基于 XML 的电子公文多重签名研究与实现

祝胜林, 肖德琴, 林丕源
(华南农业大学 信息学院, 广东 广州 510642)

摘要: 基于 XML 签名规范, 实现 XML 形式电子公文的多重数字签名方案, 利用 XPath 变换选择出签名者负责任的部分, 大大提高了签名的效率和灵活性.

关键词: XML; 电子公文; XML 签名; 广播多重签名
中图分类号: TP393.08 文献标识码: A 文章编号: 1001－411X (2005) 01－0115－04

Research and implementation on multisignature of electronic official documents based on XML

ZHU Sheng-lin, XIAO De-qin, LIN Pi-yuan
(College of Information, South China Agric. Univ. Guangzhou 510642, China)

Abstract: A multisignature scheme of XML electronic official documents based on XML is applied using the XML signature specification, in which every participant signer can select his responsible portion of a document by XPath conversion. This scheme improves the efficiency and flexibility in multisignature procedures.

Key words: XML; electronic official document; XML signature; broadcasting signature

XML(eXtensible Markup Language, 可扩展置标语言)由于其可扩展、结构化语义以及与传输协议无关的特性充分满足 Internet 和分布式等异构环境的要求, 成为了网络上数据传输的主要载体. XML 能够灵活地表示数据、以 XML 数据库存储数据和各种形式操作数据, 所以基于 XML 的电子公文格式可以比较好地解决电子政务应用中所面临的问题[1]: ①大部分电子政务系统的信息孤岛; ②不同的系统表示数据的格式千差万别; ③内容提取效率不高, 需要进行数据再加工; ④各个应用系统需要反复、重复录入数据. 目前, 我国已经启动了电子政务标准化工作, 2002 年 8 月 8 日召开了首批 6 个标准项目的启动会, 2003 年 1 月 15 日,《基于 XML 的电子公文格式规范》形成了最终征求意见稿. 随着《基于 XML 的电子公文格式规范》成为电子政务的公文标准, XML 形式的电子公文将会大量出现在各种各样的电子政务系统中, 由于开始设计 XML 时没有过多地考虑它的安全问题, 并且 XML 是纯文本的和自解释性的, 网络上任何人都能读懂其中的信息和比较容易地修改它, 如何来保证它的安全传输和验证就变得十分重要. 此外, 在电子公文的批复等处理的过程中, 常常需要多个人对 1 份电子公文进行签名, 即多重签名. 本文基于 XML 签名方案充分利用 XML 的结构特点, 实现了 XML 形式的电子公文的多重签名.

1 传统的多重签名应用模式及局限性

原始的多重签名过程: (1)将相同的整个文档的拷贝发送至每个签名者; (2)签名者利用签名算法进行签名; (3)参与签名者的签名连接起来作为文档的多重签名. 这种模式签名和验证的效率都比较低, 因为签名的大小与参与签名的人数成正比, 验证签名的时间等于独立地验证所有参与签名者个人的签名时间的总和.

1983 年, Itakura 和 Nakamura[2] 基于扩展的 RSA 模式提出第一个多重签名方案, 该方案克服了原始的多重签名的缺点, 实现了多重签名的大小与

收稿日期: 2004－07－08 作者简介: 祝胜林(1969－), 男, 讲师, 硕士; E-mail: zhushenglin@21cn.com

签名者的个数是独立的、无关的，并且与单个签名长度一致. 从那以后，不同的多重数字签名方案相继被提出. 根据签名过程的不同，传统的多重签名可以分为2类：有序多重数字签名和广播多重数字签名. 多重数字签名方案包括消息发送者、消息签名者和消息验证者；广播签名方案还包括签名收集者[3].

2001年，Tzong-Chen Wu[4]提出使用文档分解实现授权多重签名模式，允许签名者对他们负责任的子文档签名，提高了签名和通信的效率. 由于在分解文档时只是基于某些平衡策略和假设每个文档能被细分为子文档集，并没有利用XML文档结构化特点，所以也就无法保证分解成的子文档是有意义的，即该模式并不完全适合XML形式的电子公文.

因此，传统的多重签名应用模式存在如下局限性：(1)互操作性差：一方面传统的数字签名产生的值是一些BASE64码，并没有附带上下文信息，验证方无法从签名值获得摘要和签名的方法，只有与签名方事先进行约定，否则无法验证签名. 另一方面签名方将原始消息与签名值进行合并，验证方必须提前了解合并的格式，否则同样无法验证签名. (2)签名的效率低：虽然验证效率与单个签名是一样的，但签名时，需要每个签名者对整个文档进行签名，所以效率低. (3)灵活性差：虽然可以基于某种平衡策略，将文档细分为子文档，但难于保证子文档真正有意义.

2 基于XML的电子公文的多重签名方案

基于XML的多重签名方案是利用XML的签名规范[5]，并且结合基于离散对数的广播多重数字签名，实现XML形式的电子公文的签名方案. 方案的重点在于利用XPath变换选择每个签名者负责任的部分和生成XML签名.

2.1 XML签名规范

该规范包含的内容比较丰富，下面仅简单描述其签名和验证的步骤：

2.1.1 *签名生成* 在生成签名的过程中，首先通过一个统一资源标识符(URI)来标识需要签署的资源或者数据对象. 再按以下步骤生成签名：①计算每个资源的摘要信息；②创建<SignedInfo>元素；③生成签名值；④创建<Signature>元素.

2.1.2 *签名验证* 包括下面2个步骤：①验证在每个<SignedInfo>元素的<Reference>元素中所包含的摘要(即引用验证)；②验证从<SignedInfo>元素计算得来的签名值和<SignatureValue>元素内的值是否相等(即签名验证).

2.2 XPath变换

假设待签名的XML文档如下：

```
<?xml version= 1.0 encoding= UTF-8 ?>
<PurchaseOrder xmlns= urn: purchase-order >
    <Customer>
        <Name> Robert Smith </Name>
        <CustomerId> 788335 </CustomerId>
    </Customer>
    <Item partNum= C763 >
        <ProdutId> 6883-JF3 </ProductId>
        <Quantity> 3 </Quantity>
        <ShipDate> 2002-09-03 </ShipDate>
        <Name> ThinkPad </Name>
    </Item>
</PurchaseOrder>
```

对待签名XML文档应用如下XPath变换：

```
<Transform Algorithm = http://www.w3.org/TR/1999/REC-xpath-19991116 >
    <XPath> ancestor-or-self::Item </XPath>
</Transform>
```

变换的结果如下：

```
<Item partNum= C763 >
    <ProductId> 6883-JF3 </ProductId>
    <Quantity> 3 </Quantity>
    <ShipDate> 2002-09-03 </ShipDate>
    <Name> ThinkPad </Name>
</Item>
```

2.3 基于XML的电子公文的多重签名方案

本方案采用基于离散对数的多重签名方案，它包括3个阶段：密钥生成阶段、签名生成阶段和多重签名验证阶段. 第1和第3阶段与传统的签名方式相同[6]. 下面只说明第2阶段.

在签名方案中，G表示签名者集合，u_i表示签名者，$u_i \in G$. M表示待签的电子公文. $T=\{t_1, t_2, \cdots, t_m\}$表示XPath变换规则的集合. $W=\{w_1, w_2, \cdots, w_m\}$表示应用$T$后细分成的子文档，C表示XPath处理程序，则$w_i=C_{t_i}(M)$. T_i和M_i分别代表T的子集和M的子集，并授权给签名者u_i. h表示杂凑函数，如SHA、MD5. “‖”表示连接运算符. 下面列出多重签名生成步骤：

(1)文档分发者将{h(M), M_i, T_i}和{h(T), h

$(M)\}$分别发送给 u_i 和签名收集者.

(2)u_i 从授权给他的 M_i 中摘录得到子文档 w_i，所有的签名者同样摘录得到授权的子文档，之后合作检验 M 的一致性，公式如下：

$$h(M)=h(w_1 \| w_2 \| \cdots \| w_m)$$

(3)u_i 从接受到的 T_i 中摘录得到变换规则 t_i，并计算 $w_i=C_{t_i}(M)$，再验证接收到的 w_i 与计算得到的 w_i 是否相等，如果不相等，则结束签名；如果所有签名者摘录得到的子文档被成功验证，每个签名者随机选择一个整数 $z_i \in Z_q$，计算：

$$r_i=a^{z_i} \bmod p \text{ 和 } R_i=r_i^{h(T_i \| r_i)} \bmod p$$

再将 R_i 发送给其他参与签名的签名者和签名收集者.

(4)计算：

$$R=\prod_{u_i \in G} R_i \bmod p \text{ 和 } s_i=(z_i h(T_i \| r_i)R+x_i h(h(M \| R)))\bmod p,$$

其中，x_i 是签名收集者秘密传送给 u_i 的签名密钥，计算后将$\{T_i, r_i, s_i\}$发送到签名收集者. 在计算 s_i 的公式中既包含了 T_i 也包含了 M，这样尽管 u_i 只是对规则 T_i 进行签名，仍然可以保证 u_i 应用 T_i 中的规则 t_i 对 M_i 进行签名.

(5)签名收集者检查 T 的完整性：从收到的 T_i 中摘录得到 t_i，计算 $h(T)=h(t_1 \| t_2 \| \cdots \| t_m)$，判断第(1)步收到 $h(T)$和计算的 $h(T)$是否相等.

(6)验证每个签名者的签名：签名收集者通过(4)计算 R，并检验下面方程：

$$r_i^{h(T_i \| r_i)R}=(a^{s_i})(y_i^{h(h(M) \| R)})(\bmod p),$$

其中，y_i 是 u_i 的公钥.

(7)生成多重签名：如果之前步骤生成的所有的个人签名被成功验证，签名收集者计算：

$$S=\sum_{u_i \in G} s_i \bmod q,$$

并且发布(R, S)作为签名者集合 G 对 M 的多重签名.

3　方案实现

采用 Java 2 SDK 1.4.1 开发工具，Apache Xerces-1-4-3 的 XML 处理器，Apache Xalan-J-2-5-1 的XPath 处理器，以及 IBM's XML Security Suite 的 XML 签名 API[7] 组成的开发环境来实现上述方案. 其体系结构为一个基于 Browse/Server/Database(B/S/D)的 3 层体系结构(图 1)：客户端使用 XML 浏览器访问 Web 服务器，通过 Web 服务器访问 CA 服务器或信息服务器. Web 服务器提供文档中转，发布和访问 CA 服务器与信息服务器；CA 服务器提供安全注册，安全信息发布服务(发布证书和撤销列表)；信息服务器中存储各种格式的数据，譬如：数据库、XML 资源和各种应用等信息. 客户通过客户端计算机向 CA 服务器进行注册获取自己的证书，或者通过 CA 服务器查询其他客户的证书和撤销列表来获取它们的证书. 客户对生成的 XML 文档进行签名后，通过 Web 服务器提交到信息服务器分类存储；其他被授权的用户通过 Web 服务器查询到相应的信息，从签名者的证书获取他的公钥就可以对已签名的 XML 文档进行验证.

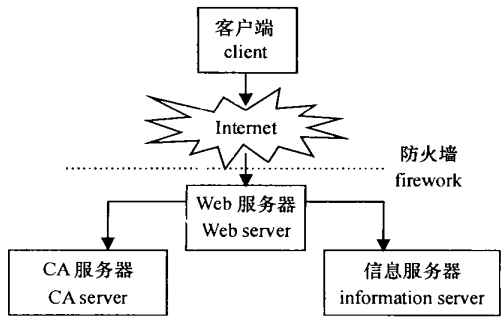

图 1　签名方案的 3 层体系结构

Fig. 1　A three-layer hierarchical structure of signature scheme

3.1　签名生成过程

(1)生成一个待填充的签名 DOM 树：TemplateGenerator siggen = new TemplateGenerator(doc, XSignature.SHA1, Canonicalizer.W3C2, SignatureMethod.RSA).

(2)对生成的DOM 对象添加参考元素. 根据签名的3 种形式，处理方式稍有不同：①分离签名(Detached)：Reference ref=siggen.createReference(URI). ②)封装签名(Enveloping)：Element objectElement=siggen.wrapWithObject(resElement, id). ③被封装签名(或称嵌入签名，Enveloped)：Reference ref=siggen.createReference(#ID name). 添加参考元素后，还需要设置 Transforms，可以使用 addTransform(String algorithm)、addXPathTransform(String expression)或者 addTransform(Element transform).

(3)构造 KeyInfo 元素，可以从 KeyStore 或其他方式获取密钥信息.

(4)将 KeyInfo 信息插入签名元素中，KeyInfo.insertTo(sigElement).

(5)生成一个签名上下文(SignatureContext)实

例: SignatureContext sigContext = new SignatureContext(); 如果应用程序需要, 还需要调用 SignatureContext 类相关的设置函数. 譬如: setIDResovler() 函数、setEntityResolver(EntityResolver).

(6)调用 sign 方法进行签名, sigContext.sign(sigElement, key).

(7)将 XML 签名输出处理, 可以输出到指定文件或保存到数据库中.

3.2 签名验证过程

(1)创建 SignatureContext 实例和设置参数, 与 3.1 中(5)相同.

(2)准备公钥: 可以从 KeyInfo 元素中获得, 也可以从 KeyStore 中获取.

(3)验证: Validity validity = sigContext.verify(sigElement, key).

3.3 多重签名实现要点

在多重签名的实现过程中, 需要SA 为每个签名者生成互不相同密钥对, 再将签名私钥通过秘密信道传送给相应的签名者, 并且公开团体公钥和每个签名者的公钥. 文档分发者根据签名消息和签名者的领域知识构建 XPath 规则集合 $T = \{t_1, t_2, \cdots, t_m\}$, 通过 T 将 M 细分成子文档集合 $W = \{w_1, w_2, \cdots, w_m\}$, 并将它们发送给相应的签名者. 签名者进行验证后签名.

签名程序有关部分是: ref.addXPathTransform(ancestor-or-self:: Item). 每一个签字者对自己负责人的部分(规则)进行签名, 最后由签名收集者收集产生一个多重签名作为团体的签名. 验证者通过团体的公钥可以验证多重签名.

4 结束语

本文提出的基于 XML 的电子公文多重签名方案的安全性是基于离散对数求解的难度, 充分利用了 XML 文档的树形结构特点, 应用 Xpath 变换规则将大的文档分解为有意义的子文档, 参与签名的签名者只需对自己负责的子文档签名, 甚至只需对代表子文档的 XPath 变换的简单表达式签名, 因此, 降低了通信量和签名计算量, 提高了签名生成的效率.

参考文献:

[1] 李光亚. 国家标准《XML 在电子政务中的应用指南》项目工作组 EGS/WG2 工作汇报[EB/OL]. http://www.egs.org.cn/, 2003-03-28.

[2] ITAKURA K, NAKAMURA K. A public-key cryptosystem suitable for digital multisignatures[J]. NEC Research and Development, 1983, (71): 1-8.

[3] 肖德琴, 祁 明, 彭丽芳. 电子商务安全保密技术与应用[M]. 广州: 华南理工大学出版社, 2003.9.

[4] WU T C, HUANG C C, GUAN D J. Delegated multi-signature scheme with document decomposition[J]. The Journal of Systems and Software, 2001, (55): 321-328.

[5] W3C XML Signature Working Group. XML Signature Syntax and Processing[EB/OL]. http://www.w3.org/TR/xmldsig-core, 2002-02-12.

[6] LU E J L, CHEN R F. An XML multi-signature scheme [J]. Applied Mathematics and Computation, 2004, (149): 1-14.

[7] IBM's XML Security Suite. XML Signature Implementation[EB/OL]. http://www.alphaworks.ibm.com/tech/xmlsecuritysuite, 2003-09-28.

【责任编辑 周志红】