Java Security

Proprietà di Sicurezza I

- Confidenzialità
  - Le **informazioni** possono essere lette **solo** dai **soggetti** che ne hanno **diritto**
- Integrità
  - Le **informazioni** possono essere modificate **solo** dai **soggetti** che ne hanno **diritto**
- Disponibilità
  - Le **informazioni** devono poter essere lette/scritte quando necessario
  - Le **risorse** devono poter essere usate dai **soggetti** che ne hanno **diritto**

Proprietà di Sicurezza II

- Identificazione
  - Ogni **soggetto** ha una propria identità unica
- Autenticazione
  - Verifica dell'identità di ogni **soggetto**
- Non ripudiabilità
  - L'**affermazione** fatta da un **soggetto** NON può essere negata in un secondo tempo

Proprietà di Sicurezza III

- Tracciabilità
  - Individuazione del **soggetto** che ha effettuato una operazione
- Accountability
  - Correlazione tra uso della **risorsa** e **soggetto** che l'ha usata

Vulnerabilità

- Punto debole di un componente del sistema
- Sfruttando la vulnerabilità si genera un comportamento anomalo nel sistema
- Il comportamento anomalo può violare la sicurezza del sistema (Attacco)
- Soluzione:

  Robustezza & Contromisure

Attacco

- Un attacco è una sequenza di azioni eseguite per ottenere il controllo di un host
- L'exploit è il programma che esegue l'attacco sfruttando una vulnerabilità dell'host
- Quando si ottiene il controllo di un host si può
  - accedere alle informazioni
  - modificare informazioni
  - impedire ad altri di accedere alle informazioni
  - utilizzare l'host come base per eseguire un attacco ad un altro host

Contromisure

- Definizione di una politica di sicurezza
  - Definizione della criticità di ogni risorsa
  - Regole per il controllo degli accessi
  - ……
- Crittografia
- Firma Digitale
- Certificati
- Firewall
- Intrusion Detection Systems (Rete/Host)

Java Security

- Platform Security: Built-in language security features enforced by the Java compiler and virtual machine
  - Strong data typing
  - Automatic memory management
  - Bytecode verification
  - Secure class loading

- Cryptography
  - digital signatures, message digests, ciphers (symmetric, asymmetric, stream & block), message authentication codes, key generators and key factories
  - Support for a wide range of standard algorithms including RSA, DSA, AES, Triple DES, SHA, PKCS#5, RC2, and RC4.

http://java.sun.com/security/

Java Security

- Authentication and Access Control
  - login mechanisms
  - policy and permissions for fine-grained access control of resources
- Secure Communications: authenticates peers over an untrusted network and protects the integrity and privacy of data transmitted between them.
- Public Key Infrastructure (PKI): Tools for managing keys and certificates
  - Certificates and Certificate Revocation Lists (CRLs): X.509
  - Certification Path Validators and Builders: PKIX (RFC 3280), On-line Certificate Status Protocol (OCSP)
  - KeyStores: PKCS#11, PKCS#12
  - Certificate Stores (Repositories): LDAP, java.util.Collection

http://java.sun.com/security/

Comunicazioni Sicure

Java Secure Socket Extension

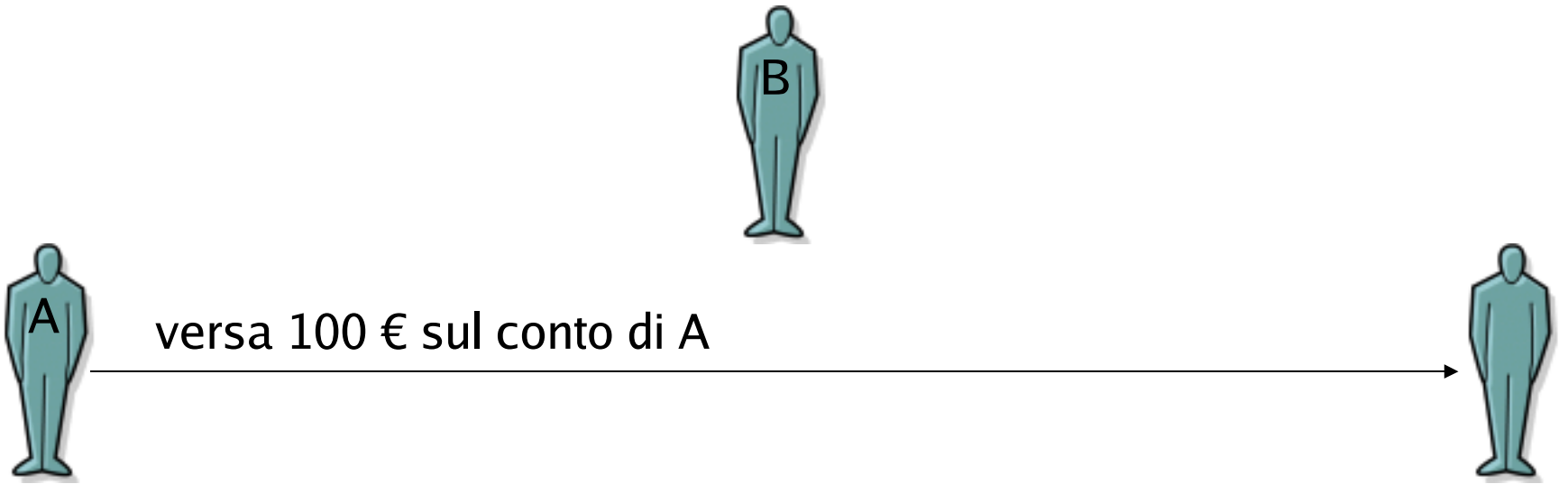
(JSSE)

Rischi nella Comunicazione

- Il tuo partner di comunicazione è chi dice di essere? (autenticazione)
- I dati possono essere intercettati quando viaggiano sulla rete (confidenzialità)
- I dati possono essere intercettati e modificati quando viaggiano sulla rete (integrità)

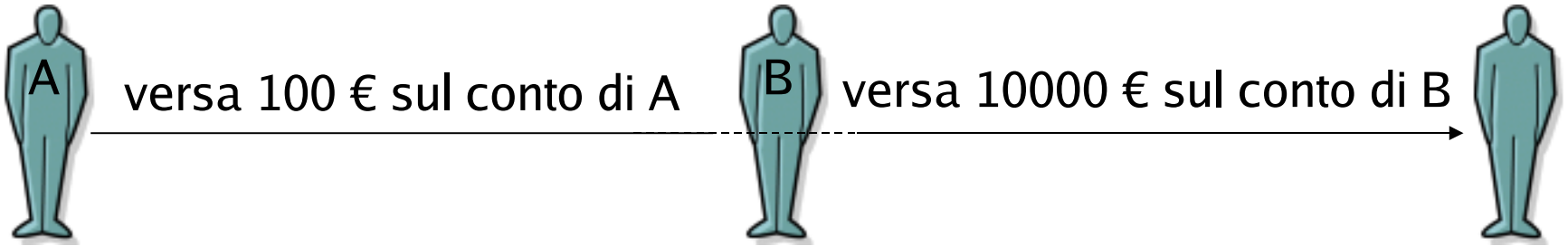
Esempio: man in the middle

violazione della confidenzialità e dell'integrità del messaggio



Esempio: man in the middle

violazione della confidenzialità e dell'integrità del messaggio



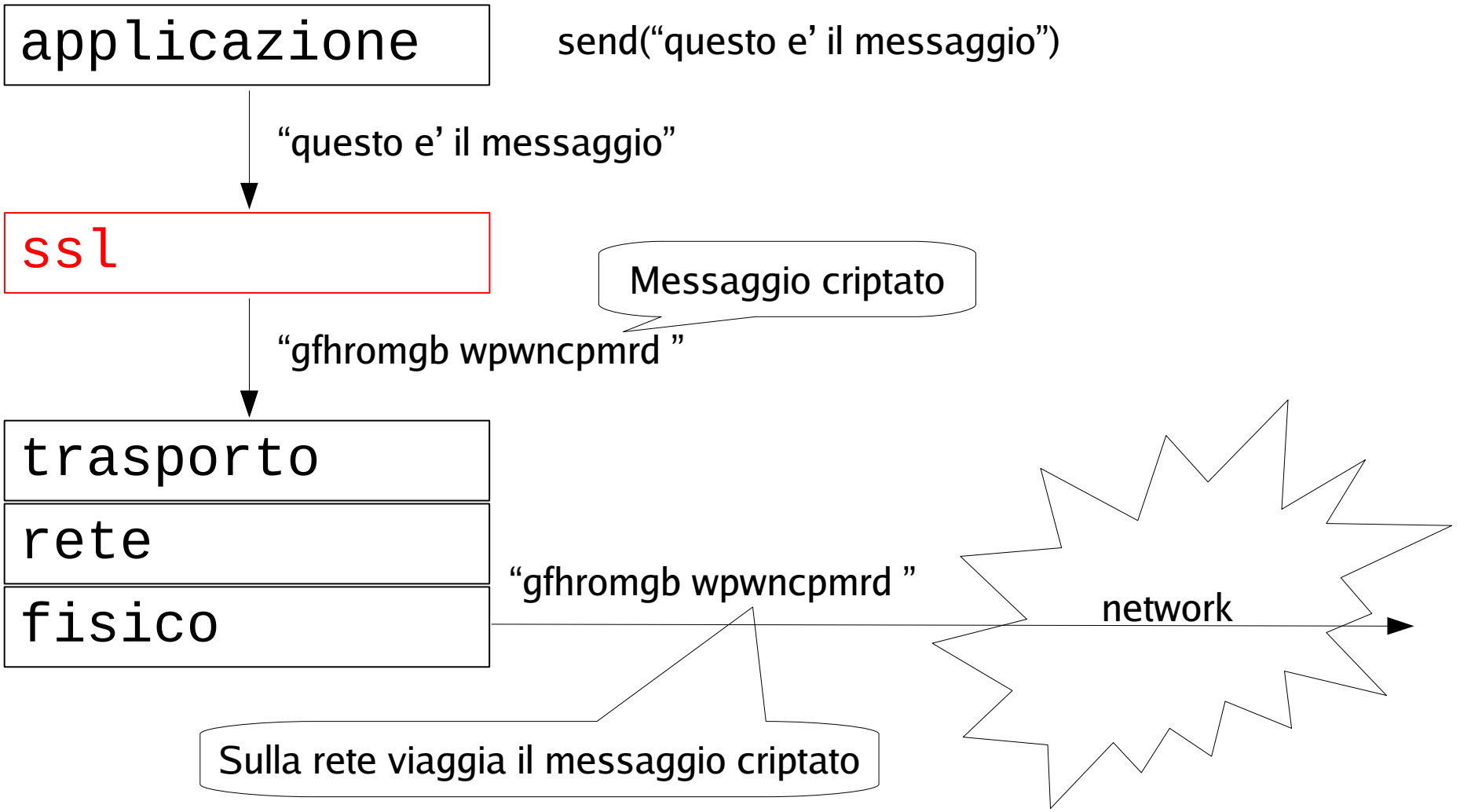
Java Secure Socket Extension

- Fornisce
  - Autenticazione Server e Client
  - Confidenzialità messaggi
  - Integrità messaggi

- E’ un nuovo layer dello stack TCP/IP

Modello TCP/IP

Nuovo layer per la sicurezza delle comunicazioni



Concetti alla base di JSSE

- Crittografia
- Certificati Digitali
- Trust

Crittografia

- Trasformazione dei dati in un formato nascosto per proteggerne la confidenzialità
  - Dati originali = dati in chiaro (plain text)
  - Dati trasformati = crittogramma (cipher text)
  - Cifratura (encryption) = plaint text ⇨ chiper text
  - Decifratura (decryption) = cipher text ⇨ plain text

- Esistono vari algoritmi di crittografia basati su concetti matematici (ad es.: numeri primi)

Crittografia

- Algoritmi a chiave segreta (simmetrici)
  - Unica chiave per criptare e decriptare
  - Chiave **segreta**
  - Problemi per lo scambio **sicuro** della chiave tra i partner di comunicazione
  - Computazionalmente leggeri
  - Ad es.: DES (e sue variazioni), IDEA, SAFER, RC2-4-5, FEAL, SKIPJACK, BLOWFISH, SEAL,....

Crittografia

- Algoritmi a chiave pubblica (asimmetrici)
  - Una chiave per criptare (pubblica)
  - Un'altra chiave per descriptare: **segreta** (privata)
  - La chiave pubblica può essere divulgata ai partner di comunicazione senza problemi di confidenzialità (ma di integrità si!!!!)
  - Computazionalmente più pesanti
  - Ad es.: RSA, Diffie-Hellman, DSA, LUC, ...

Algoritmi di Crittografia Supportati

Algoritmo	chiave (bits)
RSA (auth. and key exchange)	2048 (auth.) 2048 (key exchange) 512 (key exchange)
AES (bulk encryption)	128 - 256
RC4 (bulk encryption)	128 - 128 (40 effective)
DES (bulk encryption)	64 (56 effective) - 64 (40 effective)
Triple DES (bulk encryption)	192 (112 effective)
Diffie-Hellman (key agreement)	1024 - 512
DSA (authentication)	1024

http://java.sun.com/security/

Certificati Digitali

- Documenti di identità in formato digitale
- Rilasciati da appositi enti, Certification Authorities
  - Ad es. Verisign, Thatwe, IIT-CNR
  - A pagamento!!!!!!! (circa 15€ quelli di Verisign)
  - Standard X.509
- Contengono almeno:
  - Identità del proprietario
  - Chiave pubblica del proprietario
  - Scadenza del certificato
  - Firma digitale (basata sulla crittografia) della certification authority che ne garantisce l'autenticità

KeyStore

- File locale che contiene i certificati digitali che attestano la MIA identità (sia server che client)

- NON possiamo ora andare a comprare dei certificati, quindi usiamo `keytool` per crearli
  - Vanno bene per le nostre prove
  - NON sicuri per utilizzo serio

Creazione KeyStore con `keytool`

```
$ keytool -genkey -keystore certs -alias paolok
Enter keystore password:  paoloks
What is your first and last name?
  [Unknown]:  paolo mori
What is the name of your organizational unit?
  [Unknown]:  iit
What is the name of your organization?
  [Unknown]:  cnr
What is the name of your City or Locality?
  [Unknown]:  pisa
What is the name of your State or Province?
  [Unknown]:  pisa
What is the two-letter country code for this unit?
  [Unknown]:  it
Is CN=paolo mori, OU=iit, O=cnr, L=pisa, ST=pisa, C=it correct?
  [no]:  yes

Enter key password for <paolok>
        (RETURN if same as keystore password): paolokp
$
```

Estrazione certificato dal KeyStore

KeyStore

```
$ keytool -keystore certs -export -alias paolok -file paolocertfile.cer
Immettere la password del keystore:  paoloks
Il certificato è memorizzato nel file <paolocertfile.cer>
```

Se fosse un certificato vero qui ci sarebbe la Certification Authority

```
$ keytool -printcert -file paolocertfile.cer
Proprietario: CN=paolo mori, OU=iit, O=cnr, L=pisa, ST=pisa, C=it
Organismo di emissione: CN=paolo mori, OU=iit, O=cnr, L=pisa, ST=pisa, C=it
Numero di serie: 439c9b50
Valido da Sun Nov 20 22:34:08 CET 2005 a Sat Feb 18 22:34:08 CET 2006
Impronte digitali certificato:
        MD5: 26:D7:33:8D:0D:5D:17:B3:C4:5F:F9:BC:4E:38:53:7C
        SHA1: 26:1E:5D:5A:DD:B5:AF:AF:5F:1B:24:AE:78:13:9F:61:78:06:51:78
```

TrustStore

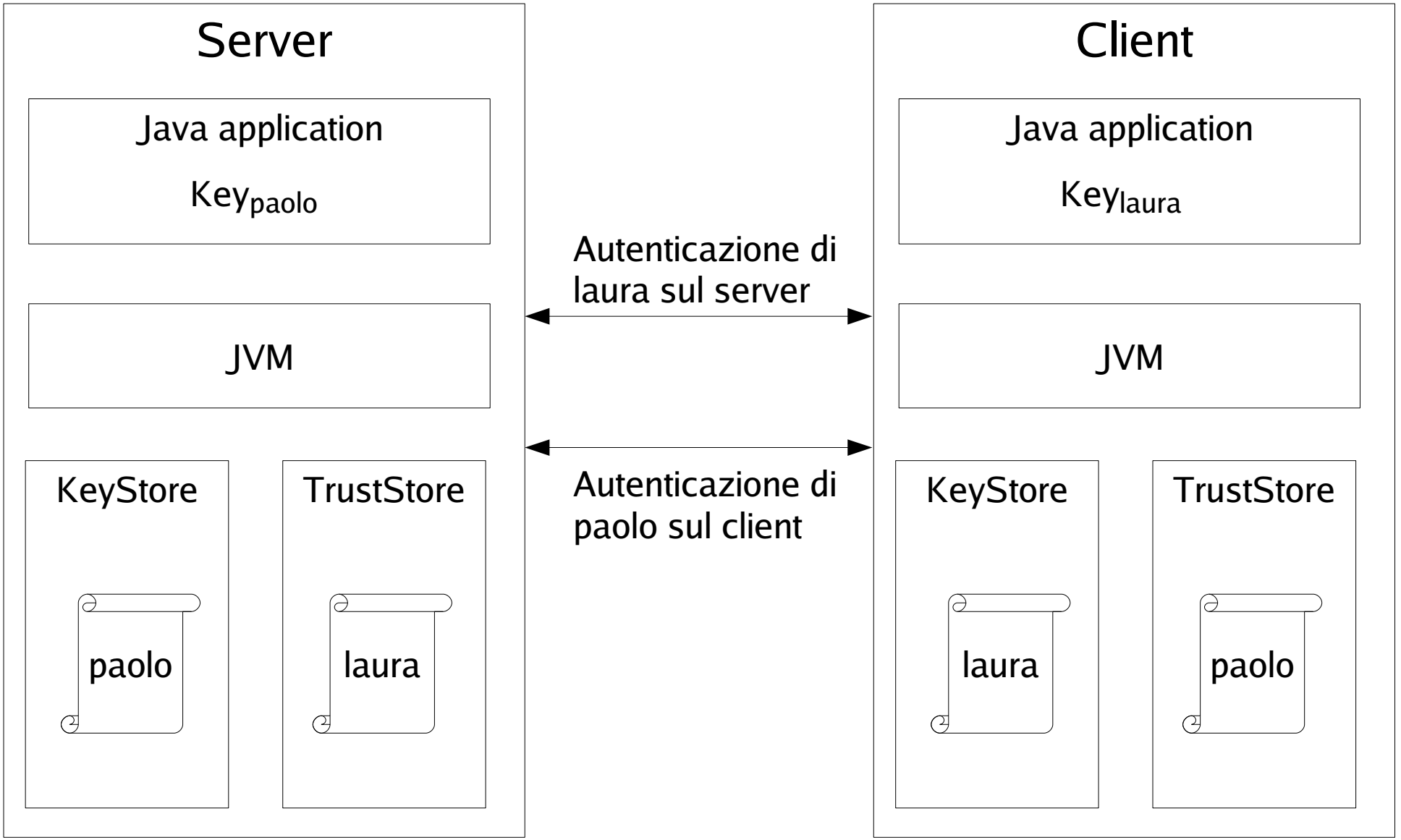
- Contiene i certificati dei partner di comunicazione di cui mi fido
- I certificati dovrei ottenerli dai partner, dalla loro CA, oppure da altre fonti

- Anche in questo caso simuliamo un TrustStore con `keytool`
  - Va bene per le nostre prove
  - NON sicuro per utilizzo serio

Inserimento Certificato nel TrustStore

$ keytool -import -keystore cacerts -alias paoloc -file paolocertfile.cer
Immettere la password del keystore: altrapass
Proprietario: CN=paolo mori, OU=iit, O=cnr, L=pisa, ST=pisa, C=it
Organismo di emissione: CN=paolo mori, OU=iit, O=cnr, L=pisa, ST=pisa, C=it
Numero di serie: 439c9b50
Valido da Sun Dec 11 22:34:08 CET 2005 a Sat Mar 11 22:34:08 CET 2006
Impronte digitali certificato:
	MD5: 26:D7:33:8D:0D:5D:17:B3:C4:5F:F9:BC:4E:38:53:7C
	SHA1: 26:1E:5D:5A:DD:B5:AF:AF:5F:1B:24:AE:78:13:9F:61:78:06:51:78
Considerare attendibile questo certificato? [no]: si
Il certificato è stato aggiunto al keystore

Mutua Autenticazione Server - Client



Lato Server

Configurazione Ambiente per Server SSL

- Creazione SSLContext
- Creazione TrustManagerFactory
- Creazione KeyManagerFactory
- Creazione KeyStore
- Carico certificati nel KeyStore
- Inizializzazione del KeyManagerFactory con il KeyStore e la password
- Inizializzazione del SSLContext con il KeyManager ed il TrustManager

Creazione SSLContext

Package `javax.net.ssl`

Non esiste un costruttore per SSLContext, bisogna utilizzare il metodo statico:

```
public static SSLContext getInstance(String protocol)
                        throws NoSuchAlgorithmException
```

`protocol` è “SSL”

`NoSuchAlgorithmException` viene sollevato se `protocol` non è supportato

Creazione SSLContext

```
import javax.net.ssl.SSLContext;

.......
public class ProvaSSLServer {

    public static void main(String[] args) {

        SSLContext c;

        try {

            c = SSLContext.getInstance("SSL");

        }
        catch(NoSuchAlgorithmException  e) {
            System.out.println("SSL non supportato");
        }
        .......
```

Creazione KeyManagerFactory

Package `javax.net.ssl`

Non esiste un costruttore per KeyManagerFactory, bisogna utilizzare il metodo statico:

```
public static final
        KeyManagerFactory getInstance(String algorithm)
                          throws NoSuchAlgorithmException
```

`algorithm` è “SunX509”, lo standard dei certificati digitali

`NoSuchAlgorithmException` viene sollevato se `algorithm` non è supportato

Creazione KeyManagerFactory

```
import javax.net.ssl.KeyManagerFactory;

.....

public static void main(String[] args) {

    .....

    KeyManagerFactory km;

    try {

        km = KeyManagerFactory.getInstance("SunX509");
    }
    catch(NoSuchAlgorithmException  e) {
            System.out.println("SunX509 non supp.");
    }
    .......
```

Creazione KeyStore

Package `java.security`

Non esiste un costruttore per KeyStore, bisogna utilizzare il metodo statico:

```
public static KeyStore getInstance(String type)
                                throws KeyStoreException
```

`type` è `"JKS"`, il tipo di KeyStore supportato da JSSE

`KeyStoreException` viene sollevato se `type` non è supportato

Creazione KeyStore

```
import java.security.KeyStore;

.....
public static void main(String[] args) {

    .....

    KeyStore ks;

    try {

        ks = KeyStore.getInstance("JKS");
    }
    catch(NoSuchAlgorithmException  e) {
        System.out.println("SunX509 non supp.");
    }

    .......
```

Carico Certificati nel KeyStore

Metodo di java.security.KeyStore

```
public final void
             load(InputStream stream, char[] password)
             throws IOException,
                    NoSuchAlgorithmException,
                    CertificateException
```

`stream` è un Inputstream aperto sul file che contiene il KeyStore

`IOException` viene sollevata se non è possibile aprire o leggere da `stream`; viene sollevata anche se la password è errata

`NoSuchAlgorithmException` viene sollevata se non viene trovato l'algoritmo per verificare l'integrità del KeyStore

`CertificateException` viene sollevata se si verificano errori nel caricare un certificato

Carico Certificati nel KeyStore

Questo è solo un esempio:
NON mettere mai la password cablata nel programma!!!!

```
char[] passwd = "paoloks".toCharArray();


ks.load(new FileInputStream("certs"), passwd);
```

Inizializzazione del KeyManagerFactory

Metodo di `javax.net.ssl.KeyManagerFactory`

```
public final void init(KeyStore ks, char[] password)
                    throws KeyStoreException,
                           NoSuchAlgorithmException,
                           UnrecoverableKeyException
```

`ks` è il KeyStore creato precedentemente

`password` è la password del KeyStore

`KeyStoreException` viene sollevata se non viene inizializzato il KeyManagerFactory

`NoSuchAlgorithmException` viene sollevata se non viene trovato l'algoritmo per verificare l'integrità del KeyStore

`UnrecoverableKeyException` viene sollevata se non è possibile accedere al KeyStore, per esempio perchè la password è errata

Inizializzazione del KeyManagerFactory

```
char[] kpasswd = "paolokp".toCharArray();

km.init(ks, kpasswd);
```

Creazione TrustManagerFactory

Package `javax.net.ssl`

Non esiste un costruttore per TrustManagerFactory, bisogna utilizzare il metodo statico:

```
public static final
      TrustManagerFactory getInstance(String algorithm)
                          throws NoSuchAlgorithmException
```

`algorithm` è “`SunX509`”, lo standard dei certificati digitali

`NoSuchAlgorithmException` viene sollevato se `algorithm` non è supportato

Creazione TrustManagerFactory

```
import javax.net.ssl.TrustManagerFactory;

.....

public static void main(String[] args) {

    .....

    TrustManagerFactory tm;

    try {

        tm = TrustManagerFactory.getInstance("SunX509");
    }
    catch(NoSuchAlgorithmException  e) {
            System.out.println("SunX509 non supp.");
    }
    .......
```

Inizializzazione del TrustManagerFactory

Metodo di `javax.net.ssl.TrustManagerFactory`

```
public final void init(KeyStore ks)
                    throws KeyStoreException,
```

`ks` è il KeyStore creato precedentemente, anche se si può usare lo stesso creato per il KeyManager, è meglio avere due KeyStores diversi:

- uno per i propri certificati
- uno per i certificati dei partners

`KeyStoreException` viene sollevata se non viene inizializzato il KeyManagerFactory

Inizializzazione del TrustManagerFactory

```
tm.init(ks);
```

Inizializzazione del SSLContext

Metodo di `javax.net.ssl.SSLContext`

```
public final void init(KeyManager[] km,
                       TrustManager[] tm,
                       SecureRandom random)
                throws KeyManagementException
```

`km` il KeyManager creato precedentemente

`tm` il TrustManager, se `tm`=null viene preso il TrustManager di default

`KeyManagementException` viene sollevata se non è possibile inizilizzare SSLContext

Inizializzazione del SSLContext

```
c.init(km.getKeyManagers(), tm.getTrustManagers(), null);
```

Creazione Server Socket con SSL

Metodo di `javax.net.ssl.SSLContext`

Non esiste un costruttore per `SSLServerSocketFactory`, bisogna utilizzare un metodo di `SSLContext`:

```
public final
      SSLServerSocketFactory getServerSocketFactory()
```

ritorna una `SSLServerSocketFactory` dalla quale creare l'`SSLServerSocket`

Creazione Server Socket con SSL

Metodo di `javax.net.ssl.SSLServerSocketFactory`

Non esiste un costruttore per `SSLServerSocket`, bisogna utilizzare un metodo di `ServerSocketFactory`, che è una superclasse di `SSLServerSocketFactory`:

```
public abstract
                ServerSocket createServerSocket(int port)
                                        throws IOException
```

ritorna un `SSLServerSocket`

Creazione Server Socket con SSL

```
import javax.net.ssl.SSLServerSocket;
import javax.net.ssl.SSLServerSocketFactory;

......

int porta = 4000;

SSLServerSocketFactory f = c.getServerSocketFactory();

SSLServerSocket ss = (SSLServerSocket)
                              f.createServerSocket(porta);


// adesso ss si tratta come un normale ServeSocket
```

Algoritmi di Crittografia

Metodi di `javax.net.ssl.SSLServerSocket`

```
public abstract String[] getSupportedCipherSuites()
```

restituisce l'elenco degli algoritmi crittografici supportati che possono essere attivati

```
public abstract String[] getEnabledCipherSuites()
```

restituisce l'elenco degli algoritmi crittografici attivati

```
public abstract void
                setEnabledCipherSuites(String[] suites)
```

modifica l'elenco degli algoritmi crittografici attivati

Algoritmi di Crittografia Supportati

```
String[] scs = ss.getSupportedCipherSuites();


for (int i=0; i<scs.length; i++)
    System.out.println(scs[i])
```

Algoritmi di Crittografia Supportati

SSL_RSA_WITH_RC4_128_MD5
SSL_RSA_WITH_RC4_128_SHA
TLS_RSA_WITH_AES_128_CBC_SHA
TLS_DHE_RSA_WITH_AES_128_CBC_SHA
TLS_DHE_DSS_WITH_AES_128_CBC_SHA
SSL_RSA_WITH_3DES_EDE_CBC_SHA
SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA
SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA
SSL_RSA_WITH_DES_CBC_SHA
SSL_DHE_RSA_WITH_DES_CBC_SHA
SSL_DHE_DSS_WITH_DES_CBC_SHA
SSL_RSA_EXPORT_WITH_RC4_40_MD5
SSL_RSA_EXPORT_WITH_DES40_CBC_SHA
SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA
SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
SSL_RSA_WITH_NULL_MD5
SSL_RSA_WITH_NULL_SHA

SSL_DH_anon_WITH_RC4_128_MD5
TLS_DH_anon_WITH_AES_128_CBC_SHA
SSL_DH_anon_WITH_3DES_EDE_CBC_SHA
SSL_DH_anon_WITH_DES_CBC_SHA
SSL_DH_anon_EXPORT_WITH_RC4_40_MD5
SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA
TLS_KRB5_WITH_RC4_128_SHA
TLS_KRB5_WITH_RC4_128_MD5
TLS_KRB5_WITH_3DES_EDE_CBC_SHA
TLS_KRB5_WITH_3DES_EDE_CBC_MD5
TLS_KRB5_WITH_DES_CBC_SHA
TLS_KRB5_WITH_DES_CBC_MD5
TLS_KRB5_EXPORT_WITH_RC4_40_SHA
TLS_KRB5_EXPORT_WITH_RC4_40_MD5
TLS_KRB5_EXPORT_WITH_DES_CBC_40_SHA
TLS_KRB5_EXPORT_WITH_DES_CBC_40_MD5

Gli algoritmi che contengono _anon_ non richiedono autenticazione

Il Server ed il Client devono attivare almeno un algoritmo in comune

Attivazione Algoritmi di Crittografia

```
String[] ascs = new String[scs.length];
int ascsl = 0;

for (int i=0; i<scs.length; i++)  {
    if (scs[i].indexOf("_anon_") > 0)
        ascs[ascsl++] = scs[i];
}

String[] oecs = ss.getEnabledCipherSuites();
String[] necs = new String[oecs.length + ascsl];

System.arraycopy(oecs, 0, necs, 0, oecs.length);
System.arraycopy(ascs, 0, necs, oecs.length, ascsl);


ss.setEnabledCipherSuites(necs);
```

Autenticazione del Client

Il server sceglie se richiedere l'autenticazione del client

Metodi di `javax.net.ssl.SSLServerSocket`

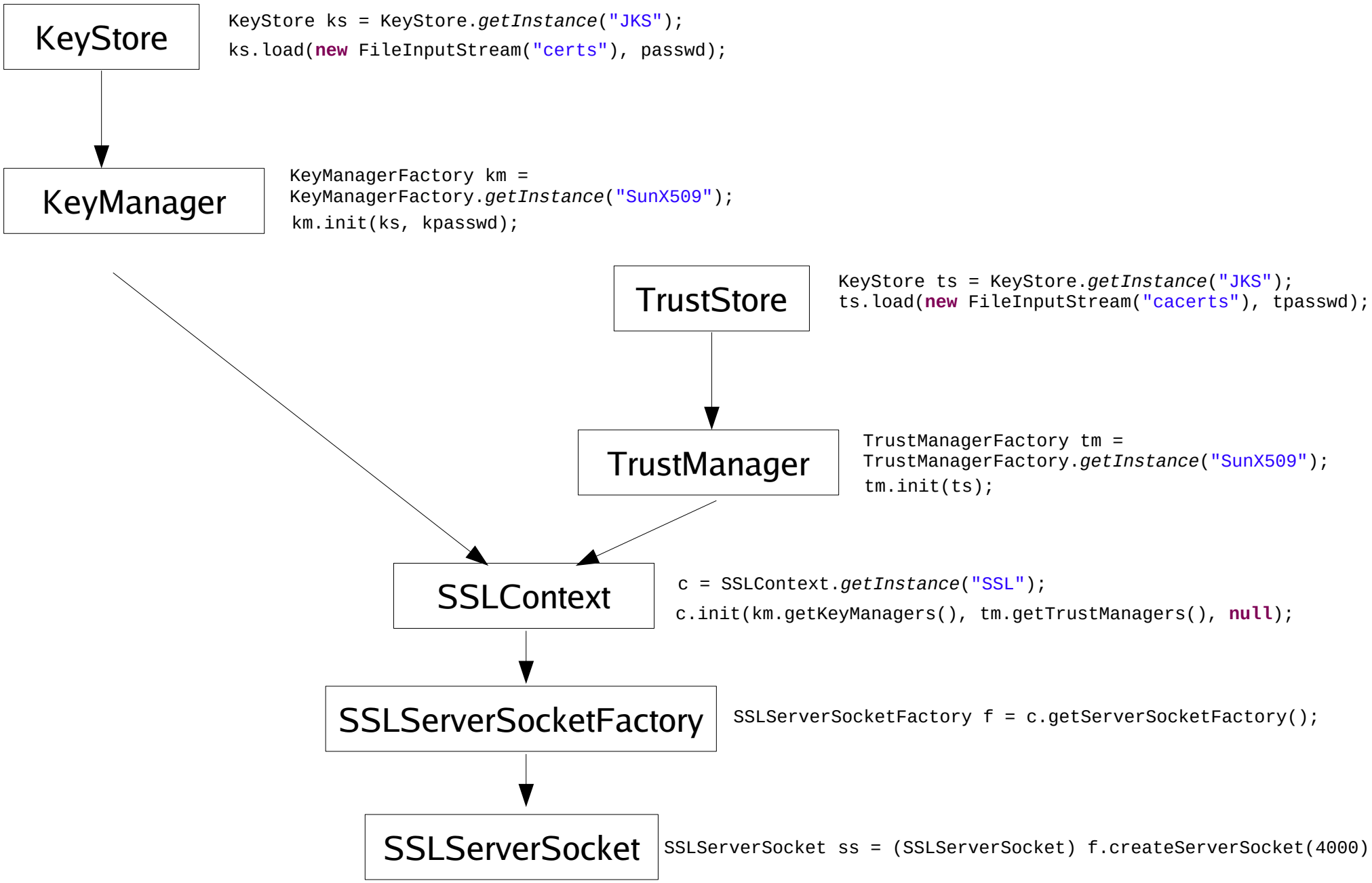
```
public abstract void setNeedClientAuth(boolean need)
```

`need` vale false se non si vuole l'autenticazione del client, true altrimenti

Autenticazione del Client

```
ss.setNeedClientAuth(false);
```

Riassumendo



Lato Client

Impostazione TrustStore

```
import javax.net.ssl.SSLException;
import javax.net.ssl.SSLSocket;
import javax.net.ssl.SSLSocketFactory;


public static void main(String[] args) {

   System.setProperty("javax.net.ssl.trustStore",
                      "cacerts");

   System.setProperty("javax.net.ssl.trustStorePassword",
                      "altrapass");


   ......
```

Creazione Socket con SSL

Metodo di `javax.net.ssl.SSLSocketFactory`

Non esiste un costruttore per `SSLSocketFactory`, bisogna utilizzare un metodo statico di `SSLSocketFactory`

```
public static ServerSocketFactory getDefault()
```

Creazione Socket con SSL

Metodi di `javax.net.ssl.SSLSocketFactory`

Non esiste un costruttore per `SSLSocket`, bisogna utilizzare i metodi di `SocketFactory`, che è una superclasse di `SSLSocketFactory`:

```
public abstract Socket createSocket(InetAddress host,
                                    int port)
                             throws IOException

public abstract Socket createSocket(String host,
                                    int port)
                             throws IOException,
                                    UnknownHostException
```

Creazione Socket con SSL

```
import javax.net.ssl.SSLException;
import javax.net.ssl.SSLSocket;
import javax.net.ssl.SSLSocketFactory;

public static void main(String[] args) {

    ....

    SSLSocketFactory f = (SSLSocketFactory)
                         SSLSocketFactory.getDefault();

    try {
        SSLSocket s = (SSLSocket)
              f.createSocket("fujih1",4000);

        // s viene utilizzato come un normale socket
    }
```

Algoritmi di Crittografia

Supportati, attivi, attivazione:

Identico al caso server

Il Client ed il Server devono attivare almeno un algortimo in comune

Credits

Prof. Fabrizio Baiardi