

Dokumentation
XML-Signaturen im Business Workflow

Hendrik Dettmer und Hyun-A Cho
Betreuer: M. Psarros

7. November 2005

Inhaltsverzeichnis

1	Aufgabenstellung	3
2	Programmübersicht	4
3	Aufbau des Programms	5
3.1	Verwendete Pakete	5
3.2	Struktur	5
3.3	Beschreibung der Funktionen	6
3.3.1	GUI2.java	6
3.3.2	signieren.java	7
3.3.3	valid.java	9
4	Anwendung des Programms	10
4.1	Die grafische Oberfläche	10
4.2	Programmanleitung	10

1 Aufgabenstellung

In einen virtuellen Betrieb durchläuft eine Bestellung verschiedene Instanzen. Dieser Ablauf wird in der Rechnung abgebildet wo die verschiedene Personen für verschiedene Teile davon zuständig sind:

Der Vertriebsbeauftragter ist zuständig für die Richtigkeit der Kundenadresse, des Preises und der Anzahl der ausgelieferten Artikel. Der BearbeiterSW-Versand ist zuständig für die Auslieferung der Bestellung. Die Rechnungsstellung ist schließlich zuständig für die Richtigkeit der Rechnung. Jeder dieser Teilnehmer ist nicht für den ganzen Bestellvorgang zuständig und deswegen wird er sich auch weigern die Rechnung als ganzes zu signieren. Da bietet XML-Sicherheit ein effizientes Werkzeug um so einen Workflow zu werkstelligen: durch das digitale Signieren von Teilkomponenten werden die Aufgabenbereiche am besten in das Programm abgebildet, jeder Teilnehmer unterschreibt nur den Teilbereich der Rechnung für die er die Verantwortung trägt. In diesem Projekt wird eine Java GUI Applikation implementiert, die diese virtuelle Rechnung durch die verschiedenen Instanzen signiert.

Die verschiedene Rollen und der zu signierende Umfang können statisch im Programm eingebunden werden, für die asymmetrische Schlüssel können einfache Javastores benutzt werden. Es wird die Apache XML-Sicherheitsbibliothek benutzt, als Entwicklungsumgebung wird Eclipse (verfügbar für Windows und Linux) empfohlen.

Optionale Zusatzaufgabe: Das XML Dokument wird in jeden Schritt zuerst validiert und auf seine Richtigkeit überprüft. Bei Fehler wird eine entsprechende Fehlermeldung ausgegeben.

2 Programmübersicht

In unserem Beispielprogramm existieren drei fiktive Personen mit folgenden Namen: Vertriebsbeauftragter, Bearbeiter-SW-Versand und Rechnungsstellung. Diese drei Personen sind berechtigt Teile des XML-Dokumentes zu signieren. Dieses XML-Dokument ist in unserem Fall eine Rechnung über zwei Produkte.

Es existiert ein RSA-Schlüssel, der in einer Keystore-Datei abgespeichert ist. Wenn eine befugte Person einen Teil der XML-Datei signieren möchte, so werden diese Daten mit dem privaten RSA-Key verschlüsselt und in die XML-Datei eingebunden. Die Verifizierung erfolgt durch Entschlüsseln mit dem öffentlichen Schlüssel und Vergleichen der Ursprungsdaten mit den entschlüsselten Daten.



3 Aufbau des Programms

3.1 Verwendete Pakete

- Pakete für die Schlüsselverwaltung:
`java.security.*`
- Pakete zur Generierung der Signatur:
`org.apache.xml.*`
- Pakete für XML-Objekt-Erzeugung, -Parsing und XSD-Validierung:
`javax.xml.parsers.*`, `org.xml.sax.*`, `org.w3c.dom.*`
- Pakete, um Dateien einzulesen und abzuspeichern:
`java.io.*`
- Pakete für die grafische Oberfläche:
`java.awt.*` und `javax.swing.*`

3.2 Struktur

Das gesamte Programm besteht aus drei Klassen:

- `GUI2.class`: Diese Klasse enthält die grafische Oberfläche, die Abfrage und der Vergleich der Benutzernamen und das Laden der XML-Datei.
- `signieren.class`: Diese Klasse lädt die XML-Datei und die Key-Store-Datei. Hier ist festgelegt, welche Teile der XML-Datei von einer Person signiert werden. Die Daten werden signiert und angehängt.
- `valid.class`: Diese Klasse öffnet die Datei und validiert diese, indem die ursprünglichen Daten mit den entschlüsselten Daten verglichen werden.

3.3 Beschreibung der Funktionen

3.3.1 GUI2.java

1. Der Deklarationsteil

Die Klasse GUI2 erbt von der Klasse **Frame** und implementiert folgende Abhörer: **TreeSelectionListener** und den normalen **ActionListener**.

Der Deklarationsteil enthält hauptsächlich Elemente der grafischen Oberfläche. Die Variablen *dateiName* und *dateiPfad* werden an die Funktionen *signieren.java* und *valid.java* weitergegeben.

Die Variable *Name* enthält die Eingabe des Textfeldes. Es wird automatisch die Funktion **main** aufgerufen, die wiederum den Konstruktor GUI2 aufruft.

2. Der Konstruktor

Die Knöpfe, Labels, Textfelder werden instanziiert und deren Aussehen und Ausrichtung werden festgelegt. Sie werden in den Containern **labelPanel**, **buttonPanel** und **treePanel** zusammengefasst. Diese werden durch den Layouttyp *borderLayout* angeordnet. Desweiteren wird noch der **WindowListener** implementiert, damit das Programm beendet wird, wenn "X" gedrückt wird. Der Konstruktor enthält noch weitere Unterfunktionen.

valueChanged(TreeSelectionEvent e): Diese Funktion aktualisiert den Baum, sobald Veränderungen an ihm vorgenommen wurden.

actionPerformed(ActionEvent e): Diese Funktion reagiert auf das Anklicken eines der vier Buttons. Wenn der Button *XML auswählen* geklickt wird, so wird die Funktion **doFile()** aufgerufen. Der Button *Valdieren* ruft **valid.main(dateiPfad, dateiName, 0)** auf und die Exception *e2* wird abgefangen. **Beenden** ruft **System.exit(0)** und beendet das Programm. Wenn Button *Signieren* gedrückt wird, so wird zuerst der Name, der in dem Textfeld *textEingabe* steht an die Variable *Name* übergeben und es wird überprüft, welche der drei berechtigten Personen signieren möchte. Falls die Person berechtigt ist, so wird die Funktion **starteSignieren(b)** aufgerufen und es wird an die Variable *b* übergeben, welche Person signieren möchte. Sollte die Person nicht berechtigt sein, so steht im Texteingabefeld "Bitte geben sie einen gültigen Namen ein". Als letztes wird die Funktion **dateiladen()** aufgerufen, damit die XML-Datei aktualisiert wird.

`starteSignieren(int b)`: In dieser Funktion wird lediglich `signieren.main(dateiPfad, dateiName, a)` aufgerufen und eine Exception abgefangen.

`doFile()`: Diese Funktion ruft den Standarddialog zum Öffnen einer Datei auf. Falls eine Datei ausgewählt wurde, füllt diese Funktion die Variablen *dateiPfad* und *dateiName*. Anschließend wird die Funktion `dateiladen()` aufgerufen. Außerdem wird noch eine Exception abgefangen.

`dateiladen()` throws Exception: Zunächst wird die virtuelle Wurzel *root* erstellt und an den Baum *tree* angehängt. Um diesen Baum zu erzeugen wird die ausgewählte Datei zunächst geparkt. Danach wird versucht die XML-Datei gegenüber der XSD zu validieren, um die Korrektheit der Struktur zu gewährleisten. Hierbei ist jedoch anzumerken, dass unsere XSD-Datei noch nicht auf unsere XML-Datei eingestellt wurde und somit jedesmal unsere XML-Datei als inkorrekt angezeigt wird, obwohl diese an sich korrekt ist. Die Funktion `SimpleErrorHandler implements ErrorHandler` überprüft, ob bei der Validierung der XML-Datei gegenüber der XSD-Datei ein *Error*, *Fatal Error* oder ein *Warning* geworfen wurde.

3.3.2 signieren.java

Klasseninterne Variablen

- *static Document doc*:
Hierin wird das geparkte Dokument abgespeichert
- *static String BaseURI*:
Diese Variable enthält den Pfad der XML-Datei
- *static Transforms transforms, transforms2, transforms3*:
In diesen Variablen wird zwischengespeichert, wie mit der XML-Datei verfahren werden soll.
- *static X509Certificate cert*:
In diesem Zertifikat stehen Information über den RSA-Schlüssel wie z.B. der Public- und der Private-Key
- *static PrivateKey privateKey*:
enthält den private Key

- *public static boolean RS, BSWV, VB = true:*

In diesen Variablen geben Auskunft darüber, ob ein Teil der Datei bereits signiert wurde oder nicht.

`main(String Pfad, String datei, int Id) throws Exception`

Nach der Initialisierung der XML-Funktion wird die Keystore-Datei geladen und der Private-Key wird extrahiert. Anschließend wird die XML-Datei geöffnet und geparkt. `transforms = new Transforms(doc)` bereitet das XML-Dokument so vor, dass es später geparkt werden kann. Man muss das Dokument insgesamt drei mal vorbereiten, da es auf insgesamt drei verschiedene Arten signiert werden kann. In die Variable *cert* werden die Schlüsseldaten und die Daten, die in der Keystore-Datei unter *certificateAlias* gespeichert waren, gemäß dem X509-Standard für Zertifikate, hineingeschrieben. Jetzt wird die Funktion `überprüfe()` ausgeführt. Anschließend wird die XML-Datei gelöscht und die verschiedenen Funktionen zum Signieren werden aufgerufen, abhängig davon, welche Person signieren möchte und die neue XML-Datei wird geschrieben.

`static boolean ueberpruefe()` Diese Funktion stellt fest, wie viele Signaturen existieren, und wer bereits signiert hat, denn wer bereits signiert hat, sollte nicht noch einmal signieren, um das Dokument nicht unnötig aufzublähen.

`static void Erste() throws Exception` Dies ist die eigentliche Signaturfunktion.

`XMLSignature sig = new XMLSignature(doc, BaseURI, XMLSignature.ALGO_ID_SIGNATURE_RSA)` Erstellt ein neues Signaturelement. Anschließend wird die ID desjenigen angehängt, der signiert. Anschließend wird festgelegt, wo die Signatur eingefügt werden soll. Danach wird festgelegt, welche Knoten von dieser Person signiert werden sollen. Dies ist der Hauptgrund, weshalb die Funktion in drei unterschiedlichen Ausführungen (`Erste()`, `Zweite()` und `Dritte()`) existiert. Anschließend wird der komplette Text transformiert. Beispielsweise werden Sonderzeichen und bestimmte Leerzeichen entfernt oder umgewandelt.

`sig.addDocument(' ' , transforms, Constants.ALGO_ID_DIGEST_SHA1);` erstellt den Hashwert der Daten. Der Hashwert wird mittels SHA1 gebildet. Desweiteren wird *transforms* übergeben, worin steht, welche Umformungen an den eigentlichen Daten vorgenommen wurden.

`sig.addKeyInfo(cert.getPublicKey())` hängt den öffentlichen Schlüssel an, damit die Daten verifiziert werden kann.
`sig.sign(privateKey)` die Daten in *sig* werden signiert.
Außerdem werden Ausnahmen abgefangen, die die Funktion `sign()` werfen könnte.

3.3.3 valid.java

Anmerkung: Diese Klasse ist Bestandteil des optionalen Teils dieses Projekts. Diese Klasse wurde lediglich zu Testzwecken erstellt um zu überprüfen, ob korrekt signiert wird.

`static void main(String Pfad, String datei, int Id) throws Exception`
Anfangs werden die XML-Funktionen initialisiert. Anschließend wird die Datei geöffnet und geparst. Die XML-Daten, die Signatur und der öffentliche Schlüssel werden aus der Datei geholt. Anschließend wird überprüft ob die Signatur gültig ist, in dem der Hashwert über die Klardaten gebildet wird und mit dem Hashwert verglichen wird, der mit dem öffentlichen Schlüssel entschlüsselt wurde.

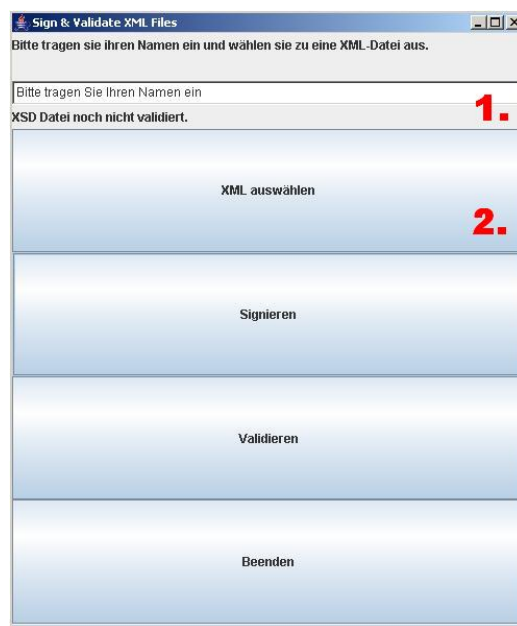
4 Anwendung des Programms

4.1 Die grafische Oberfläche

Die grafische Oberfläche besteht aus einem Textfeld und vier großen Buttons, deren Funktion noch näher erläutert wird.

4.2 Programmanleitung

1. Um eine Datei zu signieren, muss man sich lediglich von oben nach unten “durcharbeiten”. Zuerst muss der Name der Person in das Textfeld eingegeben werden, die eine Datei signieren möchte. In unserem Fall muss einer der drei folgenden Namen eingegeben werden:
 - Vertriebsbeauftragter
 - Rechnungsstellung
 - Mitarbeiter-SW-Versand
2. Danach wird die zu signierende XML-Datei ausgewählt. Dies geschieht durch Anklicken des Buttons *XML auswählen*. Es erscheint ein Standard-Dialogfenster, in dem man durch seine Dateien browsen und die gewünschte Datei öffnen kann. Wenn die XML-Datei geöffnet wurde, erscheint ein zusätzlicher Bereich, in dem die XML-Datei zur Ansicht als Baum dargestellt wird.



3. Durch Auswählen des Buttons *Signieren* werden die Teile des Dokumentes signiert, die für die Personen vorgesehen sind.
4. Der Button *Validieren* (optionale Zusatzaufgabe) überprüft grundsätzlich die Korrektheit der Signatur. Die Ausgabe erfolgt zur Zeit durch `System.out.println` auf der Kommandokonsole.
5. Der Button *Beenden* beendet das Programm, ebenso wie das “X” am oberen rechten Rand des Fensters.

