Introduzione alla sicurezza in Java

Sicurezza dell'Informazione M

AA 2017-2018

Pasquale Carlo Maiorano Picone
pasquale.maiorano4@unibo.it

Sicurezza in Java

Java è una piattaforma *"sicura"*

Linguaggio nato per:

- *applicazioni multipiattaforma*
- *applicazioni da distribuire attraverso internet*
- *applicazioni da eseguire su diversi dispositivi (cellulari, ...)*

Sicurezza: obiettivo di progetto già dalle prime release

- *security model iniziale*
- *evoluzione continua nelle versioni successive*

Sicurezza e Crittografia

Sicurezza (*computer security*)

applicazione di misure orientate a far sì che determinate informazioni, nel momento in cui vengono elaborate, archiviate o comunicate siano affidabili ed accessibili alle sole entità autorizzate

Crittografia (*cryptography*)

applicazione di tecniche di codifica dell'informazione che la trasformino in un formato decifrabile solo dal destinatario designato

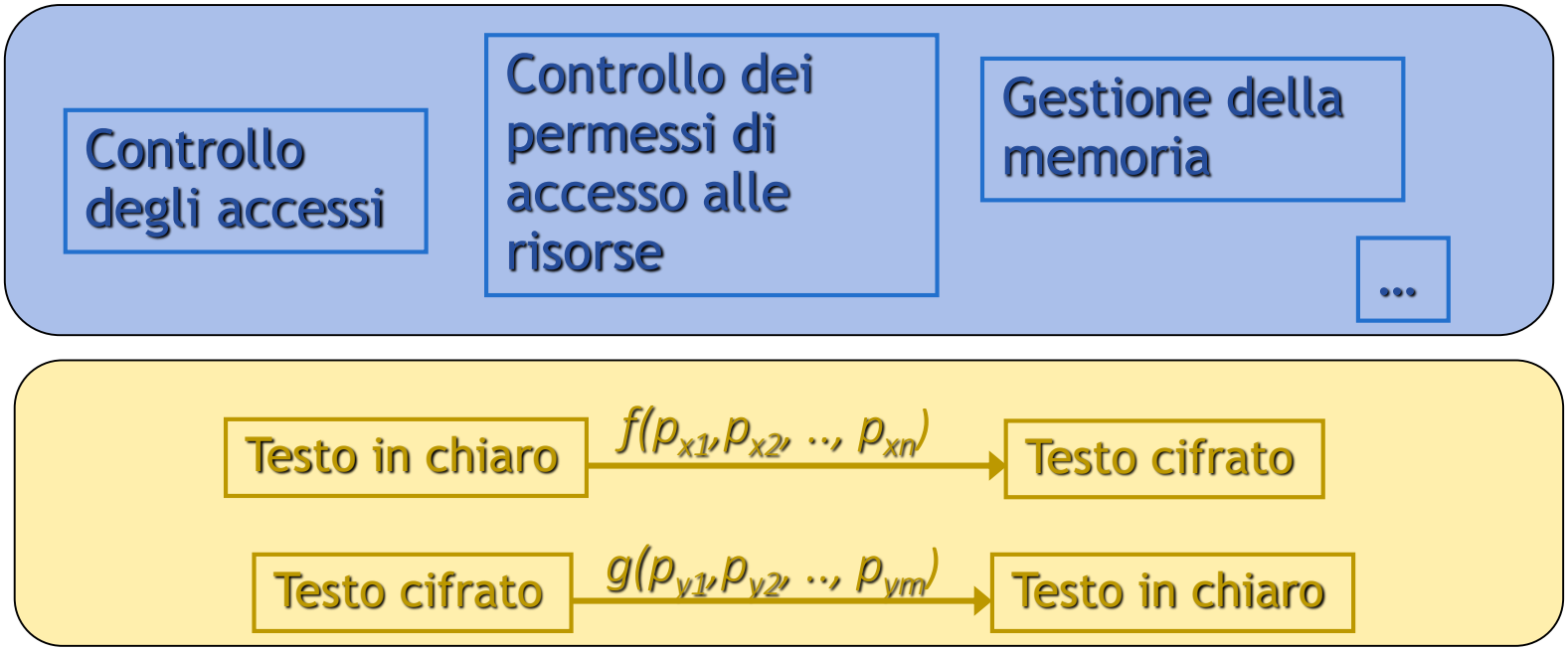
Sicurezza e Crittografia

Sicurezza (*computer security*)

applicazione di misure orientate a far sì che determinate informazioni, nel momento in cui vengono elaborate, archiviate o comunicate siano affidabili ed accessibili alle sole entità autorizzate

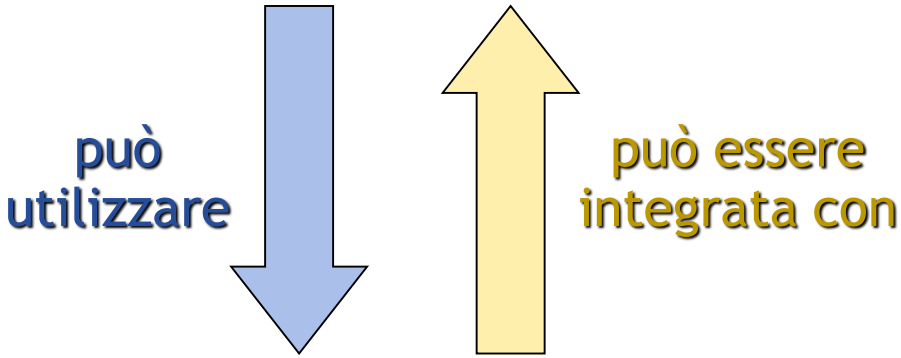
Crittografia (*cryptography*)

applicazione di tecniche di codifica dell'informazione che la trasformino in un formato decifrabile solo dal destinatario designato



Sicurezza e Crittografia

Sicurezza (*computer security*)



Crittografia (*cryptography*)

Piattaforma Java

SDK

piattaforma su cui eseguire applicazioni Java in modo sicuro

API, strumenti e servizi per la sicurezza

framework che consente ad amministratori e programmatori di gestire applicazioni in modo sicuro e di scrivere applicazioni che si appoggiano ad algoritmi, meccanismi e protocolli per la sicurezza

Sicurezza in Java

Meccanismi di controllo per evitare errori causa di comportamenti imprevedibili delle applicazioni

- linguaggio tipizzato
- controllo degli indici negli accessi ad array
- gestione automatica della memoria (garbage collection)

Sicurezza in Java

Livelli di visibilità differenziata applicabili a classi, metodi e campi

- private
- protected
- public
- package

Sicurezza in Java

Verifica, prima dell'esecuzione, della conformità del bytecode alle Java Language Specification

- rispetto del formato e vincoli strutturali imposti per i file .class che la JVM deve interpretare
- assenza di stack underflows / overflows
- nessuna violazione della memoria
- nessun cast non permesso

Sicurezza in Java

Ulteriori meccanismi di controllo a run-time

- Class Loader: gestisce il caricamento e il linking delle classi coinvolte nell'esecuzione all'interno degli opportuni namespace
- Security Manager: valida gli accessi alle risorse di sistema mediati dalla JVM
- modello a SandBox: limita il contesto di esecuzione di codice potenzialmente dannoso

Crittografia e Sicurezza in Java

Dalla versione 1.4 di Java:

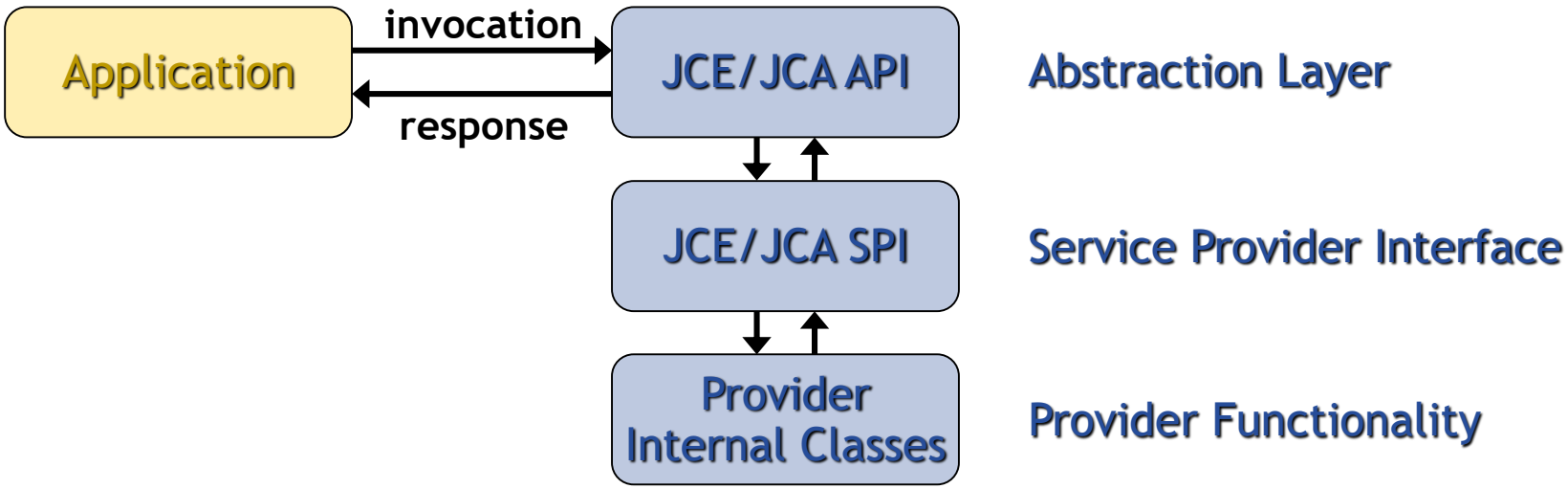
- integrazione in un'unica Java 2 Security Platform di
- *Java Cryptography Architecture*
- *Java Cryptography Extension*
- *Java Secure Socket Extension*
- *Java Authentication and Authorization Service*
- aggiunta dei package
- *Java Certification Path*
- *Java Generic Security Service*

Java Cryptography Architecture

Framework di base per la Crittografia

Provider-based Architecture

- *indipendenza dall'implementazione*
- *interoperabilità (provider e applicazioni non sono strettamente legati)*
- *estendibilità*



Provider-based Architecture

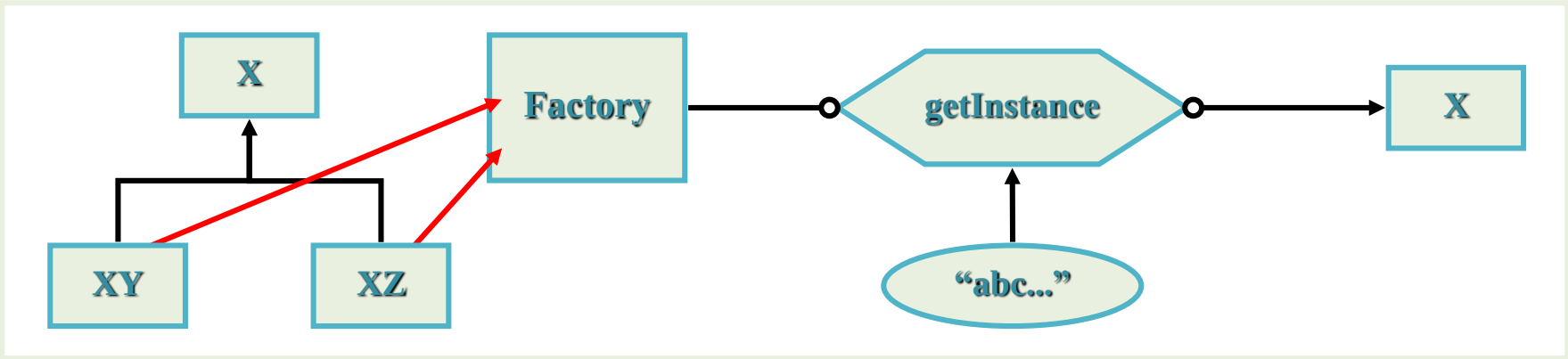
Cryptographic Service Provider

- uno o più package che implementano uno o più servizi crittografici esposti dalla JCA
- più provider installati, anche se di produttori differenti, possono coesistere

- **Responsabilità**
  - un'applicazione richiede un oggetto (`MessageDigest`)...
  - ... che implementa uno specifico servizio (hash calcolato in base all'algoritmo `SHA-1`)...
  - ... e riceve un'implementazione fornita da uno dei provider installati (`BouncyCastle`)

Provider-based Architecture

Pattern Factory



- Implementazione
  - classi astratte di tipo Engine
    - dichiarano le funzionalità offerte
  - classi concrete contenute nel provider
    - implementano le funzionalità dichiarate

Provider-based Architecture

Concretamente

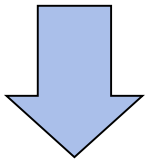
- un oggetto viene istanziato
- *non più con la parola chiave* `new`
- *ma grazie al metodo statico* `getInstance(String nome)`

- **Esempi**
  - `MessageDigest hash = MessageDigest.getInstance("SHA-1");`
  - `KeyGenerator kGen = KeyGenerator.getInstance("TripleDES");`

Provider-based Architecture

Indipendenza dall'implementazione

- più provider possono coesistere nello stesso JRE
- provider diversi possono offrire funzionalità diverse
- provider diversi possono offrire implementazioni diverse delle stesse funzionalità



Vantaggio

- scegliere autonomamente il provider più adatto alle specifiche esigenze

Provider-based Architecture

Scelta consapevole del provider

- il nome del provider viene specificato come parametro all'interno del metodo `getInstance()` utilizzato per istanziare l'oggetto richiesto

- **Esempi**
  - `MessageDigest hash = MessageDigest.getInstance("SHA-1", BouncyCastleProvider.PROVIDER_NAME);`
  - `KeyGenerator kGen = KeyGenerator.getInstance("TripleDES", BouncyCastleProvider.PROVIDER_NAME);`

Installazione di un CSP

Installazione statica

- inserire l'archivio jar che implementa il provider tra le librerie esterne del JRE
- *<JAVA_HOME>\jre\lib\ext*
- modificare il file **java.security** che si trova nella directory *<JAVA_HOME>\jre\lib\security* aggiungendo il nome del provider come ultimo elemento della lista che ha il formato
- *security.provider.n=provider*

- **Esempio**
  - `security.provider.7 = org.bouncycastle.jce.provider.BouncyCastleProvider`

Installazione di un CSP

Installazione dinamica

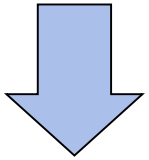
- il provider va importato all'interno dell'applicazione (jar, gradle, maven)
- il metodo `addProvider(Provider provider)` della classe `java.security.Security` consente di aggiungere il provider desiderato a run-time

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

Provider-based Architecture

Nessuna preferenza nella scelta del provider

- il nome del provider non viene specificato come parametro all'interno del metodo `getInstance()` utilizzato per istanziare l'oggetto richiesto



- Ricerca ordinata per preferenze
  - file *java.security*
  - la ricerca continua finché non si trova il primo provider che implementa la funzionalità richiesta

Provider-based Architecture

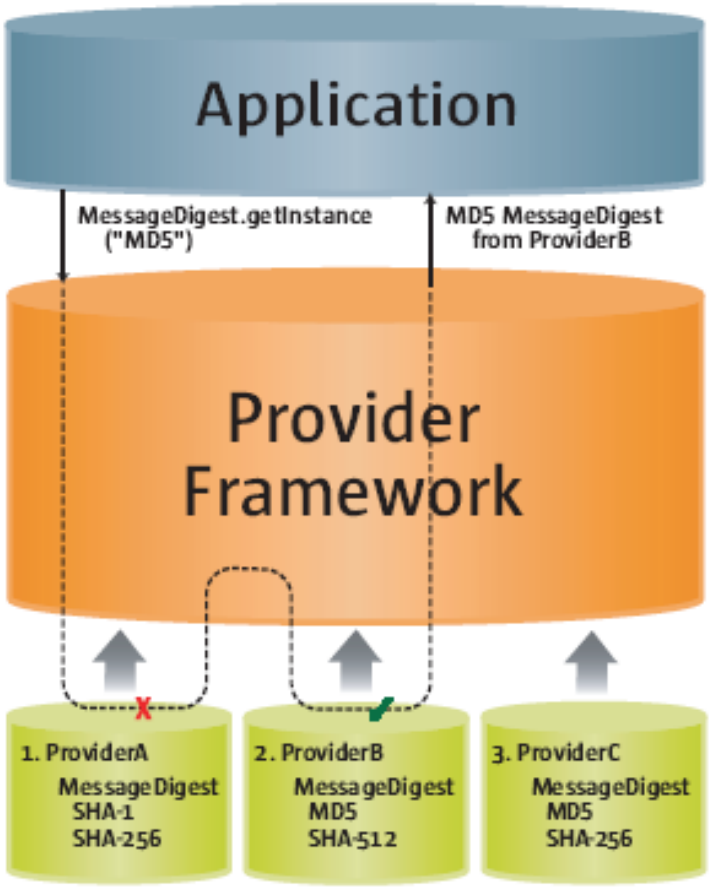


Figure 1 – Provider searching

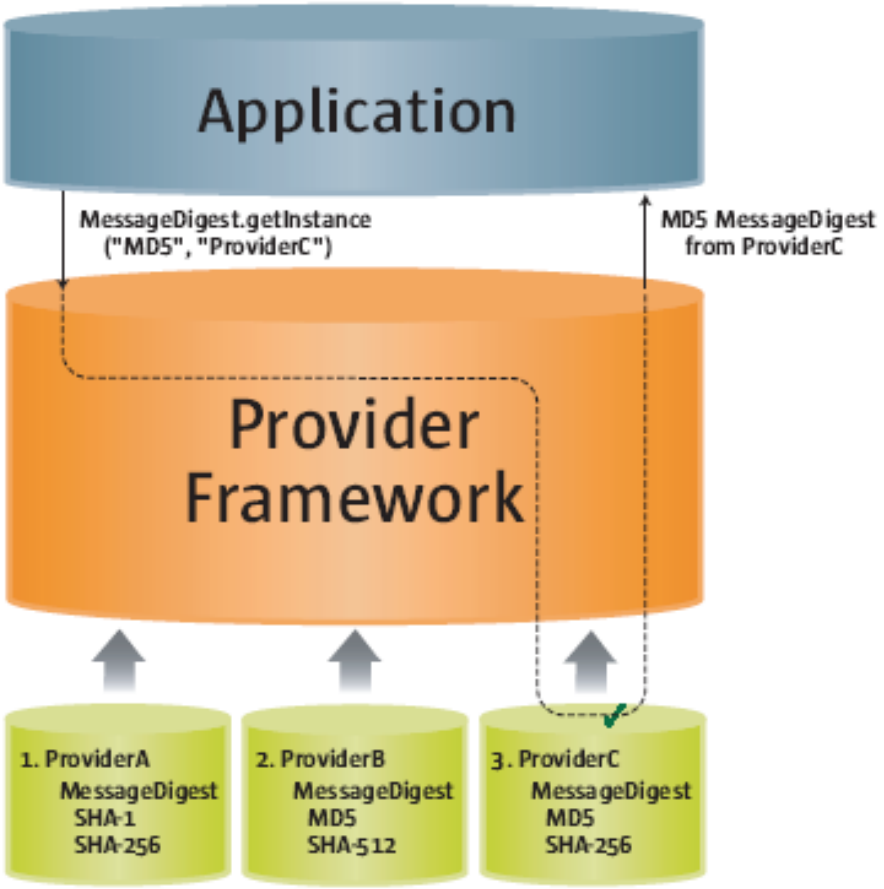


Figure 2 – Specific provider requested

Provider “Sun”

Integrato nel JDK

Include implementazioni di:

- algoritmo DSA
- *KeyPairGenerator per DSA*
- *AlgorithmParameter per DSA*
- *AlgorithmParameterGenerator per DSA*
- *KeyFactory per DSA*
- algoritmi SHA-1 e SHA-2 (da java 9 anche SHA-3, keccak)
- algoritmo SHA256PRNG (SecureRandom)
- CertificateFactory per X.509 e CRLs
- KeyStore proprietario JKS

Java Cryptography Extension

Integrato nel JDK

Implementa in modo completo le funzionalità di cifratura e decifrazione dichiarate dalla JCA

Offre il supporto all'utilizzo di

- cifrari simmetrici a blocchi e a flusso
- cifrari asimmetrici
- cifrari con password

... applicabili a

- Dati
- SerializableObjects
- I/O stream

Java Cryptography Extension

Estende le funzionalità di cifratura e decifrazione dichiarate dalla JCA

- JCE API: package javax.crypto

Classi principali

- Cipher
- *CipherInputStream / CipherOutputStream*
- KeyGenerator
- SecretKeyFactory
- SealedObject
- Mac

Java Cryptography Extension

Integrato nel JDK

Implementa ed estende le tecniche crittografiche definite dalla JCA

- provider “SunJCE”
- *algoritmi di cifratura (AES, DES, TripleDES, PBE, ...)*
- *modalità di cifratura (ECB, CBC, OFB, CFB, CTR, PCBC)*
- *padding (NoPadding, PKCS5PADDING)*
- *facilities per la conversione delle chiavi da oggetti Java in array di byte e viceversa*
- *KeyGenerator*
- *algoritmi MAC*

Provider BouncyCastle

Non integrato nel JDK (va installato/importato)

Implementa ed estende le tecniche crittografiche definite dalla JCA e dalla JCE

- *implementazione alternativa degli algoritmi di cifratura (AES, DES, TripleDES, PBE, ...)*
- *algoritmi di cifratura asimmetrica (ECIES, RSA, ElGamal)*
- *algoritmi di firma digitale*
- *padding (None, PKCS1PADDING)*
- *facilities per diversi protocolli di KeyAgreement*
- *facilities per utilizzare cifrari simmetrici a flusso e a blocchi*
- *...*

Java Secure Socket Extension

Integrato nel JDK

Implementa ed estende le tecniche crittografiche definite dalla JCA a livello di rete

- Provider “SunJSSE”
- *KeyManager (SunX509)*
- *KeyStore (PKCS12)*
- *protocolli SSL/TLS (SSLv3, TLSv1)*
- *Trust manager (PKIX)*

Java Secure Socket Extension

Offre funzionalità per

- autenticazione
- protezione dell'integrità dei dati
- protezione della riservatezza dei dati

Opera a livello di rete

- protocolli per comunicazioni sicure
- *Secure Socket Layer v.2.0 e v.3.0 (deprecati)*
- *Transport Layer Security v.1.0, v.1.1, v.1.2*

Java Secure Socket Extension

Le API JSSE (package javax.net e javax.net.ssl) estendono

- javax.crypto (JCE)
- java.security (JCA)
- java.net

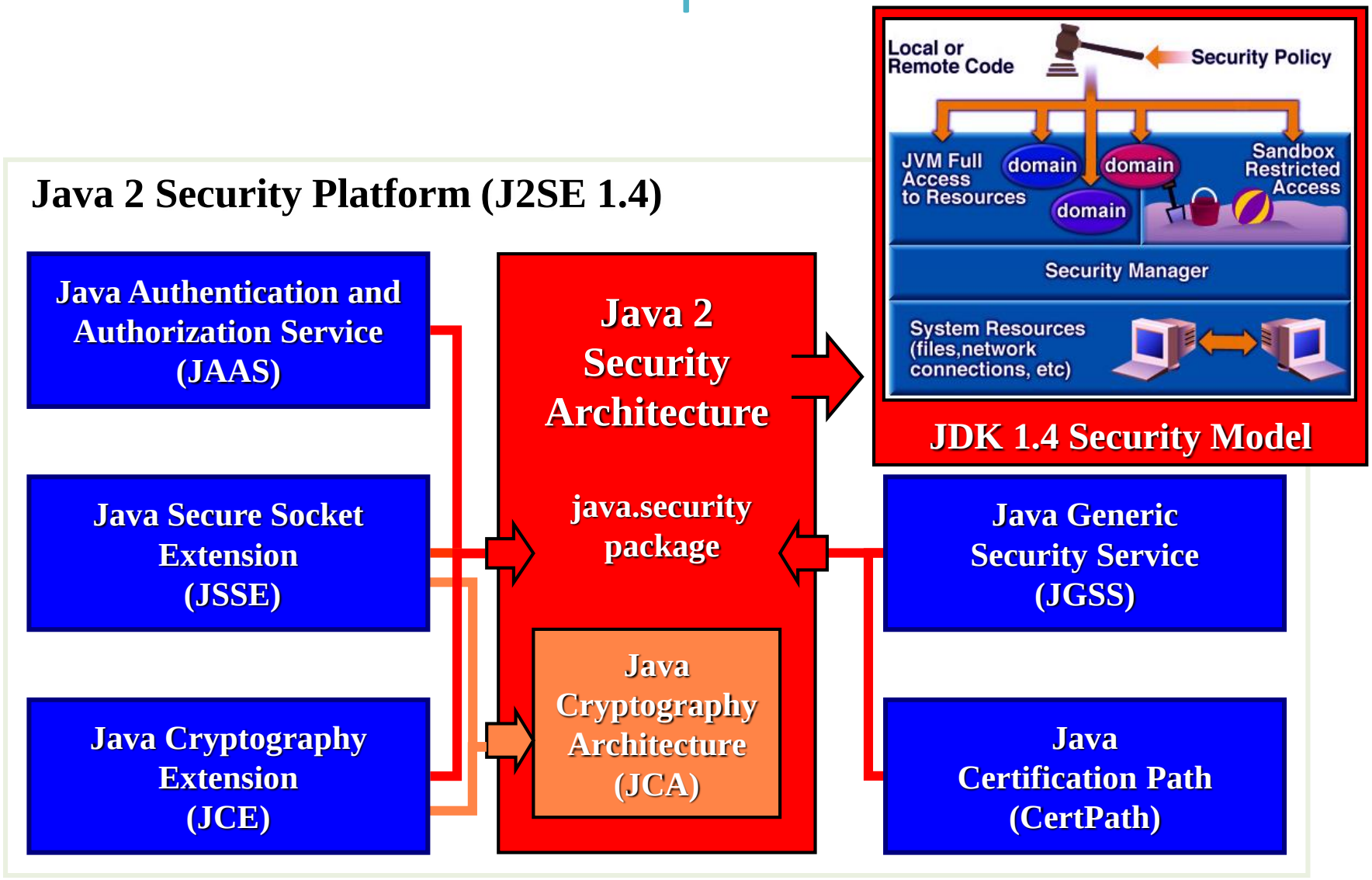
Java Authentication and Authorization Service

- Integrato nel JDK
- Offre servizi per
  - autenticazione: identificazione sicura
  - autorizzazione: verifica dei permessi
- E' composto da
  - un framework di autenticazione estendibile
  - una serie di moduli dedicati alle singole tecniche di autenticazione
    - *verifica username / password*
    - *verifica di caratteristiche biometriche*
    - *...*

Java Certification Path

- Integrato nel JDK
- Offre interfacce ed engine classes per gestire
  - certificati
  - certification path
- Le funzionalità principali riguardano
  - Basic Certification Path
  - Certification Path Validation
  - Certification Path Building
  - Certificate / CRL Storage

Java Generic Security Service

- Integrato nel JDK
- Ponte Java – Generic Security Service API
- GSS API: progettate dall'IETF per offrire un accesso uniforme a servizi di autenticazione e comunicazione peer-to-peer che adottano meccanismi di sicurezza
- I principali servizi offerti riguardano
- autenticazione
- riservatezza e integrità dei messaggi
- sequencing dei messaggi protetti
- replay detection

Architettura Java per la sicurezza