдельности закрыты с помощью пароля не стоит. Для задания пароля к хранилищу служит параметр `-storepass` и ключ для записи о секретном ключе `-keypass`.

Остается также еще несколько параметров не столь существенных и задающих провайдера криптографических услуг. Я надеюсь, вы помните, что в пакете `java.security` и других содержатся в основном не классы, выполняющие конкретные действия, а наборы интерфейсов. Конкретное же их наполнение создается различными компаниями, разрабатывающими библиотеки криптографических алгоритмов, что очень важно данные библиотеки должны соответствовать стандартным принятым интерфейсам, для обеспечения универсальности создаваемого кода, и возможности легкой миграции с одного продукта на другой.

Теперь я создам хранилище ключа и помещу в него ключ для ранее придуманных мною гипотетических организаций. Пока беспокоиться о том, что нет организации согласной подписать (удостоверить) наши сертификаты нет, и они не имеют практической ценности, не стоит все это будет впереди.

Практическая криптография в Java. Асимметричная криптография, методы безопасного обмена информацией

```
C:\WINNT\system32\cmd.exe
D:\Program_Files_2\JBuilder8\jdk1.4\bin>keytool -genkey -keystore "H:\DocsXP\My
Programming\jbuilder\keystores\store" -alias Terrana -validity 365
Enter keystore password: bigsecret
What is your first and last name?
  [Unknown]: Terrana inc.
What is the name of your organizational unit?
  [Unknown]: Solar System corp.
What is the name of your organization?
  [Unknown]: Universum ltd.
What is the name of your City or Locality?
  [Unknown]: Minsk
What is the name of your State or Province?
  [Unknown]: Republic Of Belarus
What is the two-letter country code for this unit?
  [Unknown]: BY
Is CN=Terrana inc., OU=Solar System corp., O=Universum ltd., L=Minsk, ST=Republi
c Of Belarus, C=BY correct?
  [Inol]: y

Enter key password for <Terrana>
  <RETURN if same as keystore password>:

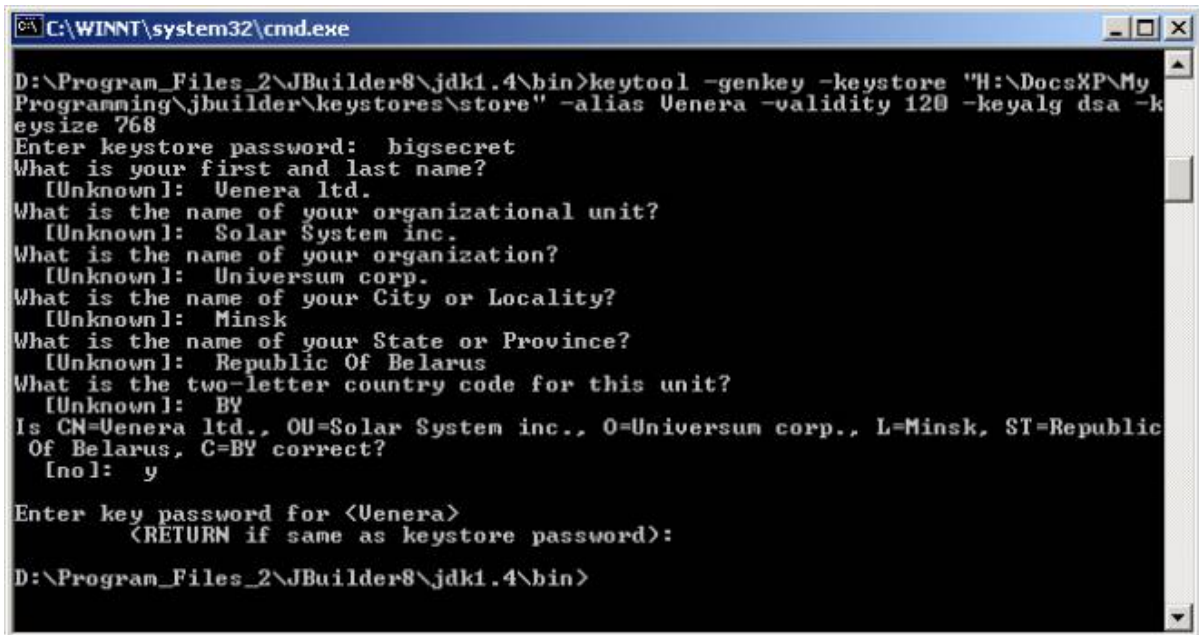
D:\Program_Files_2\JBuilder8\jdk1.4\bin>_
```

```
C:\WINNT\system32\cmd.exe
D:\Program_Files_2\JBuilder8\jdk1.4\bin>keytool -genkey -keystore "H:\DocsXP\My
Programming\jbuilder\keystores\store" -alias Mars -validity 600 -keyalg dsa
Enter keystore password: bigsecret
What is your first and last name?
  [Unknown]: Mars inc.
What is the name of your organizational unit?
  [Unknown]: Solar System inc.
What is the name of your organization?
  [Unknown]: Universum ltd.
What is the name of your City or Locality?
  [Unknown]: Minsk
What is the name of your State or Province?
  [Unknown]: Republic Of Belarus
What is the two-letter country code for this unit?
  [Unknown]: BY
Is CN=Mars inc., OU=Solar System inc., O=Universum ltd., L=Minsk, ST=Republic Of
Belarus, C=BY correct?
  [Inol]: y

Enter key password for <Mars>
  <RETURN if same as keystore password>:

D:\Program_Files_2\JBuilder8\jdk1.4\bin>_
```

*Практическая криптография в Java. Асимметричная криптография, методы безопасного обмена информацией*



```
C:\WINNT\system32\cmd.exe
D:\Program_Files_2\JBuilder8\jdk1.4\bin>keytool -genkey -keystore "H:\DocsXP\My
Programming\jbuilder\keystores\store" -alias Venera -validity 120 -keyalg dsa -k
eysize 768
Enter keystore password: bigsecret
What is your first and last name?
  [Unknown]: Venera ltd.
What is the name of your organizational unit?
  [Unknown]: Solar System inc.
What is the name of your organization?
  [Unknown]: Universum corp.
What is the name of your City or Locality?
  [Unknown]: Minsk
What is the name of your State or Province?
  [Unknown]: Republic Of Belarus
What is the two-letter country code for this unit?
  [Unknown]: BY
Is CN=Venera ltd., OU=Solar System inc., O=Universum corp., L=Minsk, ST=Republic
Of Belarus, C=BY correct?
  [no]: y

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

D:\Program_Files_2\JBuilder8\jdk1.4\bin>
```

После этого в каталоге, который я указал, как местоположение хранилища ключей появится новый файл с именем store. Если вы захотите просмотреть его содержимое, то увидите только непонятный набор символов. Говорить о том, что вся информация зашифрована и не хранится в открытом виде, пожалуй, не стоит. По аналогии я создам еще несколько сертификатов, но для них укажу дополнительные атрибуты. В первом случае я воспользовался тем, что для размера ключа, вида алгоритма приняты умолчания, сейчас же я попытаюсь указать информацию более подробную.

```
E:\Program_Files_2\j2sdk1.4.1_01\bin>keytool.exe
-genkey -keystore
  "H:\docs_xp\My Programming\jbuilder\keystores\store"
-alias Mars -validity 600 -keyalg dsa
...

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

**Замечание:**

Будьте внимательны, если вы ошибетесь в вводе данных или зададите неверные значения для параметров командной строки, то сообщение об ошибке вы получите не сразу, а только после ввода всех данных.

## Практическая криптография в Java. Асимметричная криптография, методы безопасного обмена информацией

Например далее я ошибусь в вводе длины ключа для алгоритма dsa, я задам его равным 1001, что явно недопустимо. И получу соответствующее сообщение об ошибке.

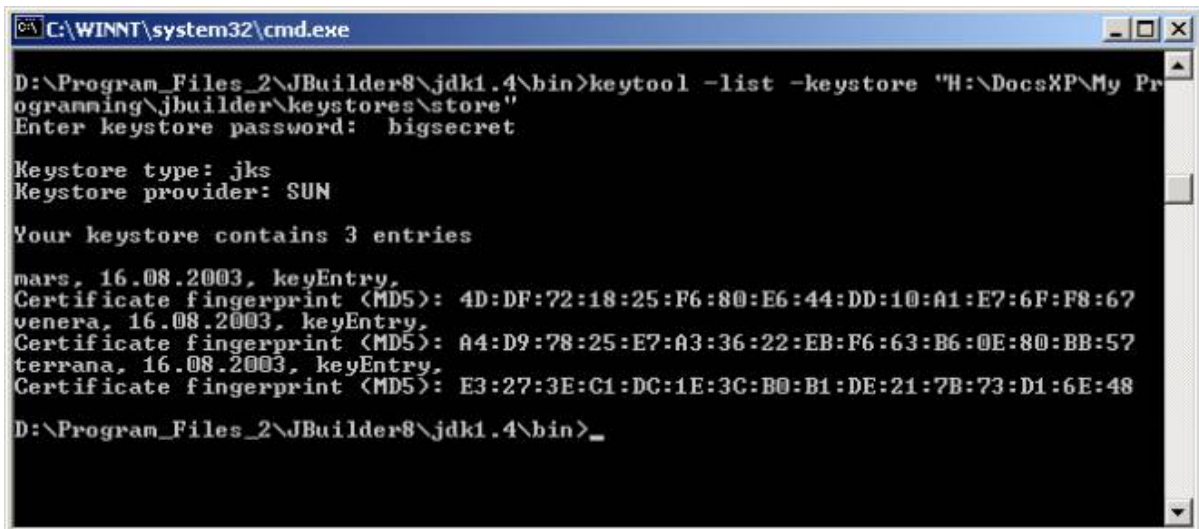
```
keytool error: Java.lang.IllegalArgumentException:
  Modulus size must range from
  512 to 1024 and be a multiple of 64

E:\Program_Files_2\jdk1.4.1_01\bin>keytool.exe
-genkey -keystore
  "H:\docs_xp\My Programming\jbuilder\keystores\store"
-alias Venera -validity 120 -keyalg dsa -keysize 768

...

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

А теперь, посмотрим содержимое данного хранилища ключей. Сделать это можно с помощью параметра командной строки `-list`.



```
C:\WINNT\system32\cmd.exe
D:\Program_Files_2\JBuilder8\jdk1.4\bin>keytool -list -keystore "H:\DocsXP\My Pr
ogramming\jbuilder\keystores\store"
Enter keystore password: bigsecret

Keystore type: jks
Keystore provider: SUN

Your keystore contains 3 entries

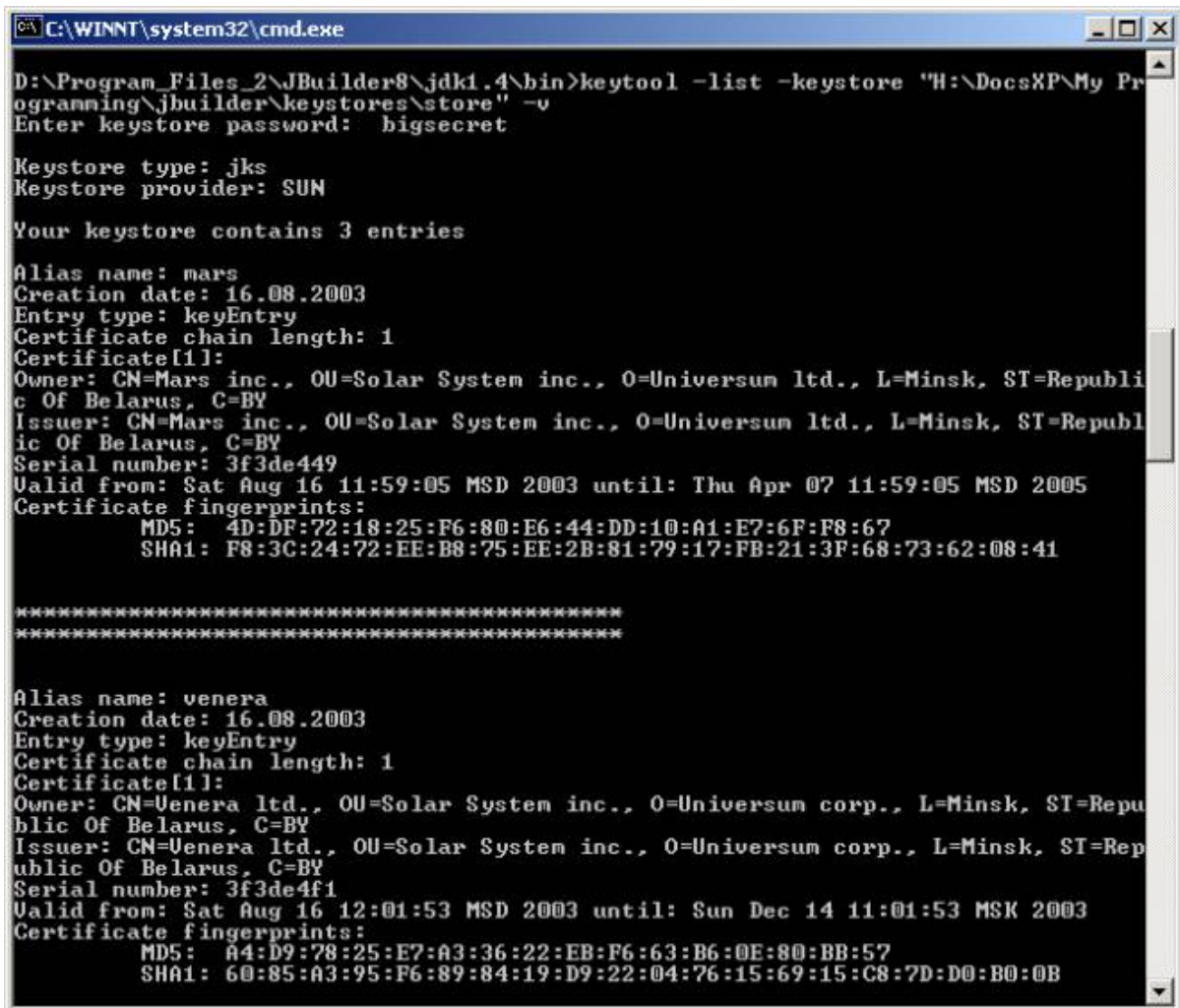
mars, 16.08.2003, keyEntry,
Certificate fingerprint (MD5): 4D:DF:72:18:25:F6:80:E6:44:DD:10:A1:E7:6F:F8:67
venera, 16.08.2003, keyEntry,
Certificate fingerprint (MD5): A4:D9:78:25:E7:A3:36:22:EB:F6:63:B6:0E:80:BB:57
terrana, 16.08.2003, keyEntry,
Certificate fingerprint (MD5): E3:27:3E:C1:DC:1E:3C:B0:B1:DE:21:7B:73:D1:6E:48

D:\Program_Files_2\JBuilder8\jdk1.4\bin>_
```

Рассмотрим более внимательно вывод содержимого данного хранилища ключей. В первой строке содержится информация о типе хранилища ключа (jks - Java keystore). Затем идет информация о провайдере криптографических услуг. И, наконец, содержимое. Для каждой записи в хранилище выводится псевдоним, под которым он в

*Практическая криптография в Java. Асимметричная криптография, методы безопасного обмена информацией*

нем зарегистрирован, дата создания, алгоритм с помощью которого была вычислена свертка данного сертификата (MD5). Разумеется, что подобный вывод информации нас не устраивает, т.к. он слишком скуден. Более подробные сведения можно получить, указав параметр `-v`. результат его применения я привожу далее. Согласитесь, что сведения более подробны. Кстати обратите внимание, какие характеристики у записи `terrana`. Помните, что когда мы ее создавали, то не указали ряд параметров.



```
C:\WINNT\system32\cmd.exe
D:\Program_Files_2\JBuilder8\jdk1.4\bin>keytool -list -keystore "H:\DocsXP\My Programming\jbuilder\keystores\store" -v
Enter keystore password: bigsecret

Keystore type: jks
Keystore provider: SUN

Your keystore contains 3 entries

Alias name: mars
Creation date: 16.08.2003
Entry type: keyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=Mars inc., OU=Solar System inc., O=Universum ltd., L=Minsk, ST=Republic Of Belarus, C=BY
Issuer: CN=Mars inc., OU=Solar System inc., O=Universum ltd., L=Minsk, ST=Republic Of Belarus, C=BY
Serial number: 3f3de449
Valid from: Sat Aug 16 11:59:05 MSD 2003 until: Thu Apr 07 11:59:05 MSD 2005
Certificate fingerprints:
    MD5:  4D:DF:72:18:25:F6:80:E6:44:DD:10:A1:E7:6F:F8:67
    SHA1: F8:3C:24:72:EE:B8:75:EE:2B:81:79:17:FB:21:3F:68:73:62:08:41

*****
*****

Alias name: venera
Creation date: 16.08.2003
Entry type: keyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=Venera ltd., OU=Solar System inc., O=Universum corp., L=Minsk, ST=Republic Of Belarus, C=BY
Issuer: CN=Venera ltd., OU=Solar System inc., O=Universum corp., L=Minsk, ST=Republic Of Belarus, C=BY
Serial number: 3f3de4f1
Valid from: Sat Aug 16 12:01:53 MSD 2003 until: Sun Dec 14 11:01:53 MSK 2003
Certificate fingerprints:
    MD5:  A4:D9:78:25:E7:A3:36:22:EB:F6:63:B6:0E:80:BB:57
    SHA1: 60:85:A3:95:F6:89:84:19:D9:22:04:76:15:69:15:C8:7D:D0:B0:0B
```

Обратите внимание, что имя владельца сертификата и имя организации, которая его выдала, совпадают, впрочем, другого и не ожидалось, ведь мы выдали его сами себе. Ниже я упомяну о том, в каких ситуациях практикуют подобный подход.

## Практическая криптография в Java. Асимметричная криптография, методы безопасного обмена информацией

Содержимое сертификата можно вывести также и в стандартизированной форме. Которая будет понятна другим приложениям, умеющим работать с форматом сертификатов X.509. За это отвечает параметр `-rfc`. Результат его работы приводится ниже.

```
C:\WINNT\system32\cmd.exe
D:\Program_Files_2\JBuilder8\jdk1.4\bin>keytool -list -rfc -keystore "H:\DocsXP\
My Programming\jbuilder\keystores\store"
Enter keystore password: bigsecret

Keystore type: jks
Keystore provider: SUN

Your keystore contains 3 entries

Alias name: mars
Creation date: 16.08.2003
Entry type: keyEntry
Certificate chain length: 1
Certificate[1]:
-----BEGIN CERTIFICATE-----
MIIDQDCCAvCBDB895EkwCwYHkoZiZjgEAWUAMI GEMQsuCQYDUQQGEwJCWT EcmBoGA1UECBMTUmUw
dWJsaWMgT2YgQmUsYXJ1czEOMAwGA1UEBxMFTWluc2sxZmFzAU BgNUBAoTD1UuaXZlcnNi bS BsdGQu
MR0wGAYDUQQLExFTb2xhc iBTExN0ZW0gaW5jLjESMBA GA1UEAxMjTWYyc yBpbmMuMB4XDTAzMDgx
NjA3NTkwNUoXDTA1MDQwNzA3NTkwNUowgYQxCzAJBgNUBA YTAkZMRwwGgY DUQqI ExNSZX B1Ym xp
YyBPZiBCZwXhcncUzMQ4wDAYDUQQHEuNaW5zaz EXMBUGA1UEChMOUW5pdmU yc3UtIGx0ZC4xGjAY
BgNUBA sIEUNuhGFyIFNSc3RlbSBpbmMuMRI wEAYDUQQDEwlnYXJzIGluYy4wggGAMI IBLAYHkoZi
ZjgEATCCAR8CgYEA/x9TgR1iEi1S30qcLuzk5/YRt1I870QAwx4/gLZRJm lFXUA iUftZPY1Y +r/F
9bow9subUWzXgTuAHTRv8mZgt2uZUKWkn5/oBHsQIsJPu6nX/rfGG/g7U +fGqKYUDwI7g/bTxR7D
AJUUE1oWkIL2dfOuK2HXKu/yIgmZndFI AccCFQCXYFCPFSMLzLKSuYKi64QL8Fgc9QKBgQD34aCF
1ps93su8q1w2uFe5eZSuu/o66oL5U0wLPQeCZiFZU4661F1P5nEHEI GatEkWcSPoTCgWE7fPCTKM
yKhhPBZ6i1R8jSjgo64eK70mdZFuo38L+iE1VvH7YnoBJDvMpg+qFGQiaID3+Fa5Z8GkotmXoB7
USUkAUw7/s9JKgOBhQACgYEA lqY5U1ZhHMujw17P1t8zH3oE1smUX10M1ofcxTQ8eioJYdzR26Ug
h2myHsdsrW8anhULHuQUtTqo92WuOdMCtId9ogS2j9CnnEiBUtH32bP1gSbmYx03BLEp8X61eR7L
uJhOmPvDd/WEpGMW1196tUgU9n5xR1NPuTRDIYP30H8wCwYHkoZiZjgEAWUAAzAAMC0CFAYF1aAH
2ICcnN3GzUHChBoZntcpAhUAhcXUkZp1tUwLd6Zicrk66TB9Qy0=
-----END CERTIFICATE-----

*****
*****

Alias name: venera
Creation date: 16.08.2003
Entry type: keyEntry
Certificate chain length: 1
Certificate[1]:
-----BEGIN CERTIFICATE-----
MIIC3TCCApsCBDB895PEwCwYHkoZiZjgEAWUAMI GHMQsuCQYDUQQGEwJCWT EcmBoGA1UECBMTUmUw
dWJsaWMgT2YgQmUsYXJ1czEOMAwGA1UEBxMFTWluc2sxZmFzAU BgNUBAoTD1UuaXZlcnNi bSBjb3Jw
```

Для иллюстрации я перенаправил вывод данной информации в файл под расширением `“.cer”`. А затем просмотрел его содержимое с помощью инструмента управления сертификатами Windows. Содержимое данного файла находится в кодировке base64.

А далее я привожу копию экрана, в котором открыт данный сертификат. Как вы

*Практическая криптография в Java. Асимметричная криптография, методы  
безопасного обмена информацией*

видите, на экране значок сертификата изображен вместе со знаком «ошибка», дело в том, что созданный нами сертификат представляет по своей сути сертификат, который выдается сам себе. И, разумеется, по умолчанию доверия к таким сертификатам нет. Подобная выдача сертификата самой себе практикуют организации, которые сами занимаются выдачей сертификатов, либо те организации, которые создают собственные версии сертификатов не нуждающиеся в чем-либо внешнем подтверждении. Разумеется, для практического применения данного сертификата необходимо, чтобы пользователь компьютера импортировал данный сертификат в свое хранилище доверенных сертификатов.



Очевидно, что выполнение столь несложной задачи как экспорт и импорт сертификатов должно быть автоматизировано. И действительно утилита `keytool` может принимать следующие параметры командной строки: `-import`, `-export`,

## *Практическая криптография в Java. Асимметричная криптография, методы безопасного обмена информацией*

-printcert. Разобраться в их использовании несложно. В качестве иллюстрации я приведу пример, в котором извлекаю из хранилища сертификат под псевдонимом "mars".

```
E:\Program_Files_2\jdk1.4.1_01\bin>keytool.exe -export
-alias mars
-keystore "H:\docs_xp\My Programming\jbuilder\keystores\store"
-file mars.cer

Enter keystore password: bigsecretpassCertificate

stored in file <mars.cer>
```

Основное внимание я теперь уделю вопросу о том, как сделать наш самопальный сертификат хоть капельку заслуживающим доверия. Для этого нам необходимо заручиться поддержкой какой-нибудь организации имеющей право заниматься подобной деятельностью, выслать им наш сертификат и запасть некоторой денежной суммой. Да, к сожалению, подобные вещи не делаются бесплатно. Ведь ваш институт СА должен вам доверять, проверить вашу информацию. В сети достаточно много страшных историй о том, как злые хакеры получили сертификат, подписанный какой-нибудь серьезной организацией, и в очередной раз творят «страшное зло». Если вы счастливый обладатель Windows2000 или выше, то должны знать, что все драйвера, которые вы устанавливаете в системе, должны иметь ЭЦП, иначе, хоть система и дает возможность установки не сертифицированного драйвера, но всячески предупреждает о страшных последствиях и ужасных ошибках которые повлечет данный шаг. Вот вам простейший пример, а подобная политика все более широко начинает распространяться.

И последняя информация, которая будет необходима вам для дальнейшего чтения данного материала — это создание запроса к центру сертификации. Я создал сертификат для сервера, который называется comp (это его DNS имя) и отправил в VeriSign. После заполнения небольшой анкеты мне на почтовый ящик пришло письмо, в котором и находился сертификат для моего сервера. Обратите внимание на то, что мы до сих пор создавали сертификаты для организации, а не для сервера. Отличие в том, что вам необходимо будет указать вместо имени вашей организации полное доменное имя сервера, разумеется, без всяких <http://>, <ftp://> [www.abc.com:8080/foo](http://www.abc.com:8080/foo). Только доменное имя и ничего более, иначе VeriSign откажется в выдаче вам

## *Практическая криптография в Java. Асимметричная криптография, методы безопасного обмена информацией*

сертификата. Для того чтобы создать файл запроса сертификата я воспользовался утилитой `keytoolc` параметром `-certreq`.

После этого я получил специальный текстовый файл с запросом на подпись сертификата. Затем я отправился на сайт `verisign`, на главной странице которого была ссылка "trial ssl certificate". Если пойти по ней дальше, то пройдя через 5 шагов мастера можно получить уже заверенный сертификат. Никакой сложности это не составляет, благо каждый шаг сопровождается небольшим комментарием. К чести VeriSign будет сказать, что они на сайте приводят примеры с иллюстрациями схемы получения сертификатов для различных программных продуктов (MIS, Java, Apache и другие). Но на всякий случай я привожу пример копии экрана того шага где в специальную форму необходимо будет поместить содержимое файла созданного шагом ранее с помощью `keytool`.

**Step 2 of 5: Submit CSR**

Before you Start  
Step 1: Generate CSR  
• Step 2: Submit CSR  
Step 3: Complete Application  
Step 4: Install Test CA Root  
Step 5: Install your Test Server ID

**Submit CSR**  
When you generated the CSR in Step 1: Generate CSR, your server software either e-mailed the CSR to you, or created a request file on your hard disk (such as key.req). Open the CSR file with an ASCII text editor such as NotePad. (Do not use a word processor such as Word that inserts formatting or control characters.)

This is an example CSR file:

```
-----BEGIN NEW CERTIFICATE REQUEST-----
MIIBCTCBtAIBADBPHQswCQYDVQQGEwJVUzEQMA4GA1UECBMRmxvcm1kYTEYMBYG
A1UEChMPRXI1leyBvbiBUaGUgV2ViHRQvEgYDVQQDFAt3d3cuZXR3Lm5ldDBcMAOG
CSqGSIb3DQEBAQUAAOsAMEgCQQQCeojtjnHqgOGTxp+XZ56RaSe1iZWpumXjU6Sx7
v1FdXzeY1oLOQaO90Jtnu1UsQRHh0yDS+45oncjKmlzCG/IZAgMBAAAGqADANBgkq
hkiG9wOBAQQAANBAFBj9g+NiUh8YWPPrFGntgf4miUd/wqUshptjJy4PjdsD3ugy
5avvuh3G//PpGh2aYXIjHpJ>
-----END NEW CERTIFICATE REQUEST-----
```

**Именно здесь необходимо поместить содержимое файла созданного с помощью keytool -certreq**

Description	
<b>Enter CSR Information:</b> Copy the entire contents of the CSR file including the lines that contain the begin and end statements into the field on the right.	<pre>-----BEGIN NEW CERTIFICATE REQUEST----- MIIBzDCCATUCAQAvGYSxCzAJBgNVBAYTAktJZMSovKAYDV LCBNaW5zayBSZWdpb24xDjAMBgNVBAcTBU1pbmNrMRswG cC4xFDASBgNVBAstC1ZlbnV5Y3BsdGQuMQ0wCwYDVQQDE A4GNADCBiQKBgQDRedJH/YcK9Dsb8Sxow4OpVHRIC1ENs dxXUwc4+E4FNLqzFrIOtXdBQ10FgynqnZgN8nePMP5F2F jPxBpeWPPZgLnnyMpGenT5wd+LeHwhkS7QIDAQABoAAwE AugMRjUB03CrTjJP47eRrzSJ1LnSxqP71DAUxkP2oB8S SFvwpt5fkFJjarDteaDOM9Tcdw1iMNnvxPGzyY70s11I LqILwm2VzsI=</pre>

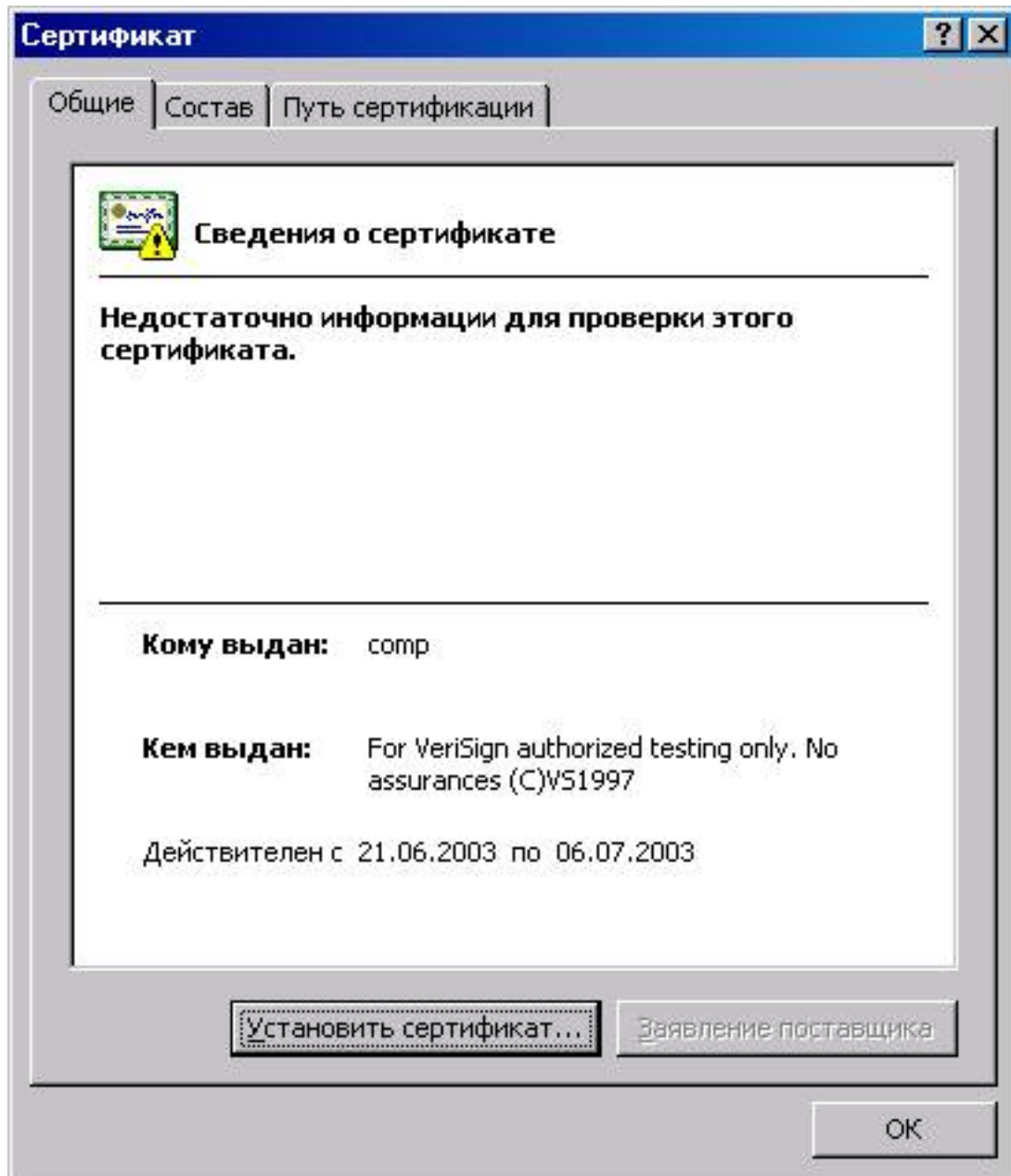
Click the CONTINUE button to submit the CSR and proceed with the Test Server ID enrollment. **Continue**

**Замечание:**

Шагов всего пять и в них нет ничего сложного, только помните что создаваемые по умолчанию сертификаты DSS не принимаются VeriSign, т.е. вам при создании сертификата необходимо будет указать что используемый алгоритм это RSA, делается это с помощью опции для keytool -keyalg rsa.

*Практическая криптография в Java. Асимметричная криптография, методы безопасного обмена информацией*

В письме, которое пришло от VeriSign, находился текст сертификата, который я скопировал в текстовый файл с расширением “cer”. Результат открытия данного сертификата я привожу на экране.



## *Практическая криптография в Java. Асимметричная криптография, методы безопасного обмена информацией*

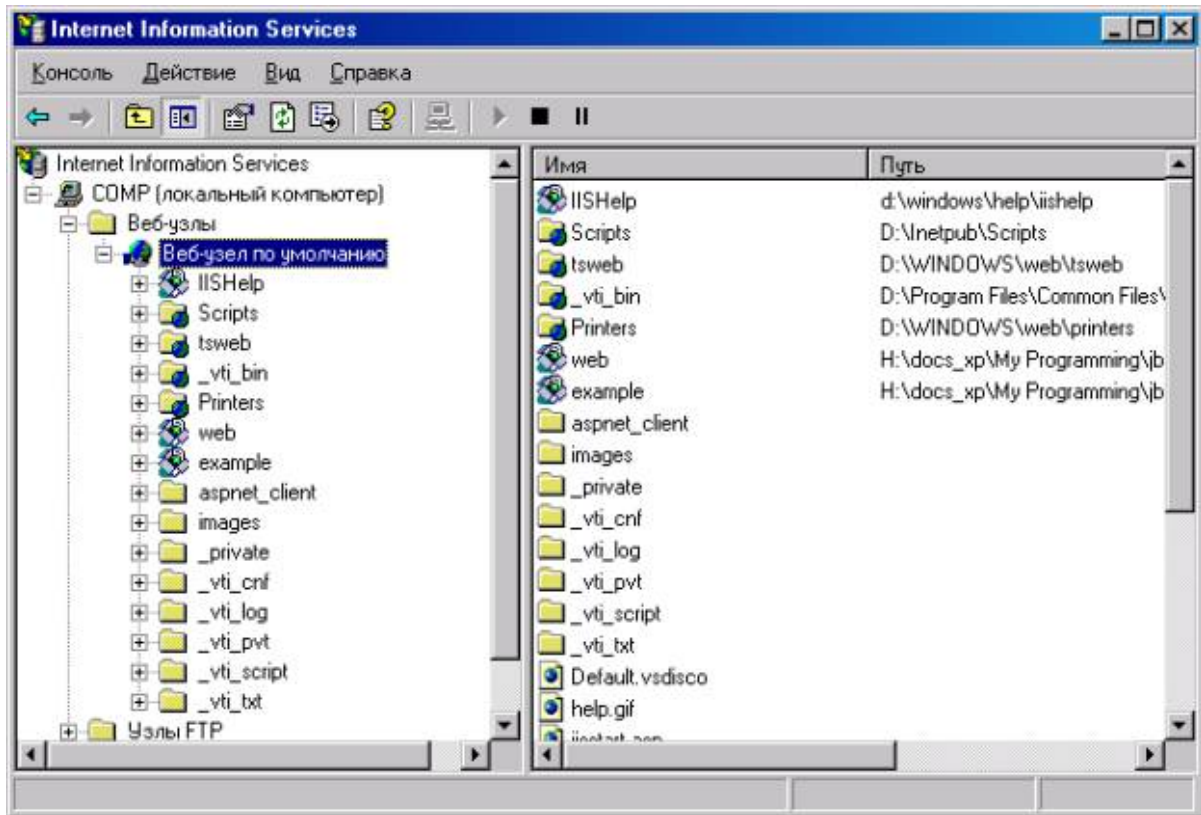
Остальные операции над хранилищем ключа, я уверен, вы сможете рассмотреть самостоятельно.

Если вы хотите создать сертификат для установки на другой веб-сервер, то следует обратиться к документации к нему. Я не могу рассмотреть последовательность действий для всех серверов. Как небольшое сравнение я выбрал схему создания сертификата или даже более точно запроса на сертификат в среде Microsoft IIS 5.1 или 5.0. По сути, любой пользователь Windows2000 или WindowsXP может установить у себя данный веб-сервер и поучиться администрировать его. В том, числе, что в данной статье наиболее важное, и попробовать получить сертификат сервера.

Я полагаю, что вы легко справитесь с установкой IIS, благо стараниями Microsoft это не сложнее чем установка MSOffice. Никаких сложностей с администрированием, удобный графический интерфейс, гораздо удобнее, чем работать с Apache. Стоп, стоп, я не за IIS и против Apache, я за то, что начинать изучение, пожалуй, стоит с более простых вещей. Если же у вас веб-ресурсы построены на использовании Apache, и вы хотите добавить поддержку работы данного веб-сервера по протоколу HTTPS, то следует либо добавить к веб-серверу Apache mod\_ssl или установить вообще отдельный продукт Apache-SSL.

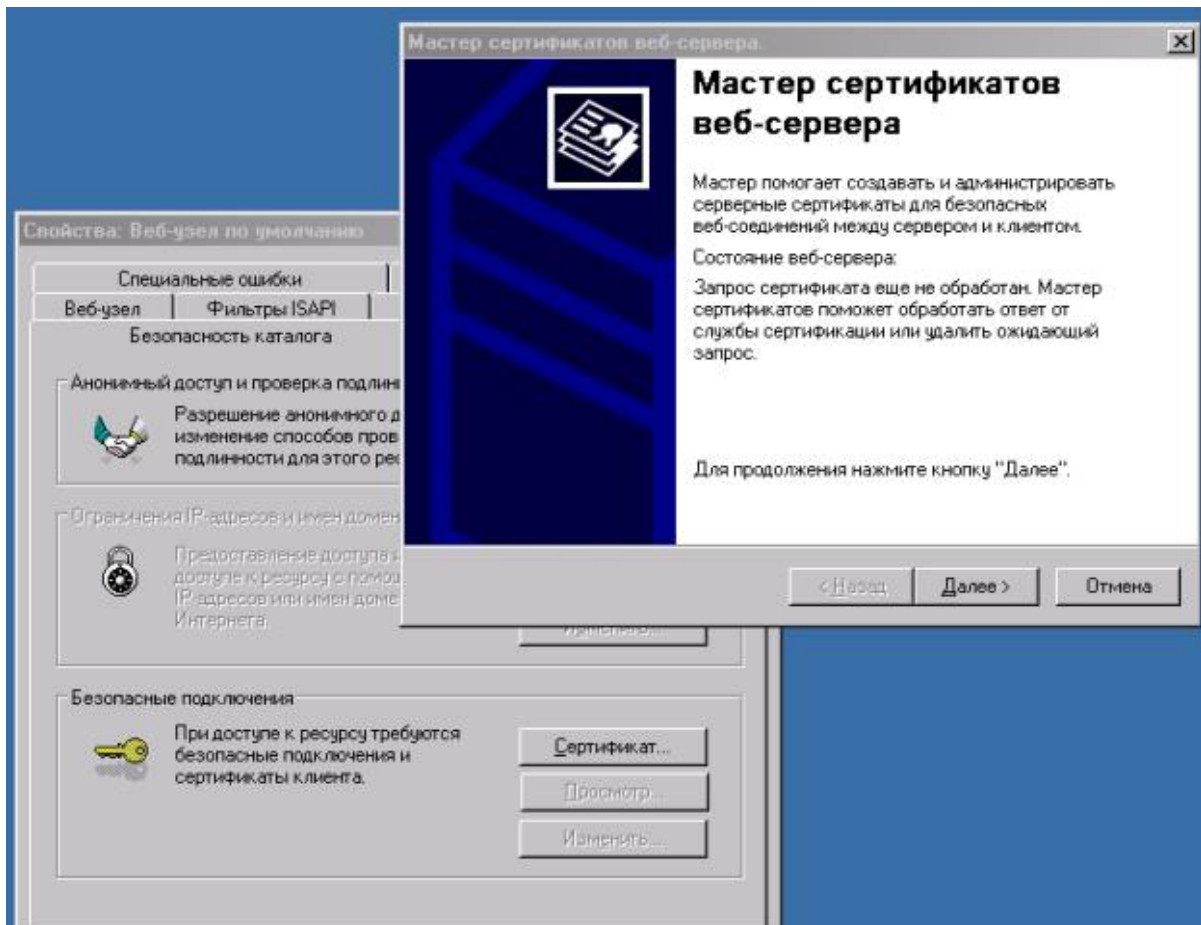
Данный веб-сервер был основан на базе решений Apache и SSLeay/OpenSSL, в настоящее время он распространяется на основе той же лицензии, что и Apache, а это означает, что возможно использовать его как для некоммерческих так и для коммерческих приложений совершенно бесплатно. Основными возможностями на момент написания данного материала была возможность аутентификации клиента и поддержка 128-битной криптографии. Более подробную документацию можно получить на сайте <http://www.apache-ssl.org>. Если вы заинтересовались mod\_ssl, то документацию по нему можно получить на сайте <http://www.modssl.org/>. Также подробные сведения о различных «mod» реализованных для Apache можно узнать, посетив сайт <http://www.opennet.ru/>.

## Практическая криптография в Java. Асимметричная криптография, методы безопасного обмена информацией



Я привожу пример изображения экрана MMC открытой оснасткой IIS. И на следующем примере просто вызываю мастера запроса сертификата. Каждый раз при запуске данного мастера он отслеживает свое состояние. Так, если вы сформировали запрос на получение сертификата, то мастер при последующем запуске учтет это и предложит на выбор либо установить возвращенный сертификат от СА или удалить запрос. При запросе сертификата мастер спросит вас относительно информации о вашей организации, предложит ввести параметры генерации ключей. В общем, говоря, почти тоже самое, что и когда мы использовали утилиту `keytool` и генерировали ключи и сертификаты для Java. Результат работы мастера сохраняется в виде текстового файла в специальном формате идентично тому, что использует `keytool` при создании запроса к центру СА.

Полученный документ для запроса следует отправить поставщику СА.



## 2. Создаем собственный защищенный веб-сервер

### 2.1. Теоретические сведения

Давайте снова рассмотрим пример установления защищенного соединения по схеме, которую мы рассматривали ранее. Первым шагом мы научимся выполнять взаимную проверку участников коммуникации. Начнем с того, что разработаем две компоненты нашей защищенной системы. Клиентская часть — ее задача заключается в том, что необходимо создать соединение с сервером и дальнейший обмен информацией пойдет по специальному криптографическому протоколу. Задача серверной части создать «ServerSocket» на некотором порте и слушать входящие запросы, и как только данный

## *Практическая криптография в Java. Асимметричная криптография, методы безопасного обмена информацией*

запрос поступил, следует начать обмен по специальному протоколу. Я уже ранее приводил его последовательность, но для удобства повторю:

- клиент посылает запрос на соединение с сервером;
- сервер генерирует случайное число  $R$  и сообщает его клиенту;
- клиент шифрует данное случайное число  $R$  с помощью секретного ключа  $x$  и отправляет серверу  $C = f(x, R)$ ;
- сервер повторно вычисляет значение  $C = f(x, R)$  и сверяет с тем, что ему прислал клиент. Если присланное значение совпадает, то пользователь наделяется соответствующими правами.

Для реализации данного протокола нам, прежде всего, будет необходимо создать пару открытого и закрытого ключа. Открытый ключ будет располагаться на сервере, а закрытый ключ будет находиться у клиента. Все созданные ключи хранятся в так называемом “keystore”. Однако, перед тем как мы продолжим работу, следует четко разграничить понятия “keystore” и “truststore”.

**Keystore** — представляет собой базу данных, в которой хранятся пары ключей и сертификаты которые используются при выполнении аутентификации клиента и сервера при SSL соединении. Когда клиент начинает соединение SSL, то он берет ключи и сертификаты из “keystore” для того чтобы предоставить информацию о себе.

**Truststore** — используется для того чтобы проверить ту информацию которую предлагает другой участник коммуникации, например он предлагает собственный сертификат который подписан некоторым институтом CA, в данном случае из truststore извлекается список доверенных CA и идет поиска того кто подписал данный сертификат, если такового издателя не будет найдено, то тому кто предоставил данный сертификат не будет доверия и соединение не состоится. Механизм проверки сертификатов встроенный в JSSE работает следующим образом:

Если свойство `Javax.net.ssl.trustStore` установлено, то его значение используется как местоположение “truststore”;

Следующим шагом идет поиск файла `lib/security/jssecacerts` если данный файл находится в каталоге определенном в системном свойстве `Java.home directory`, тогда содержимое файла `jssecacerts` используется как “truststore”;

Если файл `lib/security/cacerts` находится в каталоге `Java.home`, тогда

## *Практическая криптография в Java. Асимметричная криптография, методы безопасного обмена информацией*

данный файл используется как “truststore”.

В качестве иллюстрации далее приводится пример несложной программы, которая ищет в системе файл “truststore” и выводит его местоположение на экран.

```
package arti.security;

import java.io.*;

public class showTrustStore {

    public static void main(String[] args) {
        String s = System.getProperty("Javax.net.ssl.trustStore");

        if (s != null){
            System.out.println("Местоположение:
            Javax.net.ssl.trustStore = " + s);
            return;
        }

        String home = System.getProperty("Java.home");

        String p1 = home + File.separator +
            "lib"+File.separator+"security"
            +File.separator+"jssecacerts";

        if ( new File (p1).exists()){
            System.out.println("Местоположение: "+p1);
            return;
        }

        String p2 = home + File.separator +
            "lib"+File.separator+"security"
            +File.separator+"cacerts";

        if ( new File (p2).exists()){
            System.out.println("Местоположение: "+p2);
            return;
        }
    }
}
```

## Практическая криптография в Java. Асимметричная криптография, методы безопасного обмена информацией

```
System.out.println("хранилище не найдено");  
}  
}
```

Для решения данной задачи нам потребуется следующий набор криптографических объектов:

**Message digests** — результат вычисления хеш-функции над произвольным содержимым.

**Digital signatures** — цифровая подпись — используется для обеспечения целостности некоторого объема информации. Защита от подделки, или переделки пересылаемой информации.

**Certificates** — хранилище открытого ключа.

Для любого сложного приложения, ориентированного на использование множеством пользователей, имеющих различные задачи и соответственно права, является наличие специального диалогового окна входа в систему. Как уже говорилось ранее, передача пароля в открытом виде является небезопасной, так же как и хранение паролей на сервере. Мы создадим несколько систем для решения задачи авторизации пользователей. Сначала мы просто будем использовать хеш-функции для того, чтобы избежать открытой пересылки паролей, а следующим шагом будет отказ от использования пароля и переход к сертификатам.

### 2.2. Техническая информация о классе MessageDigest

Для вычисления хеш-функции для произвольного содержимого мы воспользуемся классом `security.MessageDigest`. Технически для создания экземпляра данного класса нам следует воспользоваться не `new`, а статическим методом `getInstance()`. Данный метод является фабричным, ему на вход в качестве параметра следует передать обязательно название алгоритма хеш-функции и опционально имя провайдера криптографических услуг в котором мы заинтересованы. Я надеюсь, что вы помните, что пакет `Java.security` большей частью представляет собой набор интерфейсов, а конкретные классы реализующие данные интерфейсы находятся в специальных пакетах которые поставляют нам «провайдеры». Если вас интересует данная информация, то узнать список провайдеров и изменить или установить

## *Практическая криптография в Java. Асимметричная криптография, методы безопасного обмена информацией*

новых вы можете редактируя специальный файл «Там где у вас установлена `jre\lib\security\java.security`» секция отвечающая за список провайдеров например, у меня выглядит так:

```
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
```

Не вдаваясь в подробности, скажу, что когда вы не указываете конкретного провайдера криптографических услуг, то Java просматривает каждого из этих провайдеров и спрашивает у всех их: реализует ли он данный алгоритм. И первый же найденный провайдер будет использован.

```
public static MessageDigest
    getInstance(String algorithm) throws
        NoSuchAlgorithmException

public static MessageDigest
    getInstance(String algorithm, String provider) throws
        NoSuchAlgorithmException, NoSuchProviderException
```

После того как мы создали экземпляр `MessageDigest` мы должны будем наполнять его набором данных для которых хотим вычислить значение хеш-функции. Осуществляется это путем вызова специальных методов:

```
public void update(byte input)

public void update(byte[] input)

public void update(byte[] input, int offset, int len)
```

И, наконец, после того как вы пропустили через `MessageDigesting` все данные мы можем вычислить значение хеш-функции, для этого мы делаем вызов одного из

## Практическая криптография в Java. Асимметричная криптография, методы безопасного обмена информацией

методов `digest()`.

```
public byte[] digest()
public byte[] digest(byte[] input)
```

Последний метод является комбинацией последовательного вызова сначала метода `update(input)`, а затем `digest()`.

Разумеется, созданный объект `MessageDigest` может быть использован для вычисления нескольких хеш-функций, однако следует помнить, что результат вычисления хеш-функции зависит от специального начального значения. Поэтому предварительно перед началом вычисления значения хеш-функции для новой последовательности байт следует очистить накопленные данные, с помощью вызова функции `reset()`.

```
public void reset()
```

Далее я привожу пример вычисления свертки для заданного файла, имя которого должно быть прочитано с клавиатуры.

```
package arti.security;

import java.security.*;
import java.io.*;

public class SimpleMDFileProcessor {

    public static void main(String[] args) throws Exception{
        MessageDigest md = MessageDigest.getInstance("SHA1");
        System.out.println("Enter Filename to Process ...");

        String fName = new DataInputStream(System.in).readLine();
        FileInputStream fin = new FileInputStream (fName);

        byte [] buf = new byte [10000];
        int rsize;

        while ((rsize = fin.read(buf)) > 0)
```

```
md.update(buf , 0 , rsize);

buf = null;
buf = md.digest();

System.out.println("MessageDigest for file
" + fName + " is equal:");

for (int i=0; i < buf.length; i++)
    System.out.print(buf[i] + " ");
}
}
```

Если внимательно рассмотреть код приведенного выше пример, то можно отметить его избыточность. Хотелось бы отметить, что при создании сложных приложений интенсивно работающих с потоками данных, необходимо иметь более удобные средства.

### **2.2.1. Потоки с поддержкой хеш-функции**

Для автоматизации работы с потоками, для которых следует вычислять значение хеш-функции в пакете `java.security` определены два класса `DigestInputStream` и `DigestOutputStream`, следует понимать, что эти два класса определены как потомки от классов фильтров потоков. Для иллюстрации я перепишу код ранее приведенного примера с использованием этих новых классов.

```
package arti.security;

import java.security.*;
import java.io.*;

public class AdvancedMDFileProcessor {

    public static void main(String[] args) throws Exception{
        MessageDigest md = MessageDigest.getInstance("SHA1");
        System.out.println("Enter Filename to Process ...");

        String fName = new DataInputStream(System.in).readLine();
        FileInputStream fin = new FileInputStream (fName);
```

## Практическая криптография в Java. Асимметричная криптография, методы безопасного обмена информацией

```
DigestInputStream din = new DigestInputStream (fin , md);

byte [] buf = new byte [10000];
int rsize;

while ((rsize = din.read(buf)) > 0);
buf = null;

buf = md.digest();

System.out.println("MessageDigest for file
" + fName + " is equal:");

for (int i=0; i < buf.length; i++)
    System.out.print(buf[i] + "; ");

}
}
```

Отдельно хотелось бы обратить ваше внимание на такой метод классов `DigestInputStream` и `DigestOutputStream`, как `public void on(boolean on)`. Вызывая данный метод с булевым параметром равным `true`, мы активизируем вычисление на основе данных пропускаемых через фильтр значения хеш-функции, если же мы вызовем данную функцию с параметром равным `false`, то проходящие через фильтр данные не будут учитываться при вычислении значения хеш-функции.

### 2.3. Пример защищенного установления соединения по сети

Давайте, теперь вернемся к нашей задаче и попытаемся создать клиент-серверное приложение, в котором мы будем передавать не открытым текстом имя пользователя и его пароль, а только значение хеш-функции над этими сведениями.

```
package arti.security;
import java.io.*;
import java.net.*;
import java.security.*;
```

## Практическая криптография в Java. Асимметричная криптография, методы безопасного обмена информацией

```
public class SecretClient {
    public static void main(String[] args) throws Exception {
        String uName , uPass;
        if (args.length != 2){
            System.out.println("Внимание: правила запуска данной программы");
            System.out.println("java arti.security.
                SecretClient username userpassword");
            return;
        }
        else{
            uName = args [0];
            uPass = args [1];
        }
        Socket sc = new Socket (InetAddress.getLocalHost() , 1001);
        DataInputStream din = new DataInputStream (sc.getInputStream());
        PrintStream pout = new PrintStream (sc.getOutputStream());
        pout.println("HELLO");
        System.out.println("Клиент говорит:HELLO");
        System.out.println("Сервер отвечает:"+ din.readLine());
        pout.println("User:"+ uName);
        System.out.println("Клиент говорит:User:"+ uName);
        String ans;
        System.out.println("Сервер отвечает:"
            + (ans = din.readLine()));
        if ( ans.indexOf("+OK") == -1){
            System.out.println("Ошибка: на сервере пользователь с именем "
                + uName+ " не зарегистрирован");
            return;
        }

        sun.misc.BASE64Encoder enc = new sun.misc.BASE64Encoder ();
        MessageDigest md = MessageDigest.getInstance("SHA1");
        byte [] buf = new byte [uPass.length()*2];
        for (int i=0; i < uPass.length(); i++){
            int code = Character.getNumericValue(uPass.charAt(i));
            buf [ i*2 ] = (byte) (code & 0xFF);
            buf [ i*2 + 1] = (byte) (code & 0xFF00);
        }
        String hpass = enc.encode(md.digest(buf));
        pout.println("Password:"+ hpass);
    }
}
```

## Практическая криптография в Java. Асимметричная криптография, методы безопасного обмена информацией

```
System.out.println("Клиент говорит password:"+ hpass);
System.out.println("Сервер отвечает:"+ (ans = din.readLine()));
if ( ans.indexOf("+OK") == -1)
    System.out.println("Ошибка: неправильный пароль");
else
    System.out.println("Успешный вход в систему:
    Секретная информация от сервера: "+ din.readLine());

sc.close();

}
}

package arti.security;

import java.io.*;
import java.net.*;
import java.util.*;
import java.security.*;

public class SecretServer {
    static Properties container = new Properties();
    protected static void loadContainer(String fName)
        throws IOException {
        /*
        Предположим, что в файле имя которого
        передается как параметр данного метода
        содержится информация о паролях пользователей в формате
        имя_пользователя=пароль
        */
        container.load(new FileInputStream(fName));
    }

    static class WorkBee
        extends Thread {
        Socket fromClient;
        public WorkBee(Socket sc) {
            fromClient = sc;
        }
    }
}
```

## Практическая криптография в Java. Асимметричная криптография, методы безопасного обмена информацией

```
public void run() {
    try {
        System.out.println(
            "-----Начало сессии с клиентом-----");
        DataInputStream din = new
            DataInputStream(fromClient.getInputStream());
        PrintStream pout = new PrintStream(fromClient.getOutputStream());
        String ask;
        System.out.println("Клиент говорит:" + (ask = din.readLine()));
        if (!ask.equalsIgnoreCase("HELLO")) {
            System.out.println("Ошибка клиент не соблюдает протокол");
            pout.println("ERROR");
            fromClient.close();
            return;
        }
        pout.println("+OK; Please Send Your Name");
        System.out.println("Клиент отвечает:" + (ask = din.readLine()));
        String cName = ask.substring(ask.indexOf(":") + 1);
        if (container.getProperty(cName) == null) {
            pout.println("-ERROR; Client With name \"" + ask +
                "\" isn't registered");
            System.out.println("Пользователь не зарегистрирован");
            fromClient.close();
            return;
        }
        pout.println("+OK; Send Hash value for password");
        System.out.println("Клиент отвечает:" + (ask = din.readLine()));

        String pass = container.getProperty(cName);
        sun.misc.BASE64Encoder enc = new sun.misc.BASE64Encoder();
        MessageDigest md = MessageDigest.getInstance("SHA1");
        byte[] buf = new byte[pass.length() * 2];
        for (int i = 0; i < pass.length(); i++) {
            int code = Character.getNumericValue(pass.charAt(i));
            buf[i * 2] = (byte) (code & 0xFF);
            buf[i * 2 + 1] = (byte) (code & 0xFF00);
        }
        String hpass = enc.encode(md.digest(buf));
        if ( ("Password:" + hpass).equals(ask)) {
            pout.println("+OK; password is valid");
        }
    }
}
```

## Практическая криптография в Java. Асимметричная криптография, методы безопасного обмена информацией

```
        pout.println("Big Secret");
        System.out.println("Авторизация прошла успешно");
    }
    else {
        pout.println("-ERROR; password is invalid");
        System.out.println("Ошибка авторизации");
    }
    fromClient.close();

    System.out.println("-----Сессия завершена-----");
}
catch (IOException ex) {
    ex.printStackTrace();
}
catch (NoSuchAlgorithmException ex) {
    ex.printStackTrace();
}
}
}

public static void main(String[] args) throws Exception {
    loadContainer(args.length == 1 ? args[0] : "default.passtore");
    ServerSocket ss = new ServerSocket(1001);
    // создаем сокет и начинаем
    // слушать приходящие запросы на порт # 1001
    Socket sc;
    while (true) {
        sc = ss.accept(); // пришел запрос описываемый сокетом sc
        new WorkBee(sc).start();

    } //end of while
} //end of main
}
```

### 2.4. Более надежный подход к решению задачи входа в систему

А теперь я предлагаю вам возможность самостоятельно найти ошибки и возможные бреши в реализованном нами криптографическом протоколе. Да, действительно, хотя

## *Практическая криптография в Java. Асимметричная криптография, методы безопасного обмена информацией*

секретная информация не передается в открытом виде, ценность нашего приложения все равно близка к нулю. Дело в том, что мы не создали средств защиты от такого вида криптографического нападения как повторная отправка.

Представьте себе ситуацию: клиент приходит на наш веб-сайт, выполняет там регистрацию и получает доступ к некоторому набору секретных ресурсов. А в это время злоумышленник перехватывает сообщение, которое мы отправили серверу, в котором и хранится значение хеш-функции для пароля и запомнил его. Хотя злоумышленник не знает пароль, он может легко получить доступ к защищенной части нашего сайта просто повторно отправив значение хеш-функции для пароля через несколько часов или дней.

Решение данной проблемы, очевидно, следует сделать так, чтобы клиент отправлял серверу не только пароль, но и информацию о моменте времени, когда он отправил данную информацию. Причем время и пароль должны быть не разделимы, чтобы невозможно было выделить из отправленной информации ее компоненты и воспользоваться ими по отдельности позже. Например, клиент вычисляет значение хеш-функции для новой строки  $S1 = \text{пароль} + \text{текущее время}$ . И именно этот результат отправляет серверу. На сервере извлекается время и проверяется величина разности между текущим моментом и временем отправки, если данная величина не превосходит некоторой заданной величины, которая в общем случае определяется временем прохождения запроса от клиента к серверу, то доступ к ресурсу разрешается, иначе происходит отказ в обслуживании.

Более качественным будет решение, основанное на использовании «секрета сервера». В общем случае такой подход выглядит следующим образом. Сервер, получив запрос клиента, вычисляет специально число по некоторому трудно детерминируемому закону и отправляет его клиенту. Клиент должен будет отправить серверу значение хеш-функции, вычисленное для аргумента  $S2 = \text{пароль} + \text{имя\_пользователя} + \text{текущее\_время} + \text{число}$ , полученное от сервера. Именно такой подход мы и попытаемся реализовать. Единственный момент, на который я хочу обратить также ваше внимание, это то, что при создании данного примера я придумал собственный протокол и правила общения между клиентом и сервером. По мере развития данного примера протокол будет усложняться и требовать большего количества усилий по проверке правильности его использования. Каждую строку, пересылаемую по сети,

## *Практическая криптография в Java. Асимметричная криптография, методы безопасного обмена информацией*

следует проверять на предмет ее корректности. И если в отдельных ситуациях можно воспользоваться правилом умолчания, то в данном случае это не применимо.

```
package arti.security;
import java.io.*;
import java.net.*;
import java.util.*;
import java.security.*;

public class SecretClient2 {
    public static void main(String[] args) throws Exception {
        String uName , uPass;
        if (args.length != 2){
            System.out.println("Внимание: правила запуска данной программы");
            System.out.println("java arti.security.SecretClient
                username userpassword");
            return;
        }
        else{
            uName = args [0];
            uPass = args [1];
        }
        Socket sc = new Socket (InetAddress.getLocalHost() , 1001);
        DataInputStream din = new DataInputStream (sc.getInputStream());
        PrintStream pout = new PrintStream (sc.getOutputStream());
        pout.println("HELLO");
        System.out.println("Клиент говорит:HELLO");
        System.out.println("Сервер отвечает:"+ din.readLine());
        pout.println("User:"+ uName);
        System.out.println("Клиент говорит:User:"+ uName);
        String ans;
        System.out.println("Сервер отвечает:"+ (ans = din.readLine()));
        if ( ans.indexOf("+OK") == -1){
            System.out.println("Ошибка: на сервере пользователь с именем "
                + uName+ " не зарегистрирован");
            return;
        }
        String dig = ans.substring(ans.indexOf("DIGIT=")+ "DIGIT=".length());
```

## Практическая криптография в Java. Асимметричная криптография, методы безопасного обмена информацией

```
System.out.println("Сервер отправил число "+ dig);
sun.misc.BASE64Encoder enc = new sun.misc.BASE64Encoder ();
MessageDigest md = MessageDigest.getInstance("SHA1");
long now;
// далее мы формируем строку включающую в себя
uPass+= dig + (now = new Date ().getTime ()) ;
// не только пароль, но и дату и время отправления,
// а также то случайное число которое послал клиенту сервер
byte [] buf = new byte [uPass.length()*2];
for (int i=0; i < uPass.length(); i++){
    int code = Character.getNumericValue(uPass.charAt(i));
    buf [ i*2 ] = (byte) (code & 0xFF);
    buf [ i*2 + 1] = (byte) (code & 0xFF00);
}
String hpass = enc.encode(md.digest(buf));

pout.println("Password:"+ hpass);
pout.println("Time:"+ now);

System.out.println("Клиент говорит password:"+ hpass);
System.out.println("Сервер отвечает:"+ (ans = din.readLine()));
if ( ans.indexOf("+OK") == -1)
    System.out.println("Ошибка: неправильный пароль");
else
    System.out.println("Успешный вход в систему:
    Секретная информация от сервера: "+ din.readLine());

sc.close();

}
}

package arti.security;

import java.io.*;
import java.net.*;
import java.util.*;
import java.security.*;

public class SecretServer2 {
```

## Практическая криптография в Java. Асимметричная криптография, методы безопасного обмена информацией

```
static Properties container = new Properties();
protected static void loadContainer(String fName) throws IOException {
    /*
     * Предположим, что в файле имя которого
     * передается как параметр данного метода
     * содержится информация о паролях пользователей в формате
     * имя_пользователя=пароль
     */
    container.load(new FileInputStream(fName));
}

static class WorkBee
    extends Thread {
    Socket fromClient;
    Random rnd = new Random ();
    public WorkBee(Socket sc) {
        fromClient = sc;
    }

    public void run() {
        try {
            System.out.println(
                "-----Начало сессии с клиентом-----");
            DataInputStream din = new DataInputStream(fromClient.getInputStream());
            PrintStream pout = new PrintStream(fromClient.getOutputStream());
            String ask , ask2;
            System.out.println("Клиент говорит:" + (ask = din.readLine()));
            if (!ask.equalsIgnoreCase("HELLO")) {
                System.out.println("Ошибка клиент не соблюдает протокол");
                pout.println("ERROR");
                fromClient.close();
                return;
            }
            pout.println("+OK; Please Send Your Name");
            System.out.println("Клиент отвечает:" + (ask = din.readLine()));
            String cName = ask.substring(ask.indexOf(":") + 1);
            if (container.getProperty(cName) == null) {
                pout.println("-ERROR; Client With name \"" + ask +
                    "\" isn't registered");
                System.out.println("Пользователь не зарегистрирован");
            }
        }
    }
}
```

## Практическая криптография в Java. Асимметричная криптография, методы безопасного обмена информацией

```
        fromClient.close();
        return;
    }
    int dig = rnd.nextInt();

    pout.println("+OK; Send Hash value for password&digit; DIGIT="+ dig);
    System.out.println("Клиент отвечает:" + (ask = din.readLine()));
    System.out.println("Клиент сообщает время отправки запроса:"
        + (ask2 = din.readLine()));
    long t = Long.parseLong(ask2.substring(ask2.indexOf(":")+1));
    long dif;
    if ((dif = (new Date ().getTime() - t)) > 5000){
        // предположим что 5 секунд = 5000 миллисекунд вполне должно хватить для
        //того чтобы сообщение дошло до сервера от клиента
        System.out.println("Ошибка слишком большой разрыв во времени
            от момента подачи сообщения, оно было отправлено
            "+ new Date(t) + " Разница во времени "+ dif);
        fromClient.close();
        return;
    }
    System.out.println("Разница во времени для отправленного
        сообщения составляет "+ dif + " миллисекунд");
    // Получаем пароль пользователя их хранилища
    String pass = container.getProperty(cName);
    // добавляем к данной строке также и число
    //которое мы послали клиенту
    pass+=dig;
    // добавляем также и время отправки запроса клиентом
    pass+=t;
    sun.misc.BASE64Encoder enc = new sun.misc.BASE64Encoder();
    MessageDigest md = MessageDigest.getInstance("SHA1");
    byte[] buf = new byte[pass.length() * 2];
    for (int i = 0; i < pass.length(); i++) {
        int code = Character.getNumericValue(pass.charAt(i));
        buf[i * 2] = (byte) (code & 0xFF);
        buf[i * 2 + 1] = (byte) (code & 0xFF00);
    }
    String hpass = enc.encode(md.digest(buf));
    if ( ("Password:" + hpass).equals(ask)) {
        pout.println("+OK; password is valid");
```

## Практическая криптография в Java. Асимметричная криптография, методы безопасного обмена информацией

```
        pout.println("Big Secret");
        System.out.println("Аутентификация прошла успешно");
    }
    else {
        pout.println("-ERROR; password is invalid");
        System.out.println("Ошибка входа в систему");
    }
    fromClient.close();

    System.out.println("-----Сессия завершена-----");
}
catch (IOException ex) {
    ex.printStackTrace();
}
catch (NoSuchAlgorithmException ex) {
    ex.printStackTrace();
}
}
}

public static void main(String[] args) throws Exception {
    loadContainer(args.length == 1 ? args[0] : "default.passtore");
    ServerSocket ss = new ServerSocket(1001);
    // создаем сокет и начинаем слушать
    // входящие запросы на порт # 1001
    Socket sc;
    while (true) {
        sc = ss.accept(); // пришел запрос описываемый сокетом sc
        new WorkBee(sc).start();

    } //end of while
} //end of main
}
```

Далее я привожу пример журнала лога, который вели клиент и сервер и честно признаюсь, что я упростил алгоритм, не сделав ряда проверок корректности передаваемых данных. Может быть, мое упущение исправите вы? Кроме того, я думаю, что неплохим бы домашним заданием могло бы включение механизма штрафов при попытке ошибочной регистрации. Подобная система давно используется в любой уважающей себя многопользовательской операционной системе (другой

*Практическая криптография в Java. Асимметричная криптография, методы безопасного обмена информацией*

правда вопрос в том, что в той же Windows2000 механизмы штрафов отключены по умолчанию).

Кратко, данная идея в том, что после каждой ошибочной попытки входа в систему сервер должен выполнить операцию блокировки дальнейшей работы с данным клиентом на несколько минут. А после, например трех ошибочных попыток, выполнить блокировку на час или выдать административное сообщение и пусть системный администратор или служба безопасности пытается разобраться в чем дело, то ли Вася Тапкин вчера очень долго в ресторане обсуждал вопросы новых инвестиций и наутро забыл все пароли, то ли нашу сеть пытаются грубо ломать методом научного тыка или говоря современного путем словарного перебора, или уж в совсем крайнем случае методом полного перебора brute force.

-----Начало сессии с клиентом-----	Клиент говорит: HELLO
Клиент говорит: HELLO	Сервер отвечает: +OK; Please Send Your Name
Клиент отвечает: User:vasya	Клиент говорит: User:vasya
Клиент отвечает: Password:uctPm7wXYXNS5TZGf/RMTjOFlag=	Сервер отвечает: +OK; Send Hash value for password&digit; DIGIT=2093139996
Клиент сообщает время отправки запроса: Time:1057007946312	Сервер отправил число 2093139996
Разница во времени для отправленного сообщения составляет 16 миллисекунд	Клиент говорит password: uctPm7wXYXNS5TZGf/RMTjOFlag=
Аутентификация прошла успешно	Сервер отвечает: +OK; pasword is valid
-----Сессия завершена-----	Успешный вход в систему: Секретная информация от сервера: Big Secret

Хотелось бы сделать маленькое отступление и сказать что многие вещи, которые мы делаем, не новы и имеют реализацию в рамках средств безопасности в Windows или Linux. Похожий подход, основанный на вычислении значения хеш-функций от паролей, случайных чисел и дат регистрации системы используется в широко известной системе Kerberos. Разумеется, более подробную информацию можно получить в MSDN, хотя Kerberos не является разработкой Microsoft, а появился в известной кузнице современных технологий и решений MIT (Массачусетский

технологический университет) еще в начале 80 годов и в настоящее время официально принят как стандарт IETF (Internet Engineering Task Force) и оформлен в RFC 1510. Так же очевидно, что возможности Kerberos больше чем то, что нам удалось реализовать, но основная цель, которую я преследовал это показать необходимость следить за последними решениями на рынке и пытаться их изменять в зависимости от нужд конкретной ситуации.

## 2.5. Сигнатуры

Хотя понятие сигнатур ранее не упоминалось явно, но тем не менее многое из того о чем я рассказывал шло именно к понятию сигнатуры. Я много внимания уделял вопросу обеспечения целостности пересылаемой информации через Интернет, так чтобы ни случайный сбой, ни целенаправленная атака не могла привести к подмене передаваемого сообщения. Для решения данной проблемы мы воспользовались понятием хеш-функций. Также я много говорил о том, что необходимо иметь способ удостовериться в том, что та информация, которая пришла к нам была отправлена именно тем, кто это заявляет.

Пришло время соединить эти пути и ввести понятие сигнатуры. Да, как я и говорил ранее, сигнатура решает два вопроса: обеспечение целостности информации и проверка отправителя. Сигнатуры основаны на использовании ассиметричных методов криптографии. Сигнатуры не ставят перед собой задачу защитить передаваемую информацию от несанкционированного просмотра, для решения такой задачи мы уже решили использовать именно симметричные методы шифрования. Итак, сделаем обобщение. **Сигнатура** — это результат вычисления хеш-функции для некоторого сообщения, который был подписан с помощью ЭЦП отправителя. Создать программную реализацию сигнатуры не очень сложно. Однако большее внимание я посвящу рассмотрению уже существующего API для Java.

В Java существует пакет `java.security`, в котором определено большое количество интерфейсов и абстрактных классов для различных направлений безопасности данных. Так класс `java.security.Signature` действительно является абстрактным и напрямую экземпляр его создать невозможно. Так же как и в случае `MessageDigest` мы должны будем воспользоваться фабричным методом `getInstance`. Данный метод имеет два варианта. В первом случае мы должны будем указать при вызове

данного метода название алгоритма, которым мы хотим воспользоваться. Как вы помните, сигнатура представляет собой объединение двух алгоритмов: один из них используется для вычисления хеш-функции, второй служит для выполнения подписи. Следовательно, имена алгоритмов представляют собой комбинацию имени алгоритма вычисления свертки и алгоритма ЭЦП.

Например, возможны следующие имена алгоритмов:

**SHA1 with DSA:** Здесь DSA — алгоритм вычисления ЭЦП, а SHA-1 алгоритм вычисления хеш-функции.

**MD2 with RSA:** MD2 — алгоритм для хеш-функции, а RSA используется для вычисления ЭЦП.

**MD5 with RSA:** как вы уже догадались MD5 — это алгоритм для вычисления хеш-функции, а RSA используется для вычисления ЭЦП.

**SHA1 with RSA:** Без комментариев.

Единственное на что я хочу дополнительно обратить ваше внимание, это то что приведенный перечень не является абсолютным и в общем случае зависит от того какие именно провайдеры криптографических услуг установлены у вас на компьютере. Для того чтобы узнать это следует посмотреть содержимое файла `jre\lib\security\java.security`.

```
1) public static Signature getInstance(String algorithm) throws  
NoSuchAlgorithmException
```

```
2) public static Signature getInstance(String algorithm, String  
provider) throws NoSuchAlgorithmException,  
NoSuchProviderException
```

Вторая версия функции `getInstance` отличается от первой тем, что мы хотим явно указать имя того провайдера услугами которого хотим воспользоваться.

После того как мы создали экземпляр класса `Signature`, мы можем его использовать как для выполнения создания подписи к информации, так и для ее проверки. Технически это осуществляется вызовом метода

```
initSign(PrivateKey privateKey)
```

## Практическая криптография в Java. Асимметричная криптография, методы безопасного обмена информацией

```
throws InvalidKeyException
```

в том случае если вы хотите выполнить подпись информации, а если же необходимо провести проверку той сигнатуры которую вы получили вместе с сообщением от вашего отправителя, то следует вызвать метод

```
initVerify(PublicKey publicKey)
    throws InvalidKeyException
```

Обратите внимание на то, что в первом случае мы передаем данной функции в качестве параметра секретный ключ, а для проверки — открытый ключ.

Следующим шагом будет вызов методов

```
public final void update(byte input) throws SignatureException
public final void update(byte[] input) throws SignatureException
public final void update(byte[] input, int offset, int len) throws
SignatureException
```

Данные методы очень похожи на те, что мы использовали в классе MessageDigest.

И, наконец, последнее, что нам будет необходимо — это вызов метода `verify()` в том случае если мы хотим выполнить проверку сигнатуры или метода `sign()` в том случае если мы хотим выполнить создание сигнатуры для некоторого документа. В качестве демонстрации я приведу пример в котором создам пару закрытого и открытого ключа, затем с помощью закрытого ключа создам подпись к данному файлу и выполню его проверку.

```
package arti.security;
import java.io.*;
import java.security.*;
import java.security.interfaces.*;

public class test_signatures1 {
    public static void main(String[] args) throws Exception{
        if (args.length != 1){
            System.out.println("Usage:
                java arti.security.test_signatures1 file.ext");
            return;
        }
    }
}
```

## Практическая криптография в Java. Асимметричная криптография, методы безопасного обмена информацией

```
KeyPair kp =
    KeyPairGenerator.getInstance("DSA").generateKeyPair();
FileInputStream fin = new FileInputStream (args[0]);
FileOutputStream fout = new FileOutputStream (args[0]+".test");
Signature sig = Signature.getInstance("SHA1withDSA");
sig.initSign(kp.getPrivate());
byte [] buf = new byte [10000];
int rsize;
while ( (rsize = fin.read(buf)) > 0){
    sig.update(buf , 0 , rsize); // обновляем значения сигнатуры
    // и вносим случайные
    int rndpos = (int)(Math.random()*(rsize - 1));
    buf [rndpos] ^= 0xFF;// искажения в файл
    fout.write(buf , 0 , rsize);
}

fin.close();
fout.close();
sun.misc.BASE64Encoder enc = new sun.misc.BASE64Encoder ();
byte [] signat;
String e64 = enc.encode(signat = sig.sign());
System.out.println("Signature=" + e64);
Signature ver = Signature.getInstance("SHA1withDSA");
ver.initVerify(kp.getPublic());
fin = new FileInputStream (args[0]+".test");

while ( (rsize = fin.read(buf)) > 0){
    ver.update(buf , 0 , rsize); // обновляем значения сигнатуры
}
fin.close();

if (ver.verify(signat))
    System.out.println("Signatures are equal");
else
    System.out.println("Signatures aren't equal");
}
}
```

Если внимательно посмотреть на приведенный фрагмент кода, то можно увидеть всю

## *Практическая криптография в Java. Асимметричная криптография, методы безопасного обмена информацией*

его надуманность и отсутствие практического значения. Глупо создавать сигнатуру для документа и проверять ее тут же. Гораздо эффективнее это вычисленную сигнатуру отправить вместе с документом по сети и чтобы получатель на другой стороне проверил ее, узнал не было ли искажений или попытки взлома. Только для этого необходимо найти способ сохранения значения ключа, который мы вычислили. В простейшем варианте можно воспользоваться сериализацией и поместить ключ в файл. Можно придумать собственный формат для хранения ключа. Однако не проще ли будет воспользоваться уже существующей и продуманной концепцией keystore. Благо в Java есть поддержка данного механизма, реализованная в виде интерфейсов `Certificate` и `KeyStore`.

Класс `KeyStore` служит для хранения двух видов объектов. Прежде всего, это ключи. Внутри хранилища может размещаться множество закрытых ключей, доступ к ним закрыт и без знания ключа с помощью которого зашифровано содержимое keystore не возможно добраться до его внутреннего устройства. Я прошу обратить внимание на то, что внутри хранилища ключей обычно размещают именно закрытые ключи. Открытые ключи должны храниться именно внутри цепочки сертификатов. Которые и являются вторым видом содержимого keystore. Каждая единица информации внутри keystore имеет имя, или псевдоним (`alias`) указание которого необходимо при выполнении всякой операции над содержимым хранилища.

Следует понимать, что хотя мы все время работаем с хранилищами организованными в виде файлов, однако архитектурой криптографических расширений Java не ограничивается внутреннее устройство и схема работы хранилища, это означает, что хранилище может располагаться на специальной smart-карте, или быть организовано в виде специального аппаратного устройства подключаемого к компьютеру или иному интеллектуальному устройству. Для создания объекта хранилища следует воспользоваться уже привычным нам способом вызова фабричного метода `getInstance`, и также как и в случае работы с `MessageDigest`, `Signature` и генераторами ключей мы можем либо просто указать тип нужного хранилища, либо сделать пожелание относительно провайдера который будет предоставлять нам криптографические услуги.

Я говорил, что хранилище всегда привязано к какому-то устройству, наиболее часто, файлу. Следовательно, перед началом использования хранилище следует загрузить, и

только потом можно обращаться к содержимому хранилища. В качестве иллюстрации я загружу из созданного ранее хранилища ключи наших гипотетических организаций Terrana inc, Venera corp. и других и перепишу код так, чтобы можно было при запуске приложения указать ему, в каком режиме он должен работать и возможность указания хранилища ключей, пароля к нему и другую необходимую информацию.

Следовательно, пример запуска нашего приложения с параметрами командной строки может выглядеть так:

Для создания сигнатуры:

```
"-operation verify -storefile xyz.store -file data.txt  
-sigfile data.txt.signature -storepass bigsecret -alias  
Terrana -aliaspass bigsecret"
```

И для проверки пример командной строки будет выглядеть очень похоже:

```
"-operation sign -storefile xyz.store -file data.txt -sigfile  
data.txt.signature -storepass bigsecret -alias Terrana  
-aliaspass bigsecret".
```

## **3. Методы получения доступа к содержимому keystore**

### **3.1. Технические сведения**

Создается объект `keystore` по аналогии с тем, как мы создавали другие криптографические объекты, например, `messagedigest`, `keygenerator` и другие. Мы воспользуемся фабричным статическим методом `getInstance`. И также как и ранее при вызове данного метода следует указать тип используемого хранилища. Я передал имя типа используемого по умолчанию, кстати, указание на этот тип, также хранится в файле конфигурации JRE вместе с указаниями списка используемых провайдеров и другой информацией необходимой для поддержки криптографических преобразований. После создания объекта `keystore` следует его загрузить используя метод `load`, в качестве параметров которому следует передать поток ввода из данного хранилища и пароль для доступа к нему. Дальнейшие действия уже предсказуемы. Для извлечения сертификата необходимо только указать имя псевдонима под которым он

## Практическая криптография в Java. Асимметричная криптография, методы безопасного обмена информацией

хранится. Для извлечения закрытого или секретного ключа дополнительно потребуется указание пароля доступа.

```
package arti.security;

import java.io.*;
import java.security.*;
import java.security.interfaces.*;

public class SigAndStore {
    /**
     * Данная функция является вспомогательной
     * и служит только для того чтобы
     * выполнять анализ параметров командной строки
     * выделяя из не необходимые
     * команды и их значения
     * @param ar - массив строк задающих
     * все параметры командной строки
     * @param command - искомая команда
     * @return - значение найденной команды
     * или если таковая не была найдена то null
     */
    public static String getValueForCommand (
        String [] ar , String command){
        for (int i=0; i < ar.length - 1; i++)
            if ( ar[i].equalsIgnoreCase(command)){
                return (ar[i+1].indexOf("-") != 0)?ar[i+1]:null;
            }
        return null;
    }

    public static void main(String[] args) throws Exception {

        System.out.println("Usage: java arti.security.SigAndAlg
        -operation verify|sign -file xyz.file.ext
        -sigfile prep.file.signature -storepass mysecret
        -storefile store.data.file -alias your.alias
```

## Практическая криптография в Java. Асимметричная криптография, методы безопасного обмена информацией

```
-aliaspass bigsecret.for.your.alias");
String operation = getValueForCommand(args , "-operation");
// Разумеется подобные проверки следовало бы
// выполнить для каждого из параметров приложения
// однако это требует больших затрат времени,
// кроме того является достаточно рутинным
// чтобы вы самостоятельно могли бы реализовать данную часть кода
if ( (! operation.equalsIgnoreCase("verify") ) &&
    (! operation.equalsIgnoreCase("sign"))) {
    System.err.println("Error -operation value is incorrect,
        value must be 'sign' or 'verify'");
    return;
}

String file = getValueForCommand(args , "-file");
String sigfile = getValueForCommand(args , "-sigfile");
String storefile = getValueForCommand(args , "-storefile");
String storepass = getValueForCommand(args , "-storepass");
String alias = getValueForCommand(args , "-alias");
String aliaspass = getValueForCommand(args , "-aliaspass");

KeyStore ks = KeyStore.getInstance(KeyStore.getDefaultType());
ks.load(new FileInputStream(storefile) , storepass.toCharArray());
Signature sig = Signature.getInstance("SHA1withDSA");
byte [] buf = new byte [10000];
int rsize;
if (operation.equalsIgnoreCase("verify")) {
    sig.initVerify(ks.getCertificate(alias));
    // пароль в случае если мы выполняем проверку и
    // нам нужен сертификат не является обязательным -
    // вспомните, что ведь сертификаты по своему
    // определению должны быть общедоступными -
    // всякий получатель сообщения с помощью сертификата
    // отправителя мог проверить его действительность и целостность
    FileInputStream fin = new FileInputStream (file);
    while ( (rsize = fin.read(buf)) > 0)
        sig.update(buf , 0 , rsize); // обновляем значения сигнатуры
    fin.close();
    fin = null;
    fin = new FileInputStream (sigfile);
```

## Практическая криптография в Java. Асимметричная криптография, методы безопасного обмена информацией

```
    rsize = fin.read (buf);
    System.out.println("Результат проверки: сигнатуры " +
        (sig.verify(buf , 0 , rsize)?"":" не ") + " совпадают");
}
else
{
    sig.initSign((PrivateKey)ks.getKey(alias ,
        aliaspass.toCharArray()));
    FileInputStream fin = new FileInputStream (file);
    while ( (rsize = fin.read(buf)) > 0)
        sig.update(buf , 0 , rsize); // обновляем значения сигнатуры
    fin.close();
    fin = null;
    FileOutputStream fout = new FileOutputStream (sigfile);
    fout.write(sig.sign());
    System.out.println("Результат вычисления
        сигнатуры был записан в файл");
}
}
}
```

## 4. Заключение

В данном материале нам удалось рассмотреть важную часть криптографических расширения для Java. Мы познакомились с сертификатами, рассмотрели процедуру их генерации с использованием встроенных средств JSDK, а также хотя и очень кратко но упомянули о схемах поддержки сертификации в Apache и IIS5.0. Нам удалось создать надежную схему аутентификации клиента в сети, как небольшое домашнее задание мне хотелось бы, что бы вы добавили и обратный процесс проверки со стороны клиента того с кем идет соединение. Достаточно глубоко причем даже не только на уровне прикладного использования были рассмотрены хеш-функции, сигнатуры, было введено понятие хранилищ ключей.

## 5. Ресурсы

- Следующим шагом нашего рассказа будет непосредственно реализация веб-сервера с поддержкой https, поэтому я рекомендую внимательно прочитать статьи

*Практическая криптография в Java. Асимметричная криптография, методы  
безопасного обмена информацией*

[http://www.javable.com/columns/serv\\_side/workshop/18](http://www.javable.com/columns/serv_side/workshop/18) посвященные работе в сети, вы должны четко понимать что такое сокет, tcp/ip знать что такое заголовки http.

- Настоятельно рекомендую почитать хорошую книжку по администрированию Windows2000/NT или Linux, при написании данного материала на автора большое влияние оказало рассмотрение Kerberos и NTLM из MSDN.
- В следующей статье серии я буду использовать xml поэтому вспомнить основные положения будет достаточно неплохо, рекомендую [xmlhack.ru](http://xmlhack.ru).
- Хорошая информация по сетям, безопасности может без слишком большого акцента на Java лежить на [www.citforum.ru](http://www.citforum.ru) а также обратите внимание на [rdsn.ru](http://rdsn.ru).
- Пожалуй, говорить о том, что вся литература, которую я рекомендовал в предыдущих статьях серии не потеряла своего значения, не стоит. Не так ли?