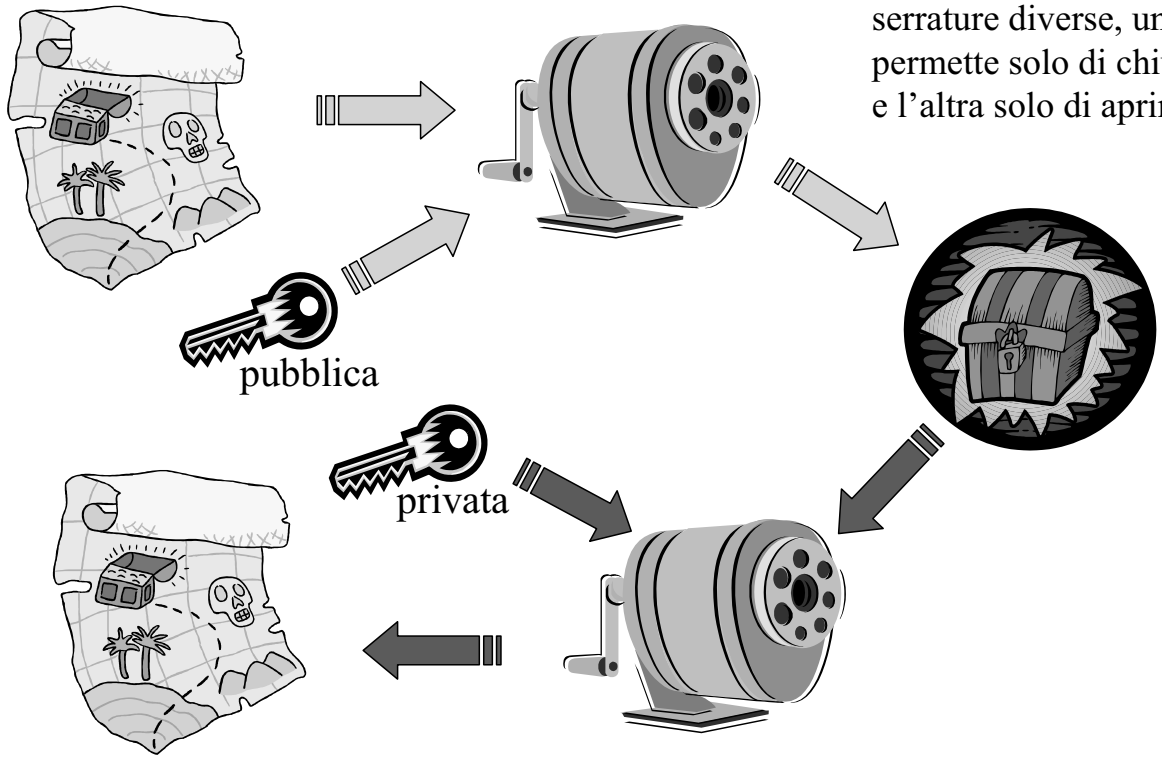
Cifratura asimmetrica

- La cifratura **SIMMETRICA** è un ottimo strumento per la sicurezza
- Il problema della cifratura simmetrica sta nel fatto che occorre avere una chiave condivisa e quindi bisogna scambiarsi questa informazione
- Questa operazione è però molto difficile da effettuarsi in modo che nessun altro scopra la chiave
- La soluzione a questo problema è risolto con la cifratura a chiave **ASIMMETRICA**

Cifratura asimmetrica

Cosa si vuole ottenere:

Il forziere ora ha due serrature diverse, una permette solo di chiudere e l'altra solo di aprire



Modalità

- Nei cifrari asimmetrici si usa quasi sempre ECB
- Tipicamente i cifrari asimmetrici sono utilizzati per cifrare un singolo blocco in chiaro
- Generalmente le dimensioni di quel blocco sono grandi quasi come quelle della chiave
- Se è necessario cifrare più dati si utilizzerà la cifratura a chiave di sessione che vedremo in seguito

Padding

- Nella cifratura asimmetrica non si usa PKCS#5
- Gli standard per cifrare con RSA sono invece PKCS#1 e OAEP (Optimal Asymmetric Encryption Padding)
- Di seguito non sarà illustrato il funzionamento di tali forme di padding
- Per chi fosse interessato : www.rsasecurity.com/rsalabs/pkcs/pkcs-1/index.html

RSA

Occorre un provider che supporti tale algoritmo
Es. BouncyCastle (←)
In J2SE1.4 RSA è supportato solo per le operazioni di firma e verifica

```
import javax.crypto.*;
import java.security.*;
…..
KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("RSA");

keyPairGenerator.initialize(1024);

KeyPair keyPair = keyPairGenerator.genKeyPair();

Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");

cipher.init(Cipher.ENCRYPT_MODE, keyPair.getPublic());

byte[] messageBytes = new String(“Combinazione:1234”).getBytes(“UTF8”);
byte[] cipherText = cipher.doFinal(messageBytes);

cipher.init(Cipher.DECRYPT_MODE, keyPair.getPrivate());

byte[] decryptedMessageBytes = cipher.doFinal(cipherText);
System.out.println(new String(decryptedMessageBytes,”UTF8”));
.....
```

Questa volta occorre una coppia di chiavi

Per la cifratura occorre inizializzare il cifrario con una PublicKey

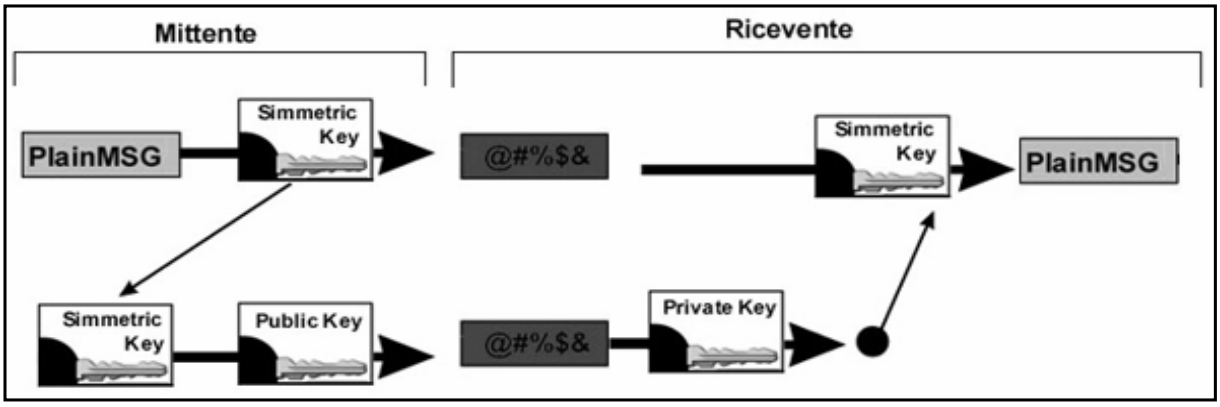
Per la decifrazione occorre inizializzare il cifrario con una PrivateKey

Cifratura a chiave di sessione

La cifratura asimmetrica è 1000 volte più lenta di quella simmetrica

Per gestire questa inefficienza in genere si utilizza la cifratura a Chiave di Sessione

Cifratura a chiave di sessione



1. Il messaggio in chiaro è cifrato con una chiave simmetrica
2. Poi questa a sua volta viene cifrata con la chiave pubblica del destinatario
3. Il destinatario quindi decifra la chiave simmetrica con la sua chiave privata
4. Infine il destinatario può decifrare il messaggio usando la chiave simmetrica ottenuta

La chiave simmetrica viene cambiata ad ogni sessione (da qui il nome **chiave di sessione**)
La cifratura asimmetrica coinvolge poco testo in chiaro risultando in questo modo efficiente

Cifratura di una chiave simmetrica

```
import java.security.*;
import javax.crypto.*;
import javax.crypto.spec.*;
.....
KeyGenerator keyGenerator = KeyGenerator.getInstance("Blowfish");
keyGenerator.init(128);
Key blowfishKey = keyGenerator.generateKey();

KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("RSA");
keyPairGenerator.initialize(1024);
KeyPair keyPair = keyPairGenerator.genKeyPair();

Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");
cipher.init(Cipher.ENCRYPT_MODE, keyPair.getPublic());

byte[] blowfishKeyBytes = blowfishKey.getEncoded();

byte[] cipherText = cipher.doFinal(blowfishKeyBytes);
cipher.init(Cipher.DECRYPT_MODE, keyPair.getPrivate());
byte[] decryptedKeyBytes = cipher.doFinal(cipherText);

SecretKey blowfishKey = new SecretKeySpec(decryptedKeyBytes,"Blowfish");
.....
```

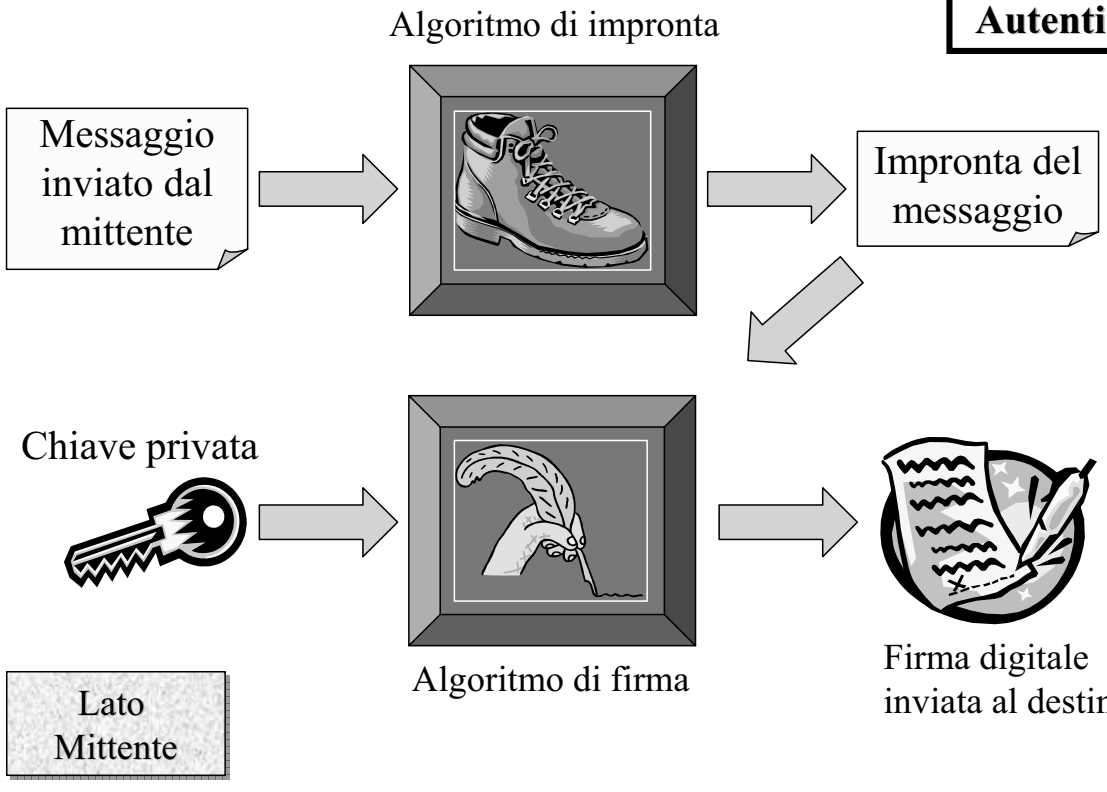
Generazione della chiave di sessione

Generazione di una coppia di chiavi RSA

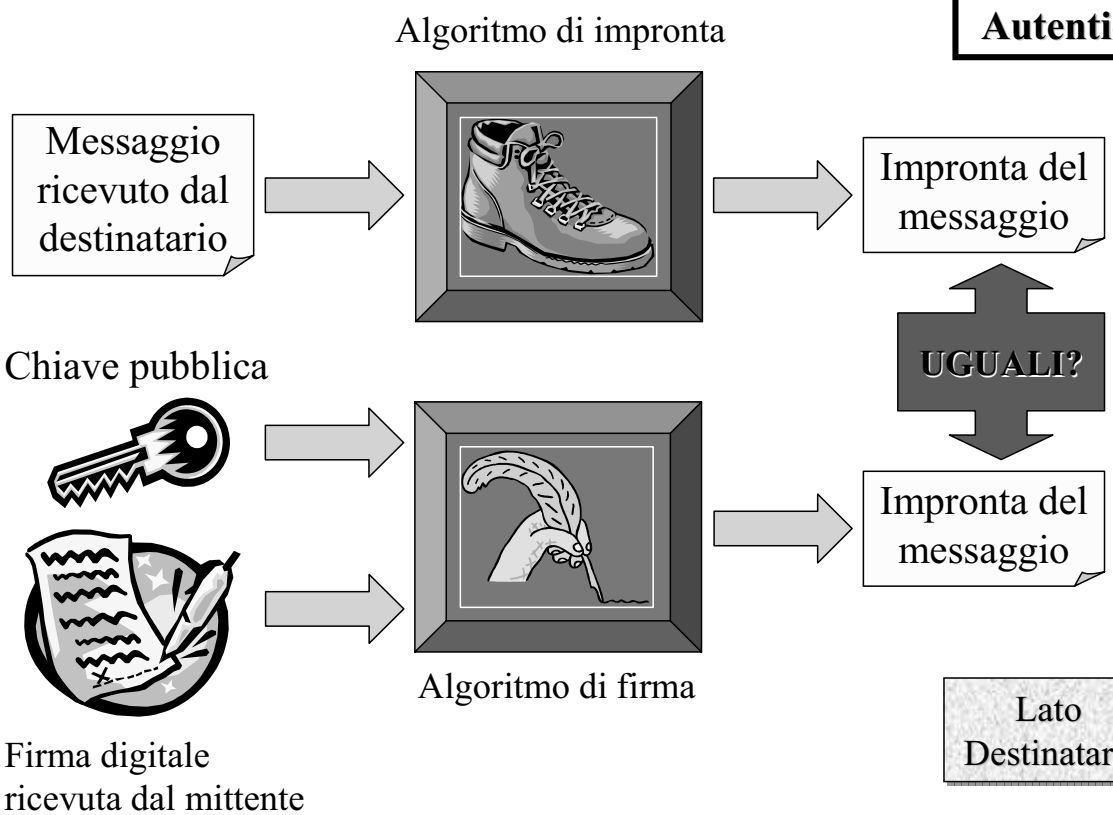
Trasformazione della chiave nel formato richiesto per l'operazione di cifratura

Riconversione dei byte in chiave

Firme digitali (1)



Firme digitali (2)



Firmare e verificare

```
import java.security.Signature;                 import java.security.SignatureException;
import java.security.KeyPair;                   import java.security.KeyPairGenerator;
.....
KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");
kpg.initialize(1024);
KeyPair keyPair = kpg.genKeyPair();
byte[] messaggio = new String("Non esistono più le mezze stagioni").getBytes("UTF8");

Signature sig = Signature.getInstance("MD5WithRSA");

sig.initSign(keyPair.getPrivate());
sig.update(messaggio);
byte[] signatureBytes = sig.sign();
.....
sig.initVerify(keyPair.getPublic());
sig.update(messaggio);

boolean verified = false;
try { verified = sig.verify(signatureBytes); }
catch (SignatureException se) { System.out.println("La firma ha un formato non valido!"); }
if (verified) { … } else { … }
.....
```

Per firmare occorre istanziare un oggetto Signature

Per firmare si inizializza con la chiave privata
update() passa i dati da firmare
sign() restituisce i byte della firma digitale

Per la verifica della firma si inizializza con la chiave pubblica
update() passa i dati da verificare

SignatureException è lanciata, ad esempio, in caso di perdita di un byte e non perchè la verifica ha dato esito negativo

RSA e DSA a confronto

- Firmare con RSA significa cifrare con la chiave privata e decifrare con la pubblica
- Questa operazione non nasconde i dati (perchè decifrabili attraverso la chiave pubblica) ma prova l'identità del firmatario
- Il DSA (Digital Signature Algorithm) funziona come RSA per la firma ma non può essere usato per cifrare
- Il DSA è più veloce nel generare le firme mentre RSA nel convalidarle

Una firma viene convalidata più spesso di quanto non venga generata
→ RSA è più veloce per la maggior parte delle applicazioni

Identità della chiave pubblica

- Per convalidare una firma occorre una chiave pubblica
- Ma come essere certi che la chiave pubblica è autentica?
- Serve un qualche mezzo per averne la certezza

I CERTIFICATI sono il tentativo di risolvere questo problema attribuendo l'identità ad una chiave pubblica in modo inconfutabile

- Java offre funzionalità di gestione dei certificati attraverso un set di packages appositamente dedicato
- Tale componente prende il nome di Java CertPath

JSSE
Java Secure Socket Extension

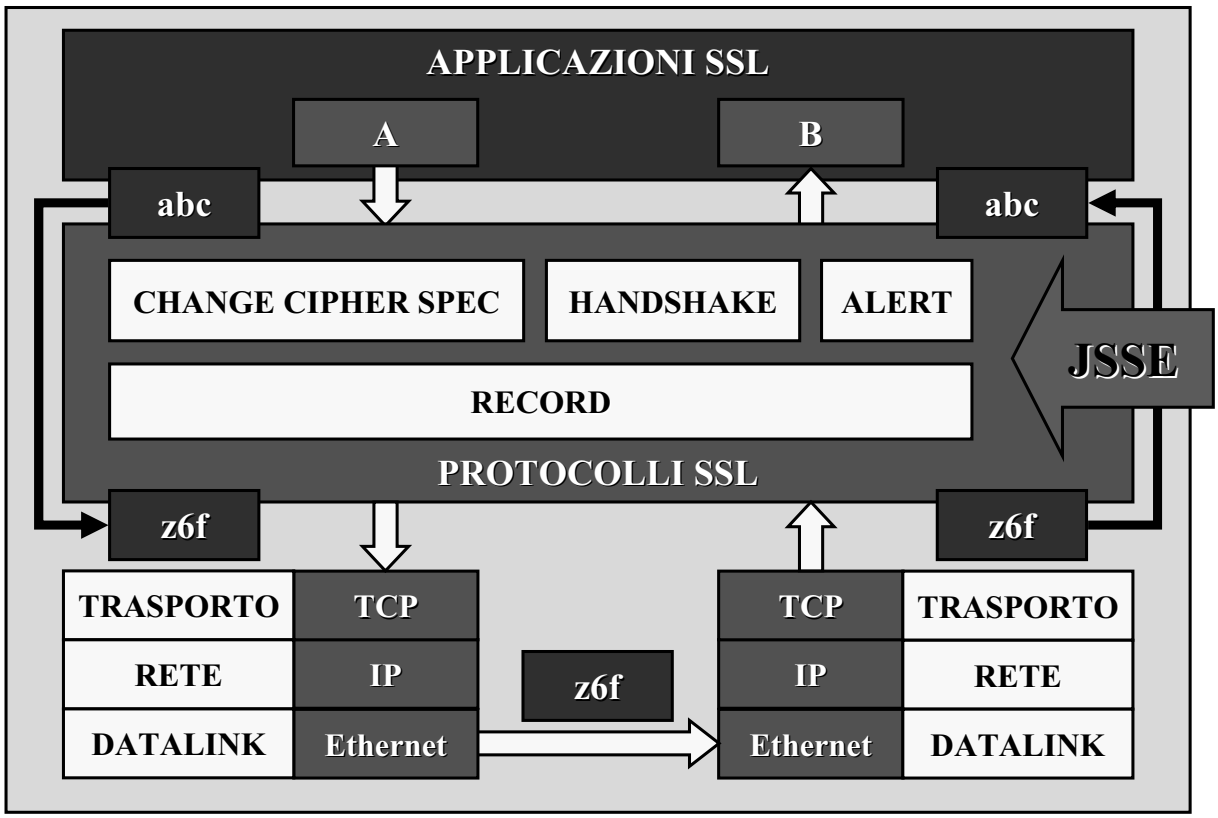
Caratteristiche

- La Java Secure Socket Extensions (JSSE) offre:
  - funzionalità di autenticazione
  - protezione dell'integrità dei dati
  - protezione della riservatezza dei dati
- Mediante i protocolli per comunicazioni sicure:
  - SSL (Secure Socket Layer) v2.0 e v3.0
  - TLS (Transport Layer Security) v1.0
- Mentre JCE opera su specifici dati locali, JSSE adotta una differente astrazione applicando meccanismi crittografici a livello di rete
- Le JSSE API contenute in javax.net e javax.net.ssl estendono:
  - javax.crypto (JCE)
  - java.security (JCA)
  - java.net

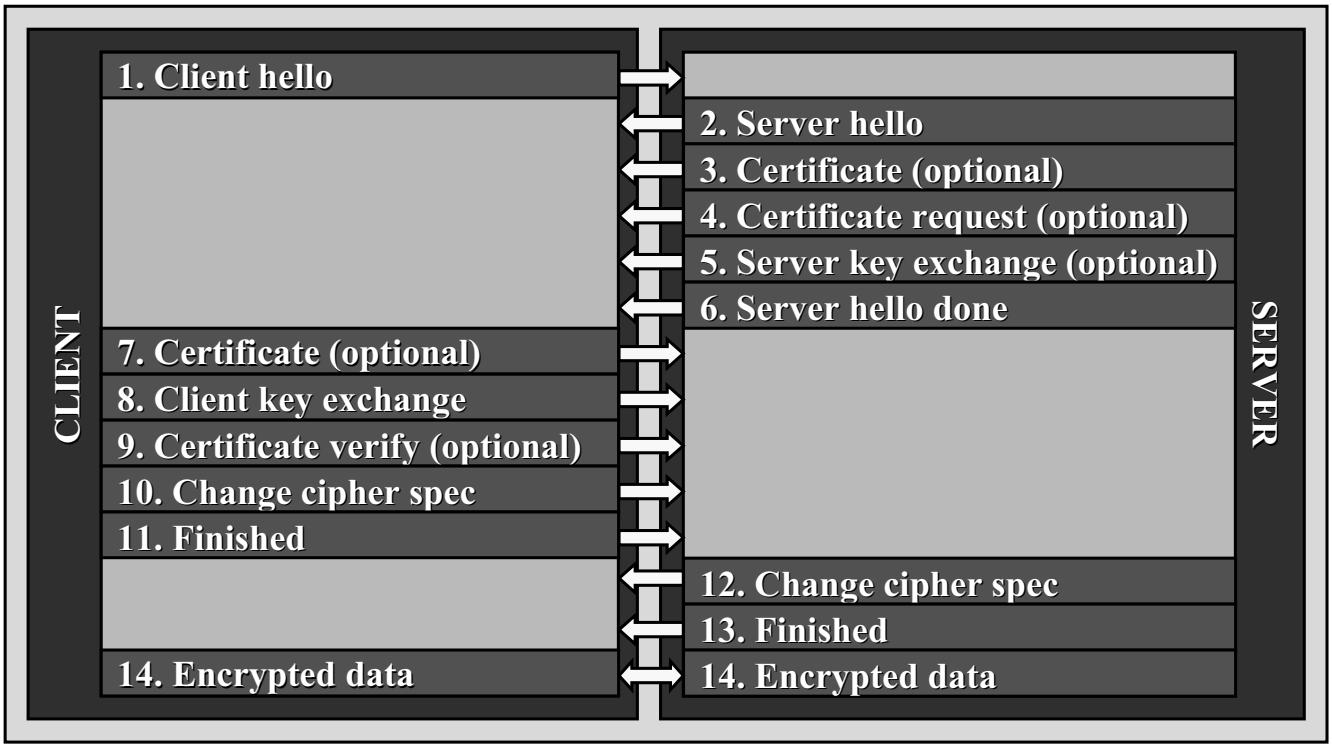
Caratteristiche

- Le funzionalità crittografiche di JSSE sono implementate dal provider "**SunJSSE**“
- Il protocollo di applicazione più adoperato con JSSE è HTTP (Hyper Text Transfer Protocol) che assume la dicitura di HTTPs
- Molti altri protocolli possono usufruire di JSSE, come NNTP (Net News Transfer Protocol), Telnet, LDAP (Lightweight Directory Access Protocol), IMAP (Interactive Message Access Protocol) ed FTP (File Transfer Protocol)
- Novità di JSSE nella versione integrata:
  - Il package javax.security.cert è presente solo per compatibilità con le applicazioni realizzate con JSSE opzionale. Ad esso è da preferire il package java.security.cert di CertPath, integrato in J2SE a partire dalla versione 1.4
  - JSSE può fare uso del provider "SunJCE" per operazioni di cifratura e decifrazione

Architettura SSL



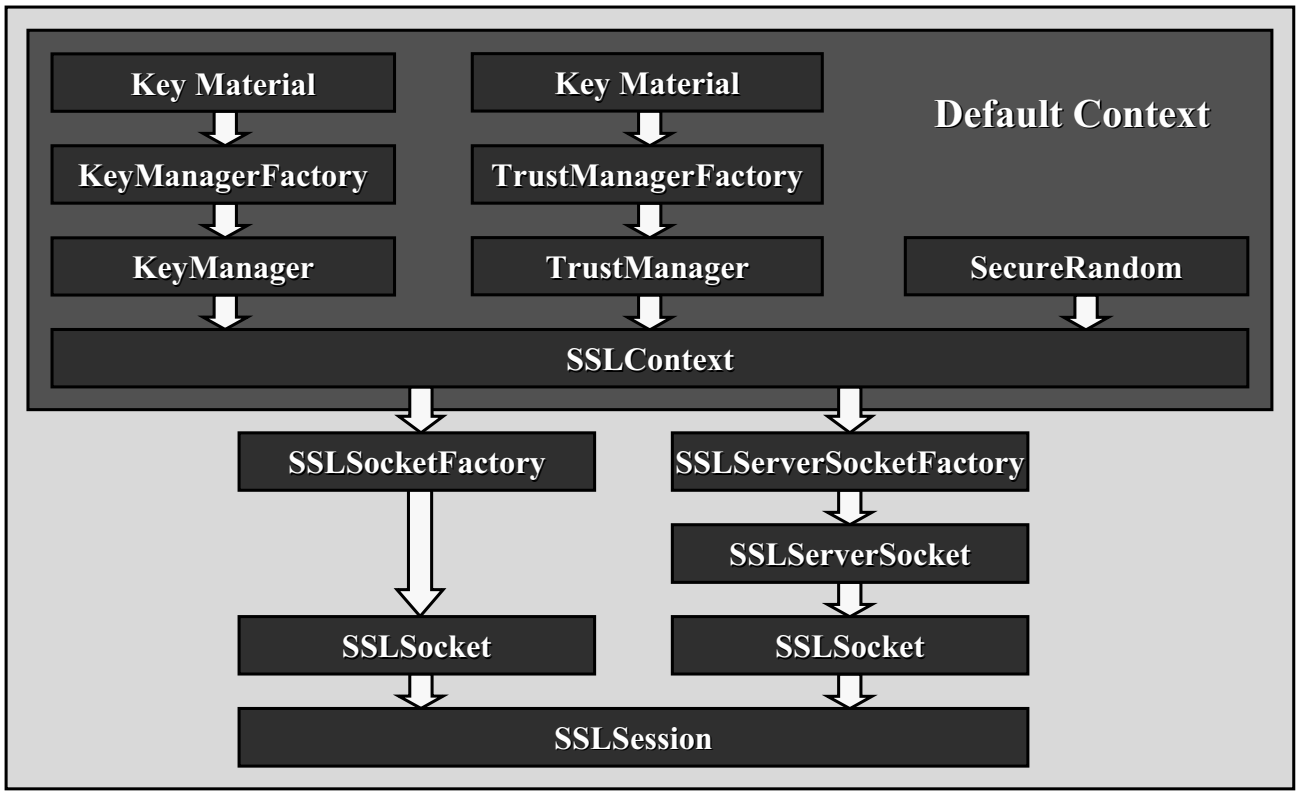
Protocollo di Handshake



Classi principali

- Factories SSLServerSocketFactory e SSLSocketFactory per la creazione di Socket, ServerSocket, SSLSocket, SSLServerSocket con incapsulamento delle caratteristiche di costruzione e configurazione
  (socket factory framework)
- Classe SSLContext che agisce da factory per le socket factories e realizza il concetto di contesto sicuro
- Interfacce KeyManager e TrustManager (incluse specifiche X.509) per la gestione delle chiavi e delle decisioni inerenti la sicurezza
  (keystore e truststore)
- Factories KeyManagerFactory e TrustManagerFactory
- SSLSession per la gestione server di sessioni SSL residenti in memoria
- Connessioni sicure con HttpsURLConnection

Relationship



Creazione di una SSLContext

```
.....
SSLContext ctx = SSLContext.getInstance("TLS","SunJSSE");

KeyManagerFactory kmf = KeyManagerFactory.getInstance("SunX509","SunJSSE");
TrustManagerFactory tmf = TrustManagerFactory.getInstance("SunX509","SunJSSE");

SecureRandom random = SecureRandom.getInstance("SHA1PRNG", "Sun");
byte seed[] = random.generateSeed(20);
random.setSeed(seed);

KeyStore ks = KeyStore.getInstance("PKCS12");
char[] passphrase = "passphrase".toCharArray();
ks.load(new FileInputStream("testkeys"), passphrase);

kmf.init(ks,passphrase);
kmt.init(ks);
ctx.init(kmf.getKeyManagers(), kmt.getTrustManagers(), random);
.....
```

Creazione di un SSLContext dal provider SunJSSE con gestione del protocollo TLS

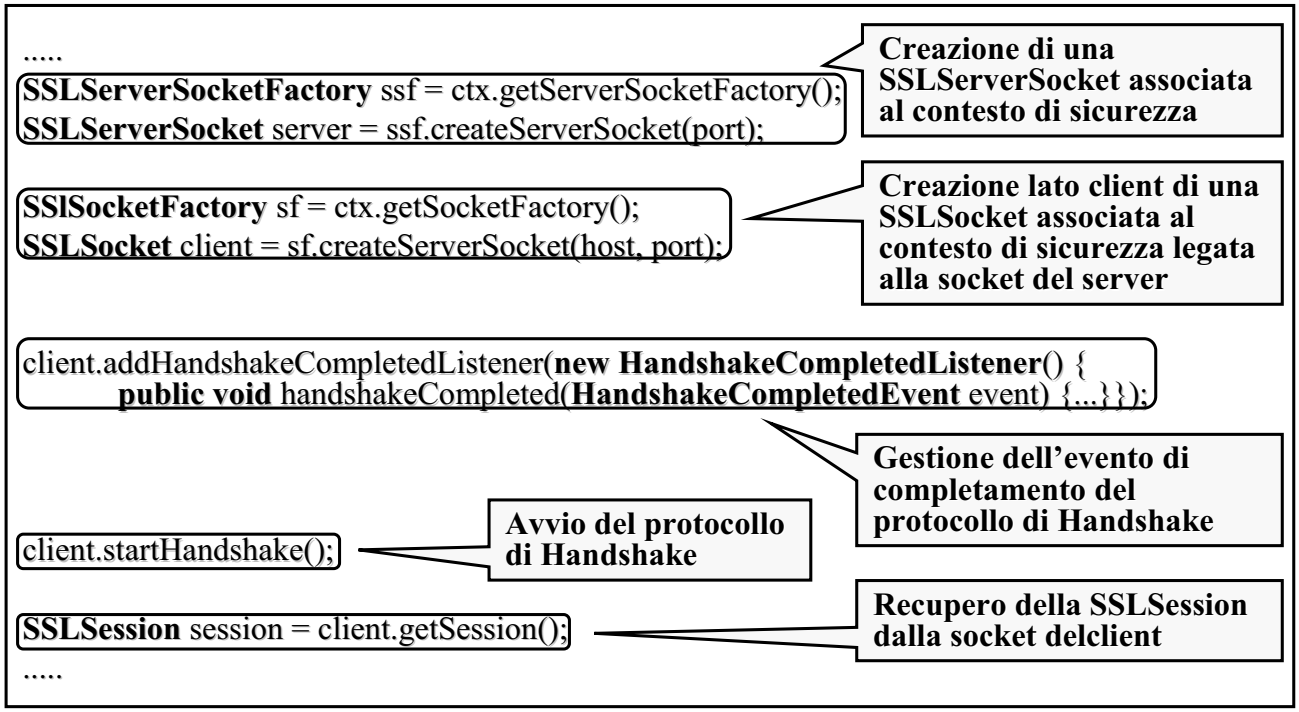
Creazione di un KeyManagerFactory e TrustManagerFactory con gestione del protocollo SunX509

Creazione di un SecureRandom e generazione di un seme di inizializzazione mediante l'algoritmo SHA1PRNG fornito dal provider Sun

Creazione di un KeyStore secondo lo standard PKCS#12
load() per caricamento da file

Inizializzazione del contesto SSL

Creazione di una SSLSession



"SunJSSE" Provider

- Funzionalità crittografiche implementate da JSSE:

ALGORITMO CRITTOGRAFICO	PROCESSO CRITTOGRAFICO	LUNGHEZZA CHIAVE (BITS)
RSA	**autenticazione e scambio della chiave**	**2048 (autenticazione)** **2048 (scambio della chiave)** **512 (scambio della chiave)**
RC4	**cifratura**	**128** **128 (40 effettivi)**
DES	**cifratura**	**64 (56 effettivi)** **64 (40 effettivi)**
TripleDES	**cifratura**	**192 (112 effettivi)**
Diffie-Hellman	**scambio della chiave**	**1024** **512**
DSA	**autenticazione**	**1024**

“SunJSSE” Provider

- Engine classes della JCA implementate dal provider della JSSE:

ENGINE CLASS	ALGORITMO - PROTOCOLLO
KeyFactory	“RSA”
KeyPairGenerator	“RSA”
KeyStore	“PKCS12”
Signature	“MD2withRSA”
Signature	“MD5withRSA”
Signature	“SHA1withRSA”
KeyManagerFactory	“SunX509”
TrustManagerFactory	“SunX509” - “SunPKIX”
SSLContext	“SSL”
SSLContext	“SSLv3”
SSLContext	“TLS”
SSLContext	“TLSv1”

Cipher Suites

- Cipher suites supportate da "SunJSSE" in ordine di preferenza per default:

CIPHER SUITES	NUOVE IN J2SEv1.4
SSL_RSA_WITH_RC4_128_MD5	
SSL_RSA_WITH_RC4_128_SHA	
TLS_RSA_WITH_AES_128_CBC_SHA	X
TLS_DHE_RSA_WITH_AES_128_CBC_SHA	X
TLS_DHE_DSS_WITH_AES_128_CBC_SHA	X
SSL_RSA_WITH_3DES_EDE_CBC_SHA	
SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA	X
SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA	
SSL_RSA_WITH_DES_CBC_SHA	
SSL_DHE_RSA_WITH_DES_CBC_SHA	X
SSL_DHE_DSS_WITH_DES_CBC_SHA	
SSL_RSA_EXPORT_WITH_RC4_40_MD5	
SSL_DSS_EXPORT_WITH_DES40_CBC_SHA	X
.....	

JAAS
Java Authentication and Authorization Service

Caratteristiche

- Java Authentication and Authorization Service (JAAS) offre due servizi:
  - autenticazione - identificazione sicura dell'utente
  - autorizzazione - verifica dei permessi necessari per le operazioni richieste
- Attraverso:
  - configurazione dinamica dei moduli di login
  - operazioni di callback
  - controllo degli accessi
  - distribuzione dei permessi
- Il framework di autenticazione è progettato per essere "pluggable" in base al modello PAM (Pluggable Authentication Module), che rende indipendenti le applicazioni dalle tecnologie di autenticazione sottostanti come:
  - verifica di username e password
  - verifica di caratteristiche biometriche (voce, impronte digitali, ...)
- Inoltre JAAS supporta lo stacking dei moduli di autenticazione
- Il componente JAAS per le autorizzazioni opera in congiunzione con il modello Java per il controllo degli accessi a risorse sensibili (Java Security Framework)

Classi principali

- Le JAAS APIs sono contenute nel set di packages javax.security.auth:

Identificazione

- Subject – sorgente di richiesta di autenticazione nell'accesso ad una risorsa
- Principal – interfaccia che rappresenta l'identità di un Subject se l'autenticazione ha successo
- Refreshable / Destroyable – interfacce che esprimono la capacità di refresh e di reset delle credenziali associate ad un Subject

Autenticazione

- LoginModule – interfaccia per l'implementazione delle diverse tecnologie di autenticazione
- LoginContext – offre i metodi per autenticare un Subject e determinare i servizi di autenticazione o i LoginModules configurati per l'applicazione
- CallbackHandler – interfaccia implementata dall'applicazione e passata al LoginContext, attraverso la quale i LoginModules comunicano bidirezionalmente con gli utenti
- Callback – package che offre diverse implementazioni da usare mediante CallbackHandler

Autorizzazione

- Policy – classe astratta per definire le politiche di accesso al sistema
- AuthPermission – incapsula i permessi di base richiesti in JAAS per l'autenticazione
- Configuration - rappresenta la configurazione di un LoginModule
- PrivateCredentialPermission – protegge l'accesso alle credenziali di un Subject privato

Identificazione

- Un Subject contiene tre tipologie di informazioni per l'identificazione:
  - Identities - nella forma di uno o più Principals
  - Public credentials - come nome e chiavi pubbliche
  - Private credentials - come password e chiavi private
- Implementazioni dell'interfaccia Principal sono le seguenti classi:
  - java.security.Identity
  - javax.security.auth.kerberos.KerberosPrincipal
  - javax.security.auth.x500.X500Principal
- Qualsiasi Object può rappresentare una credential pubblica o privata

```
Subject subject;
Principal principal;
Object credential;
subject.getPrincipals().add(principal);
subject.getPublicCredentials().add(credential);
subject.getPrivateCredentials().add(credential);
```

Login Modules

- Per costruire un modulo di autenticazione personalizzato occorre implementare l'interfaccia LoginModule
- J2SE 1.4 mette a disposizione un set di LoginModules ready-to-use:
  - JndiLoginModule - login con un directory service configurato sotto JNDI (Java Naming and Directory Interface)
  - Krb5LoginModule - login con i protocolli di Kerberos v5
  - NTLoginModule - login con le informazioni di current user in NT
  - UnixLoginModule - login con le informazioni di current user in UNIX
- In com.sun.security.auth sono disponibili alcune relative implementazioni dell'interfaccia Principal, come NTDomainPrincipal e UnixPrincipal
- La classe LoginContext carica dinamicamente la configurazione di un modulo di login da un file di testo con la seguente struttura:

```
Application {
   ModuleClass Flag ModuleOptions;
   ModuleClass Flag ModuleOptions;
   .....
};
Application { ..... };
.....
```


Callbacks e CallbackHandlers

- CallbackHandlers e Callbacks raccolgono le informazioni necessarie per l'autenticazione di un utente o un sistema
- Rendono indipendenti i LoginModules dai meccanismi di interazione
- J2SE 1.4 mette a disposizione due CallbackHandlers all'interno del package com.sun.security.auth.callback:

  DialogCallbackHandler

  TextCallbackHandler
- Sette Callbacks, presenti in javax.security.auth.callback, sono implementate da JAAS e possono essere adoperate per realizzare CallbackHandlers personalizzati:

  ChoiceCallback
  ConfirmationCallback
  LocaleCallback
  NameCallback
  PasswordCallback
  TextInputCallback
  TextOutputCallback

Esempio

Applicazione client

```
.....
MyCallbackHandler handler = new MyCallbackHandler()
LoginContext lc = new LoginContext("Example", handler);
lc.login();
.....
```

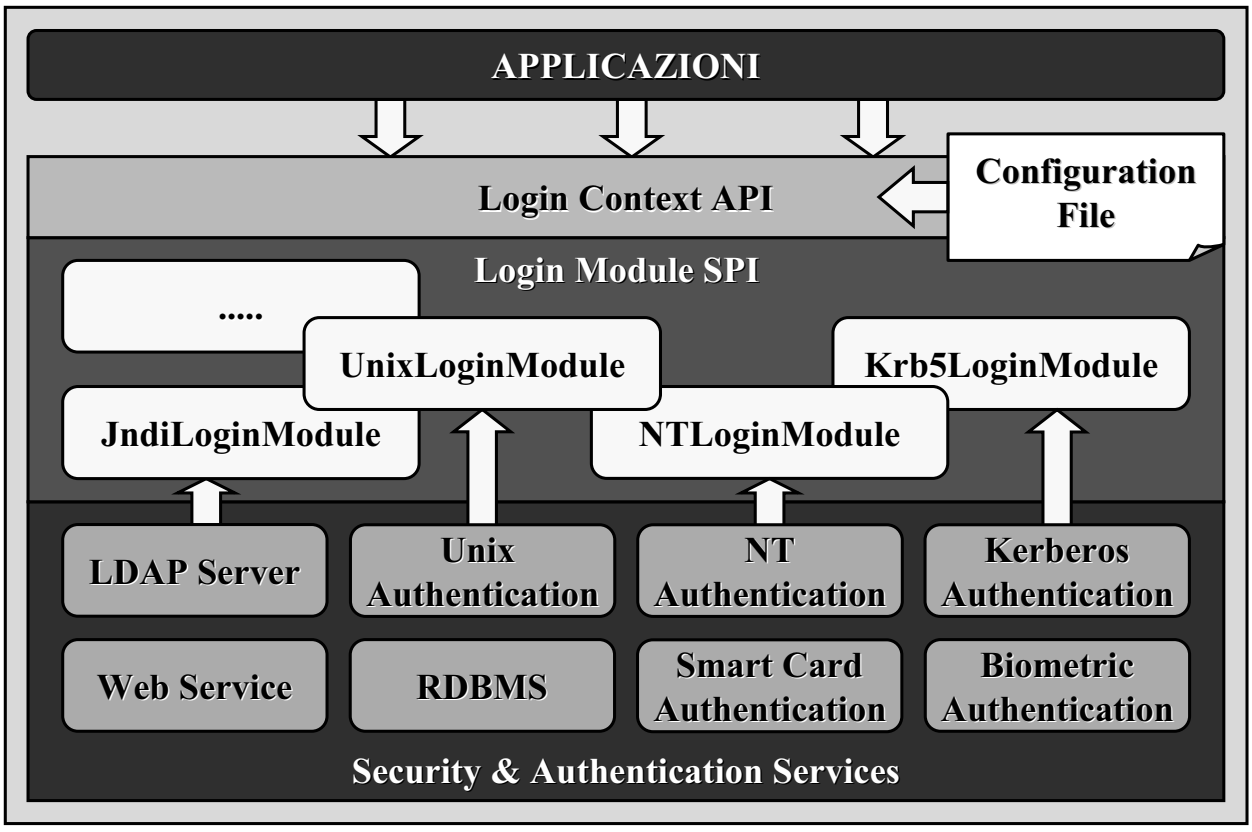
File di configurazione JAAS - "jaas.config"

```
Example {
        RdbmsLoginModule required
            debug="true"
            url="jdbc:mysql://localhost/JAAS"
            driver="gwe.sql.gweMysqlDriver";
};
```

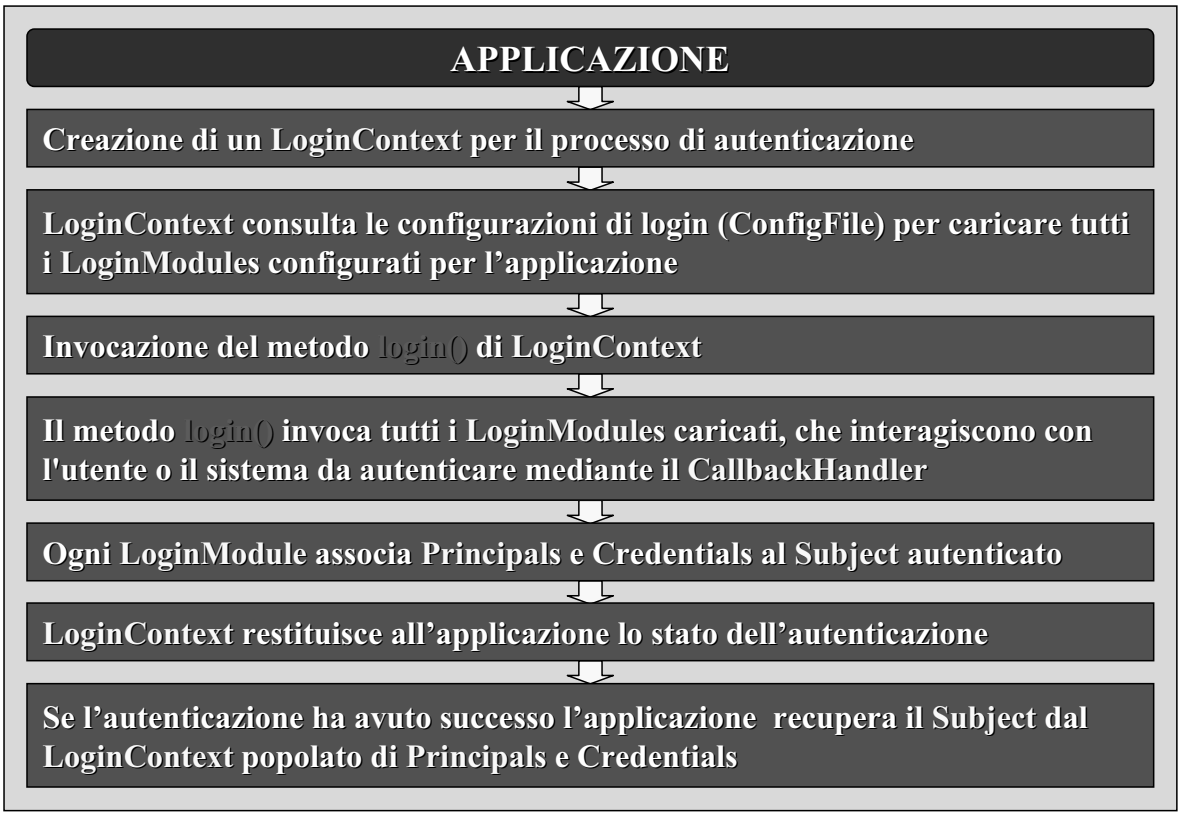
Esecuzione con impostazione della javax.security.auth.login.config

```
java -Djavax.security.auth.login.config = jaas.config MyApp"
```


Pluggable Authentication



Processo di Autenticazione



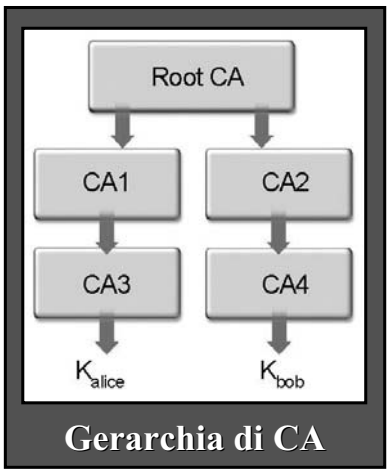
Controllo degli Accessi

- JAAS offre a tempo di esecuzione rinforza il controllo degli accessi ad un sistema
- La classe SecurityManager viene consultata ogni volta che una operazione sensibile debba essere effettuata
- Il SecurityManager a sua volta delega la responsabilità del controllo ad un AccessController
- L'AccessController si procura una immagine dell'AccessControlContext corrente e verifica se si hanno i permessi necessari per l'operazione richiesta
- JAAS offre due metodi che permettono di associare dinamicamente un Subject autenticato con il thread AccessControlContext corrente:
  - Subject.doAs()
  - Subject.doAsPrivileged()
- Dopo l'associazione ed il controllo dei permessi viene eseguita o meno l'operazione richiesta a seconda delle autorizzazioni concesse

CertPath
Java Certification Path

Caratteristiche

- Java Certification Path (CertPath) offre interfacce ed engine classes per gestire certificati e certification paths, ovvero liste ordinate di certificati derivanti da una gerarchia di Certification Authority (CA)
- Le funzionalità offerte possono essere suddivise in quattro categorie:
  - Basic Certification Path
    - CertPath, CertificateFactory, CertPathParameters
  - Certification Path Validation
    - CertPathValidator, CertPathValidatorResult
  - Certification Path Building
    - CertPathBuilder, CertPathBuilderResult
  - Certificate/CRL Storage
    - CertStore, CertStoreParameters, CertSelector, CRLSelector
- CertPath offre inoltre la possibilità di costruire e validare certification paths **X.509** secondo gli standards **PKIX**



Gerarchia di CA

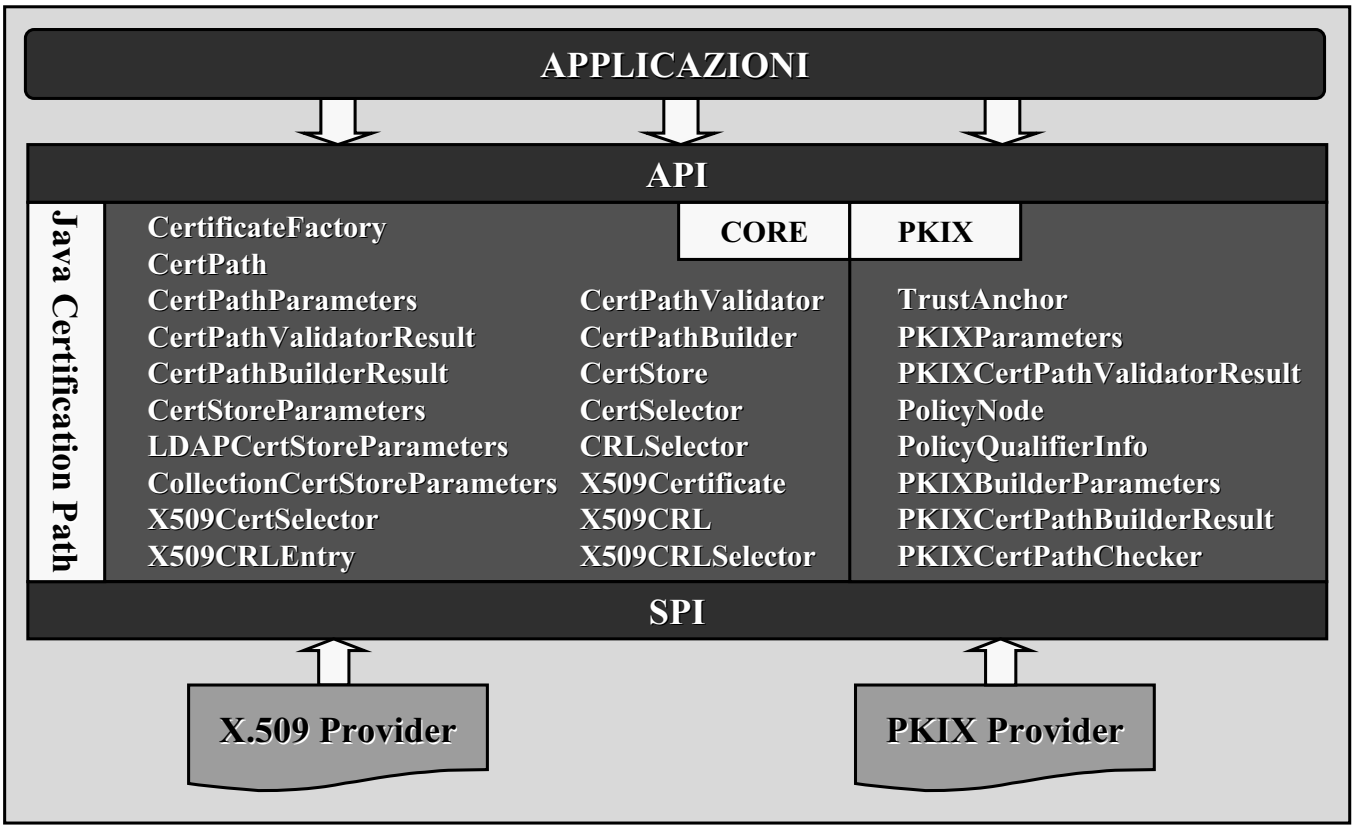
Core Classes

- Le principali CertPath APIs sono contenute in java.security.cert:
  - CertPath –definisce le funzionalità comuni per un certification path
  - CertificateFactory –definisce funzionalità di una factory di certificati
  - CertPathParameters – interfaccia per la rappresentazione trasparente del set di parametri usati con un particolare CertPathBuilder o con un algoritmo di validazione
  - CertPathValidator –valida un certification path
  - CertPathValidatorResult – interfaccia per la rappresentazione trasparente del risultato favorevole o dell'output di un algoritmo di validazione di una lista di certicati
  - CertPathBuilder –costruisce un certification path
  - CertPathBuilderResult – interfaccia per la rappresentazione trasparente del risultato o dell'output di un algoritmo di costruzione di un certification path
  - CertStore –fornisce le funzionalità di deposito di una CRL
  - CertStoreParameters – rappresentazione trasparente del set di parametri usati con un particolare CertStore (LDAPCertStoreParameters, CollectionCertStoreParameters)
  - CertSelector / CRLSelector – interfacce per la specifica dell'insieme di criteri adottati per selezionare certificati e CRLs da una collezione di certificati o di CRLs
  - X509Certificate / X509CertSelector - gestiscono certificati X.509
  - X509CRL / X509CRLEntry / X509CRLSelector - gestiscono CRL

PKIX Classes

- Le classi più importanti inerenti gli standard PKIX sono le seguenti:
  - TrustAnchor - rappresenta una "most-trusted" CA usata come root nella validazione di certification paths (non modificabile e thread-safe)
  - PKIXParameters - specifica il set di parametri di ingresso per l'algoritmo PKIX di validazione di certification paths
  - PKIXCertPathValidatorResult - rappresenta il risultato di una validazione
  - PolicyNode - interfaccia che rappresenta un nodo di un albero di policy risultante da un algoritmo di validazione eseguito con successo
  - PolicyQualifierInfo - rappesenta il policy qualifier contenuto nella Certificate Policy Extension del certificato a cui la policy fa riferimento
  - PKIXBuilderParameters - specifica il set di parametri di ingresso per l'algoritmo PKIX di costruzione di certification paths
  - PKIXCertPathBuilderResult - rappresenta il risultato di una costruzione
  - PKIXCertPathChecker - permette di eseguire controlli su certificati X.509

CertPath Architecture



Certificati digitali

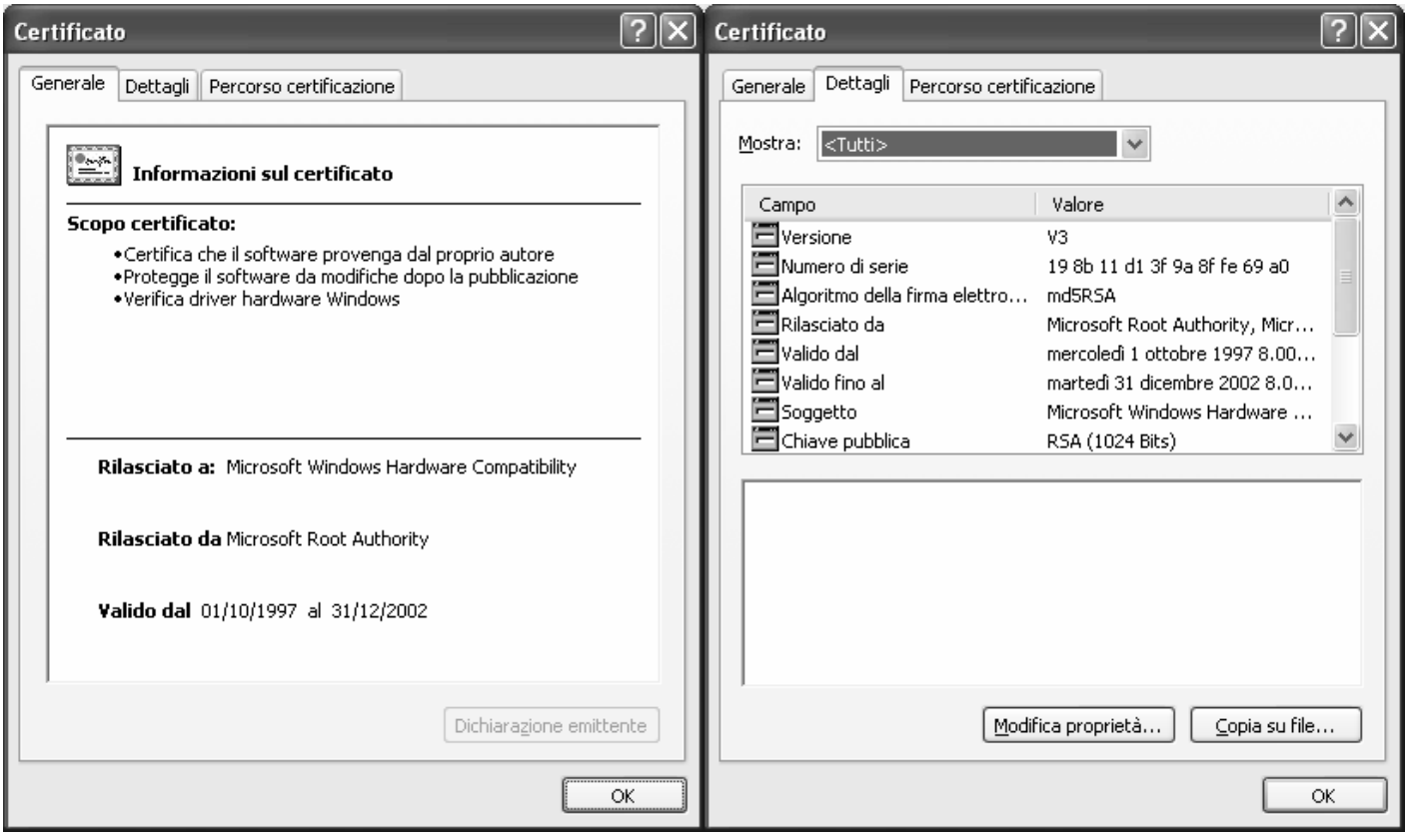
Un certificato digitale è la garanzia fornita da una terza parte che una chiave pubblica appartiene al suo proprietario

- Questa terza parte è detta *Certification Authority* (CA)
- Le due CA più note sono Verisign e Thawte (in realtà una filiale di Verisign)
  - Per l'elenco dei certificatori attivi in Italia visitare il sito del Centro Nazionale per l'Informatica nella Pubblica Amministrazione (www.cnipa.gov.it)
- La CA certifica una chiave pubblica firmandola con la sua chiave privata
- Per default, il JDK utilizza i certificati **X.509**:
  - Sono gli standard maggiormente utilizzati per i certificati digitali.
  - Sono definiti in RFC 2459 reperibile all'indirizzo www.ietf.org/rfc/rfc2459.txt.
  - Esistono 3 versioni di X.509: v1,v2 e v3.
  - *v2* e *v3* aggiungono ai certificati alcune informazioni supplementari

Contenuto dei certificati

- Certificato X.509 v1:
  - Versione
    - definisce la versione del certificato: v1,v2,v3
  - Numero seriale del certificato
    - intero univoco che identifica la CA che emette il certificato
  - Identificatore dell'algoritmo di firma
    - definisce l'algoritmo di firma usato dalla CA
  - Periodo di validità
    - definisce il periodo di tempo per cui è valido il certificato.
    - si usano 2 date: *non prima* e *non dopo*
    - il certificato è valido solo nell'intervallo
  - Soggetto
    - indica a chi è stato emesso il certificato
    - i soggetti sono memorizzati con nomi X.500
  - Chiave pubblica del soggetto
  - Firma dell'Autorità di certificazione

Esempio in Windows



Stampa di un certificato DER

```
import java.io.*;
import java.security.cert.*;
.....
String nameCert = "File_Certificate.cer";
FileInputStream fis = new FileInputStream(nameCert);
CertificateFactory cf = CertificateFactory.getInstance("X.509");
X509Certificate x509cert = (X509Certificate)cf.generateCertificate(fis);
System.out.println("\nInformazioni reperite dal certificato: " +nameCert+"\n");
System.out.println("tipo = " + x509cert.getType());
System.out.println("versione = " + x509cert.getVersion());
System.out.println("soggetto = " + x509cert.getSubjectDN().getName());
System.out.println("inizio validità = " + x509cert.getNotBefore());
System.out.println("fine validità = " + x509cert.getNotAfter());
System.out.println("numero di serie = " + x509cert.getSerialNumber().toString(16));
System.out.println("emettitore = " + x509cert.getIssuerDN().getName());
System.out.println("algoritmo di firma = " + x509cert.getSigAlgName());
System.out.println("algoritmo per la chiave pubblica = " +
                   x509cert.getPublicKey().getAlgorithm());
fis.close();
.....
```

Il certificato deve essere in formato DER (Distinguished Encoding Rules) (estensione .cer)
Un solo certificato per file

Occorre aprire il relativo file ed istanziare un certificato con CertificateFactory inizializzato allo standard X.509
generateCertificate() richiede un cast perchè restituisce un Certificate

Certificate Revocation List (CRL)

```
import java.io.*;
import java.security.cert.*;
.....
String nameCRL = "File_CRL.crl";
String nameCert = "File_Certificate.cer";
FileInputStream inCRL = new FileInputStream(nameCRL);
FileInputStream inCert = new FileInputStream(nameCert);

CertificateFactory cf = CertificateFactory.getInstance("X.509");
X509CRL crl = (X509CRL)cf.generateCRL(inCRL);

Certificate certificate = cf.generateCertificate(inCert);

inCRL.close();
inCert.close();

if (crl.isRevoked(certificate)) System.out.println("Il certificato è stato revocato!");
else System.out.println("Il certificato "+nameCert+" e' ok!");
.....
```

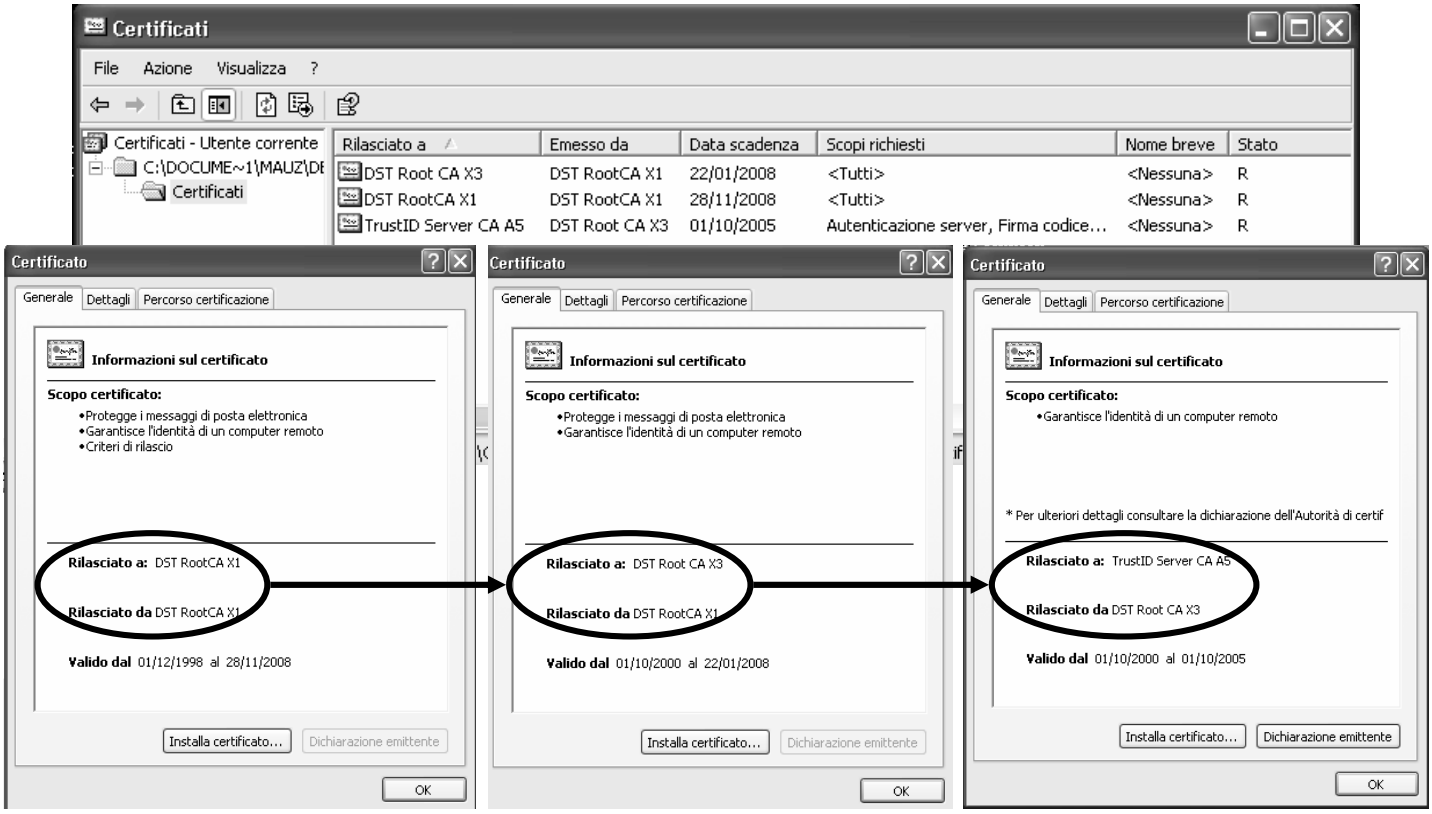
Il file deve contenere una CRL (estensione .crl)

Creazione di una CRL di X.509

Verifica del cetificato attraverso il metodo isRevoked()

Esempio di Certification Path

- CertPath Root della Digital Signature Trust (DST Root Cert.p7b):



Stampa di un certification path

```
import java.io.*;
import java.security.cert.*;
import java.util.Collection;
import java.util.Iterator;
.....
String nameCert = "File_Certificates.p7b";

FileInputStream fis = new FileInputStream(nameCert);
CertificateFactory cf = CertificateFactory.getInstance("X.509");
Collection c = cf.generateCertificates(fis);

Iterator it = c.iterator();
int j = 1;
while (it.hasNext()) {
      System.out.println("certificato N°."+j+ ": ");
      X509Certificate x509cert = (X509Certificate)it.next();
      System.out.println(x509cert);
      j++;
}
fis.close();
.....
```

Il file deve contenere una catena di certificati in formato PKCS#7 (estensione .p7b) o una sequenza di certificati di formato DER (estensione .cer)

Iterator per navigare all'interno della collezione di certificati

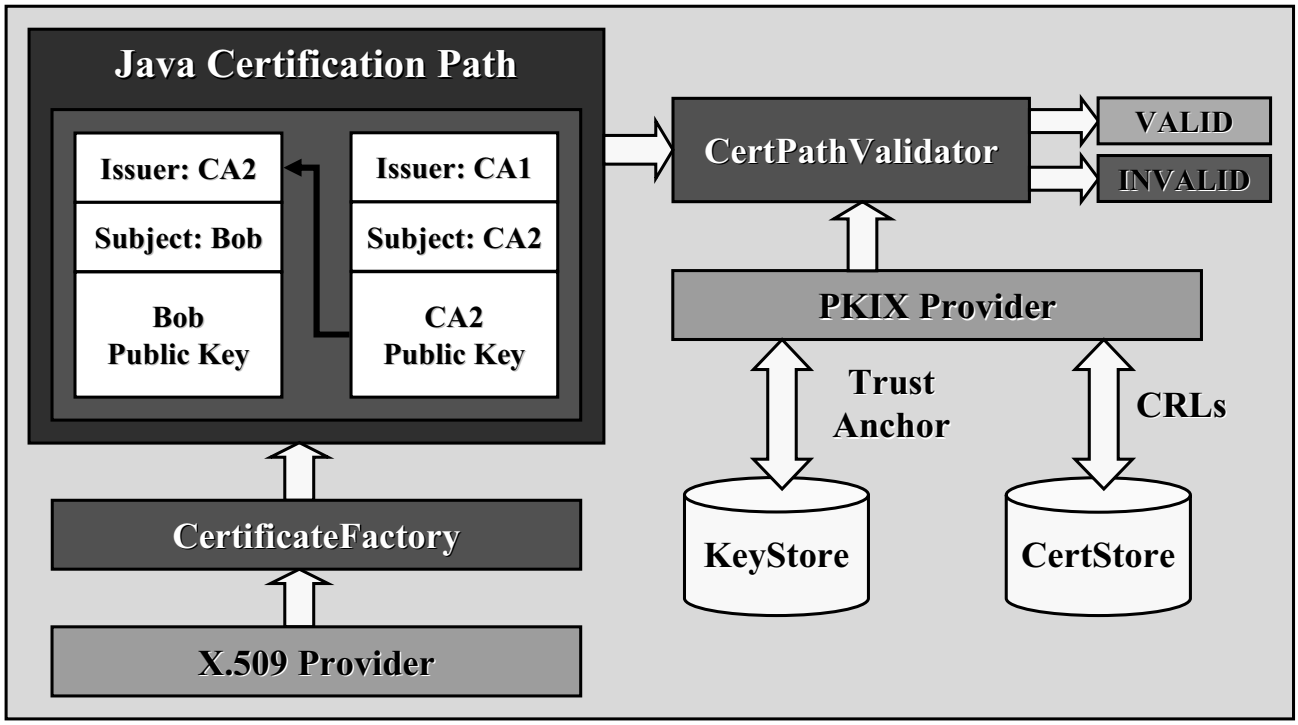
Tutto funziona correttamente solo se l'InputStream utilizzato supporta i metodi mark() e reset()

In caso contrario generateCertificates(),che si avvale di tali metodi, consumerà tutto lo stream di ingresso in un colpo solo

I vari certificati vengono inseriti in una Collection

Stampa a video usando la toString() di X509Certificate

Schema di Validazione



Processo di validazione

```
import java.security.cert.*;
.....
TrustAnchor anchor = new TrustAnchor("DST Root CA X1.cer", null);

Set anchors = Collections.singleton(anchor);
PKIXParameters params = new PKIXParameters(anchors);

params.setRevocationEnabled(false);

CertificateFactory cf = CertificateFactory.getInstance("X.509");
FileInputStream inCertPath = new FileInputStream("DST Root Cert.p7b");
CertPath co = cf.generateCertPath(inCertPath);

CertPathValidator cpv = CertPathValidator.getInstance("PKIX");
CertPathValidatorResult cpvr = cpv.validate(cp, params);
.....
```

Istanziare un TrustAnchor da un certificato root

Inserire le informazioni contenute nel TrustAnchor in un PKIXParameters

Indicare se usare un PKIX Revocation Checking Mechanism (in tal caso occorrerà specificare una CRL ed un PKIXCertPathChecker)

Istanziare il CertPath da validare

Eseguire la validazione attraverso un CertPathValidator

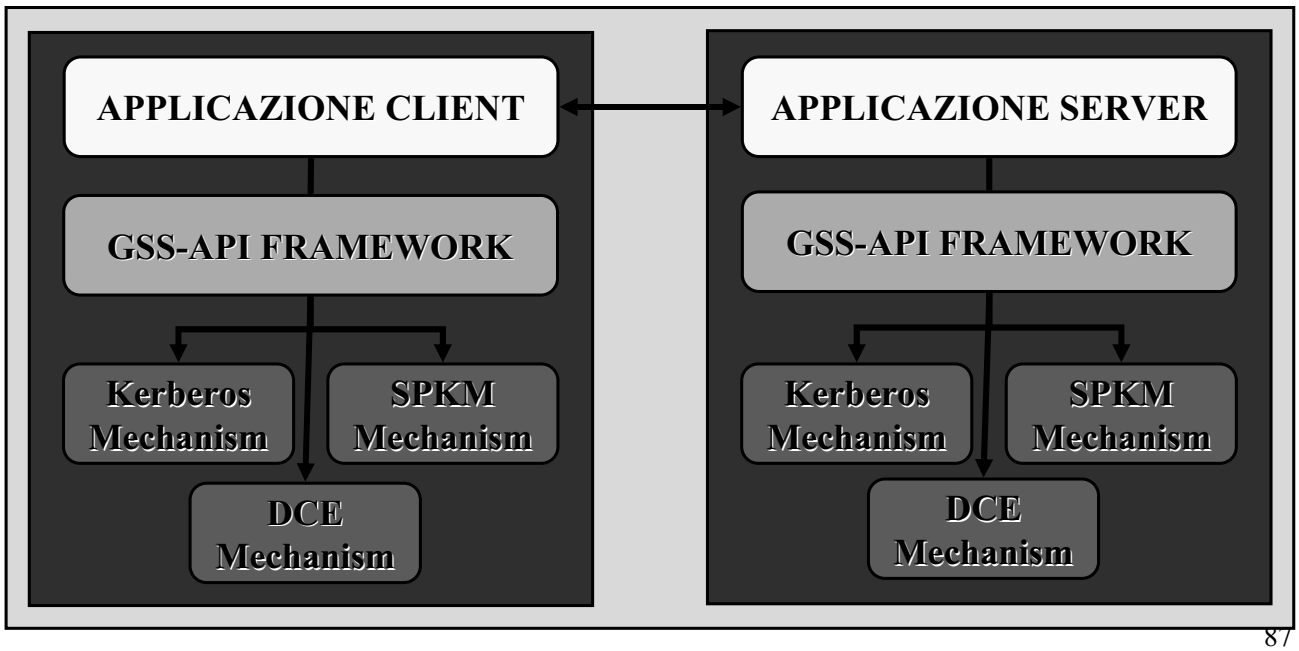
JGSS
Java Generic Security Service

Caratteristiche

- Java Generic Security Service (JGSS) contiene le Java bindings per la Generic Security Services Application Program Interface (GSS-API)
- Le GSS-API sono state progettate dal Common Authentication Technology working group dell'IETF per offrire un accesso uniforme a servizi di autenticazione e comunicazione peer-to-peer che adottano meccanismi di sicurezza sottostanti, isolando il chiamante dai dettagli implementativi
- Tali meccanismi di sicurezza possono essere impiegati in maniera simultanea e selezionati dall'applicazione a tempo di esecuzione
- Le GSS-API offrono i seguenti servizi di sicurezza:
  - autenticazione
  - riservatezza ed integrità dei messaggi
  - sequencing dei messaggi protetti
  - replay detection
  - credential delegation

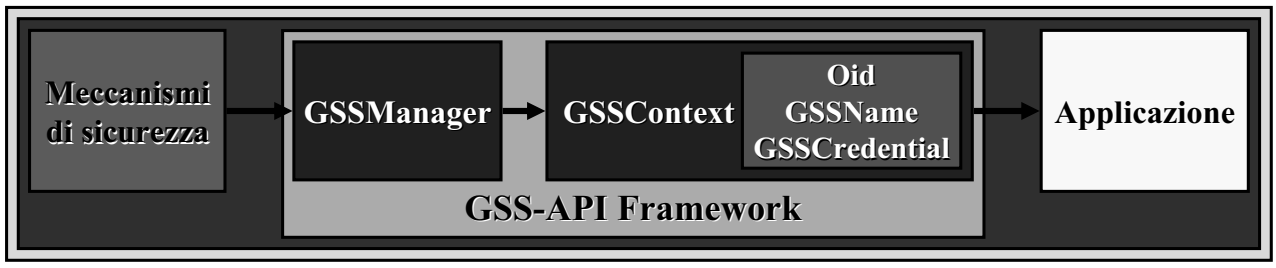
Meccanismi di sicurezza

- IETF ha definito i seguenti due meccanismi di sicurezza:
  - Kerberos v5
  - Simple Public Key Mechanism (SPKM)
- Altra tecnologia utilizzata è DCE (Distributed Computing Environment)



Kerberos v5 in JGSS

- JGSS offre la possibilità di implementare il servizio Kerberos v5 con l'ausilio dei meccanismi offerti da JAAS
- Classi principali in JGSS:
  - la classe GSSManager conosce i componenti e i meccanismi di sicurezza sottostanti ed è responsabile della loro invocazione a run-time
  - la classe Oid rappresenta l'Universal Object Identifier
  - l'interfaccia GSSName rappresenta una entità generica da instanziare e permette di definire l'user duke del servizio sicuro implementato
  - l'interfaccia GSSCredential incapsula le credentials possedute da un'entità
  - l'interfaccia GSSContext permette di definire i servizi di sicurezza offerti alle applicazioni comunicanti



Kerberos v5 in JAAS

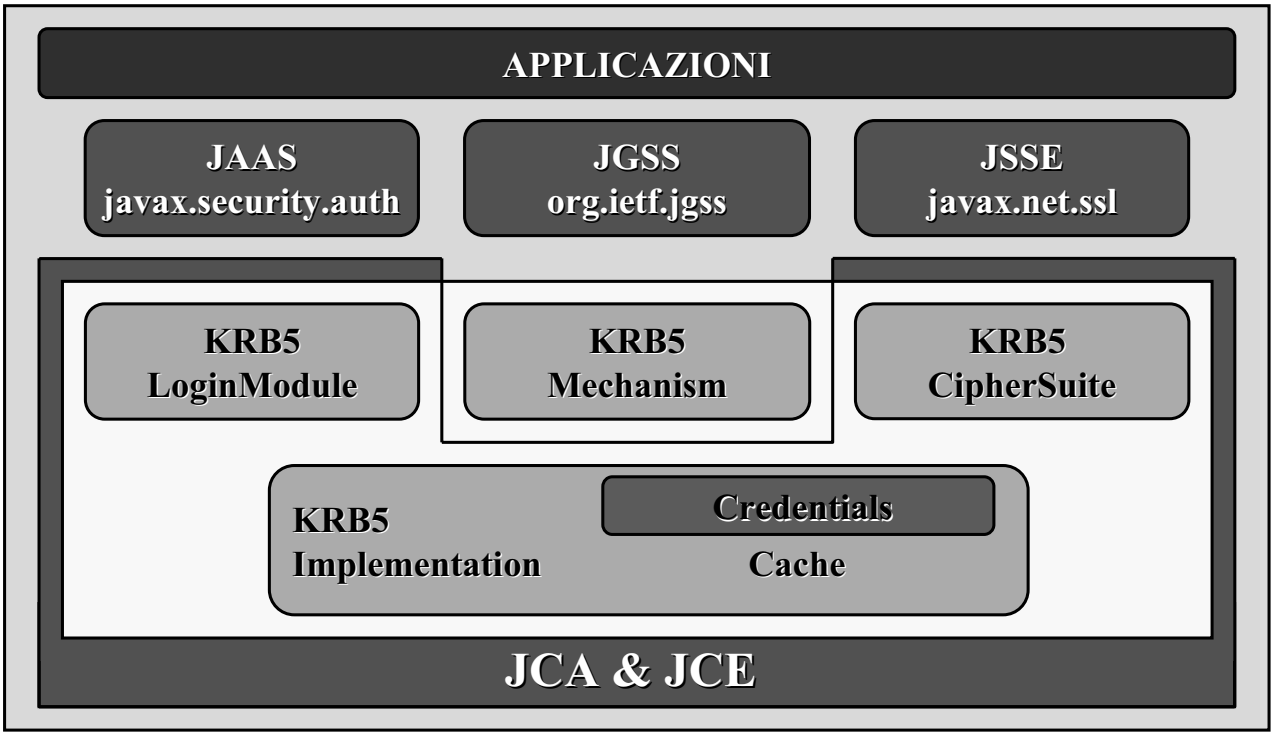
- JAAS mette a disposizione degli sviluppatori il modulo di autenticazione Krb5LoginModule e le classi del package javax.security.auth.kerberos:
  - DelegationPermission - usata per limitare l'uso del Kerberos delegation model (forwardable and proxiable tickets)
  - KerberosKey - incapsula un segreto a lungo termine per un Kerberos principal
  - KerberosPrincipal - incapsula un Kerberos principal
  - KerberosTicket - incapsula un Kerberos ticket e le informazioni ad esso associate dal punto di vista del cliente. Racchiude tutte le informazioni che il Key Distribution Center (KDC) invia al client nel messaggio di risposta KDC-REP definito in Kerberos Protocol Specification (RFC 1510)
  - ServicePermission - usata per proteggere i servizi Kerberos e le credenziali necessarie per accedere a questi servizi. Contiene un service principal name e una lista di operazioni che specificano il contesto in cui le credenziali possono essere usate

JGSS vs JSSE

- JGSS e JSSE presentano alcune analogie in termini di strumenti per comunicazioni sicure peer-to-peer
- Sono accomunati dalle seguenti caratteristiche di sicurezza:
  - autenticazione client/server
  - protezione della riservatezza e dell'integrità dei dati trasmessi
- Si differenziano per i seguenti aspetti:
  - Kerberos Single Sign-On Support
  - Communications API
  - Credential Delegation
  - Selective Encryption
  - Protocol Requirements

(When to use Java GSS-API vs. JSSE)

Kerberos v5 in J2SE



Autenticazione

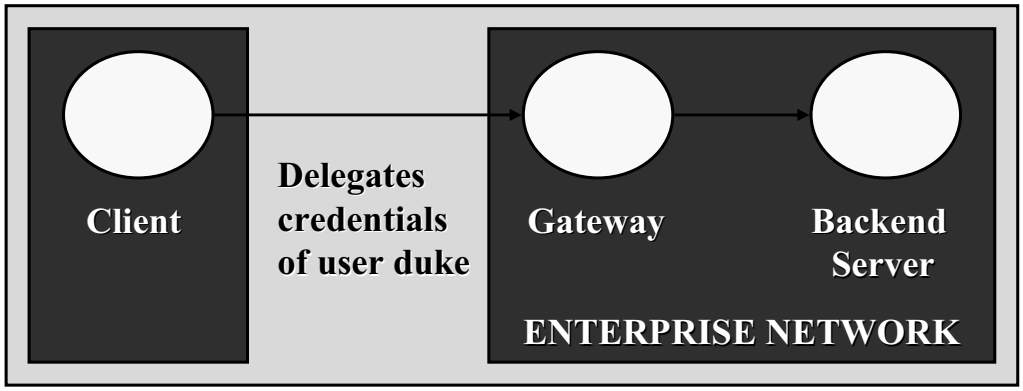
- Prima che un GSSContext possa essere usato per i servizi di sicurezza offerti, occorre realizzare uno scambio di token per l'autenticazione tra le applicazioni comunicanti
- GSSContext offre i metodi necessari a tale scambio:
  - initSecContext() - invia il token all'applicazione paritaria
  - acceptSecContext() - riceve il token
- Autenticazione in Kerberos V5:
  - Il client per primo invia il token costruito con initSecContext() contenente il messaggio AP-REQ di Kerberos
  - Il provider Kerberos ottiene dal TGT (Kerberos Ticket) del client un ticket di servizio per il target server, che viene cifrato ed inserito nel messaggio
  - Il server riceve il token e lo decifra con il metodo acceptSecContext() che autentica il client generando un nuovo token
  - In caso di autenticazione mutua, il nuovo token viene adoperato per l'autenticazione del server e rispedito al client con initSecContext()

Protezione dei messaggi

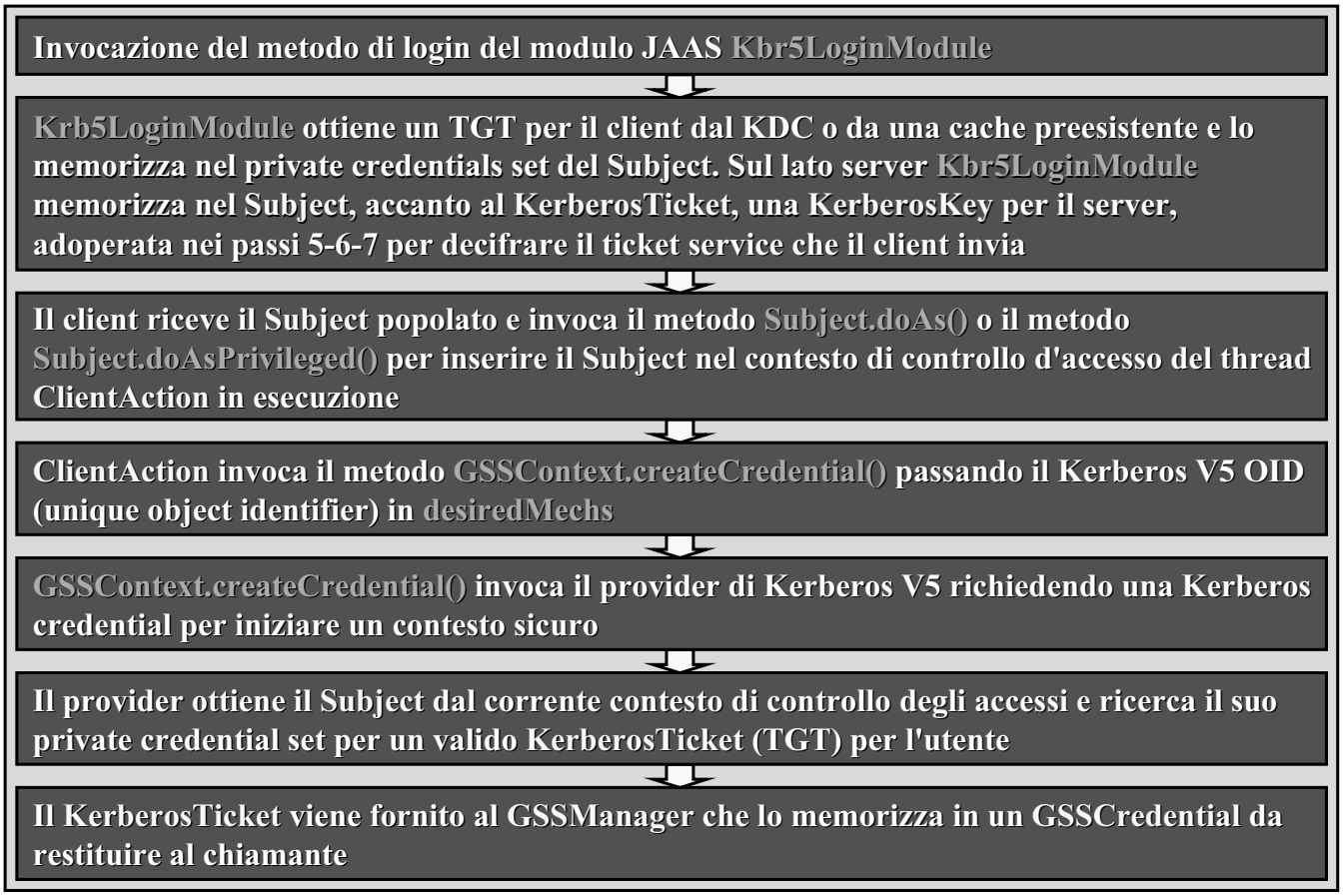
- Il contesto sicuro stabilito (GSSContext) offre protezione delle comunicazioni mediante riservatezza ed integrità dei messaggi
- I metodi relativi sono:
  - wrap() - permette di incapsulare il testo in chiaro o cifrato in un token che ne protegge l'integrità, da inviare all'applicazione paritaria
  - unwrap() - rimette in chiaro il testo originario dal token ricevuto
- L'oggetto trasmesso contiene informazioni sul testo originario (se in chiaro o cifrato) e relativi warnings per il controllo del sequencing e della duplicazione dei messaggi

Credential Delegation

- Java GSS-API aiuta il client a comunicare in maniera sicura le proprie credenziali al server e realizzare un nuovo contesto sicuro
- Nel caso di Kerberos V5, le credenziali delegate sono rappresentate da un forwarded TGT incapsulato nel primo token spedito dal client al server
- Usando questo TGT il server può ottenere un ticket service legato al client per alcuni altri servizi



Default Credential Acquisition Model



PKCS

Public Key Cryptography Standards

Public Key Cryptography Standards (PKCS)

- Le specifiche PKCS sono state prodotte dagli RSA Laboratories con lo scopo di accelerare lo sviluppo della crittografia a chiave pubblica
- Pubblicate nel 1991, sono state largamente implementate ed integrate in molti standard de facto, come **ANSI X9**, **PKIX**, **SET**, **S/MIME** e **SSL**
- JCA e JCE implementano gli standards **PKCS #1**, **#5**, **#7**, **#8**, **#9**, **#10** e **#12**
- J2SDK 1.4 Beta 3 include un miglior supporto per gli standard PKCS 1.0 con un set di APIs contenuto in JSR (Java Specification Request) 74:

  javax.security.pkcs.pkcs1
  javax.security.pkcs.pkcs5
  javax.security.pkcs.pkcs7
  javax.security.pkcs.pkcs8
  javax.security.pkcs.pkcs9
  javax.security.pkcs.pkcs10
  javax.security.pkcs.pkcs12

- La versione PKCS 2.1 presenta diversi miglioramenti che saranno implementati ed aggiunti in JSR 74 nel corrente anno

PKCS Architecture

Java Keytool
Key and Certificate Management Tool

Key Pair Generating

- Generazione di una coppia di chiavi asimmetriche e inserimento dei dati identificativi dell'utente:

keytool -genkey -keystore keystore -keyalg rsa –keysize 1024 -alias mauz -validity 180

Immettere la password del keystore: 20Colleluori03Maurizio1980
Specificare nome e cognome [Unknown]: Maurizio Colleluori
Specificare il nome dell'unità aziendale [Unknown]: MM
Specificare il nome dell'azienda [Unknown]: Mauz
Specificare la località [Unknown]: Bologna
Specificare la provincia [Unknown]: Bologna
Specificare il codice a due lettere del apese in cui si trova l'unità [Unknown]: IT
Il dato CN=Maurizio Colleluori, MM, O=Mauz, L=Bologna, ST=Bologna, C=IT è corretto? [no]: si
Immettere la password della chiave per <mauz>
(INVIO se corrisponde alla password del keystore): 20031980

Le chiavi private hanno associato un certificato che autentica le corrispondenti chiavi pubbliche
L'accesso ad un keystore è protetto da una password definita al momento della sua creazione
Si può decidere di proteggere la chiave privata con una password differente

Certificate Signing Request

- Produzione della richiesta di certificato (Certificate Signing Request) auto-firma del certificato provvisorio con la chiave privata:

keytool -certreq -keystore keystore -file csr.txt -alias mauz

Immettere la password del keystore: 20Colleluori03Maurizio1980
Immettere la password della chiave per <mauz>: 20031980

Richiesta per ogni accesso al keystore!

Richiesta per ogni accesso alla chiave segreta!

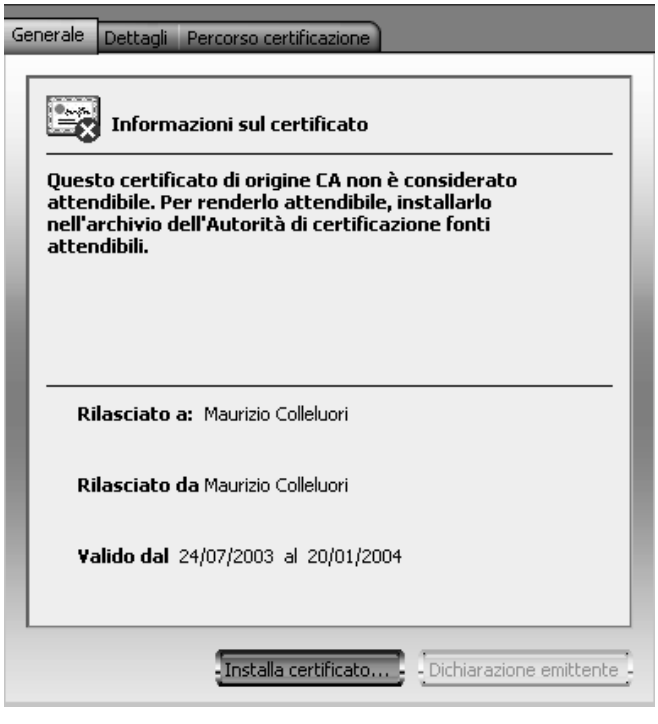
File csr.txt

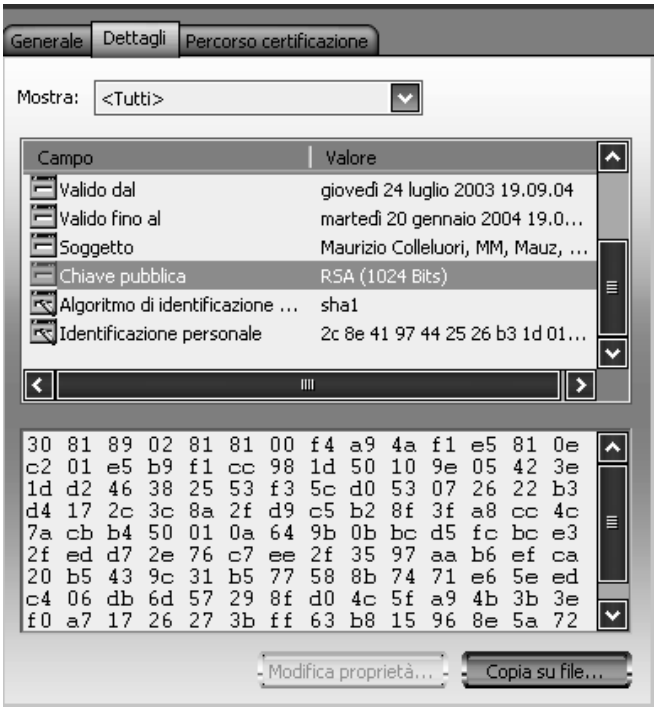
-----BEGIN NEW CERTIFICATE REQUEST-----
MIIBqzCCARQCAQAwazELMAkGA1UEBhMCSVQxEDAOBgNVBAgTI VBAcTB0Jv
bG9nbmExDTALBgNVBAoTBE1hdXoxCzAJBgNVBAsTAk1NMRwwGg pbyBDb2xs
ZWx1b3JpMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQD0qUrx5YEOwgHlufHMmB1QEJ4FQj4d
0kY4JVPzXNBTByYis9QXLDyKL9nFso8/qMxMesu0UAEKZJsLvNX8vOMv7dcudsfuLzWXqrbvyiC1
Q5wxtXdYi3Rx5l7txAbbbVcpj9BMX6lLOz7wpxcmJzv/Y7gVlo5aclnfpOQopzxRewIDAQABoAAw
DQYJKoZIhvcNAQEEBQADgYEA1hDqkWAttZEzNHAEncUodsUr0Ak8pfKhhw+BL4gmOkPo78jdqCaP
o1QIsX6KOB0X9s/+2MbWH3nVlVZP4HyQ8bv8xbDs6cY3m+adN7I01+7UHIiWqIUV2WKKmBGZSaQ+
gPNM/oV7dMZcH8mXr9zQObPqcHKqBDNjqtQC0IhE21o=
-----END NEW CERTIFICATE REQUEST-----

Self Certificate

- Generare un certificato X.509 auto-firmato:

keytool -selfcert -keystore keystore -keyalg rsa –keysize 1024 -alias mauz -validity 180





Altre funzionalità

- Download del certificato firmato dalla CA (estensione .cer secondo il formato PKCS7) e aggiornamento del certificato provvisorio auto-fimato:

  keytool -import -keystore keystore -file file.cer -alias mauz

- Esportazione di un certificato in un file esterno:

  keytool -import -keystore keystore -file file.cer -alias mauz

- Visualizzazione di una entry del keystore:

  keytool -list -keystore keystore -alias mauz

- Lettura di un file.cer:

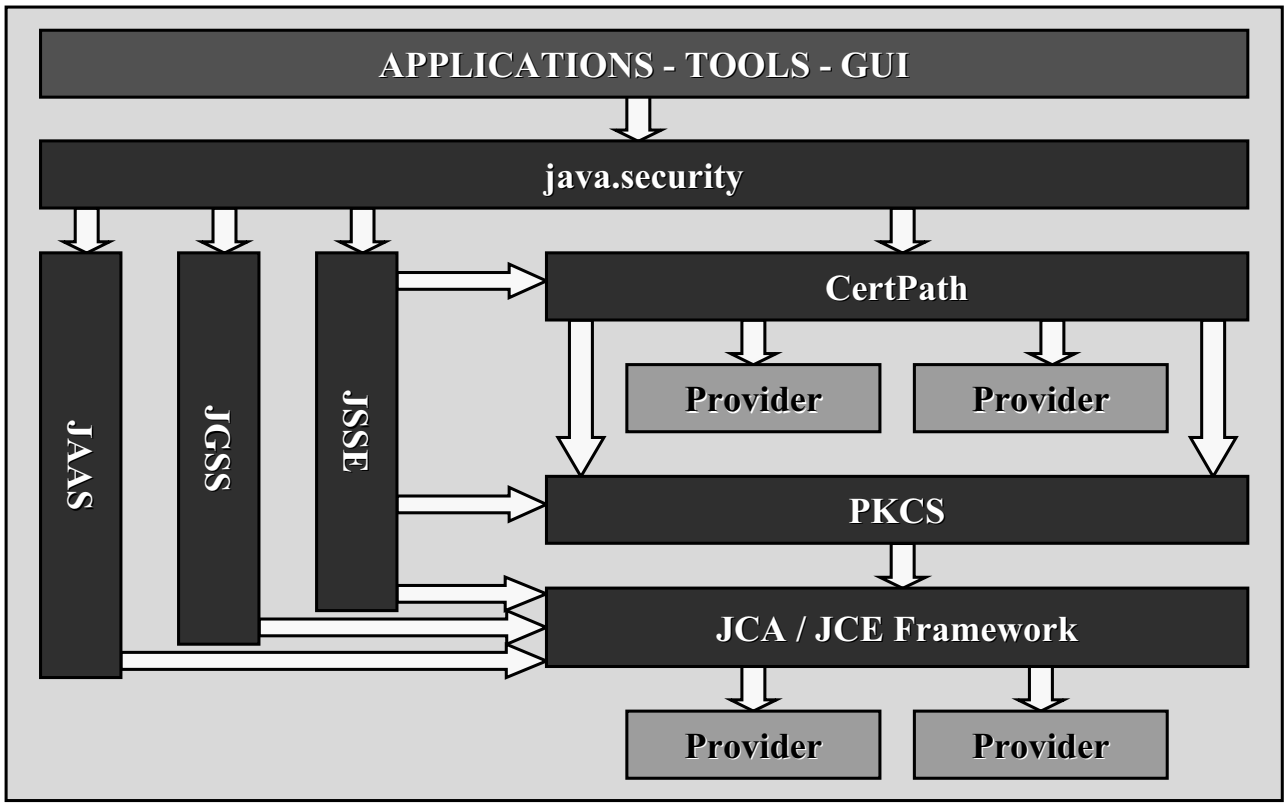
  keytool -printcert -file file.cer

- Altre utilità:

keytool -keyclone ...	clonazione di una entry del keystore
keytool -storepasswd ...	modifica della password del keystore
keytool -keypasswd ...	modifica della password della chiave privata
keytool -delete ...	cancellazione di una entry del keystore

Java Security Roadmap

Java Security Roadmap



Riferimenti

Java General Security

- http://java.sun.com/j2se/1.4.1/docs/guide/security/
  - General Security
    - Security Architecture
    - Cryptography Architecture
    - How to Implement a Provider for the Java Cryptography Architecture
    - Policy Permissions
    - Default Policy Implementation and Policy File Syntax
    - API for Privileged Blocks
    - X.509 Certificates and Certificate Revocation Lists
    - Security Managers and the JavaTM 2 SDK

Java Security Features

- http://java.sun.com/j2se/1.4.1/docs/guide/security/
  - JCE
    - JCE Reference Guide
    - How to Implement a Provider for the Java Cryptography Extension
  - JSSE
    - JSSE Reference Guide
  - JAAS
    - JAAS Reference Guide
    - JAAS Tutorials.
    - JAAS white paper
    - JAAS LoginModule Developer's Guide
  - Certification Path
    - Java Certification Path API Programmer's Guide
  - Java GSS-API
    - Java GSS-API and JAAS Tutorials for Use with Kerberos.
    - Single Sign-on Using Kerberos in Java
  - Tools
    - Security Tools

Applicazioni

- JSSE
  - RMI – SOCKETS – URLS
    http://java.sun.com/j2se/1.4.2/docs/guide/security/jsse/samples/index.html
  - SSL CLIENT / SERVER
    http://www.javaworld.com/javaworld/jw-05-2001/jw-0511-howto.html
  - HTTPS
    http://www.javaworld.com/javaworld/javatips/jw-javatip111.html
  - SECURE MAIL
    http://www.javaworld.com/javaworld/javatips/jw-javatip115.html
- JAAS
  - http://www.javaworld.com/javaworld/jw-09-2002/jw-0913-jaas.html
- Keytool
  - http://java.sun.com/products/jdk/1.2/docs/tooldocs/win32/keytool.html

Standards

- Public Key Cryptography Standards (PKCS)
  http://www.rsasecurity.com/rsalabs/pkcs/
- Java Specification Request (JSR) 74
  http://www.jcp.org/en/jsr/detail?id=74

Conferenze

- http://java.sun.com/security/
  - JavaOne 2002 Presentations (PDF documents) Sessions:
    - Java 2 Platform, Standard Edition (J2SE) Security: Present and Future
    - Authentication and Single Sign-On
    - Networking with Java 2 Platform, Standard Edition (J2SE): Present and Future
    - Single Sign-on Using the JAAS and Java GSS APIs
    - Securing the Wire
    - Java Naming and Directory Interface (JNDI)
    - JSR 105: XML Digital Signature API
  - JavaOne 2001 Presentations (PDF documents)
    - Java 2 Platform, Standard Edition (J2SE) Security: Present and Future
    - Security for the J2SE: CertPath, JCE, and JSSE
    - Java 2 Platform, Standard Edition (J2SE) Networking: Present and Future
    - Java 2 Platform, Standard Edition (J2SE) Networking and IPv6
    - Unlocking Public Key Technologies for the Java Platform Update: JSR74
    - Security for the J2SE Platform: JAAS, Java GSS-API & Kerberos
    - Security for the Java 2 Platform, Standard Edition (J2SE) and XML

Articoli

- http://java.sun.com/security/
  - FAQs, Whitepapers, Articles
    - Single Sign-on Using Kerberos in Java
    - JAAS white paper
    - Going Beyond the Sandbox: An Overview of the New Security Architecture in the JavaTM Development Kit 1.2
    - Secure Computing With Java: Now and the Future
    - CompCon 97 Abstract
    - Applet Security FAQ