Христов, Хр., Хр. Крушков, "Средства в Java платформата използващи цифрови подписи и сертификати", Научни трудове на СУБ, Серия Б. Естествени и хуманитарни науки, т. XII, Пловдив, 2010г., стр.416-422, ISSN 1311-9419.

СРЕДСТВА В JAVA ПЛАТФОРМАТА ИЗПОЛЗВАЩИ ЦИФРОВИ ПОДПИСИ И СЕРТИФИКАТИ

Христо Тошков Христов

Христо Димитров Крушков
катедра "Компютърна Информатика"
ФМИ при ПУ "Паисий Хилендарски"

TOOLS INTO JAVA™ PLATFORM UTILIZE DIGITAL SIGNATURES AND CERTIFICATIONS

Hristo Toshkov Hristov

Hristo Dimitrov Krushkov
Department of Computer Science
FMI, University of Plovdiv

Abstract

In the work are presented basic theoretical cryptographic terms connected with digital signatures and digital certificates. There are described the procedures for signing and verification of digital signature and the role of digital certificate for building "trust" in unsecure medium during exchange of digital documents between two parties. It is shown part of the ideology of the JCA specification in the Java™ platform and the principles upon which it is built. It is described the way of implementation of cryptographic services and algorithms from the Engine classes and in particular the work of the Signature class.

Keywords: *Cryptography, Digital signature, Digital certificate, Java Cryptography Architecture (JCA)*

1. Въведение.

В последните десет години Java™ платформата се наложи като една от най-широко разпространените платформи за програмиране. От самото начало на проектирането й се е заложило акцентиране върху сигурността. В ядрото на езика са интегрирани статична проверка на типовете (statically type check) и събирач на боклук (garbage collection), които подобряват качеството на написания код. При зареждането на класовете също са заложени механизми за проверка, така че да се гарантира изпълнението само легитимен за езика код. Още от първата версия Java™ създава сигурна среда за изпълнение на потенциално несигурен код, реализирано чрез Java Applets. Успоредно с разрастването на областите на приложност в платформата се развиват и технологии за защита и сигурност на електронни документи. В архитектурата на Java сигурността са включени голям брой APIs (application programming interfaces). Те дават на програмистите набор от инструменти за разработване на приложения, при които сигурността е от съществено значение. Интерфейсите в Java™ предоставят възможност да се използват реализации на алгоритми и протоколи за криптиране независимо от конкретната технология. Тези алгоритми се прилагат като готови решения предоставени *доставчици на криптографски услуги* (Cryptographic Service Provider) и се добавят като решения чрез предварително зададен интерфейс. По този начин разработчиците имат достъп до различни технологични решения и услуги по унифициран начин, без необходимостта от познаване и адаптиране към конкретните реализация. Java™ включва голям брой доставчици на криптографкси услуги, които са реализирали основните криптографски алгоритми, като освен това е възможно да бъдат инсталирани допълнителни доставчици.

Така, следва да направим кратко теоретично описание на схемите на симетрична и асиметрична криптосистема, процедурите на цифрово подписване и верификацията, ролята на цифровия подпис и сертификат в процеса на обмен на електронни документи основан на *Инфраструктура на публичния ключ*, след което ще посочим някои технологични решения на разгледаните въпроси в Java™ платформата.

2. Недостатъци на схемите на симетрична и асиметрична криптосистеми. Инфраструктура на публичния ключ.

В криптографията се разглеждат два вида криптосистеми: симетрична и асиметрична. Първите използват един и същи таен ключ за криптиране и декриптиране на съобщения, от където идва и тяхното название. В известен смисъл може да се твърди, че споделянето на тайния ключ е симетрично (той е един за двете страни!). Вторите използват двойка свързани ключове: частен и публичен.

За да могат две страни да предават криптирани съобщения, използвайки симетрична криптосистема с таен ключ, те трябва преди това да се договорят какъв ключ ще използват и да го пазят в тайна. Ако те са с различни физически местоположения, нещата допълнително се усложняват, защото трябва да се разчита на посредник или някакъв вид сигурен канал, за да се предотврати разкриване на тайния ключ по време на предаването. Тези криптосистеми макар и да имат място в съвременната компютърна криптография като част от по-сложни схеми, самостоятелно не могат да осигурят сигурна защита на електронните документи (съобщения).

Развитие в посока решение на проблема дават изследователите Whitfield Diffie и Martin Hellman през 1976 година, като представят схема на криптосистема с публичен ключ. Асиметрична криптосистема с публичен ключ съдържа двойка ключове, такива че, публичният ключ е достъпен за всеки (той би могъл да се намира на сървър, откъдето да се изтегли), а частният ключ се пази в тайна от неговия притежател. Публичния и личния ключ (public/private key pair) представляват математически свързана двойка ключове. На всеки публичен ключ съответства точно един личен ключ и обратно – на всеки личен ключ съответства точно един публичен ключ. Основно предимство на тези криптосистеми пред симетричните е, отпадането на нуждата от споделяне на тайния ключ, тъй като всяка от страните има свой частен ключ. Въпреки това криптосистемите с публичен ключ не решават един съществен проблем, как две страни първоначално да обменят публичните си ключове по несигурна среда? Решение на проблема се дава чрез така наречената Инфраструктура на публичния ключ (PKI, Public Key Infrastructure), изградена на базата на криптосистемите с публичен ключ. PKI представлява съвкупност от архитектура, организация, техники, практики и процедури предлагащи софтуерни решения интегрирани в приложни програми с цел сигурната обмяна на информация по несигурни мрежи и преносни среди. Чрез PKI базирана на криптосистема с публичен ключ се постига доверие между непознати страни като за целта се използват цифровия подпис и сертификат.

3. Цифров подпис – полагане и верификация. Цифрови сертификати.

Инфраструктурата на публичния ключ осигурява надежден метод за цифрово подписване, целта на която е да удостовери произхода на даден електронен документ и защити целостта му, когато той се предава по мрежа или несигурна среда.

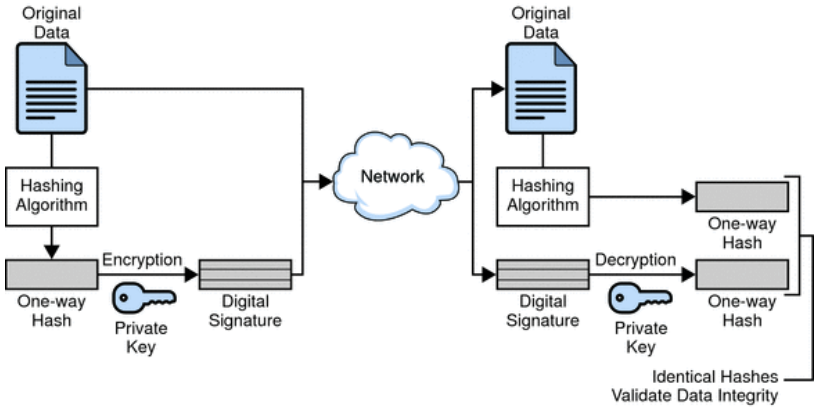
Цифровия подпис представлява число, което се изчислява математически при подписването на документа и зависи: от съдържанието на документа, от алгоритъма по който се извършва подписването и от частния ключ на притежателя подписал документа. В технологията на цифровия подпис включва две процедури: полагане и верификация.

Цифровия сертификат е електронен документ в който са записани и защитени данни с които се гарантира произхода подателя на документа. Той се издава от сертификационен орган (certification authorities – CA), трета независима страна в процеса на обмен.

3.1. Полагане на цифров подпис.

Процедурата на цифровото подписване*(фиг.№1)* е следната: Първоначално се изчислява *хеш-стойност* (message digest) на документа по някои криптографски силен алгоритъм за *хеширане* като MD4, MD5, SHA1 и др. След това, получената хеш-стойност от съобщението се *шифрира* с личния ключ на подателя. За целта се използва математически алгоритъм за цифров подпис (digital signature algorithm), който използвайки личния ключ преобразува хеш-стойността в *шифрирана хеш-стойност*. Шифрираната хеш-стойност представлява число, което наричаме *цифров подпис*. Получения цифров подпис се прикрепя към съобщение в специален формат и се изпраща заедно с открития текст.

Известни са три типа алгоритми, използвани за цифров подпис. Това са RSA, който се основава на теория на числата, DSA, които се основава на теория на дискретните логаритми и ECDSA, които се основава на теория на елиптичните криви.



Фиг.№1. Полагане и верификация на цифров подпис

3.2. Верификация на цифров подпис.

С верификацията на цифров подпис се удостоверява произходът на подателя на електронен документ и се гарантира цялост (неподправеност) на информацията която се съдържа в него. Процедурата по проверка на цифров подпис (верификация) има за цел да установи дали изпратения документ е бил подписан с личния ключ на подателя и дали действително подателя е използвал този ключ. Тази процедура е следната: Първоначално се изчислява *хеш-стойността* на подписания документ, (по същия начин и със същия алгоритъм както при процедурата на подписване). Тази хеш-стойност се нарича *текуща*, защото се получава от текущия вид на документа (не е известно дали при предаването по канала изходното съобщение не е било подправено). Следва *дешифриране* на цифровия подпис с публичния ключ на подателя, съответен на личния ключ и със същия алгоритъм използват при шифрирането. Резултата от дешифрирането е *оригинална хеш-стойност***.** Накрая се сравняват *текущата хеш-стойност* и *оригинална хеш-стойност*. Ако двете стойности съвпадат, верификацията е успешна, от което следва, че документа е подписан с личния ключ на подателя, в противен случай верификацията е неуспешна.

Верификацията на цифров подпис не може да удостовери самоличността на лицето, което е изпратило съобщението. Възможно е някой неоторизирано да използва личния ключ на друго лице, т.е. чрез верификацията на цифровия подпис не може да се проследи произхода на подателя и следователно не може

да се установи доверие между получател и подател. За да се провери и гарантира произхода на подателя на документа се използват цифрови сертификати.

3.3. Цифрови сертификати.

Цифровите сертификати са електронни документи, които съдържат публичен ключ и характеристики на притежателя си, както и специфична информация за органа от когото са издадени. Най-съществената черта на цифровите сертификати е, че свързват еднозначно публичен ключ с притежател.

Съществуват различни стандарти за цифрови сертификати като PGP (Pretty Good Privacy), SPKI/SDDI (Simple Public Key Infrastructure/Simple Distribute Security Infrastructure), X.509 и др. В практиката най-широко използван е X.509 стандарта. Синтаксиса определящ съдържанието на X.509 е съставен от следните атрибути:

- version – версия;
- serial number - сериен номер;
- signature algorithm ID - алгоритъм за цифров подпис;
- issuer name - име на издателя;
- validity period - период на валидност;
- subject (user) name - име на притежателя;
- subject public key information - публичен ключ на притежателя;
- extensions (version 3 only) - разширения;
- signature on the above fields - цифров подпис, удостоверяващ валидността на данните в сертификата.

Как цифровия сертификат гарантира произхода на притежателя си? Как две страни да си обменят първоначално публичните ключове? При обмен на електронни документи към съобщението освен цифровия подпис се прикрепя и сертификата, в който, както казахме, е записан публичния ключ. Но откъде да сме сигурни в коректността на съдържанието на сертификата, така че да имаме доверие на подателя? Съществуват различни подходи и решения за удостоверяване на автентичност на подател. Един от тези подходи е подател и получател да се доверят на трета, независима страна. Такова решение предлага *Инфраструктурата на публичния ключ* (PKI). Известни са различни „модели на доверие", които реализират PKI. Такива модели са: модел с единствена доверена страна, Web trust, йерархичен модел, модел на кръстосана сертификация (cross certification) и др. От споменатите модели, най-масово за използван е йерархичния модел предназначен за употреба в интернет среда. Този модел намира широка популярност и се използва за e-business, e-banking, e-commerce и e-goverment и др. приложения. При него доверието се изгражда, чрез сертификационни органи (Cartificate authorities, CA). Сертификационен орган е институция, която е упълномощена да издава цифрови сертификати и да ги подписва със своя личен ключ. Целта на CA е да гарантира, че даден сертификат е автентичен. Съществено в този модел са т. нар. сертификационни вериги. Подробно, каква е структурата на йерархичния модел на доверие може да се прочете от http://docs.sun.com/app/docs/doc/820-2493/6ne3feeeh?a=view#gdzdp

Така, когато до получателя достигне документ с прикрепен цифров подпис и сертификат, целостта на документа ще се установи от цифровия подпис, а автентичността на подателя ще се гарантира от цифровия сертификат. Разбира се, наред с цифровия подпис ще трябва да се верифицира и цифровия сертификат (или сертификационната верига) издаден от съответния орган. Как се верифицира цифров сертификат може да се види на http://docs.sun.com/app/docs/doc/820-2493/6ne3feeej?a=view

Сред известните сертификационни органи са: VeriSign Inc.(http://www.verisign.com); Entrust Inc. (http://www.entrust.com); DutchCrid CA (http://ca.dutchgrid.nl) и др.

4. Идеология на интерфейса в JCA.

Концепцията на Java™ сигурността обхваща различни области на приложение. В нейната архитектурата се включват голям брой APIs (application programming interfaces). Интерфейсите, предоставени от Java™, дават възможност за използването на много независими една от друга имплементации на алгоритми за криптиране и други услуги за сигурност. Тези услуги и алгоритми се имплементират в независимост от *доставчика на криптографски услуги* (Cryptographic Service Provider) и се включват в платформата чрез предварително зададен интерфейс. Java™ включва голяма част от *доставчици на криптографски услуги* с реализации на основните криптографски алгоритми, като освен това е възможно да се инсталират и допълнителни *доставчици*. Това дава възможност на разработчиците да разширяват платформата с алгоритми и механизми за сигурност. По този начин програмистите имат достъп до много и различни услуги

по унифициран начин, без необходимостта от познаването на отделните имплементации на конкретните доставчици.

За целите на криптографията, Java™ платформата ползва Java Cryptography Architecture (JCA). JCA представлява спецификация, която предоставя на програмистите средства за работа с криптографски усклуги. Наред с JCA за извършване на криптографски операции като шифриране, дешифриране, генериране на ключове и др., се използва и архитектурата Java Cryptography Extension (JCE), която не е предмет на разглеждане в изложението. Спецификацията JCA установява стандарти за различните типове криптографски услуги и специфицира начините на използване на алгоритмите.

Класовете от JCA отнасящи се към криптографските усгуги са включени в Security пакета на SDK.

4.1. Принципи в JCA.

Java Cryptography Architecture се основава на следните два приципа:

- Независимост на реализацията и обратна съвместимост (implementation independence and interoperability);
- Алгоритмична независимост и разширяемост (algorithm independence and extensibility).

Независимост на реализацията(имплементацията) и алгоритмичната независимост са допълващи се принципи. Разработчикът може да използва криптографска услуга като цифров подпис или хеширане без да се интересува от детайлите по имплементацията и дори от алгоритъма който се използва. Тъй като е невъзможна пълна алгоритмична независимост JCA предоставя стандартен API. Когато обаче разработчика иска да използва конкретна имплементация на даден алгоритъм, той може да го използва директно като го извика в програмата чрез неговото име. Алгоритмичната независимост на типове криптографски услуги е постигната чрез „engine" услуги и дефиниране на класове, които осигуряват тези услуги. Тези класове се наричени engine класове. Такива класове са: MessageDigest, Signature, KeyFactory, KeyPairGenerator и др.

Независимост на имплементацията е постигната чрез т.нар. „Provider" базирана архитектура. Терминът „Cryptographic Service Provider" – доставчик на криптографски услуги или накратко доставчик се отнася за пакети или множество от пакети, които имплементират една или повече услуги. Например, в програмата може да се извика обект на класа Signature, който имплементира даден алгоритъм като DSA (Digital Signature Algorithm), и който се получава като обект от някой от инсталираните доставчици. Друга възможност която е интегрирана в JCA е принципа на разширяемост, което ще рече, че при появата на нов алгоритъм или версия, който е подходящ за някой от engine класовете, този алгоритъм може лесно да се добави към услугите, които даден доставчик предлага.

4.2. Избор на криптографска услуга.

Доставчиците на криптографски услуги се описват в java.security файл, който се намира в директорията /java_home/jre/lib/security. В тази директория се описва приоритетът на доставчиците. Ето как става това:

```
# List of providers and their preference orders
security.provider.1=sun.security.provider.Sun
security.provider.2=sun.security.rsa.SunRsaSign
security.provider.3=com.sun.net.ssl.internal.ssl.Provider
security.provider.4=com.sun.crypto.provider.SunJCE
security.provider.5=sun.security.jgss.SunProvider
security.provider.6=com.sun.security.sasl.Provider
security.provider.7=org.jcp.xml.dsig.internal.dom.XMLDSigRI
security.provider.8=sun.security.smartcardio.SunPCSC
security.provider.9=sun.security.mscapi.SunMSCAPI
security.provider.10=org.bouncycastle.jce.provider.BouncyCastleProvider
```

Ако не е посочен конкретен доставчик, Java виртуалната машина (JVM) връща услугата от първия доставчик в списъка, който я предлага. Както се вижда от списъка, инсталацията на Java идва с голям брой предварително инсталирани доставчици.

4.3. Начин за добавяне на нов доставчик.

Има два начина да се добавят нови доставчици - статичен и динамичен. Когато се добява доставчик статично се вписва допълнителен ред в java.security файла.

Пример:

security.provider.10=org.bouncycastle.jce.provider.BouncyCastleProvider

Добавянето на доставчик динамично се извършва чрез изходен код в програмата.

Пример:

```
import java.security.Security;
import org.bouncycastle.jce.provider.BouncyCastleProvider;
. . .
Security.addProvider(new BouncyCastleProvider());
```

И в двата случая трябва предварително да се постави jar файла с имплементацията в директорията: /java_home/jre/lib/ext

5. Работа с Engine класове.

5.1. Engine класове.

Engine класовете дефинират криптографски услуги по абстрактен начин, т.е. без конкретна имплементация. Те предоставят интерфейс за дадена криптографска услуга независимо от конкретените алгоритми. Всеки engine клас има съответстващ Spi клас, напр. на класа Signature съответства SignatureSpi, на класа MessageDigest съответства MessageDigestSpi и т.н. Инстанция на engine класа капсулира като *private* поле инстанция от съответстващия му Spi клас. Освен това всички методи на engine класа са декларирани като final, т.е. не могат да се предифинират (override), а тяхната имплементация извиква съответстващия метод в Spi класа. Имплементацията на конкретен алгоритъм се предоставя от доставчика на криптографски услуги. Доставчика на услуга трябва да имплементира този Spi клас за алгоритъмите, който той иска да поддържа. Например, на класа MessageDigest е необходимо да се осигури имплементация за SHA-1, MD5, MD2 алгоритмите.

5.2. Класът Signature.

Как работят Engine класовете? За да отговорим на въпроса ще разгледаме как работи един от най-използваните Engine класове, класът Signature. Този engine клас предоставя функционалност за цифрово подписване, чрез алгоритми като DSA и RSA. Алгоритмите приемат като вход съобщение с произволна дължина и частен ключ, и връщат (генерират) масив от байтове, наричан цифров подпис. Масивът, т.е. цифровия подпис има следните свойства:

- Чрез публичния ключ съответстващ на частния ключ, с който е генериран подписа, може да се верифицира входното съобщение;
- Подписът и публичният ключ не разкриват нищо за частния ключ, с който е генериран подписа.

Обект от тип Signature може да бъде използван за подписване на електронни документи. Той също така може да се използва и за верификация. Обектите от тип Signature са модални, т.е. с тях в даден момент може да се извърши само един тип операция, задаваща определено състояние. Тези обекти могат да бъдат в три състояния: UNINITIALIZED, SIGN и VERIFY.

При първоначално създаване обектът е в състояние UNINITIALIZED. Класът Signature предоставя два инициализиращи метода initSign и initVerify, които променят състоянието съответно в SIGN и VERIFY. Инстанция на класа се създава с помощта на метода getInstance. Той приема като параметър името на алгоритъма, напр. „SHA1withDSA” и евентуално името на доставчика. Създаденият обект се инициализира с една от операциите sign или verify. Това става с един от следните методи: final void initSign(PrivateKey privateKey); final void initVerify(PublicKey publicKey); final void initVerify(Certificate certificate).

Ако създаденият обект е инициализиран и е поставен в състояние sign подписването става по следния начин:

1) Предоставя се съобщението, което трябва да се подпише с помощта на един от следните методи: final void update(byte b); final void update(byte[] data); final void update(byte[] data, int off, int len).
2) Извиква се един от методите: final byte[] sign(); final int sign(byte[] outbuf, int offset, int len) който връща цифровия подпис като масив от байтове.

Ако създаденият обект е инициализиран и е поставен в състояние VERIFY, верификацията се извършва по следния начини:

1) Данните, които трябва да бъдат верифицирани се предоставя на обекта с помощта на един от следните методи: final void update(byte b); final void update(byte[] data); final void update(byte[] data, int off, int len).
2) Извиква се един от методите: final boolean verify(byte[] signature), final boolean verify(byte[] signature, int offset, int length). Аргументите на тези методи съдържат цифровия подпис.

6. Заключение.

Може да се каже, че работата с криптографските услуги в Java е изключително лесно и не изисква много време за да започне писането на завършени програми. Езикът предлага богат избор от криптографски услуги, като техния брой се увеличава и с всяка версия на платформата се добавят новостите в тази сфера. Като заключение може да се каже, че Java™ платформата предоставя всичко необходимо за интегриране на криптографските услуги в съвременни програмни приложения.

Използвана литература:

1. Светлин Наков, Java за цивровo подписване на документи в уеб , Faber, 2005;
2. Мариян Петров, Криптографски възможности на Java платформата, Дипломна работа, ВТУ, 2009;
3. David Hook, Beginning Cryptography With Java, Wrox Press, 2005;
4. Sun Microsystems, Inc. Sun Java System Directiry Server Enterprise Edition 6.2 Reference, September 2007;
5. Sun Microsystems, Inc. , Java Cryptography Architecture API Specification & Reference, Sun Microsystems, 2004, http://java.sun.com/j2se/1.5/docs/guide/security/CryptoSpec.html
6. Sun Microsystems, Inc. , Cryptography Architecture Standard Algorithm Name Documentation, http://java.sun.com/javase/6/docs/technotes/guides/security/StandardNames.html
7. Sun Microsystems, Inc. , How to Implement a Provider for the Java™ Cryptography Architecturep, http://java.sun.com/j2se/1.4.2/docs/guide/security/HowToImplAProvider.html