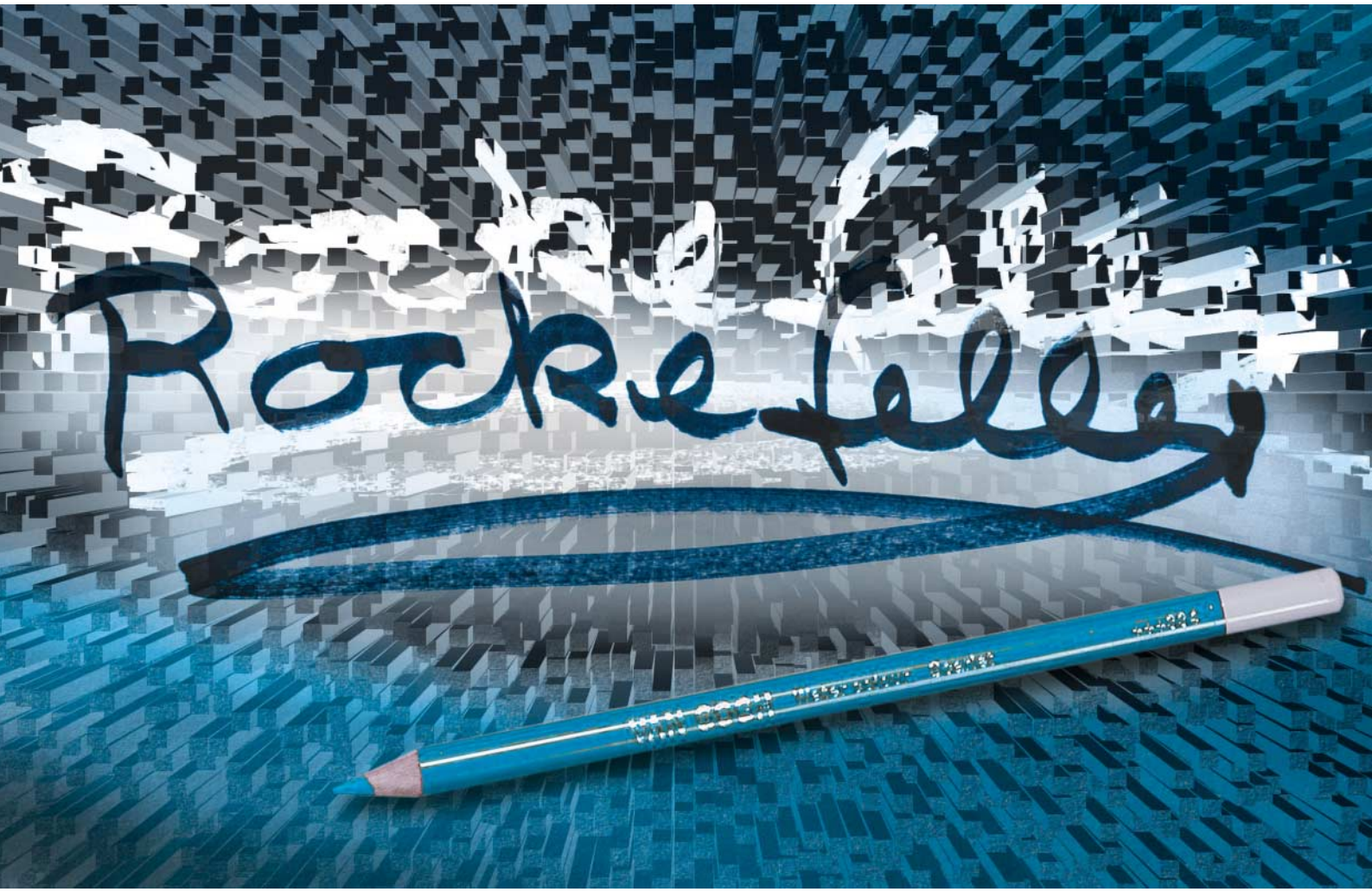


Digitales Signieren mit XML

Unterschriftsreif

Michael Merz, Michael Häusler, Michel Wichers

Wenn es um inhalts- und herstellerunabhängiges Signieren von elektronischen Verträgen oder Geschäftsdaten geht, ist XML beinahe so etwas wie eine „Wunderwaffe". Lediglich die nicht ganz übersichtliche Handhabung steht derzeit noch einem breiten Einsatz entgegen.

Vom Siegeszug der Extensible Markup Language XML durch die Welt der strukturierten Daten bleiben auch das Verschlüsseln und Signieren derselben nicht verschont. Da elektronische Signaturen ebenfalls strukturierte Daten sind, liegt es nahe, sie selbst als XML-Strukturen zu repräsentieren. Wenn man aber schon „in XML denkt", so haben wohl die Standardisierer gedacht, dann sollte man auch alle Möglichkeiten dieser Sprache ausschöpfen und elektronische Signaturen für den täglichen Gebrauch nutzbar machen.

Elektronisches Signieren ist im Grunde genommen ein alter Hut: Seitdem Rivest, Shamir und Adleman vor etwa einem Vierteljahrhundert ihren RSA-Algorithmus patentieren ließen, hat sich am Grundprinzip der elektronischen Signatur nicht viel geändert: Der Trick liegt im asymmetrischen Verschlüsselungsverfahren, das heißt, was mit dem einen Schlüssel verschlüsselt wird, lässt sich nur von einem zweiten wieder in den Klartext zurückverwandeln. Diese beiden Keys wurden zuvor paarweise erzeugt.

Man bestimmt den einen der beiden Schlüssel als privaten, den anderen als öffentlichen Schlüssel. Der private ist – natürlich – geheim, das heißt, man kann davon ausgehen, dass ein mit ihm verschlüsseltes Dokument auch tatsächlich von der Person stammt, welcher der private Schlüssel zugeordnet ist. Nachweisen lässt sich das, indem man das Dokument mit dem öffentlichen Schlüssel wieder im Klartext her-

stellt. Dazu muss man wiederum sicher sein, tatsächlich über den „richtigen“ öffentlichen Schlüssel zu verfügen.

Die mit technisch-organisatorischen Maßnahmen realisierte fälschungssichere Zuordnung von privaten und öffentlichen Schlüsseln zu Personen ist Aufgabe der PKI (Public Key Infrastructure).

Eine elektronische Signatur ist nun der mit dem privaten Key verschlüsselte Hash-Wert eines Originaldokuments. Wenn der Sender einer Nachricht diesen anhängt, kann der Empfänger seinerseits den Hash-Wert aus dem Dokument generieren, die Signatur mit dem öffentlichen Key entschlüsseln und die beiden Hash-Werte vergleichen. Sind diese gleich, so liegt der Beweis vor, dass das Dokument weder verfälscht noch von einer anderen Person signiert wurde.

Viele Wege führen zum Ziel

Soweit die graue Theorie. In der Praxis haben sich inzwischen verschiedene Standards herausgebildet, wie man diesen Algorithmus und die für ihn erforderlichen Datenstrukturen umsetzen kann. Dabei kommt den PKC-Standards (Public Key Cryptographic Standards) eine besondere Bedeutung zu (www.rsasecurity.com/rsalabs/pkcs): Sie legen verschiedene Aspekte fest, zum Beispiel grundsätzliche Ablaufbeschreibungen beim Verschlüsseln und Signieren (PKCS#1, RFC 2437), die Repräsentation verschlüsselter/signierter Daten als Binärobjekt (PKCS#7, RFC 2315) oder die Strukturierung eines privaten Schlüssels mit Metainformationen (PKCS#8).

In der praktischen Umsetzung sind PKCS-Objekte separate Dateien, die man an eine E-Mail oder an ein Dokument anhängt. Dies führt zu gewissen Nachteilen, da authentifizierte Kommunikation immer die Verwaltung zusammengehöriger Dateien erfordert und somit in vielen Fällen sogar ein aufwendiges Dokumenten-Managementsystem. Will man beispielsweise im Nachhinein ein archiviertes Dokument auf seine Authentizität überprüfen, sind unter Umständen das Dokument, seine Signatur und gegebenenfalls verschiedene Zertifikate zu ermitteln und zu verarbeiten.

Grundsätzlich ist es möglich, alle notwendigen Informationen in einem PKCS-Objekt zusammenzufassen. Da es sich hierbei um eine Binärstruktur handelt, lässt sich das Dokument in dem Fall jedoch nicht mehr ohne spezialisierte Anwendungen nutzen.

Schöner wäre es, wenn man zusammenhängende Objekte im Zusammenhang speichern und trotzdem mit den Nutzdaten arbeiten könnte. Dazu benötigt man jedoch eine Strukturierungsmöglichkeit für Inhalte, die von diesem beziehungsweise von seiner Semantik unabhängig ist. Der Hersteller könnte etwa innerhalb von Word-Dateien einen Bereich definieren, in dem man Signaturen ablegt, und der nicht zum eigentlichen (signierten) Inhalt des Dokuments zählt – das würde aber nur für den Dokumententyp eines Hersteller genügen. Ähnliches gilt für Dokumente im Portable Document Format PDF, das in der Tat seit Acrobat 5.1 das Einbetten elektronischer Signaturen vorsieht. Obwohl mit dem PD-Format die Abhängigkeit von Inhalt und Plattform schon reduziert ist, handelt es sich noch um ein herstellerspezifisches Produkt, mit signierten Binärobjekten. Damit bleibt letztlich nur XML als hersteller- und inhaltsunabhängige Auszeichnungssprache.

W3C-Standard für XML-Signaturen

Alles begann im April 1999 am MIT, als Tim Berners-Lee XML- und Kryptographieexperten dazu aufrief, gemeinsam einen W3C-Standard für das Signieren von XML-Dokumenten aus der Taufe zu heben. Es dauerte bis Februar 2002, um die daraus entstandene Empfehlung des W3C reifen zu lassen (siehe auch www.w3.org/TR/xmldsig-core/). Seitdem haben unterschiedliche Organisationen, zum Beispiel die Apache Group, IBM oder die Universität Graz, den Standard umgesetzt. Einige dieser Implementierungen sind kommerziell, andere frei verfügbar.

Einer XML-Signatur ist immer mindestens eine Ressource zugeordnet, das heißt ein XML-Baum oder beliebige Binärdaten, auf die ein XML-Link verweist. Beim XML-Baum muss sichergestellt sein, dass es zu keinen Mehrdeutigkeiten kommt (zum Beispiel bezüglich der Reihenfolge der Attribute oder des verwendeten Zeichensatzes). Um dies erreichen zu können, ist eine so genannte Kanonisierung (engl.: Canonicalization, beziehungsweise C14N) des Inhalts erforderlich. Dabei werden unter anderem nach einer Maßgabe des Standards alle Elemente in der Reihenfolge ihres Auftretens aneinander gereiht und alle Attribute alphabethisch geordnet, sodass sich ein längerer UTF8-String ergibt. Aus diesem wird der eigentliche Hash-Wert gebildet, beziehungsweise erzeugt man durch Verschlüsseln den Signaturcode – und ist damit wieder beim Standardverfahren für elektronische Signaturen angelangt (RFC 2437).

Da die Signatur eine binäre Zahlenfolge ist, lässt sie sich nicht direkt in ein XML-Dokument einbetten. Man kodiert die binären Werte im Base64-Format (RFC 1521), um aus ihnen lesbare Zeichen zu gewinnen. Die erhaltene Zeichendarstellung der Signatur findet sich schließlich als <*SignatureValue*> in der XML-Signatur wieder (siehe Listing in Abbildung 1).

Wie man am Beispiel sehen kann, gliedert sich die XML-Signatur in verschiedene Substrukturen. Im Minimalfall enthält eine XML-Signatur nur zwei Elemente: <*SignedInfo*> und <*SignatureValue*>.

<*SignedInfo*> enthält eine oder mehrere Referenzen auf die zu signierenden Daten sowie deren Hash-Werte. Außerdem ist hier der Signaturalgorithmus festgelegt. Falls vor dem Signieren noch eine oder mehrere Transformationen durchgeführt werden sollen, zum Beispiel eine Kanonisierung oder eine XLS-Transformation, ist dies auch im SignedInfo-Element definiert.

In <*SignatureValue*> ist die eigentliche Signatur gespeichert. Da die Hash-Werte aller Referenzen aneinandergehängt und zusammen verschlüsselt werden, existiert nur ein Signaturwert.

iX-TRACT

- Da elektronische Signaturen aus strukturierten Daten bestehen, liegt es nahe, sie als XML-Dokumente zu repräsentieren.
- Gegenüber anderen Signaturen weisen XML-Signaturen zahlreiche Vorteile auf: Sie sind hersteller- und inhaltsunabhängig und ermöglichen das gleichzeitige Signieren verschiedener Inhalte, die nicht einmal zusammenstehen müssen.
- Neben den hohen Anforderungen seitens der Gesetzgeber und den Kosten für die Infrastruktur ist das noch wenig komfortable Handling das Haupthindernis für den breiten Einsatz der Signaturen.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <Autokaufvertrag xmlns="http://www.ponton-consulting.de/Autokaufvertrag" xmlns:ns1="http://www.ponton-
  consulting.de/Autokaufvertrag" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.ponton-
  consulting.de/Autokaufvertrag file:C:/java/eclipse/workspace/xe/schemas/Autokauf.xsd"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#" xmlns:ocs="http://www.open-contracting.de/types">
 + <SignierterTeil id="SignedPart" parties="Parteien" partyDisplayName="./:Nachname/text()" partyElement="Partei"
    signatures="Unterschriften">
 - <Unterschriften id="Unterschriften">
   + <ds:Signature>
   - <ds:Signature>
     - <ds:SignedInfo>
         <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315" />
         <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
       - <ds:Reference URI="#partyIDRef1077008822505">
           <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
           <ds:DigestValue>2jmj7l5rSw0yVb/vIWAYkK/YBwk=</ds:DigestValue>
         </ds:Reference>
       - <ds:Reference URI="#SignedPart">
           <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
           <ds:DigestValue>TP7IOJ0ttxDZjO8EJmyuzd4crCs=</ds:DigestValue>
         </ds:Reference>
       - <ds:Reference URI="">
         - <ds:Transforms>
             <ds:Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315" />
           - <ds:Transform Algorithm="http://www.w3.org/TR/1999/REC-xpath-19991116">
               <ds:XPath>not(ancestor-or-self::ds:Signature) and not (ancestor-or-self::node() = //@xsi:schemaLocation)
               </ds:XPath>
             </ds:Transform>
           </ds:Transforms>
           <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
           <ds:DigestValue>JZnT6WSryvROS7+f00MgJ1v/Hs4=</ds:DigestValue>
         </ds:Reference>
       </ds:SignedInfo>
       <ds:SignatureValue>PEDNTjRUL7k0kH3TX8iW6UqrH4VS5FCQbOjyRNNYwPJ+4giceAwrwJJkOIYHFzEEpjs3rTc1YKJc
         rkHpO4147OIoGZJcq5+qKq2jg9Wmtc/L5LZUUKjion8u4ckrgKTO4yhFSVSP3JG+BZ0sN6wXDsV1
         4j0DGBuvWNJkOySf9jw=</ds:SignatureValue>
     - <ds:KeyInfo>
       - <ds:X509Data>
           <ds:X509Certificate>MIIC5zCCAc+gAwIBAgIGAPoyQgdtMA0GCSqGSIb3DQEBBAUAMGQxCzAJBgNVBAYTAkVVMRAwDgYD
             VQQKEwdQQVBJTkVUMRkwFwYDVQQLExBDZW50cmFsIFNlcnZpY2VzMSgwJgYDVQQDEx9QQVBJTkVU
             IENlcnRpZmljYXRpb24gQXV0aG9yaXR5MB4XDTA0MDExOTA3NTAxMloXDTA0MDMyMTA3NTAxMlow
             gYoxCzAJBgNVBAYTAkRFMR8wHQYDVQQKExZQb250b24gQ29uc3VsdGluZyBHbWJIMRQwEgYDVQQL
             EwtEZXZlbG9wbWVudDEXMBUGA1UEAxMOTWljaGVsIFdpY2hlcnMxKzApBgkqhkiG9w0BCQEWHHdp
             Y2hlcnNAcG9udG9uLWNvbnN1bHRpbmcuZGUwgZ0wDQYJKoZIhvcNAQEBBQADgYsAMIGHAoGBAKAH
             XeDfRSF1ACO+B0FvxUZAN93qf8YJpMqrNpFGM4TIbRDQWt3MNCdgFy0BVv7sEzEIVOY5R39Q6brP
             9ks2ak2XqV3JsutOXvwd2Y8Z8cyINkH9lRS/Xcw1lIUFK6O/WlQAcJS97g5yDZ8RO5ohBtwdKK8i
             pfHnQrmJEeOKooMzAgERMA0GCSqGSIb3DQEBBAUAA4IBAQAjhM6hXTZ6NBNS+XAeAJEOqhfEjnPI
             9ZesuSYO5NoHPr1FUFx4wUXkVixje8rAoJSol4V9g/Kwhizi/QjwX5u9/5q+tI+Fmk6pXuiERh0z
             j3FsMuxS6mIvCSqT8MKSJZWRnawpfgK5fPOCnwr75QwFmDLTGjvpOHosmv8Gl2KJeVuAd6BCgj6Q
             ISvRs4qKyh7bSVv5ytFp1yI4YeDr4aq0o6iXeI7252E6HIin5qJpUxFfam6b/AQ9POQ249fBdtaY
             6FCEAfgyGQYZPKUFqL+Iq1KM2EfsGCKv1o8zn62rTvZld/bxupAlt4gQypaOVav0oP+Z619fX45W
             38ryEE0n</ds:X509Certificate>
         </ds:X509Data>
       + <ds:X509Data>
         </ds:KeyInfo>
     + <ds:Object>
     + <ds:Object>
       </ds:Signature>
   </Unterschriften>
</Autokaufvertrag>
```

Um aus den binären Werten der Signatur lesbare Zeichen zu erhalten, kodiert man sie im Base64-Format. Die so erhaltene Zeichendarstellung findet sich als *<SignatureValue>* in der XML-Signatur wieder (Abb. 1).

Zusätzlich ist üblicherweise noch ein *<KeyInfo>*-Element beigefügt. Dieses kann ein Zertifikat *<X509Data>* beinhalten, das wiederum den öffentlichen Schlüssel enthält. Es ist auch möglich, den öffentlichen Schlüssel – als *<DSAKeyValue>* in seine Bestandteile zerlegt – mitzuliefern. Das hat den Vorteil, dass alle notwendigen Daten zur Überprüfung der Signatur bereits vorhanden sind. Allerdings sollte man berücksichtigen, dass im Gegenzug Manipulationen einfacher zu bewerkstelligen sind, sofern sich die Authentizität des öffentlichen Schlüssels nicht mit Hilfe eines Zertifikats prüfen lässt.

Einbetten und einbetten lassen

Der Standard erlaubt es, XML-Signaturen in die Struktur eines XML-Dokuments einzubetten. Man unterscheidet dabei drei Varianten, die Signatur mit dem Inhalt in Verbindung zu bringen (s. Abb. 2):

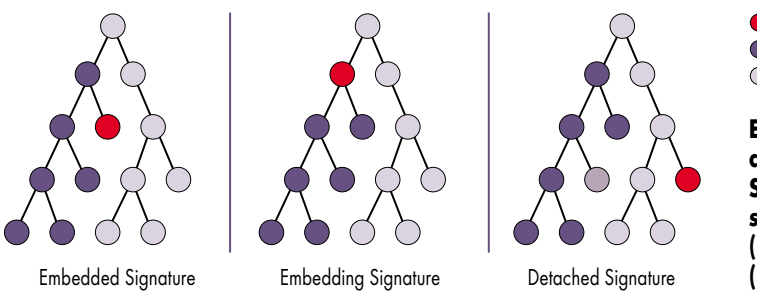
- Bei der Embedded Signature ist die Signatur als Substruktur in den Dokumenteninhalt eingebettet;
- bei der Embedding Signature umschließt die Signatur den Inhaltsteil;
- bei der Detached Signature befindet sich die Signatur an einer Stelle „neben“ dem Inhalt, es gibt also keinerlei Überschneidung.

Nun stellt sich die Frage, wie sich eine Signatur in einen Inhalt einbetten lässt, da dies doch den Inhalt verändert. Ganz einfach: Im Rahmen der Struktur einer XML-Signatur lassen sich Subelemente explizit vom Signieren ausschließen – so auch die Signatur selbst. Umgekehrt lassen sich beliebig viele Referenzen (URIs) auflisten, die gemeinsam „auf einen Streich“ zu signieren sind.

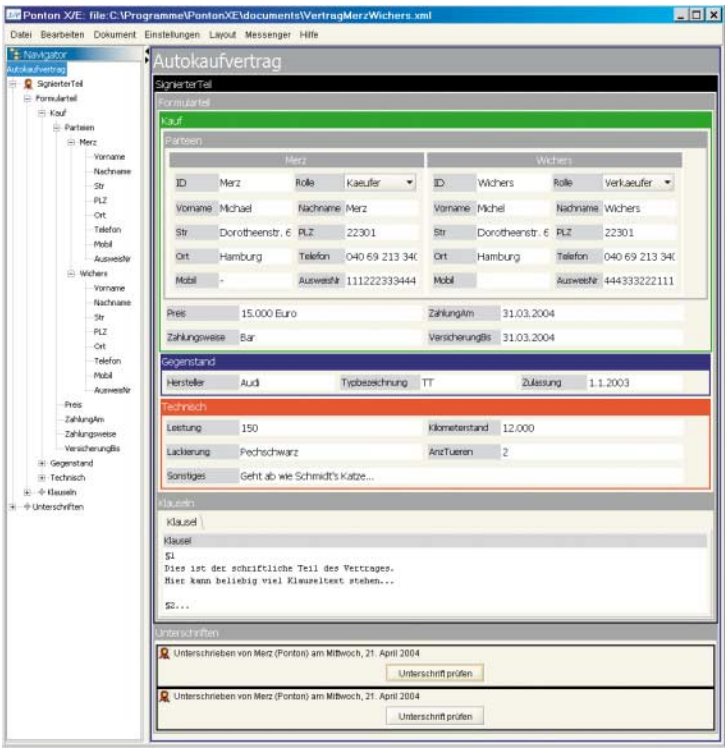
Die Elemente einer XML-Signatur sind einem festgelegten Namensraum zugeordnet (siehe www.w3.org/TR/2002/REC-xmldsig-core20020212/Overview.html#sec-NamespaceContext). Dadurch sind sie logisch von den Nutzdaten getrennt und es gibt keine Namenskollisionen. Voraussetzung ist allerdings, dass der zu verwendende XML-Parser NameSpaces verarbeiten kann.

Von der Wiege bis zur Bahre

Das Schöne an Formularen ist nicht nur, dass sich viele Textfelder ausfüllen lassen, sondern vor allem, dass man sie auch unterschreiben kann. Das gilt für Urlaubsanträge, Steuererklärungen et cetera ebenso wie für Verträge. Während die meisten Formulare nur die Unterschrift einer einzelnen Person erfordern, können es bei Verträgen beliebig viele sein (man denke beispielsweise an eine Vereinsgründung). Bei dieser Kombination „strukturierter Inhalt“ mit „beliebiger Anzahl Unterschriften“ können XML-Signaturen ihre Vorteile



Es existieren drei Varianten, die Signatur mit dem Inhalt zu verknüpfen: Sie kann entweder als Substruktur in den Dokumenteninhalt eingebettet sein (links), oder den Inhaltsteil umschließen (Mitte), oder sich neben dem Inhalt befinden (rechts) (Abb. 2).



Mit einem XML-Editor ist das Signieren dank benutzerfreundlicher Oberfläche wesentlich einfacher und übersichtlicher als in einem XML-Dokument wie in Abbildung 4 (Abb. 3).

le erst richtig ausspielen. Als Beispiel soll im Folgenden ein Autokaufvertrag dienen.

Im Rahmen des EU-Forschungsprojektes OCTANE (Open Contracting TransActions in the New Economy, siehe [1]) wurde ein Vertragseditor entwickelt, der den Umgang mit Formularen und elektronischen Verträgen vereinfacht (Abb. 3). In diesem Umfeld nutzen Teilnehmer den Editor, um etwa Bestellungen manuell in XML zu erfassen und mit Geschäftspartnern auszutauschen (siehe [2]).

Beim Beispieldokument Autokaufvertrag tragen die Vertragsparteien zunächst mit einem beliebigen XML-Editor die üblichen Daten wie Kaufpreis, Leistung, Modell et cetera ein. Um den Signiermechanismus so flexibel einsetzen zu können, dass er vom Dokumenteninhalt unabhängig ist – für elektronische Verträge eine notwendige Voraussetzung –, ist der XML-Editor durch Erweiterungen, die über den XML-Signature-Standard hinausgehen, zu ergänzen:

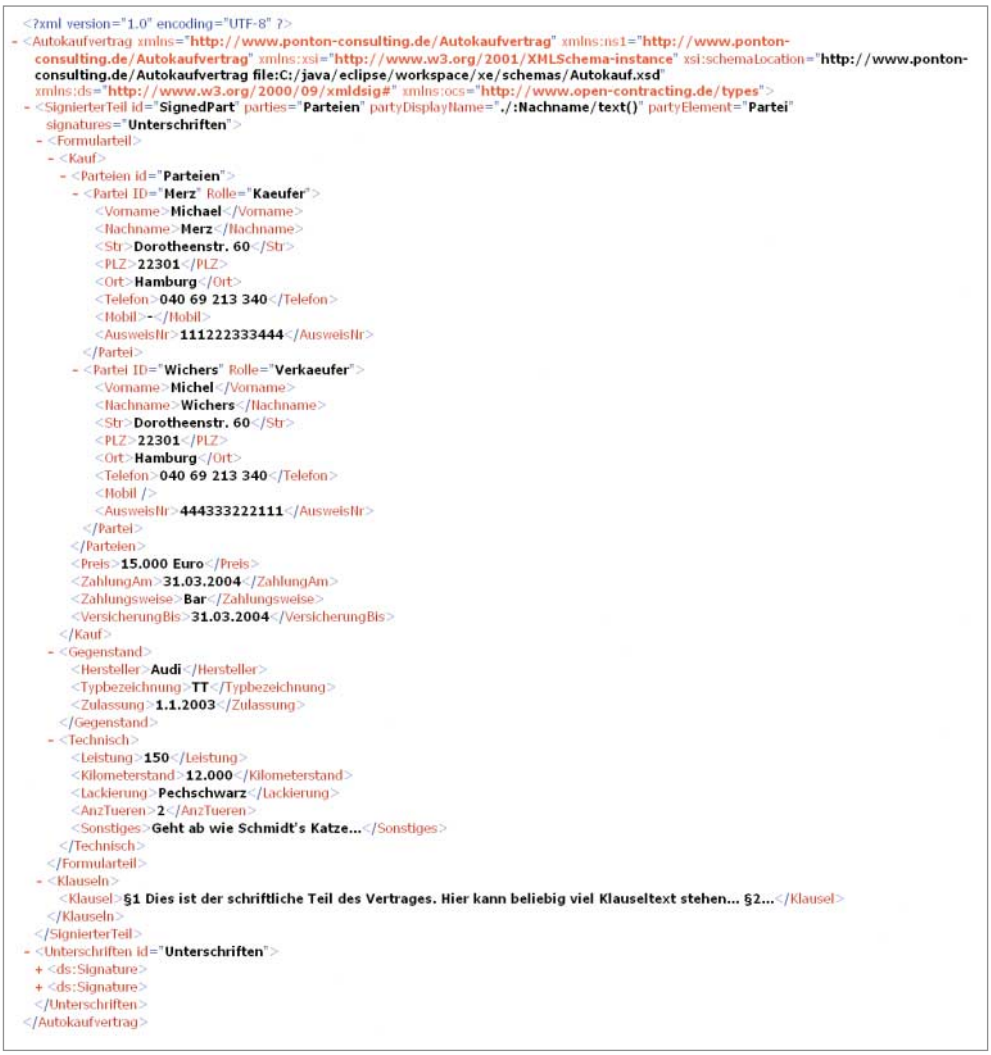
1. Man muss dem Editor mitteilen, an welcher Stelle sich der zu signierende Teil des Dokuments befindet. Dies erfolgt durch ein ID-Attribut mit dem Wert *SignedPart*.

2. Es ist festzulegen, an welcher Stelle im Dokument die Signaturen abzulegen sind. Auf diesen Signatur-Container wird vom <*SignierterTeil*>-Element aus mit dem Attribut *signatures* verwiesen.

3. Schließlich kann man bei Verträgen grundsätzlich davon ausgehen, dass mehrere Parteien unterzeichnen. Es wird vorausgesetzt, dass diese sich als Liste unterhalb eines dedizierten XML-Elements befinden. Im Beispiel ist es das Element <*Parteien*>, das eine Liste von <*Partei*>-Elementen beinhaltet. Im Element <*SignierterTeil*> gibt es dazu die Attribute *parties* und *partyElement*.

4. Ferner wird angenommen, dass ein Subelement in-

XML-Dokument eines signierten Vertrags: Zwar sind alle relevanten Informationen im Dokument enthalten, allein die Übersichtlichkeit lässt zu wünschen übrig (Abb. 4).



```
<?xml version="1.0" encoding="UTF-8" ?>
- <Autokaufvertrag xmlns="http://www.ponton-consulting.de/Autokaufvertrag" xmlns:ns1="http://www.ponton-
  consulting.de/Autokaufvertrag" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.ponton-
  consulting.de/Autokaufvertrag file:C:/java/eclipse/workspace/xe/schemas/Autokauf.xsd"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#" xmlns:ocs="http://www.open-contracting.de/types">
 - <SignierterTeil id="SignedPart" parties="Parteien" partyDisplayName="./:Nachname/text()" partyElement="Partei"
   signatures="Unterschriften">
  - <Formularteil>
   - <Kauf>
    - <Parteien id="Parteien">
     - <Partei ID="Merz" Rolle="Kaeufer">
        <Vorname>Michael</Vorname>
        <Nachname>Merz</Nachname>
        <Str>Dorotheenstr. 60</Str>
        <PLZ>22301</PLZ>
        <Ort>Hamburg</Ort>
        <Telefon>040 69 213 340</Telefon>
        <Mobil>-</Mobil>
        <AusweisNr>111222333444</AusweisNr>
       </Partei>
     - <Partei ID="Wichers" Rolle="Verkaeufer">
        <Vorname>Michel</Vorname>
        <Nachname>Wichers</Nachname>
        <Str>Dorotheenstr. 60</Str>
        <PLZ>22301</PLZ>
        <Ort>Hamburg</Ort>
        <Telefon>040 69 213 340</Telefon>
        <Mobil />
        <AusweisNr>444333222111</AusweisNr>
       </Partei>
      </Parteien>
      <Preis>15.000 Euro</Preis>
      <ZahlungAm>31.03.2004</ZahlungAm>
      <Zahlungsweise>Bar</Zahlungsweise>
      <VersicherungBis>31.03.2004</VersicherungBis>
     </Kauf>
   - <Gegenstand>
      <Hersteller>Audi</Hersteller>
      <Typbezeichnung>TT</Typbezeichnung>
      <Zulassung>1.1.2003</Zulassung>
     </Gegenstand>
   - <Technisch>
      <Leistung>150</Leistung>
      <Kilometerstand>12.000</Kilometerstand>
      <Lackierung>Pechschwarz</Lackierung>
      <AnzTueren>2</AnzTueren>
      <Sonstiges>Geht ab wie Schmidt's Katze...</Sonstiges>
     </Technisch>
    </Formularteil>
   - <Klauseln>
      <Klausel>§1 Dies ist der schriftliche Teil des Vertrages. Hier kann beliebig viel Klauseltext stehen... §2...</Klausel>
     </Klauseln>
   </SignierterTeil>
 - <Unterschriften id="Unterschriften">
   + <ds:Signature>
   + <ds:Signature>
   </Unterschriften>
</Autokaufvertrag>
```

nerhalb der jeweiligen Partei dieselbe identifiziert. Im Beispiel dient das Element *<Nachname>* als Wert für das Attribut *partyDisplayName*.

Die erweiterte Signaturfunktion des XML-Editors prüft nun, welche Parteien im Dokument definiert sind, und bietet eine entsprechende Liste zur Auswahl an. Der Benutzer wird aufgefordert, im Namen der ausgewählten Partei zu unterzeichnen.

Anschließend wird die über den Vertragsteil erzeugte XML-Signatur in dem Unterschriften-Container platziert. In der Praxis würde der erste Vertragspartner den seinerseits unterzeichneten Vertrag nun an den zweiten senden, der ebenfalls unterzeichnet. In jedem Fall ist nur eine XML-Datei auszutauschen – zum Beispiel per E-Mail.

Wenn jemand die Authentizität der Signaturen überprüfen will, so braucht er lediglich aus den Signaturelementen im Kontextmenü die entsprechende Funktion aufzurufen. Sowohl die Integrität als auch die Authentizität und Nichtabstreitbarkeit des Vertrages sind somit gewährleistet.

What you see is what you sign

Die Möglichkeiten der XML-Signaturen sind aufgrund der Kombination der zu signierenden Bestandteile vielfältig: Verschiedene Substrukturen eines Dokuments oder mehrerer Dokumente lassen sich „auf einen Streich“ unterschreiben. Dazu erlaubt es der Standard, eine Kaskade von Transformationen festzulegen, die erst durchlaufen werden muss, bevor am Ende die Erzeugung des Hash-Wertes erfolgt. Man kann bei diesem Prozess beispielsweise einige Teile des Dokuments aussparen und andere durch den Einsatz von XSLT-Stylesheets umwandeln.

So flexibel dies sein mag, es führt jedoch auch zu zusätzlichem Betrugspotenzial. Man stelle sich vor, ein Stylesheet, das beim Signieren zum Einsatz kam, verändert den Inhalt auf andere Weise als ein weiteres Stylesheet, das man beim Verifizieren verwendet. Im Vertragsbeispiel kann sich das auf die technischen Daten, auf den Preis oder sonstige Vertragsklauseln beziehen. Zusammen mit der Signatur wird lediglich die URI des Stylesheet abgelegt. Wenn sich an der dortigen Stelle später ein ganz anderes Stylesheet befindet, lässt sich nicht mehr beweisen, welches zum Signieren ursprünglich verwendet wurde.

Die Lösung besteht in der Einbeziehung des Stylesheet in die Signatur: Man referenziert die betreffende XSLT-Datei und schließt sie in den Signiervorgang ein. Gleiches kann mit der Schemadatei erfolgen, sodass sich auch hier einem nachträglichen Austauschen vorbeugen lässt.

In der Praxis wird es jedoch aus juristischen und ergonomischen Gründen schwierig sein, im Zusammenhang mit einer XML-Signatur Transformationen durchzusetzen, die die zu signierenden Dokumente stark verändern, indem sie andere Elemente referenzieren. Zum einen wäre hier das Prinzip „What you see is what you sign“ verletzt, nämlich einen Inhalt so darzustellen, wie er letztlich in die Hashcode-Generierung einfließt. Zum anderen ist einem Benutzer nur in begrenztem Maße zuzumuten, neben dem zuvor erstellten Dokument vor dem Signieren auch noch ein transformiertes – vom Original abweichendes – zu überprüfen.

Grundsätzlich interoperabel

Man kann XML-Signaturen relativ einfach selbst erzeugen, wenn man eine reguläre Kryptographiebibliothek benutzt. Problematischer ist allerdings das Verifizieren der XML-Signaturen, da eine Vielzahl von Schlüsselrepräsentationen, Datentransformationen und Ähnliches implementiert sein muss, um mit anderen Implementie-

rungen interoperabel zu sein. Glücklicherweise gibt es schon einige Bibliotheken, die das Erstellen und Verifizieren ermöglichen. Das W3C hat inzwischen mehrere Interoperabilitätstests mit ihnen durchgeführt, wobei nahezu alle verfügbaren Produkte zumindest grundsätzlich interoperabel sind (www.w3.org/Signature/2001/04/05-xmldsig-interop.html).

Von der Apache Group ist eine freie Bibliothek sowohl für Java als auch für C++ verfügbar. Weiter gibt es eine Implementierung von IBM, die für akademische Zwecke lizenzfrei ist. Darüber hinaus existieren diverse kommerzielle Implementierungen, beispielsweise von Iaik, Phaos und Infomosaic (siehe www.w3.org/Signature/Overview.html).

Beachtet werden muss hier allerdings, dass viele XML-Signatur-Bibliotheken auf generische Kryptographiebibliotheken aufsetzen, die ihrerseits Lizenzkosten verursachen können.

Das aus dem EU-Projekt hervorgegangene kostenlose Tool Ponton X/E zum Erstellen und Signieren elektronischer Verträge steht auf der Website von Ponton Consulting zur Verfügung (www.ponton-consulting.de/de/download).

Fazit

Elektronisches Signieren ist inzwischen ein gängiges Verfahren und technisch beherrschbar. Dennoch leistet sich kaum ein Normalsterblicher eine solche Signatur im täglichen Leben. Der Grund liegt zum einen in den hohen Anforderungen des Gesetzgebers, von denen Benutzer und Systemarchitekten befürchten, sie nicht erfüllen zu können, und zum anderen in den hohen Kosten für die entsprechende Infrastruktur. Und schließlich ist das elektronische Signieren vielen Menschen nicht „haptisch“ genug: Es ist ein kryptischer Prozess, der nicht nachvollziehbar tief im Inneren eines Rechners abläuft. Jede noch so gefällige Benutzerschnittstelle kann dabei nicht das Vertrauen der händischen Unterschrift vermitteln.

XML-Signaturen helfen, den Prozess des Signierens zu flexibilisieren und ihn nachvollziehbarer zu gestalten. So wird die Signatur im XML-Editor vom Benutzer als separate Substruktur des Dokuments wahrgenommen und er kann sie auch im XML-Text wiederfinden. Visualisierung und XML-Repräsentation liegen dabei so nahe beieinander, dass das Benutzervertrauen dadurch erheblich gestärkt wird. Und nicht zuletzt steht die Gesamtzahl an Transformations- und Visualisierungstechniken der XML-Welt zur Verfügung, um sowohl mit signierten Dokumenten als auch mit ihren Signaturen umzugehen. (ur)

MICHAEL MERZ,
MICHAEL HÄUSLER und
MICHEL WICHERS

haben am EU-Projekt Octane mitgewirkt.

Literatur

[1] Michael Merz; E-Business und E-Commerce; dpunkt Verlag 2002

[2] Michael Merz; E-Business; Handel mit Format; B2B-Integration mit ebXML; *iX* 2/2004, S. 90

ADRESSEN IM WEB

W3C	www.w3c.org
PKC-Standards	www.rsasecurity.com
XML-Editor	www.ponton-consulting.de/de/download