**DECRIPTRARE CON JAVA KEYSTORE (JKS)**

Il meccanismo per il quale si basa JKS è la crittografia RSA o chiave asimmetrica. In sostanza esistono due chiavi, una pubblica ed una privata. La chiave pubblica è conosciuta da tutti, la chiave privata la conosce solo la persona che vuole decriptare il testo.

Il testo in chiaro può essere criptato da chiunque conosca la chiave pubblica, mediante proprio l’utilizzo di essa.

Il testo criptato invece può essere decifrato solo da chi conosce la chiave privata.

Il file con estensione jks, ossia il keystore, contiene al suo interno la chiave privata, dunque se non si ha fisicamente questo file è impossibile effettuare la decrifratura, ma al tempo stesso un malintenzionato in possesso del jks lo può fare! Per ovviare a ciò, jks viene protetto da password.

Se in java volessi cifrare un testo in chiaro mediante questa funzionalità, dovrei prima creare un JKS (che contiene anche la chiave pubblica), criptare il testo e settarlo tranquillamente nel mio codice java. I metodi che lo utilizzano però, devono a avere a disposizione un metodo che sia in grado di decifrarlo, utilizzando il file jks creato prima. Di seguito, le istruzioni:

**1) Creazione JKS**

Da prompt windows digitare la seguente linea di comando:

*keytool -genkey -v -keystore [name.jks] -alias [alias] -keyalg RSA -keysize 2048 -validity [giorni]*

Sostituire parentesi quadre e contenuto rispettivamente con:

* Nome che voglio dargli con estensione jks
* Alias che voglio dargli
* Validità in giorni, consiglio 365000 ossia mille anni

Il comando genererà il keystore nel percorso attuale del cmd.

**2) Generazione file di configurazione**

I metodi e il tool che presenterò in questa guida, utilizzano un file di configurazione XML che definisce le caratteristiche del keystore. Sarà quindi opportuno crearlo:

*<?xml version = "1.0"?>*

*<class>*

*<keystore name = "jwtsJKS-keystore">*

*<alias>jwtsJKS</alias>*

*<extension>jks</extension>*

*<filePName>codp</filePName>*

*</keystore>*

*<keystore name = "propEncJwtsJKS-keystore">*

*<alias>propEncJwtsJKS</alias>*

*<extension>jks</extension>*

*<filePName>codp</filePName>*

*</keystore>*

*<keystore name = "bearer-keystore">*

*<alias>bearerJKS</alias>*

*<extension>jks</extension>*

*<filePName>codp</filePName>*

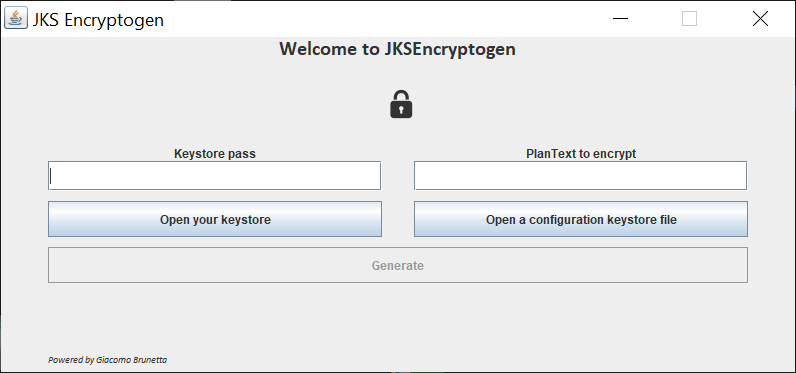
*</keystore>*

*</class>*

Il file xml (comunque un .txt ma formattato come xml) può contenere la descrizione di uno o più jks da gestire. E’ necessario definirne il nome datogli senza estensione (keystore name), l’alias, exstension (appunto, l’estensione), filePName. Il filePName è facoltativo, indica il nome (compreso di estensione) del file che contiene password per aprire il keystore, dunque si può anche lasciare vuoto se si decide di inserirla direttamente nel codice. Attenzione! Se lo si utilizza, la password deve essere criptata in un altro algoritmo conosciuto dal codice sorgente, magari in Aes, in modo da venir decriptata da esso una volta prelevata. Alternativamente appunto, si setta “a mano” all’interno del sorgente stesso.

**3) Generazione testo cifrato**

Per generare il testo cifrato si potrà utilizzare il tool apposito JKSEncryptogen.exe [](JKSEncryptogen/JKSEncryptogen.exe):



Bisogna inserire la password del keystore, il testo in chiaro da cifrare, il jks il file di configurazione creato nel punto 2 (non andrà a leggere la password da codPName).

Una volta selezionati, il pulsante “Generate” sarà cliccabile e genererà il testo cifrato che verrà copiato automaticamente tra gli appunti.

**4) Lavorare in Java con il testo cifrato**

Per prima cosa bisogna inserire il jks, il file di configurazione ed eventualmente anche il file contenente la password del keystore, tra le resources del progetto Java. La stringa che verrà manipolata sarà quella criptata, al momento giusto però dovrà essere decifrata con questi file in resources.

Una classe pubblica di Utility avrà il seguente metodo publico (meglio uno per ogni jks per sicurezza), da richiamare ogni qualvolta servisse il testo in chiaro appena criptato.



Si sta presupponendo che la password cifrata venga scritta in chiaro così, ma meglio metterla in un application.properties. Il nome keystore è il nome del file jks senza estensione.

Il su visto metodo pubblico ne richiama il ne richiama il privato getJKS.

Se non utilizziamo il file per salvare la password del keystore (comunque a sua volta criptata, in questo caso in Aes), evitiamo il codice trascritto tra “-----“ e scriviamo direttamente:

*keystore.load(keystoreIn, “passwordDelKeystore”.toCharArray());*

Di seguito il metodo getJKS:

Il metodo però ne utilizza un altro privato, il *retrieveKeystoreDetailsDomParser*, che serve a leggere (parsing) il file formattato in xml, dunque le configurazioni del jks:



Un altro metodo utilizzato è quello che va a prendere il file dalle Resources, il getResourceFile:



E ovviamente il metodo che si occupa della decifratura RSA:



NB: per un’eventuale facile analisi delle eccezioni, gestirle piuttosto che lasciarle in Throws.