Socket Sicuri in Java

Laboratorio di Programmazione di Rete A

Daniele Sgandurra

Università di Pisa

05/12/2008

Comunicazioni Sicure

Confidenzialità, Integrità, Disponibilità

- Confidenzialità: le informazioni in transito sulla rete possono essere comprese solo dai soggetti che ne hanno diritto.
- Integrità: i dati sono protetti da modifiche da soggetti non autorizzati.
- Disponibilità: i dati devono essere sempre disponibili su richiesta dei soggetti che ne hanno diritto.

Confidenzialità

- Confidenzialità: atto di nascondere informazioni o risorse.
- In un sistema che garantisce la confidenzialità, una terza parte che entri in possesso dei dati scambiati tra mittente e destinatario, non deve essere in grado di comprendere le informazioni scambiate.
- La crittografia è usata per modificare i dati per renderli incomprensibili a chi non è in possesso di una chiave crittografica. La chiave controlla l'accesso ai dati modificati per renderli comprensibili.
- Un altro aspetto importante è l'occultamento di risorse: ad es., siti che vogliono nascondere la loro configurazione o i sistemi che stanno utilizzando.

Integrità

- Integrità: si riferisce al grado di fiducia associato a dati/risorse.
- Scopo: prevenire cambiamenti impropri o non autorizzati ai dati/risorse.
- Integrità dei dati=contenuto dell'informazione; integrità dell'origine=autenticazione.
- Meccanismi di prevenzione: cercano di mantenere l'integrità dei dati bloccando ogni tentativo non autorizzato di cambiare i dati o di cambiarli in maniera non autorizzata.
- Meccanismi di protezione: riscontrano se l'integrità dei dati non è più affidabile.

Disponibilità

- Disponibilità: capacità di usare un'informazione o risorsa desiderata.
- Chi ha diritto di conoscere le informazioni deve potervi accedere.
- Un'entità può rendere impossibile l'accesso a dati o servizi rendendoli non disponibili:
  - denial of service attacks (DoS);
  - contromisure: ridondanza, backup, etc.

Definizioni

- Identificazione: ad ogni soggetto è associato un identificativo unico.
- Autenticazione: processo che verifica l'identità di un soggetto.
- Autorizzazione: controllo dei diritti associati ad un soggetto.

Crittografia

- Come trasmettere messaggi tra due persone in maniera tale che il messaggio trasmesso sia inintelligibile a terze persone?
- Crittografia: si occupa di consentire la trasmissione di un messaggio in maniera celata in modo tale che solo il destinatario possa leggere il messaggio in maniera comprensibile.
- La steganografia rende possibile celare non solo il contenuto, ma anche il fatto stesso che si stia inviando un messaggio.

Crittosistema

- Siano M = l'insieme dei messaggi in chiaro, e C l'insieme dei messaggi cifrati.
- Una trasformazione crittografica è una funzione $f : M \longrightarrow C$ iniettiva, detta funzione di cifratura. È importante che la funzione sia iniettiva per evitare ambiguità nella decifrazione dei messaggi.
- La funzione inversa f^{-1} è detta funzione di decifratura.
- Un crittosistema è una quaterna (M, C, f, f^{-1}), dove f è una trasformazione crittografica.

Chiavi

- In tutti i sistemi crittografici, i soggetti che vogliono comunicare devono quasi sempre concordare delle chiavi: i parametri dei sistemi crittografici.
- Le chiavi non devono essere trasmesse su un canale insicuro.
- Principio di Kerckhoffs: la sicurezza del crittosistema deve risiedere non sul celare l'algoritmo di cifratura/decifratura, bensì solo sul celare la chiave di decifratura.

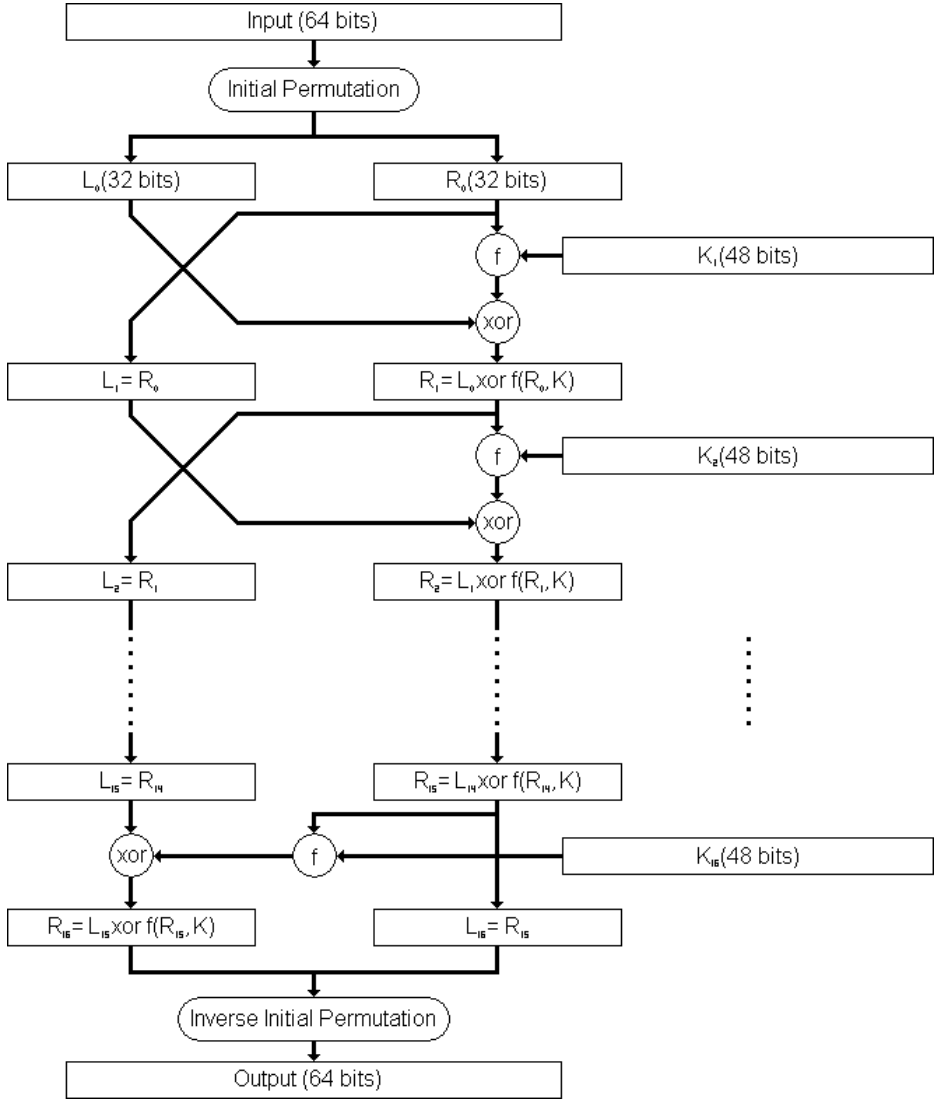
Crittografia Simmetrica e a Chiave Pubblica

- Crittografia simmetrica: nota la chiave di cifratura, la cifratura e decifratura del messaggio sono computazionalmente equivalenti:
  - due utenti, prima di cominciare a comunicare, devono concordare (oltre che il sistema crittografico stesso) anche le chiavi da utilizzare per la (de)cifratura.
- Crittografia a chiave pubblica: la funzione di decifratura ha una complessità computazionale di ordine maggiore rispetto a quella di cifratura.

Crittografia Simmetrica

- DES (Data Encryption Standard): sviluppato alla fine degli anni Settanta da un gruppo di ricercatori dell'IBM.
- Cifra blocchi di 64 bit con una chiave di 56 bit.
- Si raggruppa il messaggio in blocchi di 64 bit e:
  1. si esegue una permutazione iniziale π;
  2. si eseguono 16 cicli di cifratura (permutazioni, shift, sostituzioni);
  3. si esegue la permutazione finale π^{-1}.

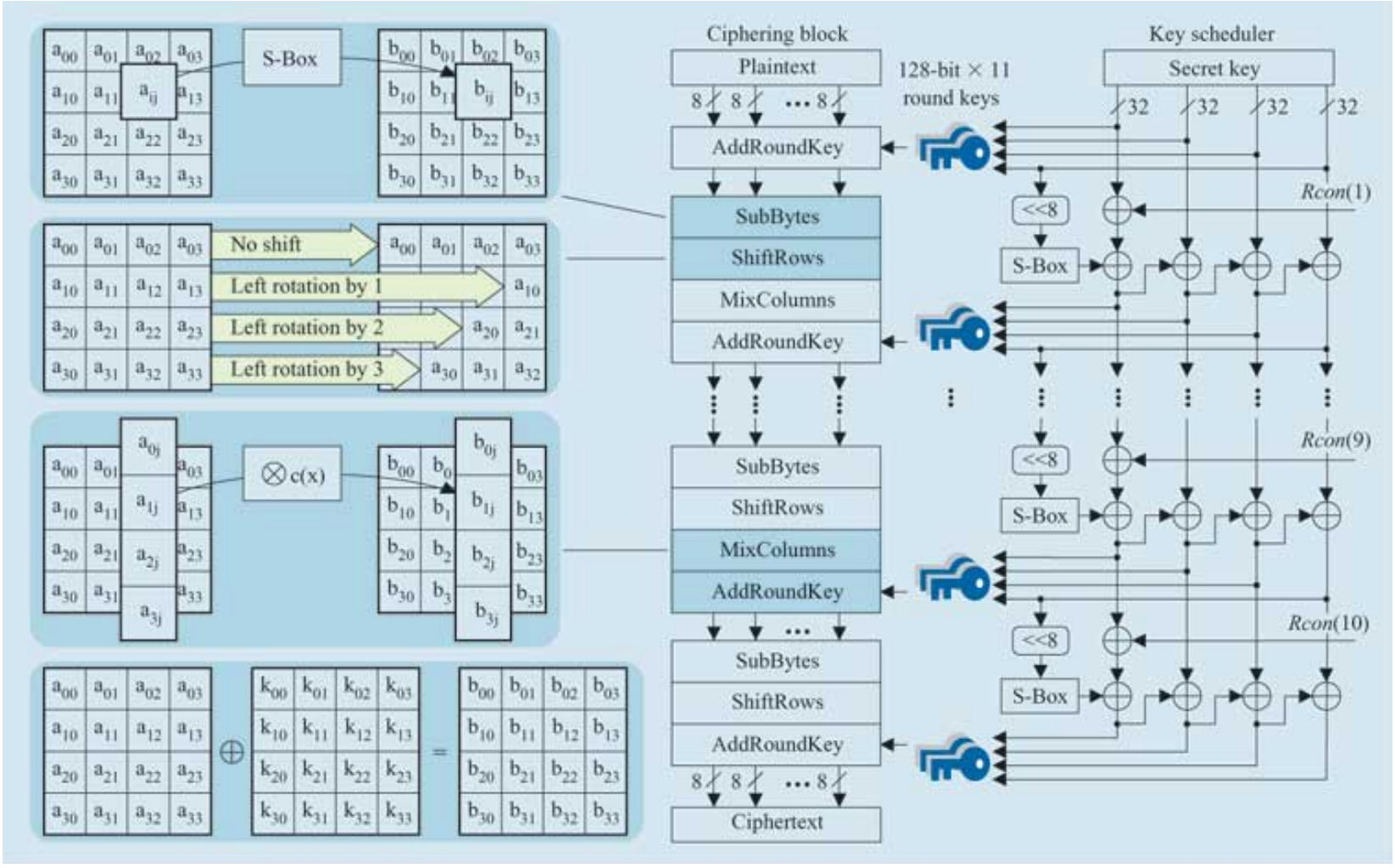
DES



Crittografia Simmetrica

- AES (Advanced Encryption Standard): in un concorso pubblico internazionale indetto dal NIST per determinare un nuovo standard di cifratura simmetrica che sostituisse il DES, venne scelto il Rijndael, che divenne l'AES.
- Blocchi di 128 bit, chiave di cifratura di 128, 192, 256 bit.
- Ciclo in cui usa quattro funzioni:
  1. sostituzione (SubBytes);
  2. spostamento righe (ShiftRows);
  3. mescolamento colonne (MixColumns);
  4. espansione (AddRoundKey).

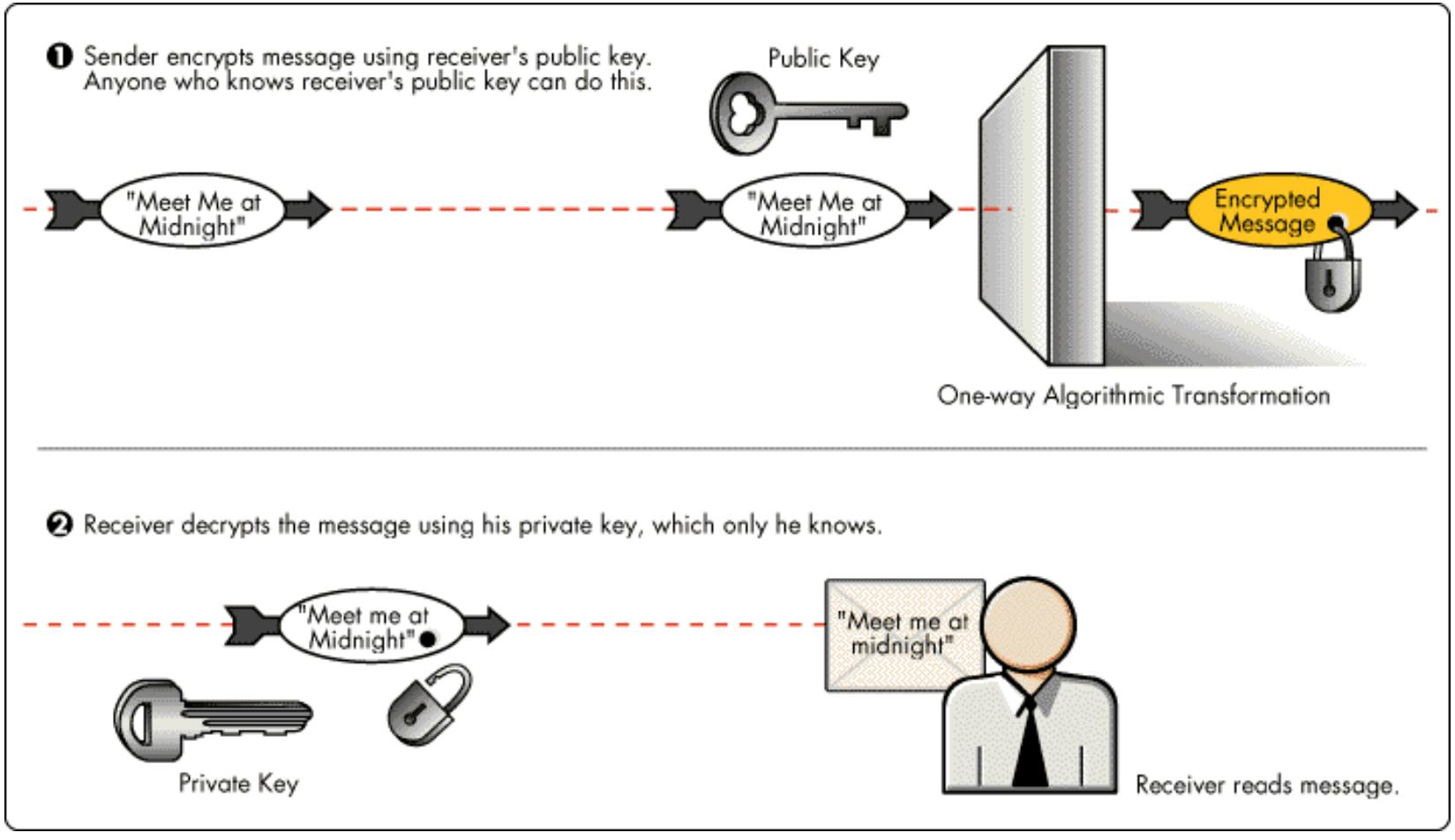
AES



Crittografia a Chiave Pubblica

- Il ruolo delle chiavi di cifratura e decifratura non è speculare: la chiave di decifratura è collegata ad un problema computazionale maggiore di quello utilizzato per costruire la prima.
- Ciascun utente sceglie una funzione crittografica che dipende da alcuni parametri, ma rende solo noti quelli che permettono di codificare i messaggi a lui diretti, mantenendo segreti quelli necessari alla decodifica.
- Chiunque può spedire un messaggio all'utente in questione senza che il messaggio, se intercettato, possa essere compreso.
- Esempi: RSA, DSA, Rabin, ElGamal.

Crittografia a Chiave Pubblica



Esempio: Doppio Lucchetto

L'utente A vuole spedire un messaggio segreto a B.

1. A mette il messaggio in una scatola che chiude con il suo lucchetto L_A (di cui solo lui ha la chiave) che spedisce a B;
2. B riceve la scatola chiusa con L_A, aggiunge il suo lucchetto L_B (di cui solo lui ha la chiave) e rispedisce il tutto ad A;
3. A, ricevuta la scatola con il doppio lucchetto, toglie il lucchetto L_A e rispedisce la scatola a B;
4. a questo punto, ricevuta la scatola, B può togliere il lucchetto L_B e leggere il messaggio di A.

Firma Digitale

Firma digitale: problema di autenticare una qualunque sequenza di cifre binarie, indipendentemente dal loro significato.

1. A, B due utenti di un sistema a chiave pubblica; f_A e f_B funzioni di cifratura pubbliche e f_A^{-1} e f_B^{-1} funzioni di decifratura private.
2. A vuole inviare un messaggio M a B: invia il messaggio cifrato $f_B(M)$ e, per certificare la propria identità, invia anche $f_B(f_A^{-1}(s_a))$, cioè la firma digitale di A. s_a è un nome convenzionale di A, che include anche un numero progressivo, un timestamp, etc.
3. B decifra il messaggio usando f_B^{-1} (che solo lui conosce) e ottiene M. Per controllare che il mittente sia proprio A, B calcola $f_A f_B^{-1}$ sulla firma digitale e ottiene s_A.

Ente Certificatore

- Il sistema funziona perché solamente *A* può aver firmato il messaggio: nessun altro conosce f_A^{-1}.
- Resta il problema di essere sicuri che sia proprio *A* a possedere f_A^{-1} e che non sia invece *C* che, rendendo pubblica una chiave, la abbia attribuita ad *A* spacciandosi per lui.
- Introduzione di una figura super partes: un Ente Certificatore delle chiavi pubbliche a cui ogni utente può rivolgersi per confrontare se la firma ottenuta coincide con quella certificata e depositata dall'ente.
- Un certificato digitale è un documento elettronico che attesta, con una firma digitale, l'associazione tra una chiave pubblica e l'identità di un soggetto.

Hash

- Di solito si fa dipendere la firma digitale dal messaggio stesso: la firma è $f_A^{-1}(h(M))$, dove $h(M)$ è una sequenza di bit di lunghezza fissata detta impronta di M, ottenuta da M tramite un'opportuna funzione detta di hash (ad es. SHA):
  - si firma l'hash del messaggio inviato.
- La funzione di hash ha la caratteristica di non consentire di risalire a M conoscendo solo $h(M)$ ed inoltre ha bassa probabilità di collisioni.

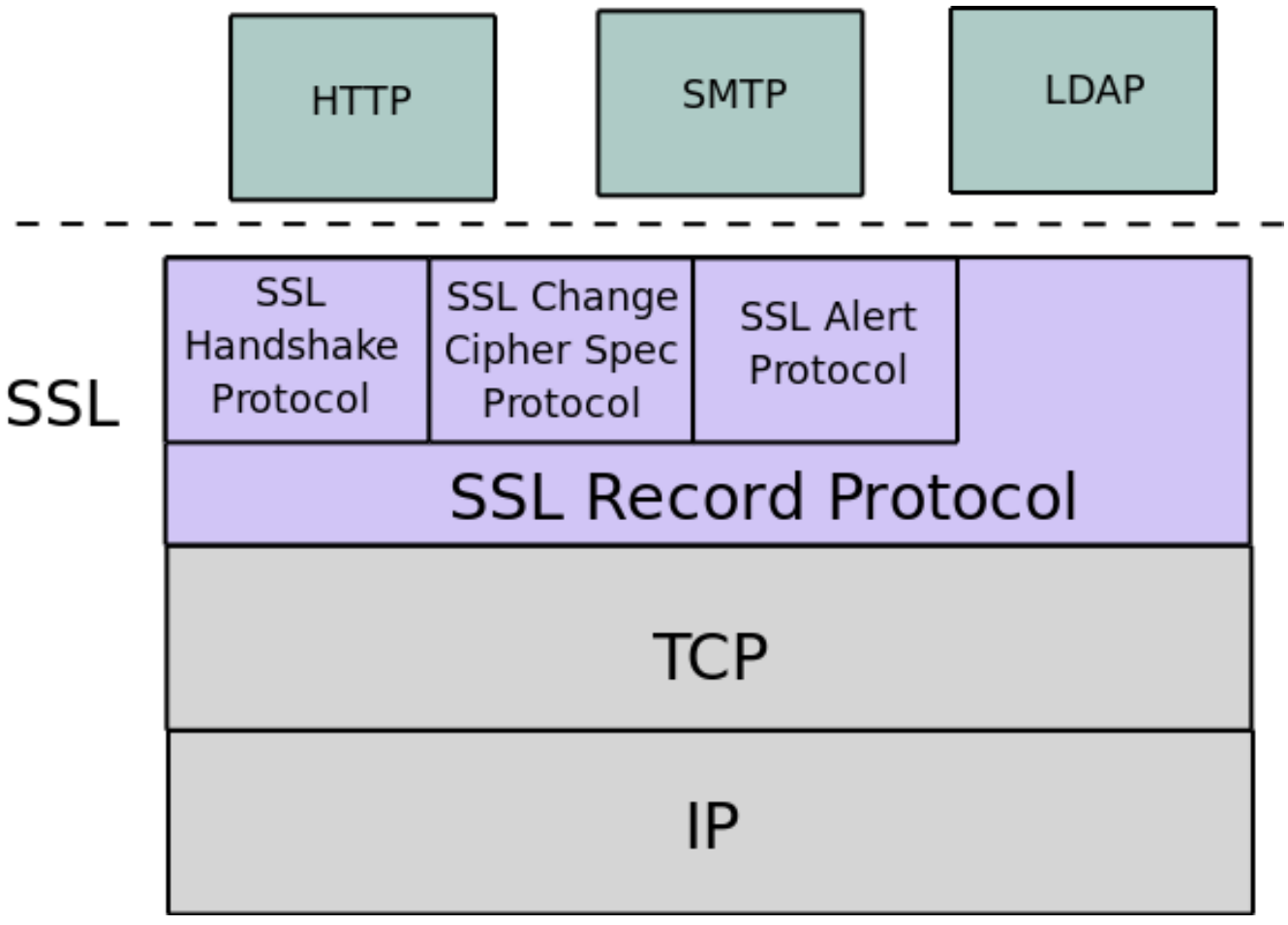
Secure Socket Layer

- Secure Socket Layer (SSL): protocollo della Netscape Communications Corporation.
- Comunicazioni sicure tramite Internet: posta elettronica, web, etc.
- Fornisce:
  - autenticazione: crittografia a chiave pubblica (es., RSA);
  - confidenzialità: crittografia simmetrica (es., AES);
  - integrità: hash del messaggio (es., SHA).

Introduzione a SSL

- Obiettivo: fornire confidenzialità e integrità dei dati tra due applicazioni che comunicano.
- Il protocollo permette alla applicazioni client/server di comunicare su un canale che impedisce la modifica dei dati, la falsificazione di messaggi e la lettura non autorizzata dei messaggi.
- TLS (Transport Security Protocol) è uno standard successivo che si basa su SSL.
- TLS 1.0, SSL 3.0 e SSL 2.0 non sono interoperabili.

SSL Layer



Record Protocol

- Sia SSL che TLS consistono di un record protocol, di un handshake protocol e di un opzionale session resumption protocol.
- Record Protocol è il livello più basso: fornisce riservatezza e affidabilità:
  - riservatezza: cifratura dei dati tramite un algoritmo di crittografia simmetrico (DES, RC4, ...). Le chiavi sono generate per ogni connessione tramite un "segreto" negoziato in precedenza.
  - affidabilità: integrità dei messaggi tramite un Message Authentication Code (MAC). Es. di funzioni di hash usate: SHA, MD5.

Handshake Protocol

- L'Handshake Protocol permette al client e al server di autenticarsi a vicenda e di negoziare un algoritmo di cifratura e le chiavi simmetriche prima di scambiarsi i dati.
- L'handshake protocol supporta le proprietà di autenticazione, riservatezza e integrità.
- L'identità del peer può essere certificata tramite crittografia asimmetrica (RSA, DSS). Di solito usata dal client per verificare l'identità del server.
- La negoziazione della chiave segreta condivisa è privata (non leggibile da terzi) e, se almeno un peer è autenticato, protetta da man-in-the-middle attack.
- I messaggi di negoziazione non possono essere modificati dagli attaccanti senza essere rilevati da i peer.

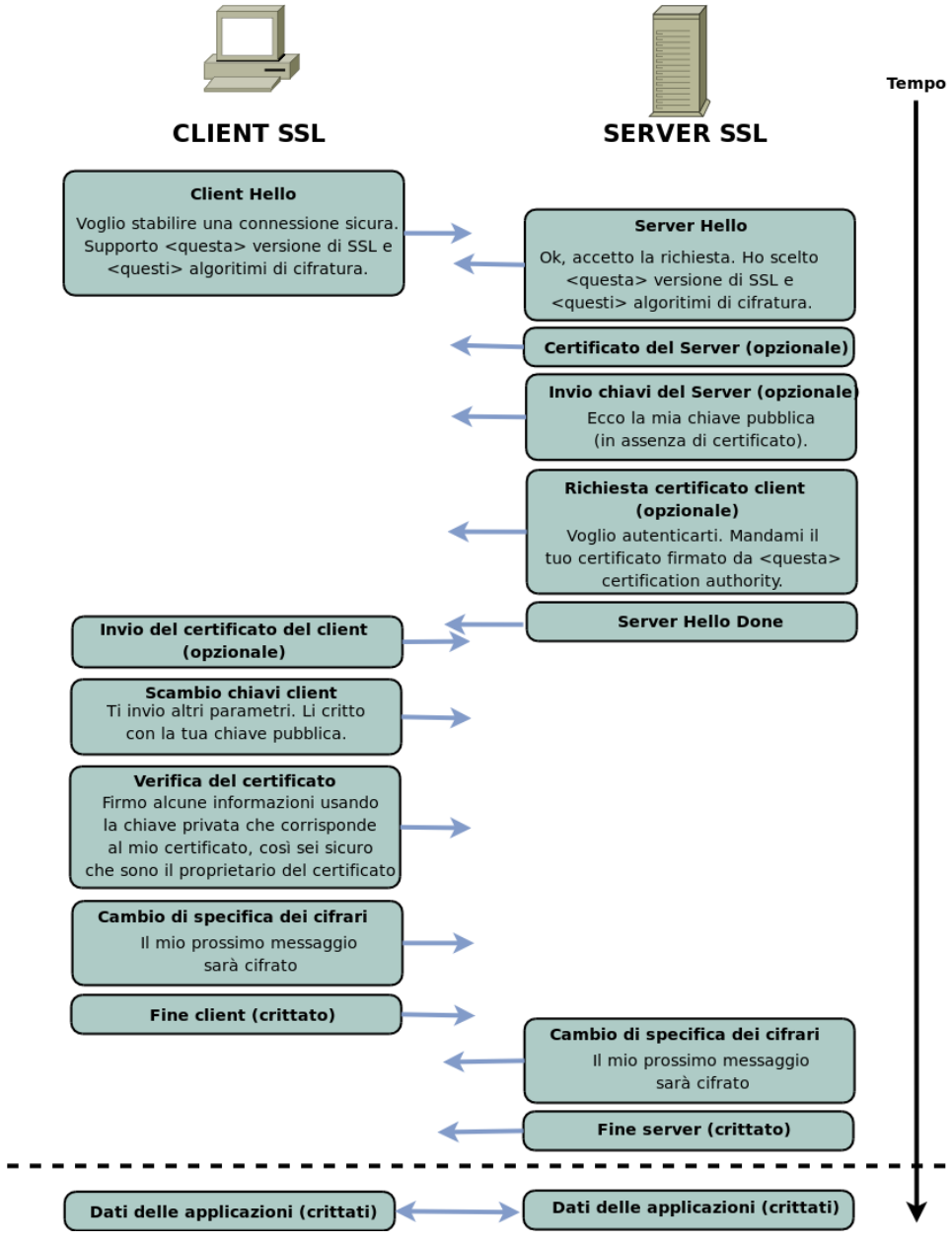
Sessione SSL

- Una sessione SSL è un insieme di parametri di sicurezza su cui entrambi i peer si sono messi d'accordo nella fase di handshake.
- Include l'autenticazione dei peer (opzionale), i metodi di cifratura e compressione, le chiavi pubbliche e private e i master secret usati da questi metodi.
- Una sessione SSL può sopravvivere anche dopo che la connessione è stata interrotta, e può essere "resuscitata" in successive connessioni tra gli stessi peer:
  - in questa maniera, gli stessi parametri di sicurezza associati ad una sessione SSL possono essere usati su varie connessioni TCP (anche in parallelo).

Suite di Cifratura

- SSL e TLS supportano vari algoritmi di scambio di chiavi, di cifratura e di MAC: questi vengono raggruppati in suite di cifratura.
- Le suite sono identificate da nomi della forma: `TLS_keyexchange_WITH_chipher_hash` o `SSL_keyexchange_WITH_chipher_hash`.
- `keyexchange` specifica l'algoritmo per lo scambio delle chiavi, `cipher` specifica l'algoritmo di cifratura, `hash` specifica l'algoritmo di autenticazione.

Il Protocollo SSL



Socket Sicuri in Java

Schema Generico Codice SSL Lato Client

- Nel 1999 la SUN introdusse le Java Secure Sockets Extensions (JSSE), che supportano TLS 1.0, SSL 3.0.
- JSSE è composto dei package `javax.net` e `javax.net.ssl`.
- Prima di potere usare JSSE sono necessari alcuni passi:
  - se si vuole usare l'autenticazione (client o server), bisogna installare un keystore;
  - può essere necessario installare un truststore.
- Riferimento libro di testo (Pitt): cap.VII.

Schema Generico Codice SSL Lato Client

In rosso le differenze rispetto all'uso di un normale socket TCP:

```
import java.io.*;
import javax.net.ssl.*;

. . .

try
{
   SSLSocketFactory sslFact =
      (SSLSocketFactory)SSLSocketFactory.getDefault();
   SSLSocket s =
      (SSLSocket)sslFact.createSocket(host, port);

   OutputStream out = s.getOutputStream();
   InputStream in = s.getInputStream();

   //invia msg al server con outputstrem
   //ricevi messaggio dal server con inputstream
}catch (IOException e) {}
```

La Classe SSLSocketFactory

Per creare un socket SSL lato client:

- Invece di usare un `java.net.socket`, si usa il metodo `createSocket()` della classe astratta `javax.net.ssl.SSLSocketFactory`.
- Per ottenere un'istanza di questa classe, si invoca il metodo `getDefault()` di `SSLSocketFactory`.

```
import javax.net.ssl.*;
...
SSLSocketFactory factory =
(SSLSocketFactory)SSLSocketFactory.getDefault();
SSLSocket socket =
(SSLSocket)factory.createSocket("www.verisign.com", 443);
```

Usare il Socket

- Il socket creato con `createSocket` può essere utilizzato come un socket normale.
- Si ottengono gli stream con `getInputStream()` e `getOututStream()` per inviare/ricevere dati.

```
PrintWriter out = new PrintWriter(
    new BufferedWriter(
    new OutputStreamWriter(
    socket.getOutputStream())));
    out.println("GET / HTTP/1.0");
    out.println();
    out.flush();

BufferedReader in = new BufferedReader(
    new InputStreamReader(
    socket.getInputStream()));
String inputLine;
while ((inputLine = in.readLine()) != null)
    System.out.println(inputLine);
```

Esempio Completo Client: Connessione Sicura con un Web Server

```
import java.net.*;
import java.io.*;
import javax.net.ssl.*;
public class SecureClientSocket
{
   public static void main(String[] args)
   {
      try{
         SSLSocketFactory factory =
            (SSLSocketFactory)SSLSocketFactory.getDefault();
         SSLSocket socket =
            (SSLSocket)factory.createSocket("www.verisign.com", 443);
         PrintWriter out = new PrintWriter(new BufferedWriter(
               new OutputStreamWriter(socket.getOutputStream())));
         out.println("GET / HTTP/1.0");
         out.println(); out.flush();
         BufferedReader in = new BufferedReader(
               new InputStreamReader(socket.getInputStream()));
         String inputLine;
         while((inputLine = in.readLine()) != null)
               System.out.println(inputLine);
         in.close(); out.close(); socket.close();
      }catch(Exception e){e.printStackTrace();}
   }
}
```

Schema Generico Codice SSL Lato Server

In rosso le differenze rispetto all'uso di un normale socket TCP:

```
import java.io.*;
import javax.net.ssl.*;
. . .

SSLServerSocket s;
try
{
   SSLServerSocketFactory sslSrvFact =
      (SSLServerSocketFactory)SSLServerSocketFactory.getDefault();
   s =(SSLServerSocket)sslSrvFact.createServerSocket(port);

   SSLSocket c = (SSLSocket)s.accept();

   OutputStream out = c.getOutputStream();
   InputStream in = c.getInputStream();

   //invia msg al client con outputstream
   //ricevi msg dal client con inputstream
}catch(IOException e){}
```

La Classe SSLServerSocket

Per creare un socket SSL lato server:

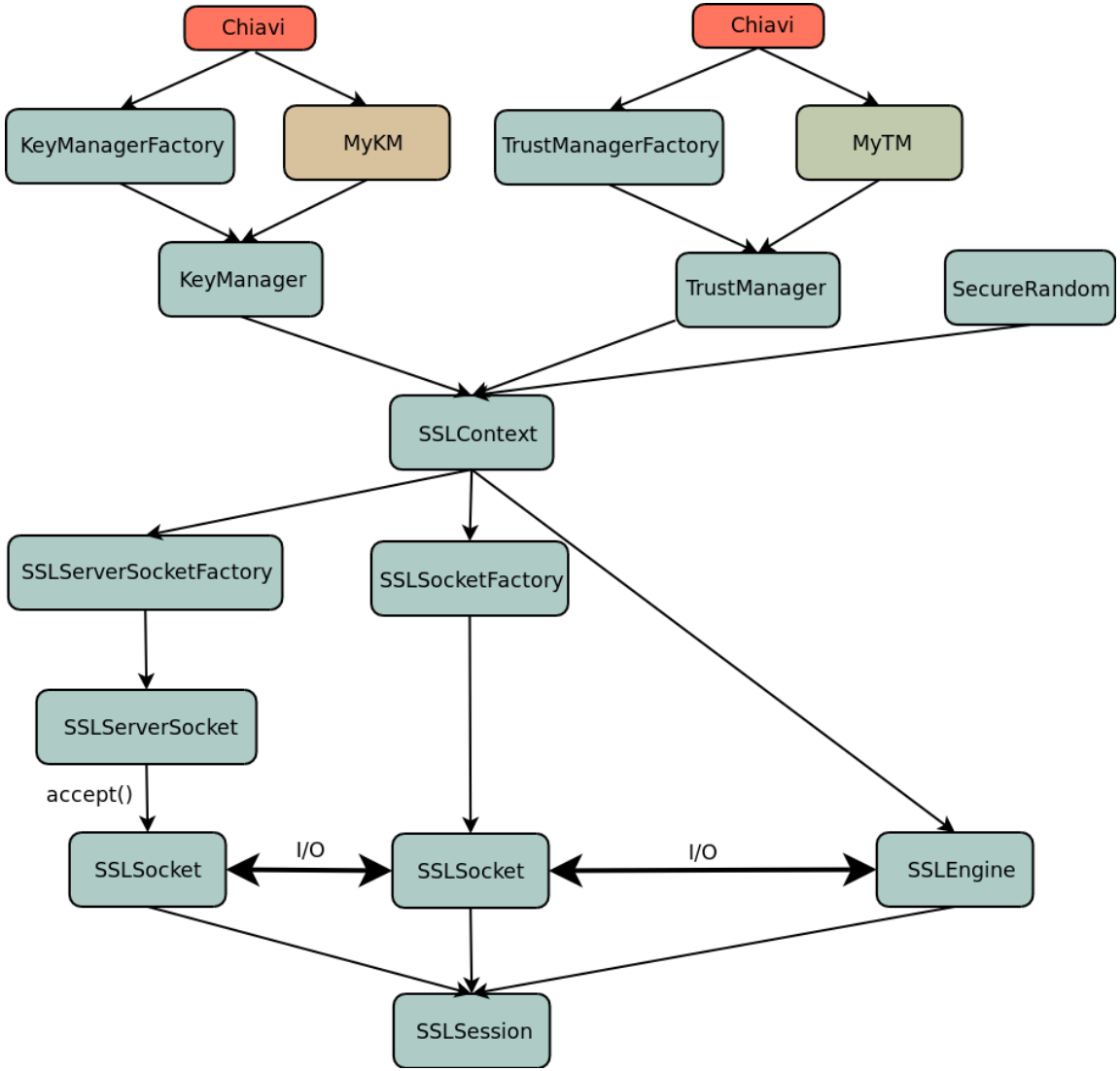
- Classe astratta `javax.net.SSLServerSocket`.
- Per ottenere un'istanza di questa classe, si invoca il metodo `getDefault()` di `SSLServerSocketFactory`.
- Si invoca il metodo `createServerSocket(int port)` sull'oggetto `SSLServerSocketFactory` ritornato.
- Di default, l'oggetto ritornato non supporta la crittatura (solo l'autenticazione).

Il Contesto SSL

Per configurare e inizializzare i socket sicuri, bisogna creare un oggetto `SSLContext`. Passi:

1. Generare le chiavi pubblica e privata e il certificato con `keytool`.
2. Creare un oggetto `SSLContext` per specificare gli algoritmi da usare.
3. (Creare un oggetto `TrustManagerFactory` per specificare le locazioni dei certificati). Opzionale
4. Creare un oggetto `KeyManagerFactory` per le chiavi utilizzate.
5. Creare un oggetto `KeyStore` per il database di chiavi e certificati.
6. Riempire il `KeyStore` con le chiavi e i certificati caricati da file.
7. Inizializzare il `KeyManagerFactory` con il `KeyStore` e la password.
8. Inizializzare l'`SSLContext`.

Schema Classi Java



Keystore

- Un keystore è un database di chiavi:
  - ad es, lato server, contiene una coppia di chiavi pubbliche/private usate per autenticarsi;
  - nel caso del keystore si usa una password per proteggere la confidenzialità delle chiavi.
- Il nome e la password del keystore possono essere passati all'avvio con: `-Djavax.net.ssl.keyStore=keystore` e `-Djavax.net.ssl.keyStorePassword=password`.

Truststore

- Un trustore è anch'esso un keystore, ma viene usato dai peer che ricevono un certificato per stabilire se ci si può fidare dell'altro peer. Serve per verificare l'idendità di un'entità:
  - ad es., lato client, contiene il database con i certificati fidati;
  - si usa una password per proteggere l'integrità del trustore (non la confidenzialità).
- Di default, Java usa il file `cacerts` nella directory `${JAVA_HOME/lib/security}`, contenente una lista di certificati fidati.
- Il nome e la password del trustore possono anche essere passati all'avvio con: `-Djavax.net.ssl.trustStore=truststore` e `-Djavax.net.ssl.trustStorePassword=trustword`.

Creare le Chiavi con Keytool

`keytool` è uno strumento per gestire e creare chiavi pubbliche/private e certificati self-signed.
Passi: (A) sul server si creano le chiavi pubbliche/private con:

```
> keytool -genkey -alias mystore -keystore mykey
Enter keystore password:  keypassword
What is your first and last name?
  [Unknown]:  Daniele Sgandurra
What is the name of your organizational unit?
  [Unknown]:  Dipartimento di Informatica
What is the name of your organization?
  [Unknown]:  Universita' di Pisa
What is the name of your City or Locality?
  [Unknown]:  Pisa
What is the name of your State or Province?
  [Unknown]:  Italy
What is the two-letter country code for this unit?
  [Unknown]:  IT
Is CN=Daniele Sgandurra, OU=Dipartimento di Informatica,
   O=Universita' di Pisa, L=Pisa, ST=Italy, C=IT correct?
  [no]:  yes

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

che genera il file "mykey" con la chiave pubblica/privata del server protette da password "keypassword".

Creare il Certificato con Keytool

(B) sul server si crea il certificato associato alle chiavi con:

```
> keytool -export -keystore mykey -alias mystore -file mycertificate.cer
Enter keystore password:  keypassword
Certificate stored in file <mycertificate.cer>
```

che genera il certificato “mycertificate.cer” (self-signed) a partire da file “mykey”. Per visualizzare il contenuto del certificato:

```
> keytool -printcert -file mycertificate.cer
Owner: CN=Daniele Sgandurra, OU=Dipartimento di Informatica, O=Universita' di Pisa,
       L=Pisa, ST=Italy, C=IT
Issuer: CN=Daniele Sgandurra, OU=Dipartimento di Informatica, O=Universita' di Pisa,
       L=Pisa, ST=Italy, C=IT
Serial number: 4747e751
Valid from: Sat Nov 24 09:56:49 CET 2007 until: Fri Feb 22 09:56:49 CET 2008
Certificate fingerprints:
         MD5:  62:15:AF:85:B6:51:CA:9E:AE:74:98:1F:5F:F9:CB:F0
         SHA1: 1F:25:2F:62:3F:93:CC:F0:DE:72:31:EE:1B:4A:EE:2D:CE:B4:AF:E1
```

Aggiungere il Certificato allo Store

(C) il certificato “mycertificate.cer” (NON le chiavi) va dato al client.
(D) il client aggiunge il certificato del server al proprio truststore con:

```
> keytool -import -alias mystore -file mycertificate.cer -keystore clientstore
Enter keystore password:  clientstorepassword
Owner: CN=Daniele Sgandurra, OU=Dipartimento di Informatica, O=Universita' di Pisa,
       L=Pisa, ST=Italy, C=IT
Issuer: CN=Daniele Sgandurra, OU=Dipartimento di Informatica, O=Universita' di Pisa,
       L=Pisa, ST=Italy, C=IT
Serial number: 4747e751
Valid from: Sat Nov 24 09:56:49 CET 2007 until: Fri Feb 22 09:56:49 CET 2008
Certificate fingerprints:
         MD5:  62:15:AF:85:B6:51:CA:9E:AE:74:98:1F:5F:F9:CB:F0
         SHA1: 1F:25:2F:62:3F:93:CC:F0:DE:72:31:EE:1B:4A:EE:2D:CE:B4:AF:E1
Trust this certificate? [no]:  yes
Certificate was added to keystore
```

che genera il file “clientstore” con lo store del client contenente il certificato del server. Per vedere il contenuto dei keystore/truststore:

```
> keytool -list -v -keystore clientstore
```

Web Server Sicuro

Esempio Completo Client: Connessione Sicura con Web Server

```
import java.net.*;
import java.io.*;
import javax.net.ssl.*;
public class SecureClientSocket
{
   public static void main(String[] args) throws Exception
   {
      int port = 3000;
      try
      {
         SSLSocketFactory factory =
            (SSLSocketFactory)SSLSocketFactory.getDefault();
         SSLSocket socket =
            (SSLSocket)factory.createSocket("localhost", port);
         PrintWriter out = new PrintWriter(new BufferedWriter(
               new OutputStreamWriter(socket.getOutputStream())));
         out.println("GET / HTTP/1.0"); out.println(); out.flush();
         BufferedReader in = new BufferedReader(
               new InputStreamReader(socket.getInputStream()));
         String inputLine;
         while((inputLine = in.readLine()) != null)
               System.out.println(inputLine);
         in.close(); out.close(); socket.close();
      }catch(Exception e){e.printStackTrace();}
   }
}
```

Esempio Completo: Web Server Sicuro

```
import java.io.*;
import java.net.*;
import java.security.KeyStore;
import javax.net.*;
import javax.net.ssl.*;
import javax.security.cert.X509Certificate;

public class SecureServerSocket
{
   public static void main(String args[])
   {
      ServerSocket server = null;
      int port = 3000;
      Socket socket=null;

      try
      {
         ServerSocketFactory ssf = getServerSocketFactory();//metodo statico definito sotto
         server = ssf.createServerSocket(port);
      }catch (IOException e)e.printStackTrace();
```

Esempio Completo: Web Server Sicuro (cont.)

```
    try
    {
       socket = server.accept();
       PrintWriter out = new PrintWriter(new BufferedWriter(
                         new OutputStreamWriter(
                         socket.getOutputStream())));
       BufferedReader in = new BufferedReader(
                         new InputStreamReader(socket.getInputStream()));

       out.print("HTTP/1.0 200 OK\r\n");
       out.print("Content-Length: " + "100" + "\r\n");
       out.print("Content-Type: text/html\r\n\r\n");
       out.flush();

     }catch (Exception e){e.printStackTrace();}
       finally {try{socket.close();}catch (IOException e){}}
}
```

Esempio Completo: Web Server Sicuro (cont.)

```
    private static ServerSocketFactory getServerSocketFactory()
    {
        SSLServerSocketFactory ssf = null;
        try
        {
            SSLContext ctx;
            KeyManagerFactory kmf;
            KeyStore ks;
            char[] passphrase = "keypassword".toCharArray();//passwd usata per salvare la chiave
            String keyfile = "mykey";//nome del file con la chiave

            ctx = SSLContext.getInstance("TLS");//TLS: successore di SSL
            kmf = KeyManagerFactory.getInstance("SunX509");//algoritmo per i certificati
            ks = KeyStore.getInstance("JKS");//JavaKeyStore: tipo di store usato da keytool
            ks.load(new FileInputStream(keyfile), passphrase);
            kmf.init(ks, passphrase);
            ctx.init(kmf.getKeyManagers(), null, null);
            ssf = ctx.getServerSocketFactory();
            return ssf;
        }catch(Exception e){e.printStackTrace();}
        return null;
    }
}
```

Esecuzione

Per avviare il server:

```
> java SecureServerSocket
```

Per avviare il client:

```
> java -Djavax.net.ssl.trustStore=clientstore SecureClientSocket
HTTP/1.0 200 OK
Content-Length: 100
Content-Type: text/html
```

Tramite la proprietà `javax.net.ssl.trustStore` si setta il truststore al file "clientstore".

URL e HTTPS

Che Cos'è un URL

URL è un acronimo per Uniform Resource Locator:

- Un riferimento (un indirizzo) per una risorsa su Internet.
- Di solito un URL è il nome di un file su un host.
- Ma può anche puntare ad altre risorse:
  - una query per un database;
  - l'output di un comando.
- Es: `http://java.sun.com`
  1. `http`: identificativo del protocollo.
  2. `java.sun.com`: nome della risorsa.

Resource Name

Il nome di una risorsa è composto da:

- Host Name: il nome dell'host su cui si trova la risorsa.
- Filename: il pathname del file sull'host.
- Port Number: il numero della porta su cui connettersi (opzionale).
- Reference: un riferimento ad una specifica locazione all'interno del file (opzionale).

Nel caso del protocollo `http`, se il Filename è omesso (o finisce per `/`), il web server è configurato per restituire un file di default all'interno del path (ad es. `index.html`, `index.php`, `index.asp`).

URL e URI

- Un URI (Uniform Resource Identifier) è un costrutto sintattico che specifica, tramite le varie parti che lo compongono, una risorsa su Internet:
  - `[schema:]ParteSpecificaSchema[#frammento]`
  - dove `ParteSpecificaSchema` ha la struttura: `[//autorita']][percorso][?query]`
- Un URL è un tipo particolare di URI: contiene sufficienti informazioni per individuare e ottenere una risorsa.
- Altre URI, ad es: `URN:ISBN:0-395-36341-1` non specificano come individuare la risorsa. In questo caso, le URI sono dette URN (Uniform Resource Name).

URL in JAVA

- In JAVA per creare un oggetto URL:

```
URL cli = new URL("http://www.cli.di.unipi.it/");
```

  che è un esempio di un URL assoluto.

- È anche possibile creare un URL relativo, che ha la forma `URL(URL baseURL, String relativeURL)`, ad es.

```
URL cli = new URL("http://www.cli.di.unipi.it/");
URL faq = new URL(cli, "faq");
```

  che risulterà puntare a `http://www.cli.di.unipi.it/faq`

- I protocolli gestiti da Java con gli URL sono `http`, `https`, `ftp`, `file` e `jar`.

- I costruttori possono lanciare una `MalformedURLException`.

Parsare un URL

Sono presenti metodi per ottenere informazioni sull'URL:

```
import java.net.*;
import java.io.*;

public class URLReader
{
   public static void main(String[] args) throws Exception
   {
      String url = "http://www.cli.di.unipi.it:80/faq";
      URL cli = new URL(url);

      System.out.println("protocol = " + cli.getProtocol());
      System.out.println("authority = " + cli.getAuthority());
      System.out.println("host = " + cli.getHost());
      System.out.println("port = " + cli.getPort());
      System.out.println("path = " + cli.getPath());
      System.out.println("query = " + cli.getQuery());
      System.out.println("filename = " + cli.getFile());
      System.out.println("ref = " + cli.getRef());
   }
}
```

Parsare un URL

Eseguendo l'esempio precedente si ottiene:

```
protocol = http
authority = www.cli.di.unipi.it:80
host = www.cli.di.unipi.it
port = 80
path = /faq
query = null
filename = /faq
ref = null
```

Leggere un URL

- Una volta creato un oggetto URL si può invocare il metodo `openStream()` per ottenere uno stream da cui poter leggere il contenuto dell'URL.
- Il metodo `openStream()` ritorna un oggetto `java.io.InputStream`: leggere da un URL è analogo a leggere da uno stream di input.

```
import java.net.*;
import java.io.*;
public class URLReader
{  public static void main(String[] args) throws Exception
   {
      String url = "http://www.cli.di.unipi.it/";
      URL cli = new URL(url);
      BufferedReader in = new BufferedReader(
         new InputStreamReader(cli.openStream()));
      String inputLine;
      while((inputLine = in.readLine()) != null)
         System.out.println(inputLine);
      in.close();
   }
}
```

Leggere un URL

Nell'esempio di prima, leggendo l'URL si ottiene:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html lang="it">
<head>
  <meta http-equiv="Content-Type" content="text/html;
                                   charset=iso-8859-1">
  <link rel="stylesheet" href="/cdc.css" type="text/css">
  <link rel="alternate" type="application/rss+xml"
                 title="Ultime notizie" href="/feed.php">
  <title>Home CdC </title>
</head>
<body bgcolor="#ced8e0">
....
```

Se necessario, settare il proxy con le proprietà:

```
java -Dhttp.proxyHost=proxyhost
    [-Dhttp.proxyPort=portNumber] URLReader
```

Connettersi ad un URL

- Nell'esempio precedente, la connessione all'URL veniva effettuata solo dopo aver invocato `openStream()`.
- In alternativa, è possibile invocare il metodo `openConnection()` per ottenere un oggetto `URLConnection`.
- Utile nel caso in cui si vogliano settare alcuni parametri o proprietà della richiesta prima di connettersi.
  - es: `cliConn.setRequestProperty("User-Agent", "Mozilla/5.0");`
- Successivamente, si invoca `URLConnection.connect()`.

```
...
String url = "http://www.cli.di.unipi.it/";
URL cli = new URL(url);
URLConnection cliConn = cli.openConnection();
cliConn.connect();
BufferedReader in = new BufferedReader(
      new InputStreamReader(
      cliConn.getInputStream()));
...
```

URL e HTTPS

Si possono usare le URL anche per connessioni sicure su HTTPS:

```
import java.net.*;
import java.io.*;
public class SecureClientUrl
{
   public static void main(String[] args)
    {
      try
      {
         URL url = new URL("https://www.verisign.com");
         URLConnection conn = url.openConnection();
         BufferedReader in = new BufferedReader(
                new InputStreamReader(conn.getInputStream()));
         String inputLine;
         while((inputLine = in.readLine()) != null)
            System.out.println(inputLine);
         in.close();
      }catch(Exception e){e.printStackTrace();}
    }
}
```

RMI Sicuro

Custom Socket Factory

Le custom socket factory possono essere usate per controllare come le invocazioni dei metodi remoti sono eseguite a livello di rete:

- per settare le opzioni dei socket;
- per stabilire il tipo di comunicazione (es., con autenticazione);
- per controllare la codifica dei dati (crittatura, compressione).

Custom Socket Factory e RMI

Se si usa RMI, quando un oggetto remoto è esportato con `exportObject()`, è possibile specificare come parametri:

- un custom socket factory per il client (un'istanza di `java.rmi.server.RMIClientSocketFactory`),
- un custom socket factory per il server (un'istanza di `java.rmi.server.RMIServerSocketFactory`),

che saranno utilizzati per effettuare le comunicazioni tramite RMI, cioè per invocare metodi remoti sugli oggetti remoti.

Client e Server Socket Factory

Un client socket factory controlla la creazione dei socket utilizzati per stabilire invocazioni remote:

- come le connessioni sono stabilite;
- il tipo di socket da usare.

Un server socket factory controlla la creazione dei socket lato server:

- come le connessioni in ingresso sono accettate;
- il tipo di socket da usare per accettare le connessioni in ingresso.

Lo stub per un oggetto remoto è associato al client socket factory:

- un client socket factory deve implementare `Serializable`;
- il codice del client socket factory può essere scaricato a runtime dal client, settando il `codebase`.

Esempio di Client RMI

```
import java.net.InetAddress;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.RemoteException;

public class HelloClient
{
   public static void main(String args[])
   {
      try
      {
         Registry registry = LocateRegistry.getRegistry("localhost");
         Hello obj = (Hello)registry.lookup("Hello");
         String message = "";
         message = obj.sayHello();
         System.out.println(message);
      }catch(Exception e){e.printStackTrace();}
   }
}
```

Esempio di Server RMI: l'Oggetto Remoto

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Hello extends Remote
{
   String sayHello() throws RemoteException;
}
```

```
import java.net.InetAddress;
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
import java.rmi.RMISecurityManager;
import java.rmi.server.*;

public class HelloImpl implements Hello
{
   public HelloImpl() throws RemoteException
   {
   }
```

Esempio di Server RMI: l'Oggetto Remoto e il Main

```
    public String sayHello()
    {
      return "Hello World!";
    }

    public static void main(String args[])
    {
      if(System.getSecurityManager() == null)
         System.setSecurityManager(new RMISecurityManager());

      try
      {
         HelloImpl obj = new HelloImpl();
         RMIClientSocketFactory csf = new RMISSLClientSocketFactory();
         RMIServerSocketFactory ssf = new RMISSLServerSocketFactory();
         Hello stub = (Hello)UnicastRemoteObject.exportObject(obj, 0, csf, ssf);
         Registry registry = LocateRegistry.getRegistry();
         registry.rebind("Hello", stub);
      }catch (Exception e){e.printStackTrace();}
   }
}
```

Esempio di Server RMI: il Client Socket Factory

```
import java.io.*;
import java.net.*;
import java.rmi.server.*;
import javax.net.ssl.*;

public class RMISSLClientSocketFactory implements RMIClientSocketFactory, Serializable
{
   public Socket createSocket(String host, int port) throws IOException
   {
      SSLSocketFactory factory = (SSLSocketFactory)SSLSocketFactory.getDefault();
      return (SSLSocket)factory.createSocket(host, port);
   }

   public int hashCode()
   { return getClass().hashCode(); }

   public boolean equals(Object obj)
   {
      if(obj == this)
         return true;
      else if (obj == null || getClass() != obj.getClass())
         return false;
      return true;
   }
}
```

Esempio di Server RMI: il Server Socket Factory

```
import java.io.*;
import java.net.*;
import java.rmi.server.*;
import javax.net.ssl.*;
import java.security.KeyStore;
import javax.net.ssl.*;
public class RMISSLServerSocketFactory implements RMIServerSocketFactory
{
   private SSLServerSocketFactory ssf = null;
   public RMISSLServerSocketFactory()
   {
      try
      {  SSLContext ctx;
         KeyManagerFactory kmf;
         KeyStore ks;
         char[] passphrase = "keypassword".toCharArray();
         ks = KeyStore.getInstance("JKS");
         ks.load(new FileInputStream("keyfile"), passphrase);
         kmf = KeyManagerFactory.getInstance("SunX509");
         kmf.init(ks, passphrase);
         ctx = SSLContext.getInstance("TLS");
         ctx.init(kmf.getKeyManagers(), null, null);
         ssf = ctx.getServerSocketFactory();
      }catch(Exception e){e.printStackTrace();}
   }
```

Esempio di Server RMI: il Server Socket Factory

```
    public ServerSocket createServerSocket(int port) throws IOException
    {
       return ssf.createServerSocket(port);
    }

    public int hashCode()
    { return getClass().hashCode(); }

    public boolean equals(Object obj)
    {
       if(obj == this)
          return true;
       else if (obj == null || getClass() != obj.getClass())
          return false;
       return true;
    }
}
```

Il File di Policy per il Server

Usando le Server Socket Factory, il server ha bisogno di installare un SecurityManager e di possedere un file di policy che specifica da quale host accettare le connessioni. Per semplicità accettiamo tutte le connessioni.

```
grant {
   permission java.net.SocketPermission "*", "accept,resolve";
   permission java.net.SocketPermission "*", "connect,resolve";
};
```

Quindi, il server oltre alle classi ha due file: il file di policy "server.policy" e il file con le chiavi "mykey". Invece, il client oltre alle classi ha il file "clientstore". Sia "mykey" che "clientstore" sono stati creati con `keytool`. Per avviare il server ed il Client:

```
> java -Djava.security.policy=server.policy HelloImpl

> java -Djavax.net.ssl.trustStore=clientstore HelloClient
Hello World!
```

Codebase

- Il codebase identifica una locazione da cui è possibile caricare classi in una JVM
- La proprietà `java.rmi.server.codebase` specifica le URL:
  - se client e server sono sullo stesso host, si può usare `file:///`. Altrimenti, si usa `http`, `ftp` o `nfs` e, in questi casi, il server deve rendere disponibili i file tramite questi protocolli.
- Nell'esempio precedente si supponeva che il client possedesse il file `RMISSLClientSocketFactory.class`. In alternativa, il client può scaricare il file class dal server a runtime:
  1. installando un security manager;
  2. specificando un file di policy opportuno;
  3. settando la proprietà `java.rmi.server.codebase` da cui scaricare il file.

Esercizi

Esercizi

- Per prima cosa, tramite keytool creare le chiavi pubbliche/private con i vostri dati (che andranno associate al server) ed estrarre il certificato che va importato in un truststore (sul client). Eseguire uno dei due esercizi seguenti:
  1. effettuare una comunicazione sicura su TCP (come nell'esempio del Web Server) modificando uno degli esercizi assegnati nelle lezioni precedenti in maniera da usare socket sicuri invece dei socket TCP;
  2. modificare l'esercizio sulla votazione RMI in modo che la comunicazione tramite RMI sia sicura.
- In entrambi i casi, lato client deve essere presente solo il file con il truststore (non il file con le chiavi).
- Tutte le password usate (ad es. per il trustore/keystore) NON devono essere inserite nei file sorgenti, ma (ad es.) passate come parametri da riga di comando o inserite a runtime.